REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE
MINISTERE DE L'ENSEIGNEMENT SUPERIEUR ET DE LA RECHERCHE SCIENTIFIQUE
UNIVERSITE D'OUM EL BOUAGHI
Faculté : sciences exactes et des sciences de la nature et de la vie

Thèse

POUR OBTENIR LE DIPLOME DE

Doctorat 3ème Cycle

Filière : Informatique

Spécialité : Imagerie pour systèmes embarqués

Thème :

Aide à la conception des systèmes embarqués intelligents

Présenté Par :
MECIBAH Zina

Thèse soutenue le 08/04/2025 devant le jury composé de :

N°	Nom et prénom	Grade	Etablissement	Qualité
01	**MOKHATI Farid**	Prof.	Université Larbi Ben M'Hidi - OEB	Président
02	**BOUTEKKOUK Fateh**	Prof.	Université Larbi Ben M'Hidi - OEB	Rapporteur
03	**BENABOUD Rohallah**	MCA	Université Larbi Ben M'Hidi - OEB	Examinateur
04	**MENASSEL Rafik**	MCA	Université Larbi Tébessi -Tébessa	Examinateur
05	**NESSAH Djamel**	MCA	Université Abbes Laghrour- Khenchela	Examinateur


بسم الله الرحمن الرحيم

Remerciements

Εl-Hamdou Li ALLAH, le Tout-Puissant, pour m'avoir accordé la force morale et physique nécessaire à l'achèvement de cette thèse.

Je tiens à exprimer toute ma gratitude au **Prof.BOUTEKKOUK Fateh**, mon directeur de thèse, pour la confiance qu'il m'a témoignée et pour son encadrement tout au long de ces années.

Je tiens à exprimer mes sincères remerciements au **Prof. MOKHATI Farid** pour l'honneur qu'il me fait en acceptant de présider le jury de cette thèse. J'adresse également toute ma gratitude au **Dr. BENABOUD Rohallah,** au **Dr. MENASSEL Rafik** et au **Dr.NESSAH Djamel** pour avoir généreusement accepté d'en être membres examinateurs. Leur expertise et leurs précieux retours contribueront, sans nul doute, à enrichir la qualité de ce travail.

Je souhaite également exprimer ma reconnaissance au **Dr. Gueram Tahar** pour son soutien et ses précieux conseils.

Enfin, Je présente mes remerciements à tous mes collègues du laboratoire RELA(CS) [2].

Dédicaces

Je tiens tout d'abord à exprimer ma profonde gratitude à **mon cher père**, à qui je dédie ce travail. Il a été, tout au long de sa vie, une source inépuisable d'inspiration, de sagesse et de courage. Bien qu'il ne soit plus parmi nous pour voir aboutir ce projet et partager les fruits de ses sacrifices, son souvenir reste vivant dans mon cœur. Sa présence, bien que désormais invisible, m'accompagne chaque jour et continue de guider mes pas avec amour et détermination,

رحمك الله أبي الغالي و أسكنك فسيح جناته

A **ma très chère mère**, dont l'amour inconditionnel, les encouragements constants et les innombrables sacrifices ont été pour moi une source inestimable de force et d'inspiration. Merci du fond du cœur pour ton soutien indéfectible et ta présence réconfortante à mes côtés, dans les moments de joie comme dans ceux de difficulté. Tu as toujours cru en moi, même lorsque je doutais de moi-même, et tes prières, tes paroles réconfortantes et ton sourire plein de bienveillance ont illuminé mon chemin. Je te suis infiniment reconnaissant pour tout ce que tu as fait et continues de faire pour moi. Que Dieu te protège, t'accorde une longue vie en bonne santé et te comble de bonheur, car tu es et resteras mon plus grand trésor,

A **mon cher mari, BOUNAB Saleh**, pour la patience, la compréhension et le soutien indéfectible dont il a fait preuve depuis le jour de notre mariage. Je tiens à te témoigner toute ma gratitude et à exprimer, du fond du cœur, mon amour pour toi. Merci infiniment pour tout ce que tu es et tout ce que tu fais pour moi,

A **ma puce FARAH**. Ton sourire éclaire mon monde et ton amour me donne la force de persévérer dans tout ce que j'entreprends. Merci d'être une source constante de bonheur et de tendresse dans ma vie,

A **mes chers sœurs** : Malika, Amina, Sabrina, Souad et Choubaila, ainsi que leurs maris et leurs enfants,

A tous ceux qui m'aiment,

A tous ceux qui m'ont dit : quand termines-tu ton travail ?

Ainsi qu'à tous ceux qui, par un mot, m'ont donné la force de continuer .....

Résumé

De nos jours, de nombreux systèmes embarqués (SEs) doivent s'adapter automatiquement en cours d'exécution pour répondre aux conditions changeantes ou aux exigences évolutives, nécessitant ainsi la capacité d'auto-adaptation. Cette catégorie de SEs est largement utilisée dans des secteurs tels que les maisons intelligentes, les systèmes automobiles, les réseaux de communication, la surveillance environnementale, et bien plus encore. Malgré sa prévalence, les recherches sur la conception de haut niveau des systèmes embarqués auto-adaptatifs (SEAA) restent limitées, en particulier dans le domaine de l'ingénierie des exigences (IE). De plus, il n'existe actuellement aucune référence établie pour guider le développement de l'IE spécifiquement pour les SEs. Notre recherche vise donc à:

1. Développer un métamodèle (MM4SAES) définissant les concepts essentiels et les relations à prendre en compte dans le développement des SEAAs en s'appuyant sur le modèle MAPE-K.
2. Proposer, sur la base de MM4SAES, un processus complet d'ingénierie des exigences pour les systèmes embarqués auto-adaptatifs (REP4SAES). Ce processus intègre trois approches clés de l'ingénierie des exigences pour les systèmes auto-adaptatifs : la mitigation des incertitudes à travers des exigences flexibles, la modélisation des exigences pour le comportement adaptatif (objectifs d'adaptation) en tant que méta-exigences (exigences concernant les exigences régulières du système), et l'examen des exigences pour le comportement fonctionnel des boucles de rétroaction.
3. Propose, durant la phase de spécification des exigences, un nouveau profil SysML nommé SysML4SAS visant à enrichir les diagrammes d'exigences et le diagramme de définition de blocs avec de nouveaux stéréotypes et relations.
4. Faciliter notre processus grâce à une suite d'outils permettant de rationaliser et simplifier le flux de travail de l'ingénierie des exigences.

Mots clés : Système embarqué; Auto-adaptatif ; Processus de conception; Conception de haut niveau; Ingénierie des exigences; Métamodèle; MAPE-K; Boucle de rétroaction; SysML.

Abstract

In today's world, many embedded systems (ESs) need to adapt dynamically at runtime in response to changing conditions or evolving requirements, necessitating the capability for self-adaptation. This class of ES is widely used across sectors such as smart homes, automotive systems, communication networks, environmental monitoring, and more. Despite its prevalence, there remains limited research into the high-level design of self-adaptive embedded systems (SAES), particularly within the field of requirements engineering (RE). Additionally, no established references currently exist to guide RE development specifically for ES. Our research thus seeks to:

1. Develop a metamodel (MM4SAES) that defines the essential concepts and relationships to consider in the development of SAES based on MAPE-K.
2. Proposes, based on MM4SAES, a comprehensive requirements engineering process for self-adaptive embedded systems (REP4SAES). This process incorporates three key approaches to requirements engineering for self-adaptive systems: mitigating uncertainties through relaxed requirements, modeling requirements for adaptive behavior (adaptation goals) as meta-requirements (requirements concerning the system's regular requirements), and examining requirements for the functional behavior of feedback loops.
3. Proposes, during the requirements specification phase, a new SysML profile named SysML4SAS aimed at enriching requirement diagrams and the block definition diagram with new stereotypes and relationships.
4. Facilitate this process through a suite of tools that streamline and simplify the requirements engineering workflow.

Keywords:

Embedded System; Self-Adaptive; Design Process; High-level design, Requirement Engineering; Metamodel; MAPE-K; FeedBack Loop; SysML.

الملخص

في عالم اليوم، تحتاج العديد من الأنظمة المدمجة إلى التكيف ديناميكيًا أثناء التشغيل استجابةً للظروف المتغيرة أو المتطلبات المتطورة، مما يستدعي القدرة على التكيف الذاتي. يُستخدم هذا النوع من الأنظمة المدمجة على نطاق واسع في مجالات متعددة مثل المنازل الذكية، أنظمة السيارات، شبكات الاتصال، مراقبة البيئة، وغيرها. و على الرغم من انتشارها، لا تزال الأبحاث حول التصميم عالي المستوى للأنظمة المدمجة ذاتية التكيف محدودة، خاصة في مجال هندسة المتطلبات. علاوة على ذلك، لا توجد مراجع معتمدة حاليًا لتوجيه تطوير هندسة المتطلبات الخاصة بالأنظمة المدمجة.

تهدف أبحاثنا إلى:

1. تطوير ميتاموديل (MM4SAES) يحدد المفاهيم الأساسية والعلاقات التي يجب مراعاتها في تطوير الأنظمة المدمجة ذاتية التكيف، وذلك بناءً على نموذج MAPE-K .
2. اقتراح عملية شاملة لهندسة المتطلبات للأنظمة المدمجة ذاتية التكيف (REP4SAES) تعتمد على MM4SAES. تدمج هذه العملية ثلاث مقاربات رئيسية لهندسة المتطلبات للأنظمة ذاتية التكيف: التخفيف من الشكوك من خلال متطلبات مرنة، ونمذجة المتطلبات للسلوك التكيفي (أهداف التكيف) كمتطلبات فوقية (متطلبات تتعلق بالمتطلبات الاعتيادية للنظام)، ودراسة المتطلبات للسلوك الوظيفي لدورات التغذية الراجعة.
3. اقتراح خلال مرحلة تحديد المتطلبات، تعريف جديد لـ SysML يُسمى SysML4SAS يهدف إلى إثراء مخططات المتطلبات ومخطط تعريف الكتل بأنماط جديدة وعلاقات جديدة.
4. تسهيل هذه العملية من خلال مجموعة من الأدوات التي تبسط وتسهل سير عمل هندسة المتطلبات.

الكلمات المفتاحية: نظام مدمج؛ ذاتي التكيف؛ عملية التصميم؛ التصميم عالي المستوى؛ هندسة المتطلبات؛ ميتاموديل؛ دورة التغذية الراجعة.

Table des matières

Liste des figures

Liste des tableaux

Liste des acronymes

La signification d'un sigle ou d'une abréviation est indiquée lors de la première apparition dans la thèse. De plus, l'abréviation peut être soit en français soit en anglais. Le plus souvent le terme anglais est le plus répandu.

ADELFE	Atelier de Développement de Logiciels à Fonctionnalité Emergente.
AGC	Apollo Guidance Computer.
AGORA	Attributed Goal Oriented Requirements Analysis.
AMAS	Adaptive Multi Agent System.
AORE	Aspect Oriented Requirements Engineering.
AORE4PF	Aspect-Oriented Requirements Engineering for Problem Frames.
API	Application Programm Interface.
ARGM	Aspects in Requirements Goal Models.
ASIC	Application Specific Integrated Circuit.
BDD	Block Definition Diagram.
CAMA	CAN, Martins and Almudi.
CBHSCD	Component Based Hardware Software CoDesign.
CBIC	Cell Based Integrated Circuit.
CBSE	Component Based Software Engineering.
CCodesign	Conventional Codesign.
Codesign	Concurrent design.
CPLD	Complex Programmable Logic Device.
CPS	Cyber Physical Systems.
CPU	Central Processing Unit.
DFG	Data Flow Graph.

DOORS	Dynamic Object-Oriented Requirements System.
DSP	Digital Signal Processor.
EMF	Eclipse Modeling Framework.
ES	Embedded System.
FCodesign	Formal codesign.
FPGA	Field Programmable Gate Array.
FR	Functional Requirement.
FSA	Finite State Automaton.
FSM	Finite State Machine.
GA	Gate Array.
GAL	Generic Array of Logic.
GBRAM	Goal-Based Requirements Analysis Method.
GERSE	Guia de Elicitação de Requisitos para Sistemas Embarcados (Un acronyme portugais).
GORE	Goal Oriented Requirements Engineering.
HLD	High Level Design.
IC	Integrated Circuit.
IDE	Integrated Development environment.
IE	Ingénierie des Exigences.
IEEE	Institute of Electrical and Electronics Engineers.
IHM	Interface Homme Machine.
IoT	Internet of Things.
JAD	Joint Application Development.
KAOS	Knowledge Acquisition in autOmated Specification OR Keep All Objectives Satisfied.
LAB	Logic Array Bloc.
LLD	Low Level Design.
LOTOS	Langage Of Temporal Ordering Specification.
MAPE-K	Monitor, Analyzer, Planner, Executer, Knowledge.
MARTE	Modeling and Analysis of Real Time Embedded systems.

MAS	Multi Agent System.
MM4SAES	Metamodel for Self-Adaptive Embedded System.
NFR	Non-Functional Requirement.
OCL	Object Constraint Language
OMG	Object Management Group.
PAL	Programmable Array of Logic.
PBD	Platform Based Design.
PIA	Programmable Interconnect Array.
PLA	Programmable Logic Array.
PLD	Programmable Logic Device.
PN	Petri Nets.
RAND	Research ANd Development.
RDP	Reconfiguration Dynamique Partielle.
RDT	Reconfigaration Dynamique Totale.
RE	Requirement Engineering.
RELAX	Requirement Engineering Language for Adaptive Systems.
REP4SAES	Requirement Engineering Process for Self-Adaptive Embedded System.
REPES	Requirements Engineering Process for Embedded Systems.
RTOS	Real Time Operating System.
SAES	Self-Adaptive Embedded System.
SAS	Self-Adaptive System.
SE	Système Embarqué.
SEAA	Système Embarqué Auto Adaptatif.
SLR	Systematic Literature Review.
SoaML	Service oriented architecture Modeling Language
SoC	System on Chip.
SoPC	System on Programmable Chip.
SoS	System of System.
SPLD	Simple Programmable Logic Device.

SRD	System Requirements Document.
SysML	Systems Modeling Language.
SysML4SAS	SysML for Self-Adaptive System.
TERASE	Template para Especificação de Requisitos de Ambiente em Sistemas Embarcados (Un acronyme portugais).
TTM	Time To Market.
URD	User Requirements Document.
V&V	Verification and Validation.
VHDL	Very Hign Description Language.
WSN	Wireless Sensor Network.
XMI	XML Metadata Interchange.
XML	eXtensible Markup Language.

INTRODUCTION GENERALE

"Ce que vous faites aujourd'hui détermine votre avenir."

Gandhi

Sommaire:

1. Contexte général de la thèse

Les systèmes embarqués (SEs) sont des dispositifs électroniques intégrés à des machines plus grandes, conçus pour effectuer des tâches spécifiques généralement en temps réel. Ils sont omniprésents dans notre quotidien. Il suffit de jeter un coup d'œil autour de nous pour le constater. Retirer de l'argent à un distributeur, c'est utiliser un SE. Lorsqu'on prend sa voiture, le régulateur de vitesse, la direction assistée ou encore le système de contrôle de trajectoire sont autant de SEs. Même le pilote automatique d'un avion en fait partie. Le système de contrôle d'un drone, bien qu'installé généralement dans une station au sol, est également considéré comme un SE, et les exemples abondent. Dans de nombreux secteurs, ces dernières années, les avancées technologiques ont transformé des fonctions autrefois effectuées manuellement ou mécaniquement, au profit de l'électronique et de l'informatique embarquée. Ce confort accru devient de plus en plus difficile à abandonner. Parallèlement, les systèmes intelligents se distinguent par leur capacité à traiter des données, à prendre des décisions autonomes, et à s'adapter à leur environnement. Donc, ils fournissent une approche pour résoudre des problèmes importants et assez complexes et obtenir des résultats cohérents et fiables dans le temps [1]. En plus, la définition des systèmes intelligents est un problème difficile et fait l'objet de nombreux débats. Du point de vue du calcul, l'intelligence d'un système peut être caractérisée par sa flexibilité, son adaptabilité, sa mémoire, son apprentissage, sa dynamique temporelle, son raisonnement et sa capacité à gérer des informations incertaines et imprécises.

En combinant ces deux concepts, les systèmes embarqués intelligents gagnent en importance, car ils permettent aux dispositifs intégrés d'évoluer, de s'optimiser et de répondre à des situations dynamiques sans intervention humaine. Plus spécifiquement, les systèmes embarqués auto-adaptatifs (SEAA) sont cruciaux dans des environnements complexes et incertains, comme ceux des systèmes critiques de sécurité, où ils peuvent ajuster automatiquement leur comportement pour maintenir les performances même en cas de perturbations. Cela les rend indispensables dans des domaines tels que les véhicules autonomes, les dispositifs médicaux et les infrastructures critiques, où une adaptation en temps réel est nécessaire pour assurer la fiabilité, la sécurité et l'efficacité. Donc, les SEAAs se tiennent à l'intersection de l'informatique embarquée et de l'informatique auto-adaptative.

Dans notre thèse, nous portons une attention particulière aux systèmes embarqués auto-

adaptatif. Plusieurs termes ont été utilisés pour désigner des systèmes dotés de capacités d'auto-adaptation, notamment les systèmes autonomes, les systèmes informatiques organiques (organic computing systems), les systèmes adaptatifs dynamiques et les systèmes auto-adaptatifs. Dans cette thèse, nous utilisons le dernier terme et nous nous concentrons sur les systèmes embarqués auto-adaptatifs basés sur le modèle MAPE-K.

Le modèle MAPE-K (Monitor, Analyze, Plan, Execute - Knowledge) est d'une importance cruciale pour les systèmes embarqués auto-adaptatifs, car il fournit un cadre structuré permettant une gestion efficace de l'auto-adaptation. En surveillant en continu les conditions d'exécution et en analysant les données en temps réel, ce modèle permet aux systèmes de détecter les anomalies et de réagir de manière appropriée. La phase de planification génère des stratégies d'adaptation basées sur l'analyse des données, tandis que l'exécution met en œuvre ces stratégies pour ajuster le comportement du système. De plus, le composant "Knowledge" assure une gestion des connaissances, facilitant l'apprentissage et l'amélioration continue des processus d'adaptation. En intégrant ces mécanismes, le modèle MAPE-K permet d'optimiser les performances, d'améliorer la résilience face aux défaillances et de garantir une réponse appropriée aux changements contextuels, rendant ainsi les systèmes embarqués auto-adaptatifs plus intelligents et réactifs.

La conception d’un système embarqué auto-adaptatif basée sur le modèle MAPE-K suit plusieurs étapes cruciales, parmi lesquelles la phase d'ingénierie des exigences qui s’occupe une place centrale, car elle est primordiale et doit être réalisée avec beaucoup d’attention. En intégrant les exigences dynamiques dans le processus d'adaptation, les systèmes peuvent mieux répondre aux changements et maintenir leur performance et leur fiabilité dans des environnements dynamiques et imprévisibles. C'est pourquoi cette thèse met en évidence le rôle essentiel des exigences dans la conception de ces systèmes.

2. Problématique et motivations

Bien que plusieurs techniques aient été développées pour soutenir l'ingénierie des exigences des systèmes auto-adaptatifs [1], [2], [3], [4], [5], [6], [7], [8], l'élucidation, l'analyse, la spécification, la validation et la vérification des exigences des systèmes embarqués auto-adaptatifs restent largement sous-explorées. Cela conduit souvent à une conception ad hoc de l'auto-adaptation, au détriment de la rigueur nécessaire dans ce domaine complexe. Dans le secteur des systèmes embarqués, cette lacune est particulièrement problématique : plus de 50% des problèmes identifiés lors de la livraison découlent d'une mauvaise compréhension lors de la capture des exigences. Notre revue systématique de la littérature révèle qu'aucune approche ne décrit explicitement comment les exigences d'un système embarqué doivent être élucidées, analysées, spécifiées et validées.

En ingénierie des exigences pour les systèmes auto-adaptatifs, la plupart des travaux supposent que les exigences sont déjà définies, limitant ainsi leur portée à la gestion et au raisonnement autour de l'adaptation. Or, l'ingénierie des exigences demeurent des tâches complexes nécessitant un processus systématique encore à formaliser. Dans ce contexte, la traçabilité des exigences devient cruciale, notamment pour les systèmes embarqués auto-adaptatifs, car elle assure une continuité entre la collecte des besoins et des contraintes du nouveau système jusqu'à la validation finale. Elle facilite également la gestion des impacts des changements, la gestion des dépendances entre les exigences et la justification des décisions prises tout au long du processus. Enfin, le recours à des outils spécifiques permettant de spécifier, documenter et suivre les exigences s'avère essentiel pour gérer l'implication de multiples parties prenantes et répondre à l'évolution des exigences.

3. Objectifs et contributions

L'objectif de notre thèse est de proposer un processus détaillé pour l'ingénierie des exigences des systèmes embarqués auto-adaptatifs (SEAAs). Il convient de noter que ce processus peut également être appliqué aux systèmes embarqués traditionnels, non auto-adaptatifs. Les contributions de cette thèse sont les suivantes :

1. **Élaboration d'un métamodèle unifié (MM4SAES)** : L'un des principaux défis dans le domaine des SEAAs résidera dans l'absence d'une terminologie commune, en raison de la phase exploratoire de la recherche. Pour répondre à ce défi, un métamodèle bien défini, capable de capturer les concepts des SEAAs ainsi que les relations qui les relient, sera élaboré. Ce métamodèle représentera une avancée significative pour améliorer la qualité des exigences dans ce contexte. Idéalement, les exigences se limiteront à décrire le comportement externe du système et ses contraintes, sans inclure d'informations de conception. Cependant, au niveau de détail requis pour spécifier un système complexe, il deviendra essentiel de concevoir une architecture initiale pour structurer la spécification des exigences. Ainsi, un métamodèle nommé MM4SAES (MetaModel for Self-Adaptive Embedded Systems) sera proposé. Ce métamodèle sera enrichi de concepts architecturaux conformes au modèle MAPE-K et complété par des contraintes OCL, permettant d'exprimer des relations complexes difficilement représentables graphiquement.
2. **Définition d'un processus d'ingénierie des exigences (REP4SAES)** : À partir du métamodèle, un processus détaillé pour le développement des exigences des SEAAs basé sur le modèle MAPE-K sera défini. Ce processus, baptisé REP4SAES (Requirements Engineering Process for Self-Adaptive Embedded Systems), intégrera trois approches principales pour l'ingénierie des exigences des systèmes auto-adaptatifs : (1) l'assouplissement des exigences pour atténuer les incertitudes, (2) la modélisation des objectifs d'adaptation sous forme de méta-exigences (c'est-à-dire des exigences portant sur les exigences de base), et (3) la prise en compte des exigences de comportement fonctionnel des boucles de rétroaction.
3. **Génération de plugins EMF pour Eclipse** : Pour valider et documenter les informations produites à chaque étape du processus, tout en assurant la traçabilité des besoins et des contraintes, des plugins pour Eclipse seront développés. Ces plugins seront générés à partir du métamodèle MM4SAES en utilisant Ecore EMF, un outil intégré à la plateforme Eclipse.

4. **Proposition d'un nouveau profil SysML pour les SEAAs** *:* Lors de la phase de spécification des exigences, un profil SysML novateur, baptisé SysML4SAS (Systems Modeling Language for Self-Adaptive Systems), sera introduit. Ce profil sera conçu pour enrichir les capacités de modélisation offertes par SysML, en particulier en ce qui concerne les SEAAs. SysML4SAS apportera de nouveaux stéréotypes et relations adaptés à la spécification des exigences des SEAAs, permettant d'intégrer plus efficacement des concepts clés tels que l'incertitude, les exigences de conscience, et les mécanismes d'évolution. Enrichissant les diagrammes d'exigences, ce profil facilitera la représentation des relations entre exigences invariantes, exigences relaxées, exigences de conscience et exigences d'évolution. De plus, SysML4SAS étendra le diagramme de définition de blocs (BDD) en introduisant des éléments pour modéliser explicitement les composants d'auto-adaptation, tels que les capteurs et les effecteurs. Cette approche renforcera la traçabilité entre les exigences et les solutions architecturales, contribuant à une spécification plus rigoureuse et adaptée aux besoins des SEAAs.
5. **Automatisation de la génération des documents d'exigences** : Pour automatiser la génération des documents d'exigences utilisateur et système, la bibliothèque Apache POI de Java sera utilisée. Cette bibliothèque permettra de créer des fichiers Excel organisant les informations de manière tabulaire. Ces fichiers Excel seront produits à partir de fichiers XMI, eux-mêmes générés par les plugins EMF d'Eclipse. L'objectif sera d'extraire les informations contenues dans les fichiers XMI et de les intégrer dans des fichiers Excel de manière structurée et automatisée grâce à Apache POI.

4. Plan de la thèse

Ce manuscrit est composé de cinq chapitres, les deux premiers chapitres sont consacrés à la présentation d'un état de l'art sur les travaux en liaison avec notre problématique. Ensuite, nos contributions seront présentées dans les trois derniers chapitres.

Donc, Nous évoquerons dans le premier chapitre les concepts relatifs à la conception des systèmes embarqués auto-adaptatif.

Dans le deuxième chapitre, nous aborderons l'ingénierie des exigences des systèmes embarqués auto-adaptatifs.

Au cours du troisième chapitre, nous proposerons un métamodèle pour les systèmes embarqués auto-adaptatif. Notre proposition est nommée MM4SAES pour : Metamodel for Self Adaptive Embedded System.

Dans le quatrième chapitre, nous présenterons le nouveau processus d'ingénierie des exigences pour les systèmes embarqués auto-adaptatifs. Ce processus, nommé REP4SAES (Requirement Engineering Process for Self-Adaptive Embedded Systems), sera détaillé. Nous introduirons également notre profil SysML et les différents outils développés dans ce cadre.

Et dans le dernier chapitre, l'application de notre processus, ainsi que l'utilisation de nos outils dans une étude de cas dans le domaine de la santé concernent un système de surveillance intelligent destiné aux patients atteints de diabète.

Enfin, cette thèse se conclut par un récapitulatif des principales contributions de notre recherche dans le domaine de l'ingénierie des exigences des systèmes embarqués auto-adaptatifs. Nous mettons également en lumière les perspectives futures qu'elle ouvre, en envisageant les pistes de développement et d'amélioration possibles.

Le tableau ci-dessous résume la structure générale de cette thèse en deux parties.

Tableau 0-1 : Plan de la thèse.

ETAT DE L'ART	**CHAPITRE 01 :** La conception des systèmes embarqués auto-adaptatif	La présentation générale des méthodologies de conception des systèmes embarqués ainsi que des méthodologies de conception des systèmes embarqués auto-adaptatif.
	CHAPITRE 02 : L'ingénierie des exigences des SEAAs	La présentation des notions de bases ainsi qu'un état de l'art sur l'ingénierie des exigences des systèmes embarqués et un état de l'art des travaux les plus pertinents sur l'ingénierie des exigences des systèmes embarqués auto adaptatifs.
CONTRIBUTIONS	**CHAPITRE 03 :** Métamodèle pour les systèmes embarqués auto-adaptatifs basé sur MAPE-K	La présentation de la démarche de réalisation de notre revue systématique de la littérature (SLR), couvrant les études de 1980 à 2024, inclut la méthodologie de développement de notre processus, ainsi que le processus de création de notre métamodèle. Enfin, une étude comparative avec d'autres métamodèles souligne l'importance de notre proposition.
	CHAPITRE 04 : Un processus d'ingenierie des exigences pour les SEAAs – REP4SAES	La présentation de notre processus détaillé pour l'ingénierie des exigences des systèmes embarqués auto-adaptatifs, de notre nouveau profil SysML et de nos outils.
	CHAPITRE 05 : Etude de cas	L'application de notre processus, ainsi que l'utilisation de nos outils dans une étude de cas portant sur un système de surveillance intelligent destiné aux patients atteints de diabète.

Etat de l'art

CHAPITRE 1 : La conception des systèmes embarqués auto adaptatif

« *Qui veut faire quelque chose trouve un moyen, qui ne veut rien faire trouve une excuse.* »

Proverbe arabe

SOMMAIRE :

1. Introduction

Les systèmes embarqués (SE) traditionnels sont généralement fermés et fonctionnent dans un environnement bien connu et fixe (prévisible) où toutes leurs configurations possibles sont connues durant la conception. Cependant, les SEs doivent aujourd'hui être opérationnels 24/7, malgré que, dans de nombreux cas, les conditions de fonctionnement changent fréquemment de manière imprévisible.

La plupart des approches traditionnelles des systèmes critiques prennent en compte les scénarios les plus défavorables au moment de la conception pour les systèmes critiques avec des contraintes de synchronisations strictes, par contre, les scénarios des cas moyens sont pris en compte dans le cas des systèmes temps réel souples. Par conséquent, l'utilisation des ressources d'un SE comme les CPU[1], la mémoire et l'énergie sont généralement surestimées. Donc, En raison de la complexité croissante des SEs et des contraintes de coût strictes y inhérentes, les approches statiques ne sont plus envisageables.

La situation des SEs change radicalement dans des environnements imprévisibles très dynamiques, où toutes les configurations possibles ne sont pas connues d'avance ou sont incertaines. Dans ce cas, l'adaptation automatique de ces systèmes aux différentes situations est très utile car elle aide à réduire les coûts et à augmenter leur fiabilité. De nos jours, L'informatique auto-adaptative est devenue un sujet de recherche très actif (dans les deux communautés : matérielles et logicielles) comprenant de nombreux axes de recherche tels que la modélisation de l'incertitude, la prédiction, le contrôle, la vérification et la mise en œuvre.

Dans le domaine matériel, les composants matériels deviennent de plus en plus auto-adaptatifs grâce à la reconfiguration dynamique et automatique. Selon [9] « *les technologies reconfigurables permettent des phases de développement et de prototypage rapides par rapport à la conception d'un circuit spécifique ASIC*[2] ». Donc, Les technologies reconfigurables offrent une solution intermédiaire entre les ASIC et les processeurs, permettant ainsi un compromis entre la puissance de calcul (supérieure à celle des processeurs) et le degré de flexibilité (plus élevé que celui des ASIC). En outre, Les systèmes reconfigurables les plus souvent utilisés dans le domaine des SEs sont les réseaux logiques programmables (FPGA[3]).

[1] Central Processing Unit.
[2] Application Specific Integrated Circuit.
[3] Field Programmable Logic Device.

Dans le domaine des logiciels, ces derniers doivent être auto-adaptatifs pour répondre aux exigences de spécification en évolution rapide et à l'environnement dynamique imprévisible.

De plus et contrairement aux systèmes embarqués traditionnels qui étaient centrés et fermés, les systèmes embarqués de nos jours sont plus en réseau ; utiliser la technologie sans fil comme les WSN[1] « réseaux de capteurs sans fils » et ouvrir avec un support qui permet l'installation en ligne de logiciels tiers. Donc, l'auto-adaptation complique considérablement la conception du système embarqué à développer et crée de nouveaux défis pour sa vérification, sa prévisibilité et sa sécurité. Par exemple, l'adaptation d'un composant dans un système déclenche souvent des réactions en chaîne provoquant d'autres adaptations dans d'autres composants, des configurations incohérentes et instables peuvent être atteintes. Une adaptation incontrôlée peut conduire à un comportement émergent et imprévisible, ce qui la rend inappropriée pour la sécurité des systèmes critiques. D'autre part, La gestion de l'incertitude est un autre problème majeur causé par l'adaptabilité.

Le présent chapitre constitue une présentation générale des méthodologies de conception des systèmes embarqués ainsi que des méthodologies de conception des systèmes embarqués auto-adaptatif. Une brève description des systèmes embarqués est d'abord donnée. Ensuite, une revue de différentes méthodologies de conception des systèmes embarqués est présentée avec les avantages et les limites de chaque méthodologie. Par la suite, une description des systèmes embarqués auto-adaptatif (les approches d'adaptation, la gestion de l'incertitude ainsi que les niveaux d'adaptation) est donnée. A la fin de ce chapitre, les méthodologies de conception des systèmes embarqués auto-adaptatif seront passées en revues.

[1] Wireless Sensor Network.

2. Système embarqué

2.1. Définition

Suite à l'apparition d'Apollo Guidance Computer (AGC) dans les années 60 [10], Le concept de systèmes embarqué (SE) a vu le jour. Il est connu encore beaucoup plus avec l'invention du premier microprocesseur (Intel 4004). Au début, ces systèmes contenaient un seul processeur mais actuellement ils peuvent même contenir des centaines.

Le terme de «système embarqué» a été traduit de l'expression anglaise «embedded system», qui signifie «système diffus» ou «système enfouis». Selon [11] *« un système embarqué est un système contenant du matériel et du logiciel complètement intégré dans l'environnement qu'il contrôle. Il est généralement autonome, exécute une tâche précise dédiée à une application spécifique, remplace souvent des composants électromécaniques, n'a généralement pas de périphérique standard et possède une interface IHM*[1] *qui peut être aussi simple qu'une diode électroluminescente qui clignote ou aussi complexe qu'un système de vision de nuit en Temps Réel*».

L'architecture d'un système embarqué (voir la Figure1-1) peut varier selon les systèmes. Généralement, si le SE est autonome et indépendant, donc on ne trouve pas de systèmes auxiliaires. En revanche, l'architecture de base d'un SE est généralement composée de:

a. **Capteurs** : utilisé pour recueillir les informations de l'environnement comme la température, mouvement, vibration…etc.
b. **Mémoire** : L'espace mémoire des SEs peut être très limité pour les petits microcontrôleurs embarqués dans des systèmes embarqués simples, jusqu'à plusieurs Giga de Flash pour des processeurs avec des systèmes d'exploitation Linux. Il est donc primordial d'identifier précisément les besoins de l'appareil, afin de concevoir un SE adapté.
c. **Unité de traitement (CPU)** : Le processeur est le cœur d'un SE. Il prend des entrées et produit une sortie après le traitement des données. Ainsi, le processeur traite les données pour mesurer la sortie et les stocker dans la mémoire
d. **Actionneurs (également appelés effecteurs)** : utilisé pour exécuter les actions comme l'électrovanne, aimant, moteur, bobine…etc.

[1] Interface Homme Machine.

Donc, Un SE reçoit des informations de l'environnement par le biais de capteurs. Ensuite, Il mémorise et traite ces informations avec des algorithmes. Enfin, Il renvoie des informations vers l'environnement par le biais d'actionneurs.

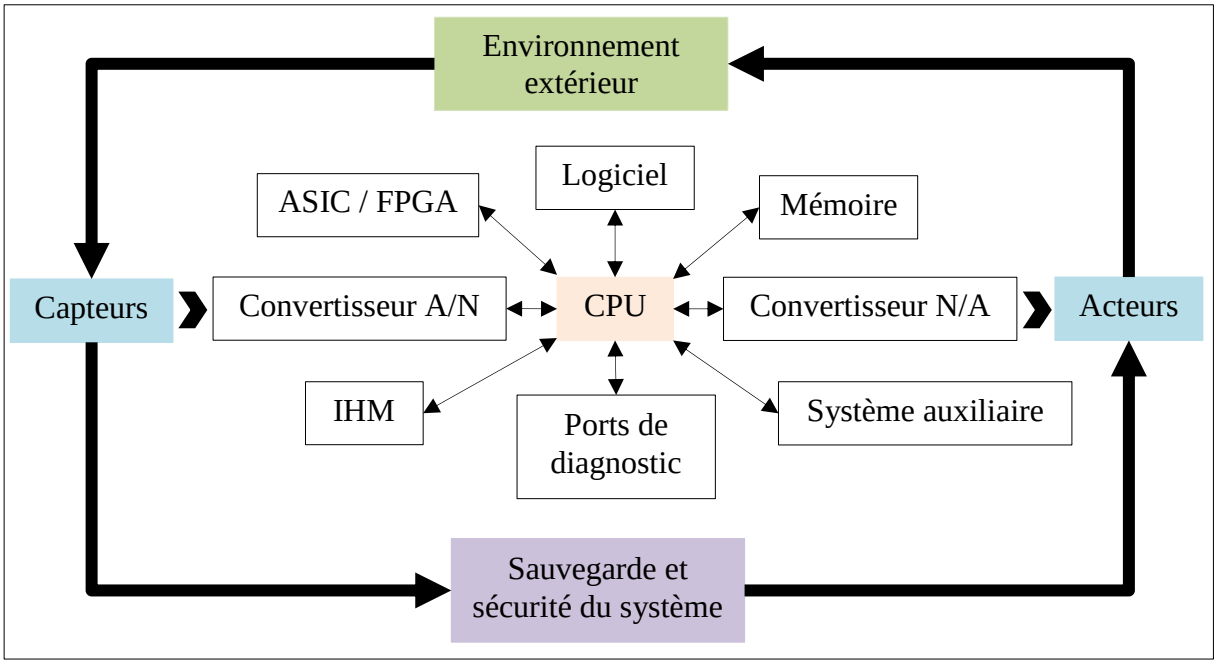


Figure 1-1 : Modèle de Système Embarqué.

2.2. Domaines d'application

Contrairement à un ordinateur, un SE n'a pas de raison d'être seul. Il est généralement un composant faisant partie d'un système plus large. Par exemple, actuellement la plus part des voitures sont équipées d'un système de climatisation de l'air contrôlé par un système embarqué. Ce dernier collecte de l'information sur l'humidité et la température de l'air au sein de la voiture et, sur la base de cette dernière, active ou non le climatiseur. Un autre exemple, le vélo à assistance électrique. Il dispose d'un ou plusieurs capteurs (pour détecter la couple, la vitesse et la pression sur la pédale), le moteur qui contient généralement le processeur et l'actionneur, et l'IHM (écran tactile ou bien un écran en couleur de haute définition, commande vocale…) pour lire diverses informations comme la vitesse.

Donc, les SEs se retrouvent vraiment partout, dans le transport (automobile, ferroviaire, avionique), dans les équipements mobile et bureautiques (Copieurs, Répondeurs, Imprimante, Téléphone portable), dans l'équipement d'un bâtiment (escalators, ascenseurs, système de surveillance, Systèmes d'éclairage, Contrôle d'accès). Ils sont aussi présents dans le monde industriel (les chaines de production, stations de contrôle). Notons que, les SEs dans le monde

industriel doivent être beaucoup plus robustes afin de supporter des conditions bien plus strictes que celles des applications destinées au grand public.

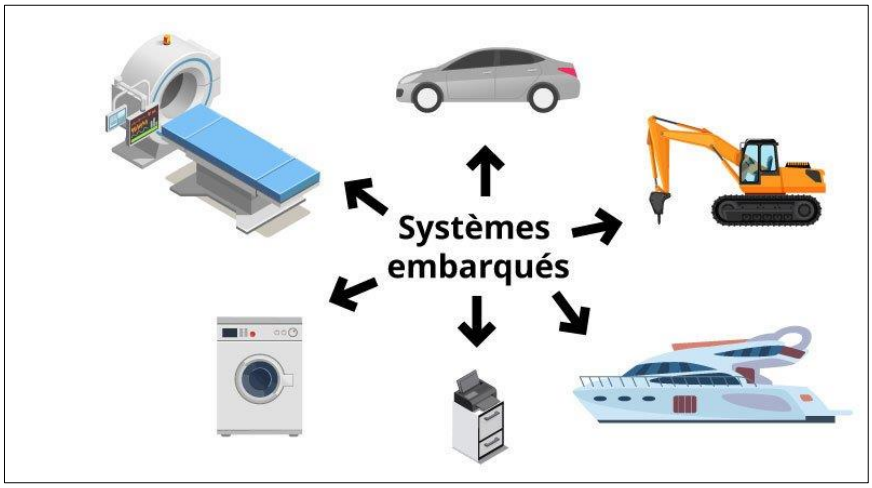


Figure 1-2 : domaines d'application des systèmes embarqués.

2.3. Classification des systèmes embarqués

2.3.1. Selon le type d'application visé

Selon le type d'application visé, On peut distinguer quatre principaux types de SEs:

a. Les systèmes embarqués à usage général : similaire aux applications bureau mais embarqué. Comme par exemple téléphone portable, assistant personnel…etc.
b. Les systèmes embarqués de contrôle en temps réel : Système de navigation aérien.
c. Les systèmes embarqués pour le traitement du signal : Sonar, Radar.
d. Systèmes embarqués pour les réseaux et communications : Téléphone, routage, transmission de données.

On peut retrouver certains SEs dans plusieurs catégories.

2.3.2. Selon la taille et la complexité

Selon la taille et la complexité, on peut distinguer trois principaux types de SEs :

a. **Système embarqué de petite échelle** : la conception de ces systèmes se retrouve généralement au niveau d'une carte et les ressources, en particulier en termes de capacité de stockage, sont très limitées pour limiter la dissipation de l'énergie.
b. **Système embarqué de moyenne échelle** : Le matériel et le logiciel de ces systèmes sont plus complexes

c. **Systèmes embarqués sophistiqués** : ont une complexité matérielle et logiciel très grande.

2.4. Contraintes des systèmes embarqués

Quel que soit le champ d'application du SE, il est soumis à de nombreuses contraintes :

a. **Time to market (TTM)** : ou bien temps de mise sur le marché, est très important dans l'industrie des SEs car le marché est concurrentiel. TTM est définit comme suit : « *le temps qu'il faut à un nouveau produit pour passer du concept à sa mise en vente* ».
b. **Le cout de production** : il faut réduire le cout de production. Pour cela, il faut réduire les matières utilisées pour la fabrication.
c. **Ressources limitées** : les ressources disponibles d'un SE sont généralement limitées.
   - **la consommation d'énergie** : est un point critique pour les SE avec autonomie (système alimenté par des batteries), si la consommation est excessive donc le prix de revient est augmenter car il faut utiliser des batteries de forte capacité.
   - **Capacité de stockage** : dans les SEs, la capacité de stockage de données est généralement limitée.
   - **Puissance de calcul** : dans les SEs, la puissance de calcul (nombres d'opérations/seconde) pour effectuer les différents tâches est généralement limitée.
   - **Taille limitée.**
d. **La fiabilité et la sécurité** : le SE doit être toujours fonctionner correctement et doit pouvoir réagir en cas de panne de l'un de ses composants car ils sont utilisés dans des applications de plus en plus critiques dans lesquels leur dysfonctionnement peut générer des catastrophes.
e. **Fonctionnement en Temps Réel** : Dans les systèmes temps réel, les calculs doivent être effectués en réponse à un événement externe. La validité d'un résultat dépend du respect des délais (le moment où il est produit). Le non-respect d'une échéance peut entraîner une défaillance du système. Il est important de noter que la plupart des systèmes embarqués (SE) sont multirates, c'est-à-dire que le traitement des informations se fait à des rythmes différents.

Donc, durant la conception des systèmes embarqués, il est nécessaire de trouver un compromis entre toutes ces contraintes.

3. La conception des systèmes embarqués

Cette section donne un aperçu sur les différentes méthodologies de conception des SEs. Tout d'abord, nous allons commencer par la présentation des différents défis de conception des SEs. Puis, nous expliquons les différents niveaux d'abstraction dans le processus de conception des SEs. Par la suite, nous aborderons brièvement les principales méthodologies de conception des SEs avec les principaux avantages et inconvénients de chaque méthodologie.

3.1. Défis de conception des SEs

Les principaux défis durant la conception d'un SE sont les suivants :

- De quoi avons-nous besoin ?;
- La communication entre les membres de l'équipe matérielle et logicielle ;
- Solution matériel ou bien logiciel ? (Performance VS flexibilité) ;
- Délai de mise sur le marché (TTM) VS le cout de fabrication ;
- Mémoire limitée et puissance de calcul restreinte ;
- Comment choisir l'architecture?;
- Comment minimiser la consommation d'énergie ;
- Et est-ce que le futur système fonctionne vraiment?

Actuellement, Les SEs sont de plus en plus complexes et doivent être produits à un coût de plus en plus bas suivant des cycles de développement de plus en plus courts. Donc, concevoir un SE revient finalement à un exercice d'optimisation c'est-à-dire essayer au maximum de minimiser le coût de production d'un nouveau produit pour des fonctionnalités optimales.

3.2. Niveaux d'abstraction

La figure suivante résume les principaux niveaux d'abstraction dans le processus de conception des SEs. Dans chaque niveau, il faut analyser le système pour déterminer les caractéristiques actuelles et pour l'améliorer dans les prochains niveaux s'il existe des détails manquants.

Il existe deux techniques pour la conception des SEs, Top-down design et Bottom-up design. Dans la première technique on part du plus haut niveau d'abstraction et on « descend »vers le

plus détaillé. Donc, Il faut commencer par la définition des exigences. Puis, dans la phase de spécification, il faut créer une description détaillé de ce que nous voulons (la spécification indique comment le futur système se comporte et pas comment il est construit). Ensuite, dans la phase de conception d'architecture, le but est de décrire comment le système implémente ces fonctions. Malheureusement, un grand nombre de concepteurs négligent la phase d'exigence et de spécification et commence la conception par la définition de l'architecture du système [4]. A l'étape suivante, les concepteurs déterminent les grands composants matériels/logiciels. Enfin, afin d'obtenir un système fonctionnel, les différents composants construits son mis ensemble. Notons que, l'activité d'intégration n'est pas un simple branchement des différents composants.

Par contre, dans la deuxième technique, on part des composants de base et on « remonte » vers le système.

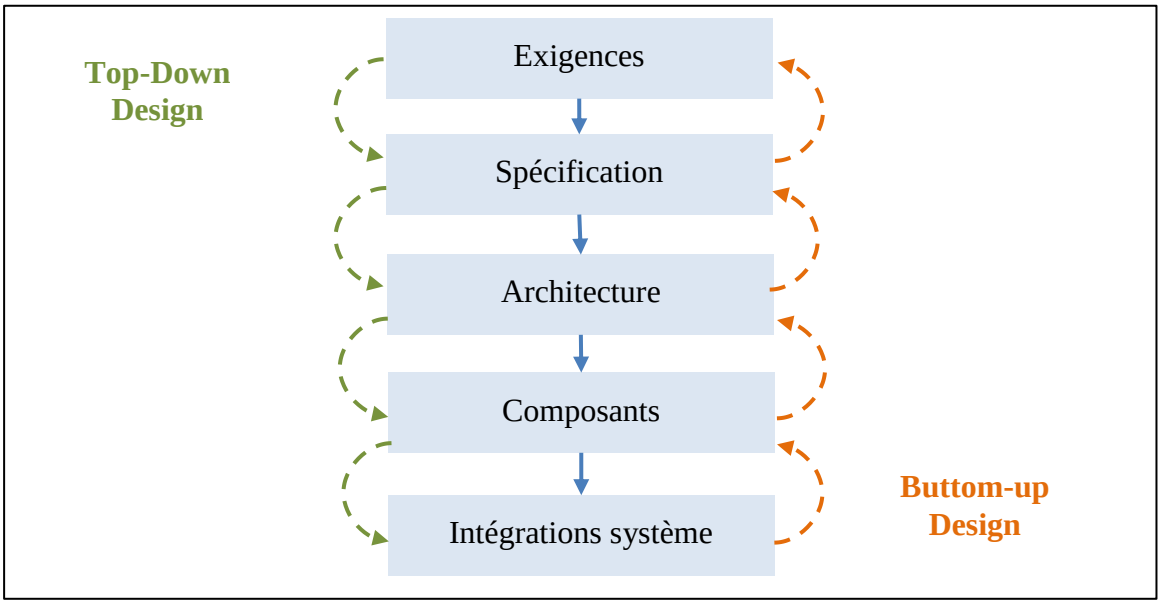


Figure 1-3 : Niveaux d'abstraction dans le processus de conception des SEs.

3.3. Méthodologies de conception des systèmes embarqués

3.3.1. Méthode classique de conception

Depuis longtemps, Les systèmes embarqués sont traités de la manière suivante :

Une fois les différents besoins du nouveau système identifiés, il convient de séparer les éléments qui constitueront la partie matérielle de ceux qui constitueront la partie logicielle. Cette séparation est essentielle, car elle nécessite des compétences distinctes dans les

domaines de l'électronique, de la mécanique et de l'informatique. Le processus décisionnel repose sur les différentes contraintes du système comme niveau de la consommation d'énergie, la surface qu'il occupe, les performances d'exécution …etc. par la suite, une équipe s'occupera de la partie logicielle et une autre équipe concevra l'équipement électronique. En fin du processus de conception, le logiciel sera intégré dans le matériel (voir figure suivante).

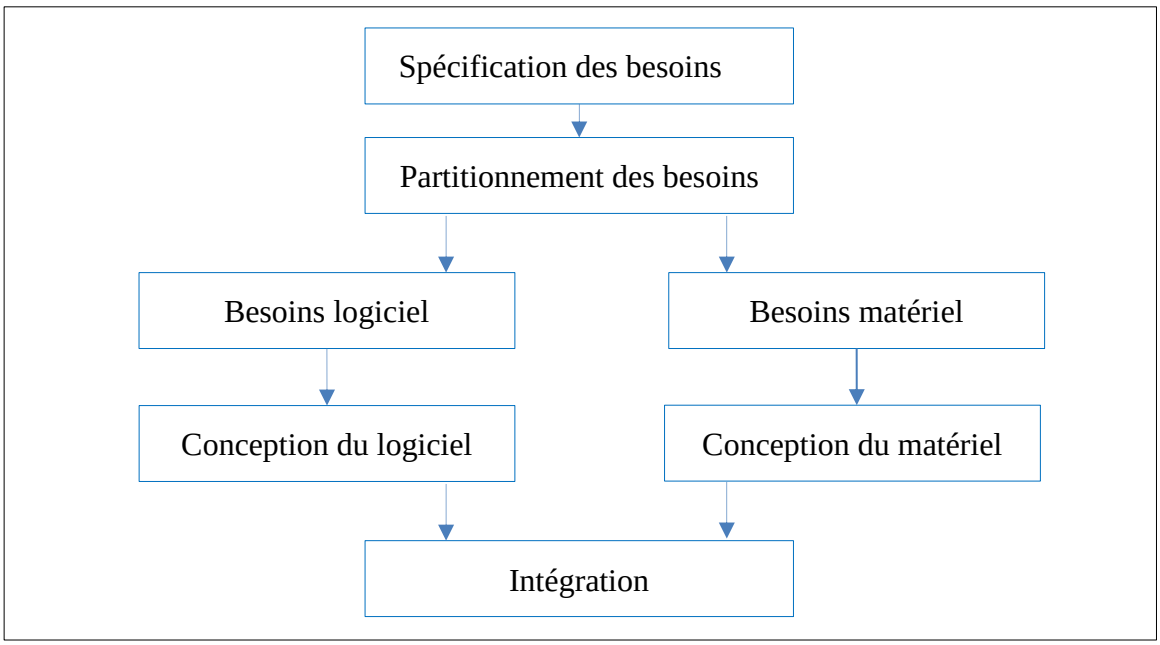


Figure 1-4 : Approche classique pour le développement d'un système mixte [12].

Les principaux problèmes rencontrés avec la méthode classique sont les suivants :

- Réalisé par des ingénieurs indépendants, chaque équipe « matérielle et logicielle » possède souvent une connaissance limitée des fonctionnalités de l'autre. En conséquence, les développeurs logiciels exploitent mal les capacités matérielles et peuvent parfois recréer des fonctions déjà existantes. De leur côté, les spécialistes du matériel ont parfois une vision imprécise des besoins des ingénieurs logiciels, ce qui peut entraîner un mauvais dimensionnement de la partie logicielle du système.
- L'ajustement et/ou le changement tardif des spécifications obligent des fois les concepteurs à refaire le travail.
- La solution proposée étant fortement dépendante de la partition choisie parce que la décision du partitionnement se fait a priori sans expérimentation concrète sur le système cible.
- Dans la phase d'intégration nous pouvons rencontrer des problèmes qui nécessitent parfois d'importantes modifications d'une ou l'autre des parties.

- Cette approche restreint les possibilités d'exploration de différentes solutions où certaines fonctionnalités pourraient migrer du logiciel vers le matériel ou vice versa.

3.3.2. Conception conjointe matériel/logiciel (Codesign)

Pour traiter les différents problèmes de la méthode classique « traditionnelle » de conception des SEs, le Codesign est apparu.

Le terme Hardware/Software Concurrent Design (HW/SW Codesign) et qui se traduit par conception conjointe matériel et logiciel ou simplement Co-conception est apparue comme une nouvelle discipline pour concevoir des circuits intégrés complexes (ICs[1]) au début des années 1990 [13].

Selon [12] « *la Co-conception est une méthode unique de conception, couvrant l'intégralité du cycle de vie du système mixte, en unifiant la conception des parties logicielles et matérielles*». Ainsi, le Codesign permet aux concepteurs de transformer de manière optimale les spécifications d'un système en un produit industriel intégrant à la fois une partie logicielle et une partie matérielle, tout en respectant les contraintes fonctionnelles et non fonctionnelles définies dans le cahier des charges. De plus, cette approche cherche à différer autant que possible les décisions concernant le choix du matériel (ASIC, FPGA, etc.) et du logiciel (firmware, système d'exploitation, etc.) tout au long du processus de conception, contrairement à l'approche traditionnelle où ces choix sont effectués dès le début du projet. Elle cherche à exploiter la synergie entre le matériel et le logiciel afin d'optimiser et/ou de satisfaire les contraintes de conception telles que le coût, les performances et la consommation énergétique du produit final. En parallèle, elle vise également à réduire considérablement le délai de mise sur le marché (TTM[2]).

Notre taxonomie des approches de Codesign des systèmes embarqués les plus importantes est résumé dans le tableau suivant :

[1] Integrated Circuit.
[2] Time-to-Market.

Tableau 1-1 : Taxonomie des approches de Codesign des SEs.

N°	Type	Brève description	Env. / Tools	Réf.
01	CCodesign	Le Codesign traditionnel	POLIS, COSYMA	[13], [14], [15], [16], [17], [18], [19]
02	IP-based Codesign	CCodesign + IPs reuse	Xilinx Vivado Design Suite	[20], [21], [22]
03	Platform-based Codesign	CCodesign + Platform reuse	HSCDE	[23], [24]
04	Model-based Codesign	CCodesign + models transformations technology	SONORA, GASPARD2	[25], [26], [27], [28]
05	Formal Codesign	CCodesign + formal specifications and verifications	PeaCE	[29], [30], [31], [32], [33]
06	CBHSCD	Component Based Hardware Software CoDesign CCodesign + Component	ESIDE	[34], [35], [36], [37], [38], [39]

A. CCodesign

Les étapes du CCodesign sont différentes des étapes de la conception classique. Le graphique de synthèse établi par [16]montre l'enchaînement de celles-ci (voir figure 1-5).

Voici une brève description de ces différentes étapes :

- **La définition des exigences**

Les exigences d'un SE sont les descriptions de ce que le système doit faire, les services qu'il fournit et les contraintes sur son fonctionnement (comme les contraintes de sécurité). Donc, ils traduisent les besoins des clients pour un système qui répond à un certain objectif.

Dans cette étape, il faut bien déterminer toutes les exigences du système (fonctionnelles et non fonctionnelles), car les erreurs à ce stade conduisent nécessairement à des problèmes dans les phases suivantes.

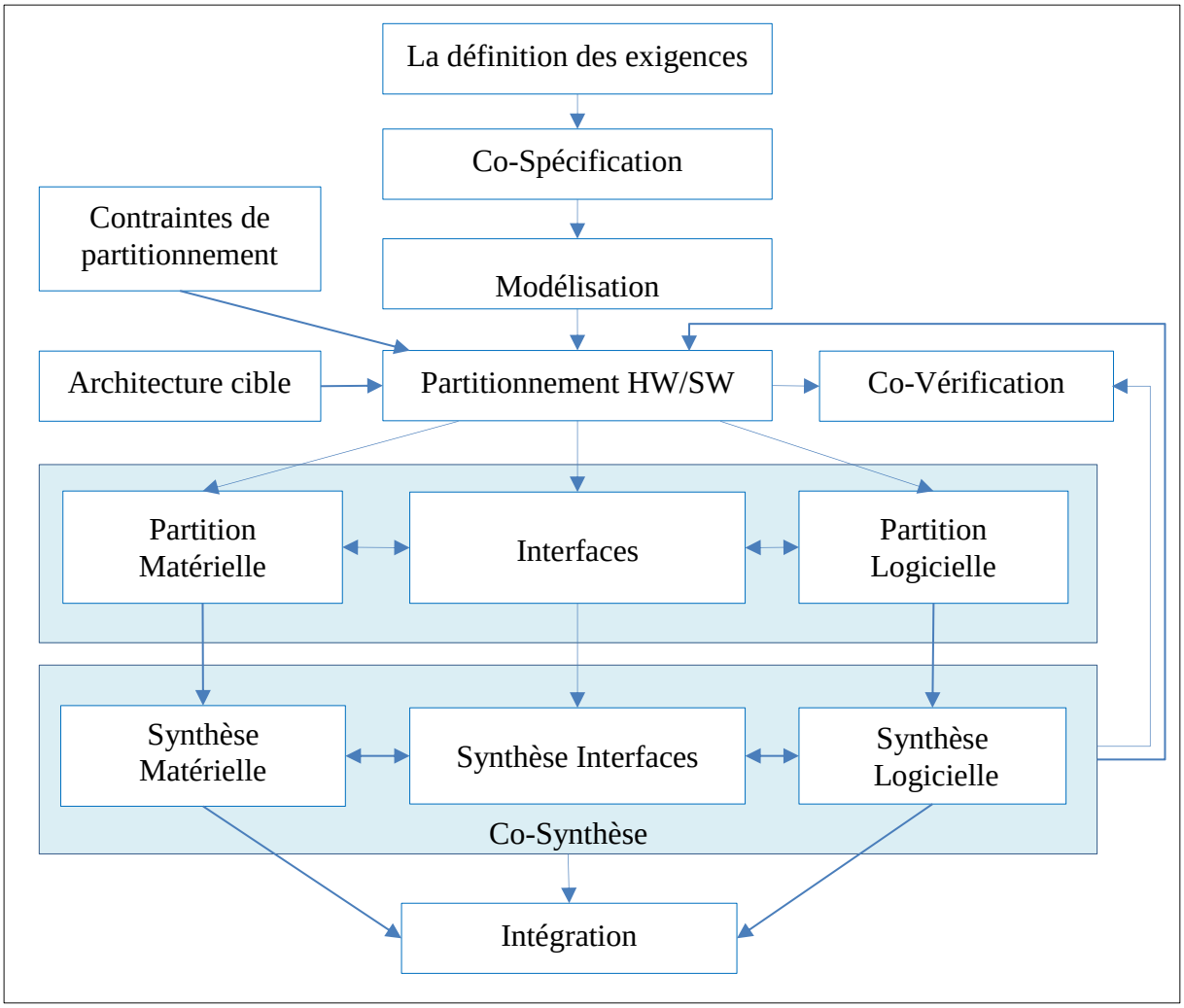


Figure 1-5 : Démarche du Codesign **[5], [16]**.

- **Co-spécification :**

Spécifier un système c'est le définir de manière la plus rigoureuse possible. En d'autre terme, La spécification d'un système est une description de ce système en vue de sa réalisation (Pas de concept de matériel ou de logiciel). Depuis longtemps, le langage naturel était le seul mode de spécification. Il souffre de son manque de précision et ambigüité. Ce qui a conduit à définir des langages mieux adaptés comme le langage SystemC (basé sur C++).

Généralement, pour la spécification des systèmes mixtes, il existe deux méthodes, la spécification homogène et la spécification hétérogène[40]. Le principe de la spécification homogène consiste à utiliser un langage de spécification pour spécifier les composants matériels et logiciel du système. Le grand défi de cette méthode est : comment analyser et partager la spécification entre les parties matérielles et logicielles ?

Il existe quelques méthodologies de codesign qui commencent la spécification avec un langage de haut niveau. Par exemple, Polis [18] qui utilise le langage Esterel pour la spécification.

La spécification hétérogène consiste à utiliser un langage spécifique pour le matériel et un langage spécifique pour le logiciel. Le grand problème de cette approche est l'affectation du matériel et du logiciel avant la phase de partitionnement qui va à l'encontre de l'idée de base du codesign (la séparation entre le matériel et le logiciel le plus tardivement possible). Parmi les méthodologies de codesign qui supportent la spécification hétérogène, CoWare [41] qui utilise le langage VHDL[1] pour le matériel et langage C pour le logiciel.

À l'issue de cette étape, le contrat entre le concepteur et l'utilisateur sera établi. Pour l'utilisateur, La spécification décrit le mode d'emploi et le comportement de l'utilisateur dans toutes les situations. Pour le concepteur, la spécification est l'objectif à atteindre. Pour cette raison, le cahier des charges doit être soigneusement écrit pour refléter fidèlement les exigences du client.

- **La modélisation**

Le processus de conceptualisation et d'affinement des spécifications est désigné sous le terme de modélisation. Le formalisme choisi pour cette modélisation doit être indépendant des technologies de mise en œuvre et ne doit pas imposer une implémentation spécifique pour les systèmes embarqués, afin de préserver la liberté d'explorer diverses solutions potentielles[40].

Il existe plusieurs formalismes pour la modélisation des SEs, comme les automates à états finis (FSA[2] ou bien FSM[3]), les graphes à flots de données (DFG[4]) et les réseaux de Petri (PN[5]).

- **Partitionnement matériel/logiciel**

À cette étape, il s'agit de regrouper les variables et comportements étroitement liés, en se basant sur la spécification et la modélisation du système, puis de déterminer pour chaque regroupement s'il sera réalisé par voie logicielle ou matérielle. Les paramètres de choix entre le matériel et logiciel sont par exemple : coût, poids, temps de conception, temps d'exécution

[1] Very Hign Description Language.

[2] Finite State Automaton.
[3] Finite State Machine.
[4] Data Flow Graph.
[5] Petri Nets.

et consommation d'énergie. En général, si le composant nécessite des performances élevées alors il est réalisé en matériel et si nécessite de la flexibilité alors il est réalisé en logiciel.

Le partitionnement est généralement réalisé en deux étapes[40]: la sélection d'une architecture matérielle et l'allocation des éléments du modèle fonctionnel représentant l'application sur les composants de cette architecture. Il est donc essentiel de disposer d'une connaissance précise, d'une part, des caractéristiques logicielles et matérielles des fonctions qui modélisent l'application, et d'autre part, du modèle d'architecture cible considéré.

L'architecture cible peut être monoprocesseur (constituée d'un processeur) ou distribuée (constituée d'un ensemble de processeurs). Dans le domaine du codesign, il est constitué d'un ensemble de composants matériels (ASIC, FPGA), logiciels (processeurs) et de communication.

Cette étape peut être manuelle (le partitionnement est réalisé par les concepteurs du système), interactive (le partitionnement est assisté par des outils) ou bien automatique (le partitionnement est réalisé complètement par des outils).

Après avoir sélectionné la plate-forme (FPGA ou bien ASIC). Il existe deux approches assez complémentaires qui méritent d'être soulignées :

Dans l'approche Vulcan développée par [42], l'idée était de commencer avec une solution uniquement matérielle (utilise le langage HardwareC pour la spécification). Ensuite, migrer le plus de tâches possible vers un logiciel tout en satisfaisant les contraintes de performances dans le but de réduire les coûts de conception. Par contre, dans le système de conception Cosyma développé simultanément à l'Université technique de Braunschweig [43] a pris exactement le point de départ inverse en commençant par une partition uniquement logicielle (utilise le langage Cx pour la spécification) des blocs avec une migration ultérieure des tâches vers le matériel afin de satisfaire les contraintes de performances tout en essayant de minimiser les coût des blocs matériels résultants.

Donc, à la fin de cette étape, nous allons obtenir les spécifications de tous les composants logiciels et matériels.

- **Co-synthèse**

Selon [40] « *l'étape du Co-Synthèse permet de transformer les descriptions fonctionnelles en descriptions directement implantables sur les processeurs matériels et logiciels de*

l'architecture cible ». Cette étape regroupe les synthèses de la partie matérielle (Synthèse matérielle), logiciel (Synthèse logicielle) et interfaces de communication (Synthèse interfaces).

Dans les SEs, il existe trois type de communication entre les différents composants : communication logiciel/logiciel, communication matériel/matériel et communication logiciel/matériel. En raison de l'hétérogénéité et les différences de vitesse entre les composants matériels et logiciels, le dernier type pose beaucoup problèmes. Les principales approches de synthèse d'interfaces de communication [5] sont l'utilisation des protocoles de communication, des systèmes d'exploitation et des primitives ou des bibliothèques de communication.

- **Co-vérification**

La vérification des SEs (systèmes matériels/logiciels) est appelée Co-Vérificarion. Elle est défini comme suit : «*le processus qui détermine que la conception, à différents niveaux d'abstractions, est correcte* »[40].

Afin d'éviter la propagation des erreurs entre les différentes étapes du processus de conception des SEs et dans le but de garantir que Les performances et les fonctionnalités du système à concevoir sont conformes à la Co-Spécification, une phase de vérification est obligatoire à la fin de chaque étape.

En général, il existe deux techniques pour la Co-Vérification des SEs qui sont : la vérification formelle et la Co-Simulation.

- **Intégration**

Le rôle de la dernière étape du codesign est l'intégration finale des composants matériels et logiciels du système. Malheureusement, durant cette étape peuvent émerger des problèmes qui nécessitent parfois d'importantes modifications dans le matériel ou le logiciel.

B. IP-based Codesign

Dans les années soixante, l'apparition des SoC[1] a remplacé les systèmes embarqués traditionnels bien connus ce qui offre une solution plus pratique et plus rentable (practical and cost-effective) pour la mise en œuvre de systèmes embarqués.

[1] System On Chip.

Selon la loi de Moores, le nombre de transistors intégrés double tous les 18 mois, alors que la conception et la vérification des SoC deviennent plus complexes et sujettes aux erreurs [44]. De plus, selon [45], la complexité de tels systèmes augmente d'année en année alors que la productivité des développeurs de matériel et de logiciels n'augmente pas à un rythme comparable (voir la figure suivante).

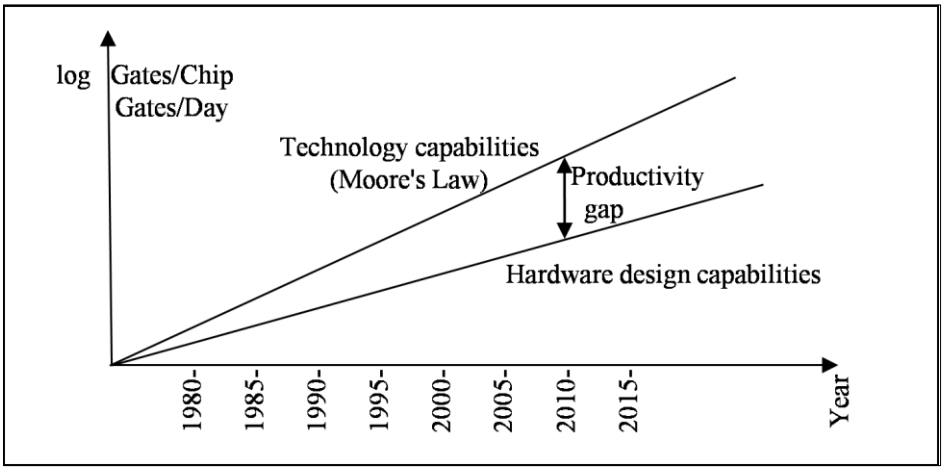


Figure 1-6 : Écart de productivité **[45]** .

Donc, toutes ces raisons (La complexité croissante des SEs principalement les SoC, les contraintes de temps de mise sur le marché et les concepteurs de compétences différentes) ont amené les chercheurs à proposer une nouvelle méthodologie de conception des SEs basée sur la réutilisation d'éléments préexistants (conçus dans des projets antérieurs ou fournis par des fournisseurs externes et vérifiés), appelés « composant virtuel ».

Le concept de composant virtuel est né dans le milieu des années 90, a donné lieu à plusieurs termes pour désigner ces blocs réutilisables : composants virtuels, composants réutilisables, ou plus simplement Intellectual Properties (IPs). Donc, L'IP est un composant électronique matériel ou logiciel réalisant une fonction bien déterminée pouvant être réutilisé par un utilisateur n'ayant pas participé à la spécification de ce composant, il peut être Hard (non modifiable), Firm (avec des possibilités de modification restreintes) ou Soft (avec de grandes possibilités de modification). Il est fourni par un tiers qui le crée et le qualifie pour qu'il soit utilisé par une autre personne ou équipe de travail. Il doit également être classé dans des catalogues sur le Web [44]. Après leur création, leur qualification à l'aide de certains attributs et leur classement dans un catalogue web, l'architecte SoC, appelé aussi L'intégrateur IP ou simplement concepteur SoC [44] pourrait rechercher des IPs appropriés pour son SoC, les valider puis les intégrer dans sa conception SoC.

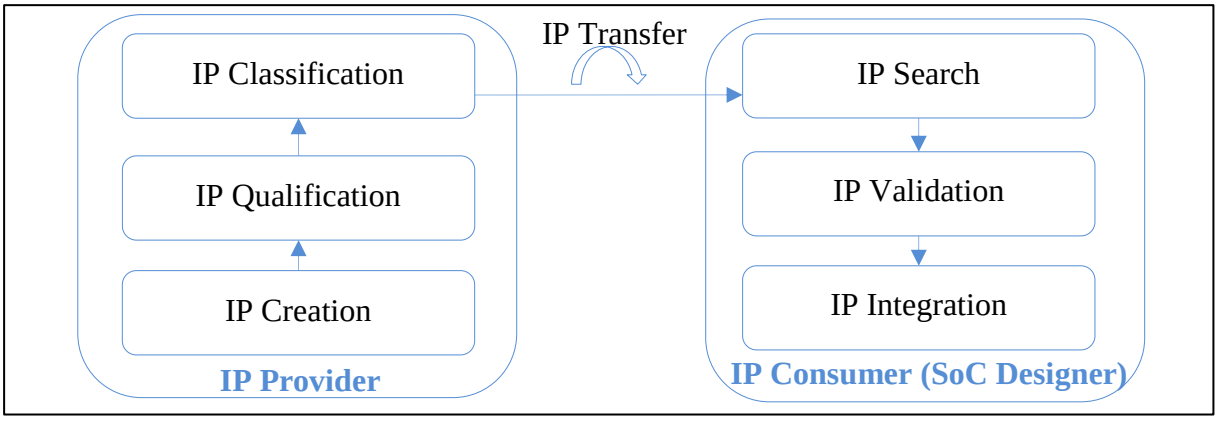


Figure 1-7 : Processus de réutilisation IP.

A base de HwIP, SWIP et InterfaceIP, [21] propose flux de Codesign rapide pour les SoPC[1].

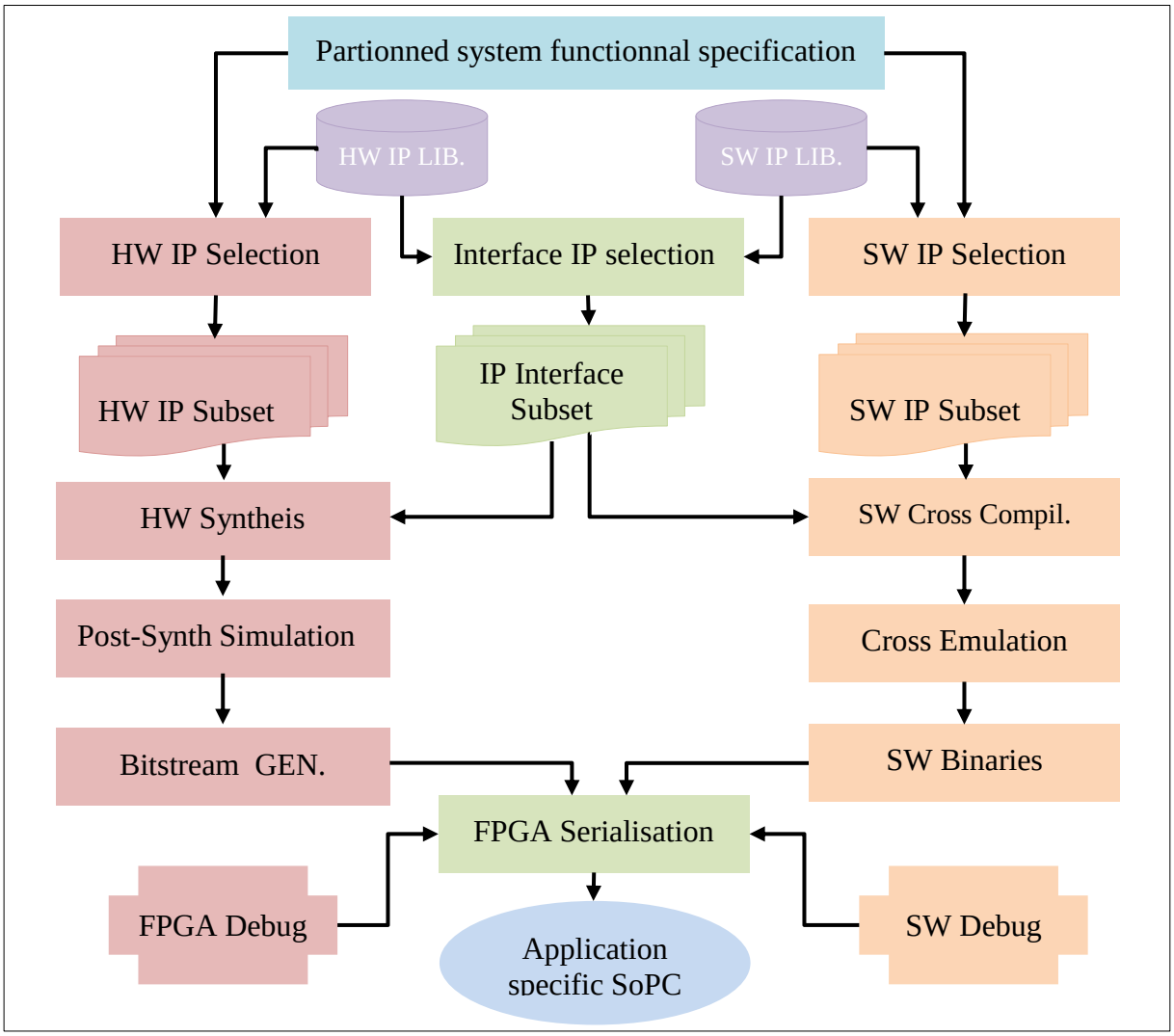


Figure 1-8 : Codesign rapide pour les SoPC.

[1] System on Programmable Chip.

L'IP est un composant matériel ou logiciel exécutant une fonction précise, pouvant être réutilisé par un utilisateur qui n'a pas participé à la spécification de ce composant. Cette approche implique que seule la compréhension de la fonction réalisée est requise, sans nécessiter la connaissance des détails de la spécification. Pour cela, désormais, les concepteurs ont concentré leurs efforts sur la partie réellement innovante d'un système plutôt que sur la conception de composant déjà existant (conçus dans des projets antérieurs ou fournis par des fournisseurs externes). Malheureusement, Les systèmes construits autour d'IPs discrète multi-sources sont gourmands en énergie, longs et coûteux à développer. Des négociations de licence prolongées peuvent également être nécessaires pour certaines parties. La résolution de tous ces problèmes peut nécessiter de trois à six mois avant même que les ingénieurs ne commencent la conception elle-même.

C. Platform-based Codesign

Depuis le début des années 2000, la conception des SEs à base de plateforme (Platform-Based Design ou PBD) est apparu.

Selon le domaine d'application, Il existe de nombreuses définitions du terme « plateforme ». Notamment, dans le domaine des circuits intégrés : « *une plateforme est considérée comme un circuit intégré flexible où la personnalisation pour une application particulière est obtenue en programmant un ou plusieurs des composants de la puce. La programmation peut impliquer une personnalisation du métal (réseaux de portes), une modification électrique (personnalisation FPGA) ou un logiciel à exécuter sur un microprocesseur ou un DSP* ». En général, selon [46] *« Une plateforme est une couche d'abstraction qui masque les détails de plusieurs raffinements de mise en œuvre possibles des couches sous-jacentes. C'est une bibliothèque d'éléments caractérisés par des modèles qui représentent leurs fonctionnalités et offrent une estimation de grandeurs (physiques) importantes pour le concepteur. La bibliothèque contient des interconnexions et des règles qui définissent quelle est la composition juridique des éléments. Une composition juridique d'éléments et d'interconnexions est appelée une instance de plateforme (platform instance)*».

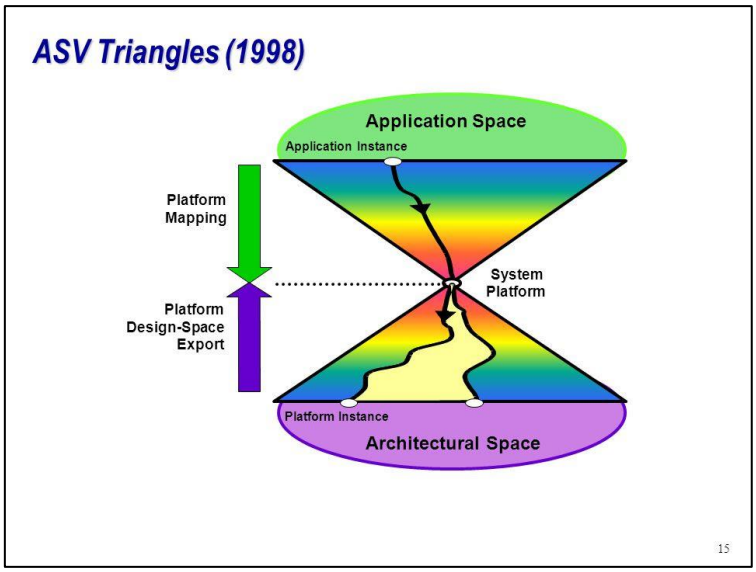


Figure 1-9 : Platform-based Design **[46]**.

Comme le montre la Figure 1-9, les principes de base de la conception à base de plateforme sont les suivants [24]:

- Considérer la conception comme un processus de rencontre au milieu «meet-in-the-middle process », où les raffinements successifs des spécifications rencontrent des abstractions des implémentations potentielles ;
- L'identification de couches précisément définies, où se déroulent le processus de raffinement et d'abstraction. Les couches permettent ensuite aux conceptions construites sur elles d'être isolées des détails de niveau inférieur, mais laissent suffisamment d'informations être transmises sur les niveaux d'abstraction inférieurs pour permettre l'exploration de l'espace de conception avec une prédiction assez précise des propriétés de l'implémentation finale.

Pour les SEs, La plateforme système est la combinaison des plateformes matérielles (plateforme de micro-architecture) et logicielles (plateforme d'API).

La plateforme matérielle est une famille d'architectures qui permet une réutilisation substantielle des logiciels [47]. L'architecture "de base" se compose de cœurs programmables, d'un sous-système d'E/S et de mémoires. Alors que, le rôle de La plateforme API[1] est de permettre aux concepteurs de systèmes d'utiliser les services offerts par une micro-architecture [46].

[1] Application Programm Interface.

Plusieurs livres et articles sont parus dans la littérature traitant les plateformes et de leurs utilisation dans la conception de systèmes embarqués (voir par exemple [48]) car elles offrent aux concepteurs de nombreux avantages par rapport à la conception de systèmes basée sur des éléments IP distincts provenant de plusieurs sources, citons par exemple :

- Le concept de plateforme ne fait pas la distinction entre matériel et logiciel mais entre fonctionnalité et architecture. Par conséquent, le partitionnement entre composants matériels et logiciels peut se faire de manière intelligente et optimisée.
- La plateforme permet la réutilisation et la facilitation du travail d'adaptation d'une conception commune à une variété d'applications différentes.
- La plateforme réduit considérablement le temps et le coût de développement et le risque global de conception, car elle limite le nombre de fournisseurs de matériel et de logiciels requis.

Mais, la question qui se pose est, comment s'assurer que les exigences et les spécifications sont satisfaites et combien de temps faut-il pour transformer une plateforme en produit ?

D. Model-based Codesign

La conception basée sur des modèles (MBCD[1]) est une approche de conception des SEs relativement récente et devient très populaire. Elle est basée sur le raffinement progressif des modèles simulables. Donc, elle offre la possibilité d'abstraire les composants du système à plusieurs niveaux de représentation.

Dans cette méthodologie, un ensemble d'exigences et de contraintes est obtenu pour le système à modéliser. Le système est alors décrit comme un modèle abstrait qui est une combinaison de ses spécifications structurelles et comportementales associées. Les composants du modèle sont spécifiés à un haut niveau d'abstraction pour rester indépendants de la mise en œuvre.

Donc, dans le MBCD, les développeurs modélisent une spécification de système indépendamment de la mise en œuvre et utilisent une conception basée sur la simulation pour évaluer des prototypes virtuels avant la construction du système. À la fin du processus de simulation, un prototype de système virtuel est obtenu. La conception est ensuite partitionnée en matériel, logiciel et interfaces correspondantes à l'aide d'un processus que nous appelons la cartographie des modèles.

[1] Model Based CoDesign.

[26] Propose une approche de conception pour les SEs qui permet aux développeurs de créer des modèles d'une représentation formelle du système indépendamment de l'implémentation matérielle et logicielle. Donc, les développeurs utilisent la modélisation basée sur la simulation pour explorer la faisabilité de prototypes virtuels, puis mappent de manière interactive la spécification sur une architecture logicielle-matérielle.

[49] Donne un panorama de nombreux profils UML2.0 utilisés pour le codesign des SOCs. Pour le Codesign basé sur des modèles, on peut utiliser l'un des profils suivants : MARTE[1], UML SOC, UML SystemC ou bien GASPARD2.

Avantages

- Favorise la réutilisation : les concepteurs peuvent réutiliser des composants élémentaires quelle que soit leur technologie, en les agrégeant dans des structures de conception plus complexes.
- prend en charge la prise de décision en matière de conception (design decision-making) : Les concepteurs peuvent échanger des alternatives de conception et sélectionner une solution de conception qui répond le mieux aux spécifications et aux exigences.

E. Formal Codesign (FCodesign)

Le Codesign formelle (Codesign basée sur spécification et vérification formelle) est une approche de conception pour les systèmes embarqués critiques avec des contraintes strictes c.-à-d. les systèmes embarqués dont une panne peut avoir des conséquences dramatiques comme les systèmes de pilotage des avions, les systèmes de paiement électronique, les appareil médicaux de contrôle de dosages …etc.,

Pour garantir l'exactitude du système, Cette approche repose sur l'utilisation des langages formels de spécification tels que le langage B, le langage Esterel, LOTOS[2], PN[3] et des techniques de vérification formelles telles que la vérification de modèles (model checking) et preuve de théorèmes (theorem proving). Donc, La méthodologie elle-même part d'une spécification formelle initiale puis procède par raffinement jusqu'à la génération du code.

Le caractère formel de cette approche permet de produire des spécifications exemptes d'ambiguïté, mais malheureusement, les concepteurs de SEs ne sont pas très familiers avec

[1] Modeling and Analysis of Real Time Embedded systems.
[2] Langage Of Temporal Ordering Specification.
[3] Petri Nets.

les spécifications formelles nécessitant une formation mathématique approfondie ; pour cette raison, au lieu de traiter directement de telles spécifications, de nombreux outils ont été développés pour générer des spécifications formelles à partir de notations graphiques (voir [50], [51]).

F. Component-Based Hardware–Software Codesign

L'ingénierie logicielle basée sur les composants (CBSE[1]) préconise la réutilisation et l'adaptation des composants logiciels existants. Cependant, les systèmes embarqués, se composent non seulement de logiciels, mais également de composants matériels. Ainsi, la conception basée sur les composants devrait être étendue aux systèmes comportant à la fois des composants matériels et logiciels. Une telle extension n'est cependant pas sans défis. La méthodologie étendue doit tenir compte de contraintes strictes sur les performances ainsi que de différents facteurs de coût. En outre, les dissemblances entre le matériel et le logiciel (telles que les primitives de communication, le niveau d'abstraction, etc.) doivent être résolu.

[38]Propose la méthodologie CBHSCD[2] qui est une méthodologie de conception qui gère les composants matériels et logiciels de manière uniforme en utilisant une notion de composant générique axée sur la fonctionnalité et en utilisant des logiciels comme des adaptateurs d'interface pour le matériel. Les principales étapes de cette méthodologie sont illustrées dans la figure suivante. Par conséquent, l'objectif principal de cette méthodologie est d'assembler le système à partir de blocs de construction pré-vérifiés existants permettant au concepteur un prototypage rapide à un très haut niveau d'abstraction. A ce niveau d'abstraction, les composants ne connaissent aucun détail d'implémentation les uns des autres, pas même si l'autre est implémenté en tant que matériel ou logiciel.

Dès les premières étapes du processus de conception, il est possible de simuler et de valider le comportement de ce système prototype.

[1] Component Based Software Engineering.
[2] Component Based Hardware Software CoDesign.

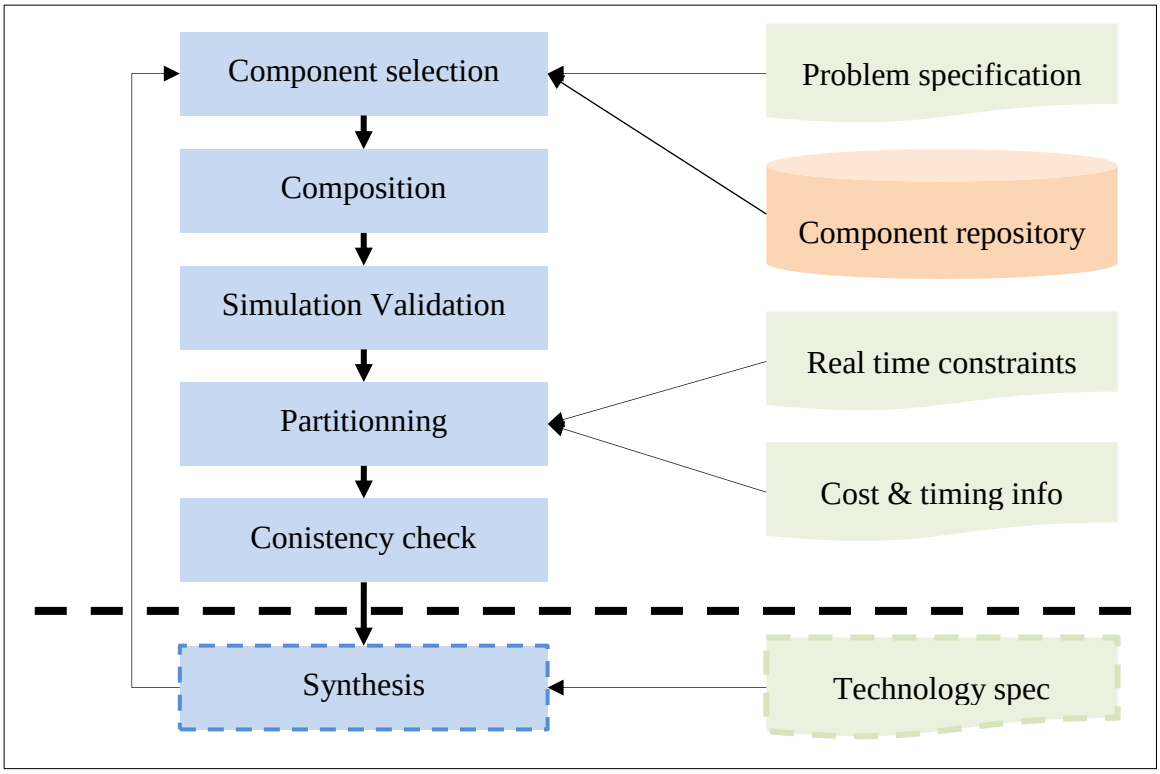


Figure 1-10 : Phases de la méthodologie CBHSCD**[38]**.

4. Système embarqué auto adaptatif

4.1. Les systèmes embarqués auto-adaptatifs

On peut considérer l'auto-adaptation comme un moyen de gérer la complexité croissante des systèmes informatiques [52]. Il a été introduite il y a environ deux décennies.

Dans [53], la définition proposée est «*Les systèmes auto-adaptatifs sont des systèmes capables de modifier leur comportement au moment d'exécution afin d'atteindre les objectifs du système* ». Des circonstances imprévisibles telles que des changements dans l'environnement du système, des défauts du système, de nouvelles exigences et des changements dans la priorité des exigences sont quelques-unes des raisons pour déclencher une action d'adaptation dans un système auto-adaptatif. Donc, un système auto-adaptatif (SAA) est censé se reconfigurer automatiquement pour faire face aux variations de son contexte, de son environnement et de ses exigences. Pour cela, ils doivent fonctionner dans des conditions incertaines, sans interruption. Notons que, les causes possibles d'incertitudes comprennent les changements dans l'environnement opérationnel, la dynamique de la disponibilité des ressources et les variations des objectifs des utilisateurs. Le but de l'auto-adaptation est de laisser le système collecter des données supplémentaires sur les incertitudes

pendant le fonctionnement afin de se gérer en fonction d'objectifs de haut niveau. Le système utilise les données supplémentaires pour résoudre les incertitudes et, en fonction de ses objectifs, se reconfigure ou s'ajuste pour satisfaire les conditions changeantes.

Un système embarqué est dit auto-adaptatif (SEAA) s'il est conscient de son propre état, de ses exigences ou de son environnement au moment de l'exécution. Ces informations d'exécution seront utilisées pour ajuster son comportement en fonction des objectifs de système. En d'autres termes, un SEAA est un système qui s'adapte automatiquement sans aucune intervention extérieure. Les SAES appartiennent à une classe plus large de systèmes appelés Cyber Physical Systems (CPS) [54]. Les CPS sont considérés comme la prochaine révolution informatique et la nouvelle génération de systèmes complexes de systèmes (SoS[1]). Ils sont des dispositifs embarqués complexes et ubiquitaires couplés à une intégration globale respectant la loi de Moore [54].

Donc, les SEAA se tiennent à l'intersection de l'informatique embarquée et de l'informatique adaptative. Un bon exemple de SEAA typique est le téléphone portable. Ce dernier peut ajuster automatiquement les paramètres du réseau sans fil lors de l'entrée dans une nouvelle région ou un pays.

4.2. Approches d'adaptation

Selon [55] « *La capacité d'un système à s'adapter au moment de l'exécution afin d'atteindre ses objectifs dans des conditions changeantes n'est pas l'exclusivité de l'auto-adaptation, mais peut être réalisée par d'autres moyens comme les systèmes autonomes, les systèmes multi-agents (MAS[2]), les systèmes auto-organisés et les systèmes sensibles au contexte* ». Ces approches se distinguent de l'auto-adaptation, notamment par rapport au principe interne des systèmes auto-adaptatif « Un système auto-adaptatif comprend deux parties distinctes : la première partie interagit avec l'environnement et est responsable des préoccupations du domaine - c'est-à-dire les préoccupations des utilisateurs pour lesquels le système est construit; la deuxième partie consiste en une boucle de rétroaction qui interagit avec la première partie (et surveille son environnement) et est responsable des préoccupations d'adaptation - c'est-à-dire des préoccupations concernant les préoccupations du domaine ».

Cependant, le principe interne des systèmes auto-adaptatif peut être appliqué à ces approches en ajoutant un système de gestion réalisant l'auto-adaptation.

[1] System Of System.
[2] Multi Agent System.

1. Le domaine des systèmes autonomes a une longue tradition d'étude des systèmes qui peuvent changer leur comportement pendant le fonctionnement en réponse à des événements qui n'ont peut-être pas été entièrement anticipés. Une idée centrale des systèmes autonomes est d'imiter le comportement humain (ou animal), qui a été une source d'inspiration pendant très longtemps.

   L'intérêt pour les systèmes autonomes s'est considérablement accru ces dernières années, avec divers domaines d'application tels que les véhicules autonomes. Bien que ces applications aient un potentiel extrême, leurs succès jusqu'à présent se sont également accompagnés de quelques échecs dramatiques, tels que les accidents causés par les voitures autonomes de première génération. Les conséquences de tels échecs démontrent les réelles difficultés techniques liées à la réalisation de systèmes réellement autonomes. Un sous-domaine important des systèmes autonomes est celui des SMA[1] [55]. Actuellement, il existe plus de 80 méthodologies de conception des SMA [56], l'un de ces méthodologies est la méthodologie ADELFE[2]. Il est basé sur les systèmes multi-agents adaptatifs (AMAS[3]) et utilisé uniquement lorsque l'environnement est imprévisible ou que le système est ouvert [56].
2. Les systèmes auto-organisés (les systèmes s'organisent lui-même) mettent l'accent sur le contrôle décentralisé. La conception de systèmes décentralisés qui exposent le comportement global requis tout en évitant les phénomènes émergents indésirables reste un défi majeur [55].
3. La sensibilité au contexte (Context Awareness) met l'accent sur la gestion des éléments pertinents de l'environnement (physique, virtuel ou utilisateur) en tant que citoyens de première classe dans la conception et l'exploitation du système. Les systèmes informatiques sensibles au contexte concernent l'acquisition du contexte (par exemple, grâce à des capteurs pour percevoir une situation), la représentation et la compréhension du contexte, et le pilotage du comportement en fonction du contexte reconnu (par exemple, le déclenchement d'actions en fonction du contexte réel). Les systèmes sensibles au contexte ont généralement une architecture en couches, où un gestionnaire de contexte ou un middleware dédié est responsable de la détection et de la gestion des changements de contexte.

[1] Système Multi Agent.
[2] Atelier de Développement de Logiciels à Fonctionnalité Emergente.
[3] Adaptive Multi Agent System.

4. Les systèmes informatiques conscients de soi (Self Awareness) contrastent avec les systèmes informatiques sensibles au contexte dans le sens où ces systèmes capturent et apprennent des connaissances non seulement sur l'environnement mais aussi sur eux-mêmes. Ces connaissances sont encodées sous la forme de modèles d'exécution (runtime models), qu'un système conscient de lui-même utilise pour raisonner au moment de l'exécution, lui permettant d'agir conformément à des objectifs de niveau supérieur.

4.3. L'incertitude dans les systèmes auto-adaptatif

Assurer que les systèmes atteignent leurs objectifs dans un contexte/environnement incertain est un critère essentiel pour l'auto-adaptation. Néanmoins, le concept d'incertitude dans les systèmes auto-adaptatifs est encore insuffisamment compris.

Selon [55], L'incertitude dans les systèmes auto-adaptatifs est défini comme suit : « *l'incertitude est toute déviation des connaissances déterministes qui peut réduire la confiance dans les décisions d'adaptation prises sur la base des connaissances* ».

Généralement, L'incertitude dans les systèmes informatiques est liée aux connaissances disponibles pour effectuer les différentes tâches à accomplir [55]. Donc, il peut faire référence à l'absence de connaissances, à l'insuffisance des connaissances et à la différence entre les connaissances requises pour effectuer une tâche et les connaissances déjà possédées, entre autres éléments. Aussi, Il faut bien distinguer entre l'incertitude aléatoire et l'incertitude épistémique [55]. Le premier, fait référence à l'imprécision des connaissances. Tandis que, le deuxième fait référence au manque de connaissances. Notons que, l'objectif principal de la recherche dans le domaine de l'auto-adaptation est l'incertitude aléatoire.

Sans une atténuation appropriée de l'incertitude, les décisions d'adaptation peuvent être inexactes, peu fiables ou non concluantes [55]. Pour cette raison, Au cours des dernières années, des efforts substantiels ont été consacrés à apprivoiser l'incertitude, avec un accent particulier sur la conception de taxonomies de l'incertitude pour les systèmes auto-adaptatifs et sur le développement de méthodes de gestion de l'incertitude.

Apprivoiser l'incertitude consiste à fournir des garanties sur la conformité d'un système auto-adaptatif à ses objectifs d'adaptation, quelle que soit l'incertitude à laquelle il est confronté. Parmi les approches qui existent pour apprivoiser l'incertitude, on peut citer : quantitative verification at runtime [57] (La vérification quantitative est une technique basée sur les

mathématiques pour analyser l'exactitude, les performances et la fiabilité des systèmes présentant un comportement stochastique), ActivFORMS [58] (est une approche qui applique la vérification statistique ou bien statistical model checking at runtime), et proactive decision-making.

Plusieurs taxonomies de la source d'incertitude ont été proposées. Par exemple, Weyns [55] classe les sources d'incertitude en quatre groupes (l'incertitude liée au système lui-même « managed and managing », l'incertitude liée aux objectifs du système, l'incertitude dans le contexte d'exécution et l'incertitude liée aux aspects humains). Dans chaque groupe, un ensemble non exhaustif de sources d'incertitude est répertorié (voir le tableau suivant). Les sources d'incertitude peuvent se manifester au moment de la conception, de l'exécution ou bien les deux.

Tableau 1-2 : Sources d'incertitude dans les systèmes auto-adaptatifs.

Groupe	Source d'incertitude	Brève explication
Système	Simplification des hypothèses	Modélisation d'abstractions qui introduisent un certain degré d'incertitude.
	modèle dérive	Décalage entre les éléments du système et leurs représentations.
	Incomplétude	certaines parties du système ou de son modèle sont manquantes et peuvent être ajoutées au moment de l'exécution.
	Valeurs de futurs paramètres	Manque de connaissances sur les valeurs futures des paramètres pertinents pour la prise de décision.
	Fonctions d'adaptation	Fonctions de surveillance, de prise de décision et d'exécution imparfaites pour réaliser l'adaptation.
	Décentralisation	Manque de connaissances précises sur les effets au niveau du système de la prise de décision locale.
	L'apprentissage automatique	Apprendre avec des données imparfaites et limitées, ou un caractère aléatoire dans le modèle et l'analyse.

But	Spécification des buts	Difficulté à spécifier précisément les préférences des parties prenantes.
	Futurs changements d'objectifs	Changements d'objectifs en raison de nouveaux besoins des clients, de nouvelles réglementations ou de nouvelles règles du marché.
Contexte	Contexte d'exécution	Le modèle de contexte basé sur la surveillance peut ne pas déterminer avec précision le contexte et son évolution.
	Bruit de détection	Les capteurs/sondes (Sensors/probes) ne sont pas des appareils idéaux, et ils peuvent fournir des données (légèrement) inexactes.
	Différentes sources d'informations	Imprécision due à la composition et à l'intégration de données provenant de différentes sources.
Humains	Humain dans la boucle	Le comportement humain est intrinsèquement incertain; il peut s'écarter du comportement attendu.
	Multi-propriété	Certaines parties du système fournies par différentes parties prenantes peuvent être partiellement inconnues.

4.4. Niveau d'auto-adaptation

4.4.1. *Auto-adaptation au niveau matériel*

Dans les années 70, un nouveau type de système complexe apparait. C'est les systèmes sur puce (SoC) qu'ils sont définit par [59] comme suit : « *Un système sur puce est un système hétérogène capable d'intégrer plusieurs composants sur une même puce. Parmi les composants qu'on peut trouver dans un SoC, on compte les processeurs, les mémoires, les interconnexions qui peuvent suivre plusieurs topologiques, les interfaces externes, les convertisseurs analogiques-numériques, les capteurs et les horloges*».

Il existe deux types de système sur puce, statique ou reconfigurable. La figure suivante illustre

une classification des différents circuits intégrés (ICs[1]) numérique.

Les systèmes reconfigurables sur puce sont des systèmes qui comportent plusieurs composants reconfigurables, interconnectées par un réseau d'interconnexion reconfigurable. Ils offrent une grande flexibilité par rapport aux SoCs statiques parce qu'ils sont implémenté sur du matériel reconfigurable. Donc, ces systèmes peuvent être reconfigurés un nombre illimité de fois et le concepteur du système capable d'ajouter d'autres fonctionnalités et/ou d'effectuer des modifications sur le système après sa fabrication. L'architecture reconfigurable peut être configurée de manière statique ou dynamique.

La reconfiguration statique est le chargement d'un circuit reconfigurable avec une configuration avant le commencement de l'exécution de l'application. Tandis que, la reconfiguration dynamique (run-time reconfiguration) est le chargement d'un circuit reconfigurable avec une configuration durant l'exécution de l'application. Donc, il permet aux systèmes de s'adapter à des changements au cours de l'exécution.

Donc, désormais l'auto-adaptation dans les systèmes embarqués peut non seulement être mise en œuvre dans le logiciel mais encore de plus efficacement dans le matériel.

La reconfiguration dynamique est implémentée à travers les réseaux de portes programmables (FPGA).

[1] Integrated Circuit.

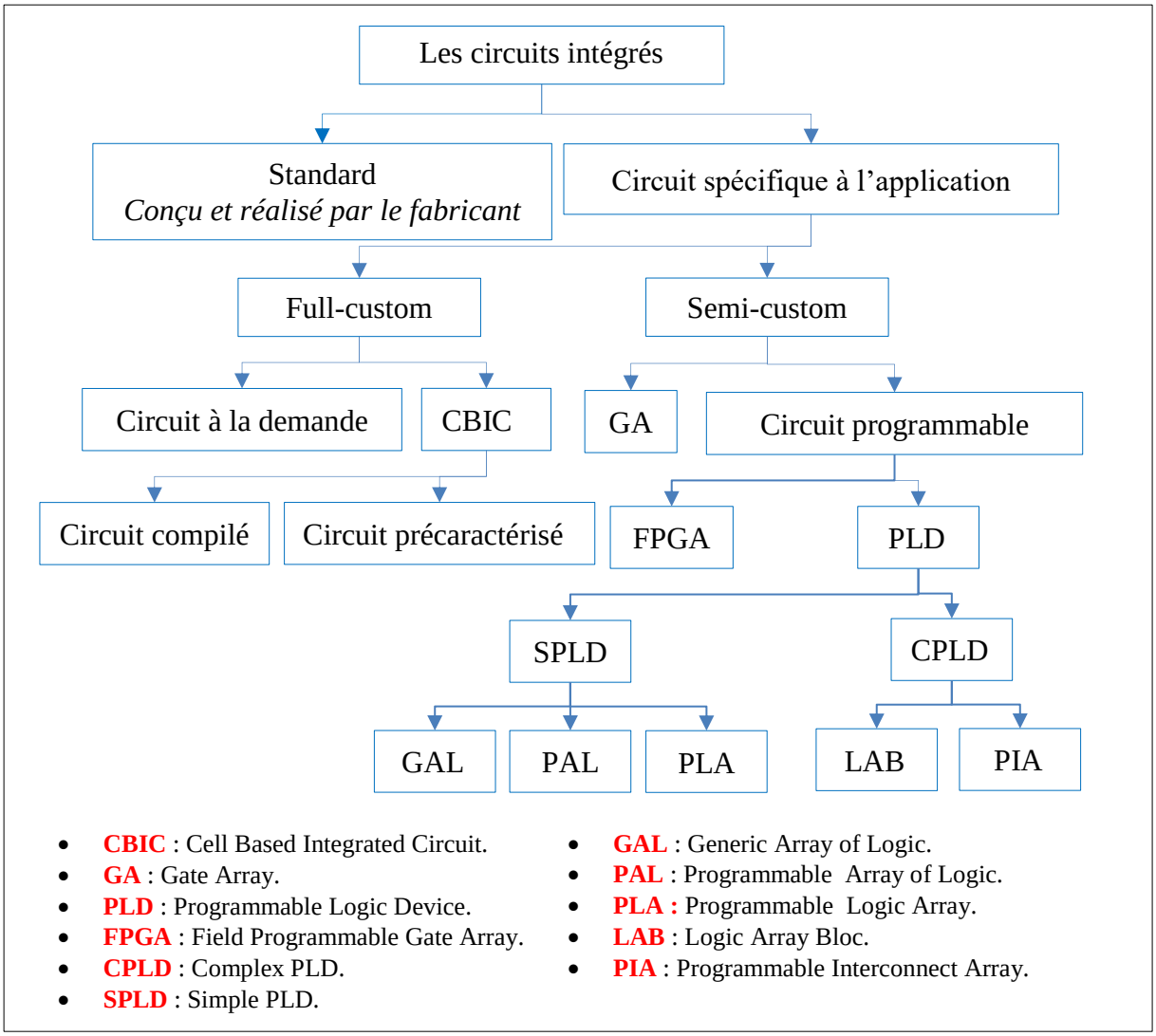


Figure 1-11 : Classification des circuits intégrés numérique.

En 1984, les FPGAs lancés sur le marché par la firme Xilinx. Certains FPGAs supportent la reconfiguration dynamique totale (RDT[1]). Tandis que, certains supportent la reconfiguration dynamique partielle (RDP[2]) qui permet de reconfigurer une partie du FPGA sans perturber le fonctionnement du reste. Notons que, La reconfiguration dynamique introduise le concept du matériel virtuel [59].

[1] Reconfiguration Dyanmique Totale.
[2] Reconfiguration Dynamique Partielle

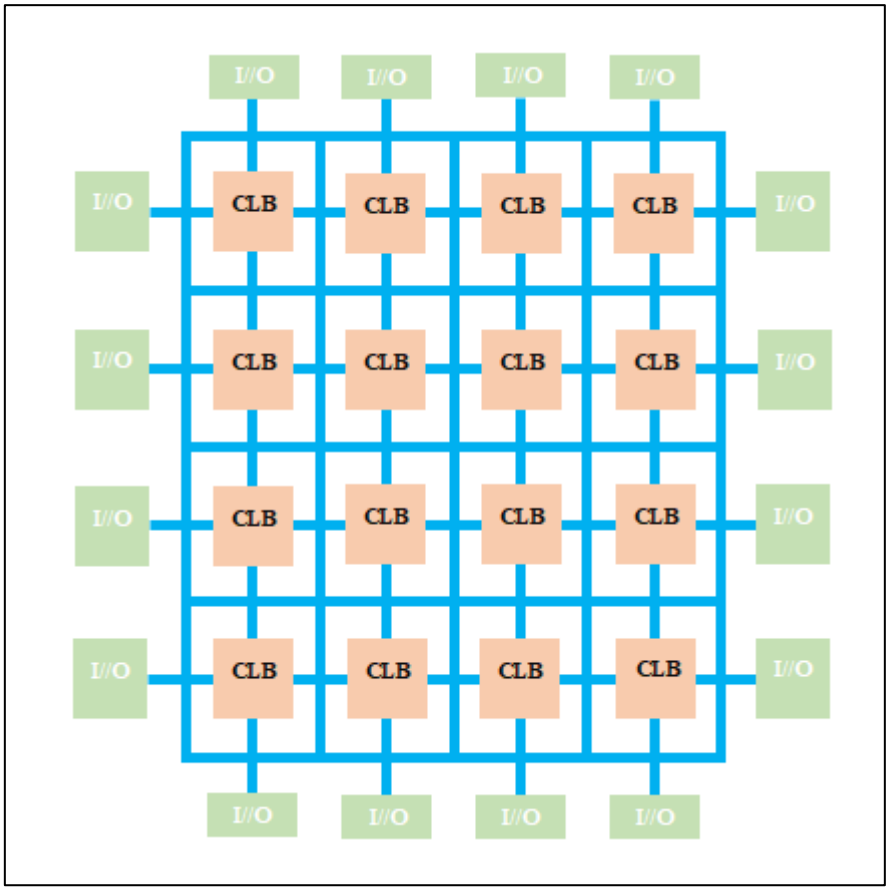


Figure 1-12 : Architecture générique d'un FPGA [60].

4.4.2. *Auto-adaptation au niveau logiciel*

Évidemment, « *la reprogrammation est plus flexible que la modification d'architecture matérielle. Cependant, la solution logicielle est moins performante en termes de traitement et de consommation par rapport à la solution matérielle basée sur les ASIC et FPGA* »[61].

L'auto-adaptation au niveau logiciel d'un système embarqué est généralement divisée en trois catégories : l'auto-adaptation au niveau du système d'exploitation, l'auto-adaptation au niveau du middleware et/ou l'auto-adaptation au niveau de l'application.

Au niveau logiciel, pour aborder l'aspect de l'auto-adaptation, il semble naturel d'utiliser le contrôle en boucle fermée comme paradigme principal. Un modèle bien connu pour appliquer le contrôle en boucle fermée au logiciel est l'architecture MAPE-K (Monitor-Analyze-Plan-Execute-Knowledge). Il convient de souligner que le modèle MAPE-K peut gérer des reconfigurations à la fois au niveau logiciel (ajustement des algorithmes, modification des paramètres) et au niveau matériel (reprogrammation des FPGA). Cependant, la

reconfiguration matérielle avec des FPGA est généralement orchestrée par des composants logiciels qui suivent les étapes du modèle MAPE-K pour surveiller et décider des ajustements nécessaires.

4.5.Modèle conceptuel d'un système auto-adaptatif

Selon [62] « *Le modèle conceptuel d'un SAA (voir la figure suivante) introduit un vocabulaire de base pour le domaine de l'auto-adaptation et sert de guide pour l'organisation et la focalisation des connaissances du domaine* ». Il comprend quatre éléments de base : boucle de rétroaction (Feedback Loop), objectifs d'adaptation (Adaptation Goals), système géré (Managed System) et environnement. La boucle de rétroaction et les objectifs d'adaptation forment le système de gestion (Managing System) [55].

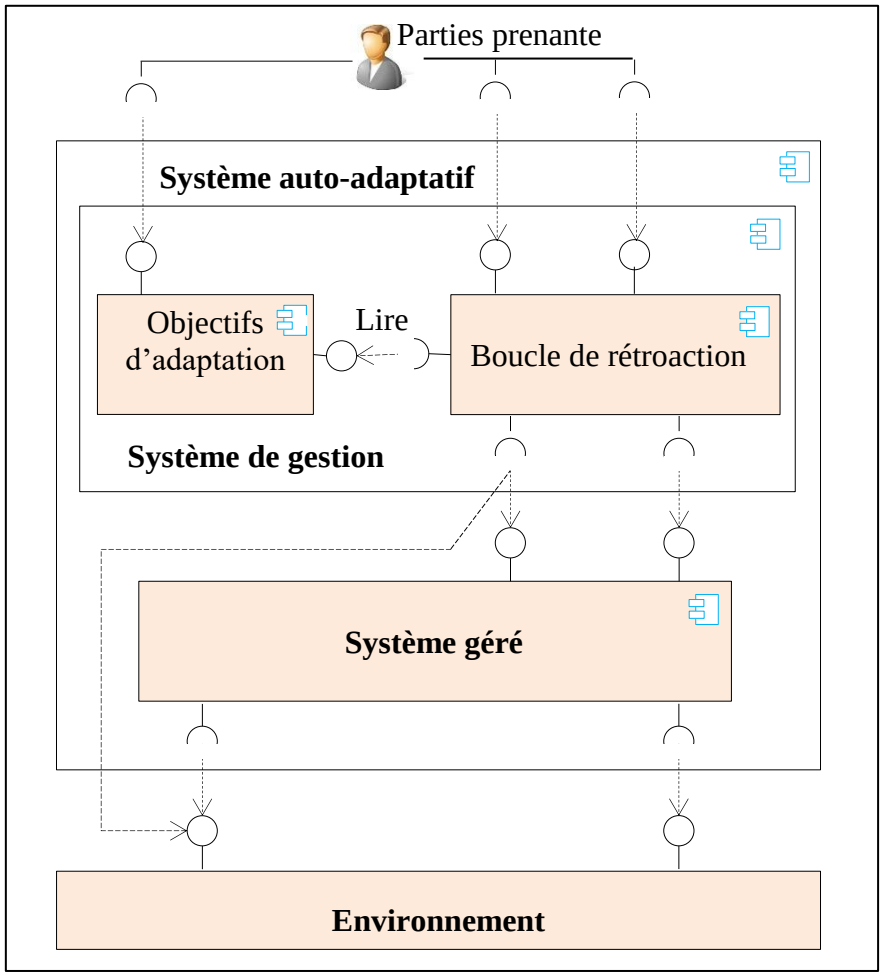


Figure 1-13 : Le modèle conceptuel d'un SAA**[55]**.

4.5.1. *Système de gestion*

1. Boucle de rétroaction (Feedback Loop)

L'auto-adaptation est obtenue en améliorant un système avec une boucle de rétroaction (ou une combinaison de boucles de rétroaction) [52]. Il est proposé par [63] et considéré comme le modèle de référence pour la conception de logiciels auto-adaptatifs. Donc, le rôle de cette boucle de rétroaction est de s'assurer que le système respecte toujours un ensemble d'objectifs d'adaptation lorsque les conditions de fonctionnement du système changent. Pour cette raison, il surveille le système et son environnement (Monitor) pour collecter des données d'exécution qui n'étaient pas disponibles auparavant, analyse (Analyzer) et planifie (Planer) des configurations alternatives et exécute des adaptations (Executer) pour atteindre les objectifs d'adaptation, ou de se dégrader gracieusement si besoin [52]. Les fonctions de la boucle de rétroaction et les connaissances qui sous-tendent leur fonctionnement sont souvent appelées une boucle MAPE-K [62]. Selon [64] « *Les activités MAPE sont centrées sur les modèles de connaissances qui incluent généralement diverses formes de modèles d'exécution (runtime models) tels que les modèles d'architecture logicielle du système et de l'environnement gérés, les modèles d'objectifs, les réseaux de Markov qui permettent de prédire les qualités de différentes configurations de système, etc* ».

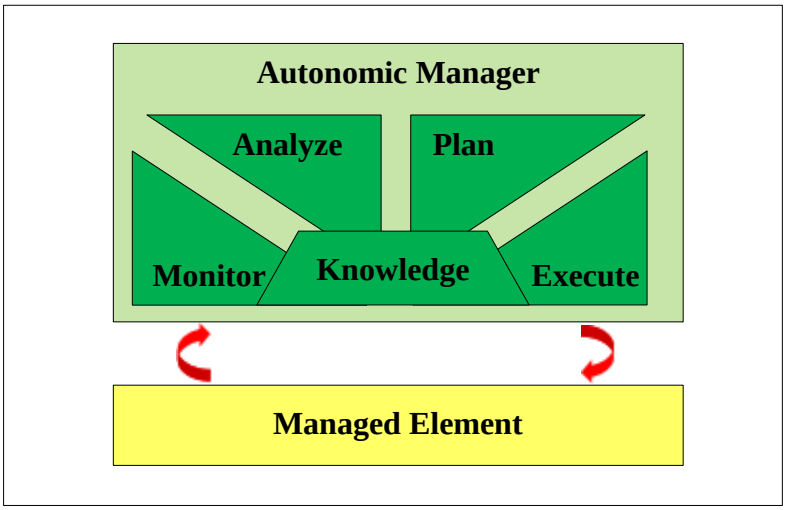


Figure 1-14 : La boucle MAPE-K.

2. Objectifs d'adaptation

Les objectifs d'adaptation représentent les préoccupations du système de gestion par rapport au système géré ; ils concernent les propriétés de qualité du système géré (les propriétés non

fonctionnelles du système, telles que la fiabilité, les performances, le coût ….etc.). D'autre part, Les systèmes auto-adaptatifs ont plusieurs propriétés Self-x. La figure 1-15 montre la hiérarchie de ces propriétés [65].

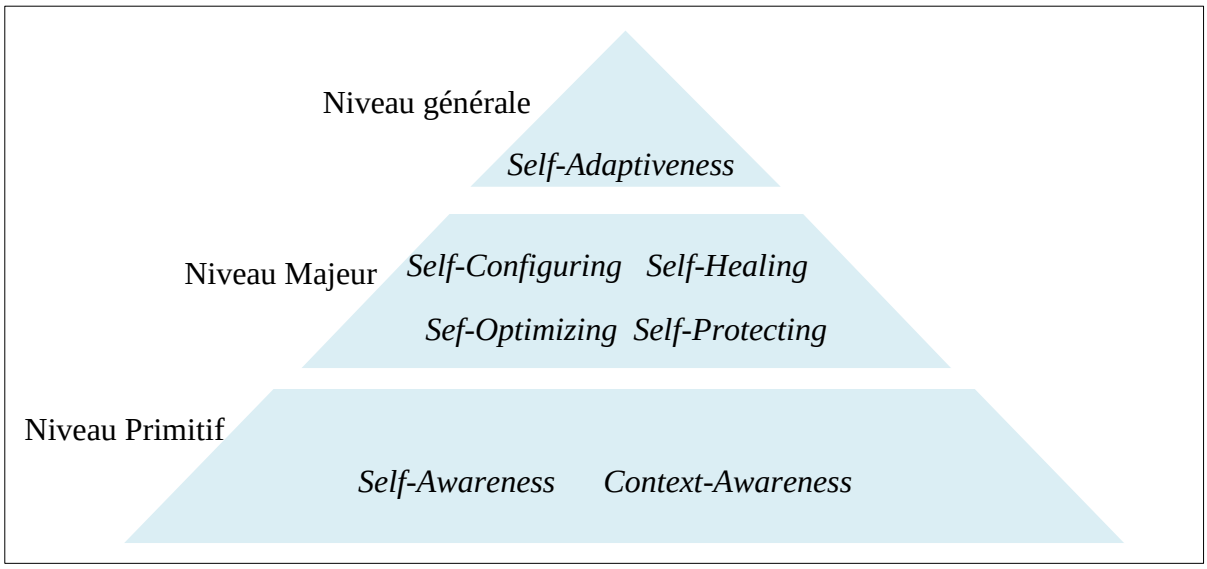


Figure 1-15 : Hiérarchie des propriétés Self-* d'un SAA **[65]**.

Donc, selon les propriétés Self-x d'un SAA, on peut distinguer quatre principaux types d'objectifs d'adaptation de haut niveau [55] :

- **l'Auto-Configuration (Self-Configuration)** : les systèmes qui se configurent automatiquement,
- **l'Auto-Optimisation (Self-Optimization)** : les systèmes qui cherchent continuellement des moyens d'améliorer leurs performances ou de réduire leurs coûts,
- **l'Auto-Guérison (Self-Healing)** : les systèmes qui détectent, diagnostiquent et réparent les problèmes résultant de bogues ou de défaillances.
- **l'Auto-Protection (Self-Protection)** : les systèmes qui se défendent contre les attaques malveillantes ou les défaillances en cascade.

Généralement, les objectifs d'adaptation sont utilisés par le système pour raisonner sur lui-même pendant le fonctionnement. Donc, ils doivent être représentés dans un format lisible par machine [55]. En plus, Les objectifs d'adaptation sont souvent exprimés en termes d'incertitude avec laquelle ils doivent composer. Des exemples d'approches sont :

- la spécification d'objectifs de qualité de service à l'aide de logiques temporelles probabilistes qui permettent une quantification probabiliste des propriétés,
- la spécification d'objectifs flous dont la satisfaction est représentée par des contraintes

floues,

- et une spécification déclarative d'objectifs (contrairement à l'énumération) permettant l'introduction flexibilité dans la spécification des objectifs.

Enfin, les objectifs d'adaptation peuvent eux-mêmes être sujets à changement, ce qui est représenté dans la Figure 1-6 au moyen de l'interface d'évolution. L'ajout de nouveaux objectifs ou la suppression d'objectifs pendant le fonctionnement nécessitera des mises à jour du système de gestion, et nécessitera souvent également des mises à jour des sondes et des actionneurs.

4.5.2. *Système géré*

Le système géré c'est le système qui fait l'objet d'une adaptation. Il comprend à la fois le logiciel embarqué et le matériel, qui ensemble réalisent les fonctions nécessaires pour répondre aux besoins des utilisateurs du système. Par conséquent, les préoccupations du système géré concernent le domaine (l'environnement du système). Différentes terminologies ont été utilisées pour désigner le système géré [55], telles que l'élément géré (managed element), la couche système (system layer), la fonction centrale (core function), le système de niveau de base (base-level system) et l'installation contrôlable (controllable plant). Pour réaliser ses fonctions, le système géré détecte et affecte l'environnement à travers des capteurs pour permettre la surveillance et des effecteurs (également appelés actionneurs) pour exécuter les actions d'adaptation.

L'exécution en toute sécurité des adaptations nécessite que les actions appliquées aux systèmes gérés n'interfèrent pas avec l'activité normale du système. En général, ils peuvent affecter les activités en cours du système - par exemple, la mise à l'échelle d'un système dans le domaine du Cloud Computing peut nécessiter l'arrêt d'un conteneur (container) et son redémarrage. Une approche classique pour réaliser des adaptations sûres consiste à appliquer des actions d'adaptation uniquement lorsqu'un système (ou les parties soumises à l'adaptation) est dans un état de repos (un état dans lequel aucune activité n'est en cours dans le système géré ou dans les parties de celui-ci qui sont sujettes à adaptation) afin que le système puisse être mis à jour en toute sécurité [55].

4.5.3. *Environnement*

L'environnement signifie la partie du monde extérieur avec laquelle un système auto-adaptatif

interagit et dans laquelle les effets du système seront observés et évalués. Il peut inclure des utilisateurs, des éléments physiques et des éléments virtuels. La distinction entre l'environnement et le système auto-adaptatif est faite en fonction de degré de contrôle [55]. L'environnement peut être détecté et affecté respectivement par des capteurs et des actionneurs. Cependant, comme l'environnement n'est pas sous le contrôle de l'ingénieur logiciel du système, il peut y avoir une incertitude quant à ce qui est détecté par les capteurs ou quels seront les résultats des effecteurs.

5. La conception des systèmes embarqués auto adaptatifs

L'idée derrière un système auto-adaptatif est de concevoir et de développer un système capable de s'adapter de manière autonome aux conditions changeantes au moment de l'exécution. L'avantage de cette adaptabilité autonome est de diminue le coût et le temps nécessaires pour adapter le système tout en répondant à certaines exigences de qualité lors de l'exécution [53]. Donc, pendant la conception et de la mise en œuvre d'un système embarqué auto-adaptatif il faut prendre en considération que non seulement le système embarqué doit appliquer les modifications au moment de l'exécution, mais aussi répondre aux exigences du système jusqu'à un niveau satisfaisant. Pour cela, L'ingénierie de tels systèmes est souvent difficile car les informations disponibles au moment de la conception ne sont pas suffisantes pour anticiper toutes les conditions d'exécution Par conséquent, les concepteurs préfèrent souvent gérer cette incertitude au moment de l'exécution.

Bien que l'informatique adaptative ait atteint un bon niveau de maturité pour les systèmes logiciels, il est encore relativement immature pour le SEAA. Les méthodes existantes d'ingénierie logicielle adaptative peuvent facilement échouer lorsqu'elles sont appliquées au SEAA car elles doivent faire face à un ensemble de défis supplémentaires tels que les contraintes d'énergie/temps réel, la fiabilité des logiciels et du matériel et les exigences de sécurité, notamment dans les SEAAs critiques, où toute forme de bogue est inacceptable[54]. En plus, malgré l'énorme effort fourni par les chercheurs pour développer des méthodologies de conception et leurs outils associés pour les SEAA, nous pouvons facilement observer qu'aucune des approches existantes ne fournit un processus de développement complet couvrant toutes les phases de développement. De plus, la plupart des outils existants ne sont pas interopérables. Ainsi, une méthodologie standard ou au moins spécifique à un domaine sera nécessaire pour une conception efficace.

Plusieurs stratégies ont été élaborées pour la conception de SEAA. Ces méthodes peuvent être divisées en deux niveaux : la conception de haut niveau (HLD[1]) et l'approche de conception au niveau bas (LLD[2]).

5.1. Conception de haut niveau (HLD)

Selon la littérature, peu de recherches ont été préparées autour de la conception de haut niveau des SEAA. En 2014, Marco, W. et al. [66] introduit SOMA4DDAS[3], un processus de développement basé sur le profil UML SoaML[4], pour générer des systèmes d'aide à la conduite auto-adaptatifs. En 2018, dans [67] une méthodology de co-conception innovante appelée R-Codesign est présentée pour des systèmes embarqués reconfigurables soumis à des contraintes d'énergie. La méthodologie implique des techniques de modélisation et de partitionnement pour l'allocation des tâches des fonctions logicielles et des comportements matériels en fonction des contraintes de l'utilisateur et à l'aide d'heuristiques. En 2020, Alwyn B. et al. Proposent la plate-forme Elastic IoT[5] [68], un cadre qui simplifie le développement et le déploiement d'applications IoT distribuées qui utilisent des périphériques embarqués adaptatifs. Il utilise une approche axée sur les ressources pour mettre à disposition des données et des capacités de traitement en tant que service, ce qui permet le développement de systèmes IoT véritablement auto-intégrants et auto-adaptants.

5.2. Conception de bas niveau (LLD)

De nombreuses contributions ont été apportées à l'incorporation de stratégies d'auto-adaptation dans le processus de développement des systèmes embarqués. Ces travaux sont classés en adaptations au niveau matériel (conception basée sur la reconfiguration dynamique du matériel), adaptations au niveau logiciel (conception basée sur l'architecture logicielle) et adaptation de couche croisée, c'est-à-dire une combinaison des deux.

Concernant les stratégies d'auto adaptation au niveau logiciel, voici quelques proposition : En 2011, Diguet et al. [69] ont proposé une méthode pour mettre en œuvre l'auto-adaptabilité dans un système embarqué reconfigurable basé sur un système d'exploitation pour atteindre des objectifs tels que la qualité de service, les performances ou la consommation d'énergie. En

[1] High Level Design.
[2] Low Level Design.
[3] Service-Oriented Modeling and Architecture for Distributed Driver Assistance Systems.
[4] Service oriented architecture Modeling Language.
[5] Internet of Things.

2017, D. M. R. Babu et Y. M. Roopa [70] ont proposé une architecture du middleware pour les systèmes embarqués en réseau qui prend en charge les fonctionnalités interchangeables et personnalisables de middleware auto-adaptatives basées sur des composants. L'architecture est conçue pour être reconfigurable en cours d'exécution pour appuyer l'adaptation dans des environnements embarqués spécifiques. En 2022, H. Nakagawa et al. [71] proposent un mécanisme d'évolution pour mettre à jour les fonctionnalités des systèmes embarqués existants sans les modifier. Le mécanisme utilise une unité de contrôle déployée à l'extérieur du système embarqué et un processus d'évolution qui utilise efficacement le diagramme de la machine d'état au moment de la conception et de l'exécution pour mettre à jour les systèmes embarqués. Enfin, concernant les stratégies d'auto adaptation au niveau matériel, L. Fiack, B. et al. [72] Proposent en 2014 une plate-forme multi-FPGA permet une reconfiguration parallèle dynamique de divers composants du système tout en maintenant le routage global et le calcul local.

6. Conclusion

Dans ce chapitre, nous avons présenté les principales idées sur les SEs et les systèmes auto-adaptatifs. Un intérêt particulier est donné aux méthodologies de conception des SEs ainsi qu'aux méthodologies de conception des SEs auto-adaptatifs.

Dans cette thèse, nous nous concentrons sur la première phase i.e. la définition des exigences qui est l'étape fondamentale dans n'importe quel cycle de développement. Cette étape est primordiale, et doit être menée avec beaucoup d'attention.

Pour cela, dans le chapitre suivant, nous allons mettre au clair les énoncés de base de l'ingénierie d'exigences des systèmes embarqués tels que : la catégorisation des exigences, le processus à suivre et les différents approches de l'ingénierie des exigences. Enfin, nous allons citer les différents travaux qui sont réaliser dans le cadre de l'ingénierie des exigences des SEs auto-adaptatifs.

CHAPITRE 2 : L'ingénierie Des Exigences Des SEAAs

« On ne récolte que ce que l'on sème »

Proverbe français

SOMMAIRE :

1. Introduction

Les exigences d'un système sont les descriptions de ce que le système doit faire, les services qu'il fournit et les contraintes sur son fonctionnement. Ces exigences reflètent les besoins des clients pour un système qui répond à un certain objectif, comme contrôler un appareil, passer une commande ou trouver des informations. Le processus de découverte, d'analyse, de documentation et de vérification de ces services et contraintes est appelé ingénierie des exigences. Notons que, La RAND[1] Corporation, fondée en 1946, a introduit la notion d'analyse de systèmes, qui a évolué en ingénierie des exigences [73].

L'ingénierie des exigences (RE) est une démarche méthodologique dont l'objectif est de construire un référentiel d'exigences et de le maintenir à jour au fil du temps, en tenant compte des évolutions. Elle repose sur une séparation claire entre le domaine du problème et celui de la solution. Le Quoi pas le Comment. Donc, l'ingénierie des exigences est un ensemble d'activités, de méthode et d'outils permettant de développer et de gérer les exigences d'un système.

Ce chapitre est divisé en deux parties, dans la première partie, nous présentons les notions de bases ainsi qu'un état de l'art sur l'ingénierie des exigences des systèmes embarqués. Donc, nous commençons par la présentation des catégories des exigences ainsi que les caractéristiques de chaque catégorie, ensuite nous mettons l'accent sur le processus de l'ingénierie des exigences des systèmes embarqués à suivre, les approches et les différents outils disponible. Dans la deuxième partie, un état de l'art des travaux les plus pertinents sur l'ingénierie des exigences des systèmes embarqués auto adaptatifs sera élaboré.

[1] Research And Development.

2. Catégorisation des exigences

2.1. Selon le niveau d'abstraction

Les trois niveaux d'abstraction sont les suivants : Besoins, Exigences d'utilisateur et Exigences système. Donc, il ne faut pas confondre entre les deux termes Besoin et exigence.

Un besoin est l'expression par un utilisateur du système d'un manque, d'une nécessité, d'une insatisfaction ou d'un désir, comme par exemple : j'ai besoin de … je veux que …. Il me faut un …etc. Donc, les besoins sont écrits par les clients et exprimé souvent dans des termes ambigus et vagues.

Selon la norme IEEE[1] Std 610.12-1990[74], une exigence (parfois appelée requis ; anglais : requirement) est définit comme suit :

« *Condition ou capacité dont un utilisateur a besoin pour résoudre un problème ou atteindre un objectif*».

Ou bien

« *Condition ou capacité que doit présenter un système pour satisfaire un contrat, un standard, une spécification ou tout autre document formel imposé* ».

Donc, vous devez rédiger les exigences à différents niveaux de détail car différents lecteurs les utilisent de différentes manières. Les utilisateurs finaux et les clients ont besoin d'une déclaration de haut niveau d'exigences (Exigences d'utilisateurs); Les développeurs du système ont besoin d'une spécification plus détaillée (Exigences du système). Notons que, certains des problèmes qui surviennent au cours du processus d'ingénierie des exigences résultent de l'absence de séparation claire entre ces différents niveaux de détail.

Différents niveaux d'exigences sont utiles car ils communiquent des informations sur le système à différents types de lecteurs. L'exemple suivant [75] illustre la distinction entre les exigences de l'utilisateur et celles du système. Cet exemple d'un système de gestion de patients en soins psychiatriques (MHC-PMS[2]) montre comment une exigence de l'utilisateur peut être étendue à plusieurs exigences du système.

[1] Institute of Electrical and Electronics Engineers.
[2] Patient Management System for Mental Health Care.

Tableau 2-1 : Exigence d'utilisateur VS Exigence système.

Exigence d'utilisateur
▪ Le MHC-PMS doit générer des rapports de gestion mensuels indiquant le coût des médicaments prescrits par chaque clinique au cours de ce mois.
Exigences du système
▪ Le dernier jour de travail de chaque mois, un récapitulatif des médicaments prescrits, de leur coût et des cliniques prescripteurs est généré. ▪ Le système génère automatiquement le rapport pour impression après 17h30 le dernier jour de travail du mois. ▪ Un rapport doit être créé pour chaque clinique et doit énumérer les noms de médicaments individuels, le nombre total d'ordonnances, le nombre de doses prescrites et le coût total des médicaments prescrits. ▪ Si les médicaments sont disponibles dans différentes unités de dose (par exemple, 10 mg, 20 mg), des rapports distincts doivent être créés pour chaque unité de dose. ▪ L'accès à tous les rapports de coûts doit être limité aux utilisateurs autorisés figurant sur une liste de contrôle d'accès de gestion.

2.2. Exigences fonctionnelles et non fonctionnelles

Les exigences du système sont souvent classées en exigences fonctionnelles ou en exigences non fonctionnelles.

2.2.1. Exigences fonctionnelles

Ce sont des déclarations de services que le système devrait fournir, comment le système devrait réagir à des entrées particulières et comment le système devrait se comporter dans des situations particulières. Dans certains cas, les exigences fonctionnelles peuvent également indiquer explicitement ce que le système ne doit pas faire [75].

Lorsque les exigences fonctionnelles sont exprimées en tant qu'exigences de l'utilisateur, les exigences fonctionnelles sont généralement décrites d'une manière abstraite qui peut être comprise par les utilisateurs du système. Cependant, des exigences système fonctionnelles plus spécifiques décrivent en détail les fonctions du système, ses entrées et ses sorties, les exceptions, etc. Les exigences fonctionnelles du système varient d'exigences générales couvrant ce que le système doit faire à des exigences très spécifiques reflétant les méthodes de

travail locales ou les systèmes existants d'une organisation.

Remarque : Les exigences de domaine sont des contraintes sur le système dans son contexte d'utilisation. Donc, ils sont dérivés du domaine d'application du système plutôt que des besoins spécifiques des utilisateurs du système. Il peut s'agir de nouvelles exigences fonctionnelles à part entière, limiter les exigences fonctionnelles existantes ou définir la manière dont des calculs particuliers doivent être effectués. Le problème avec les exigences de domaine est que les ingénieurs logiciels peuvent ne pas comprendre les caractéristiques du domaine dans lequel le système fonctionne. Souvent, ils ne peuvent pas dire si une exigence de domaine a été manquée ou est en conflit avec d'autres exigences.

2.2.2. Exigences non fonctionnelles

Ce sont des contraintes sur les services ou fonctions offerts par le système. Ils peuvent inclure de nombreuses préoccupations telles que les contraintes de temps, les contraintes de sécurité, les contraintes de confidentialité, les contraintes de fiabilité, de flexibilité …etc. ils sont parfois appelées exigences de qualité, exigences extra-fonctionnelles ou qualités de service.

Les exigences non fonctionnelles s'appliquent souvent au système dans son ensemble, plutôt qu'à des fonctionnalités ou services individuels du système [75]. Donc, les exigences non fonctionnelles sont liées aux propriétés globales qui sont généralement difficiles à vérifier dans leur première formulation. Notons que, Les exigences d'un système embarqué sont plus orientées vers les contraintes matérielles.

En réalité, la distinction entre les différents types d'exigences n'est pas aussi claire. Une exigence d'utilisateur concernant la sécurité, telle qu'une déclaration limitant l'accès aux utilisateurs autorisés, peut semblé être une exigence non fonctionnelle. Cependant, lorsqu'elle est développée plus en détail, cette exigence peut générer d'autres exigences qui sont clairement fonctionnelles, telles que la nécessité d'inclure des fonctions d'authentification des utilisateurs dans le système. Cela montre que les exigences ne sont pas indépendantes et qu'une exigence génère ou contraint souvent d'autres exigences. Les exigences du système ne se contentent donc pas de spécifier les services ou les fonctionnalités du système qui sont requis ; ils spécifient également les fonctionnalités nécessaires pour garantir que ces services et fonctionnalités sont correctement fournis.

Les exigences non fonctionnelles sont souvent plus critiques que les exigences fonctionnelles

individuelles [75]. Les utilisateurs du système peuvent généralement trouver des moyens de contourner une fonction du système qui ne répond pas vraiment à leurs besoins. Toutefois, le fait de ne pas répondre à une exigence non fonctionnelle peut signifier que l'ensemble du système est inutilisable. Par exemple, si un système d'aéronef ne répond pas à ses exigences de fiabilité, il ne sera pas certifié comme sûr pour l'exploitation ; si un système de contrôle embarqué ne répond pas à ses exigences de performance, les fonctions de contrôle ne fonctionneront pas correctement.

Bien qu'il soit souvent possible d'identifier les composants du système qui implémentent des exigences fonctionnelles spécifiques (par exemple, il peut y avoir des composants de formatage qui implémentent des exigences de rapport), il est souvent plus difficile de relier les composants à des exigences non fonctionnelles. La mise en œuvre de ces exigences peut être diffusée dans tout le système. Il y a deux raisons à cela :

a. Les exigences non fonctionnelles peuvent affecter l'architecture globale d'un système plutôt que les composants individuels. Par exemple, pour vous assurer que les exigences de performances sont satisfaites, vous devrez peut-être organiser le système pour minimiser les communications entre les composants.
b. Une seule exigence non fonctionnelle, telle qu'une exigence de sécurité, peut générer un certain nombre d'exigences fonctionnelles connexes qui définissent les nouveaux services système requis. En outre, cela peut également générer des exigences qui restreignent les exigences existantes.

Finalement, La figure suivante présente une classification des exigences non fonctionnelles. Vous pouvez voir sur ce diagramme que les exigences non fonctionnelles peuvent provenir des caractéristiques requises du système comme le temps de réponse, l'utilisation des ressources, la disponibilité, la fiabilité, etc. (exigences produit), de l'organisation développant le système (exigences organisationnelles) ou bien de sources externes (exigences externes).

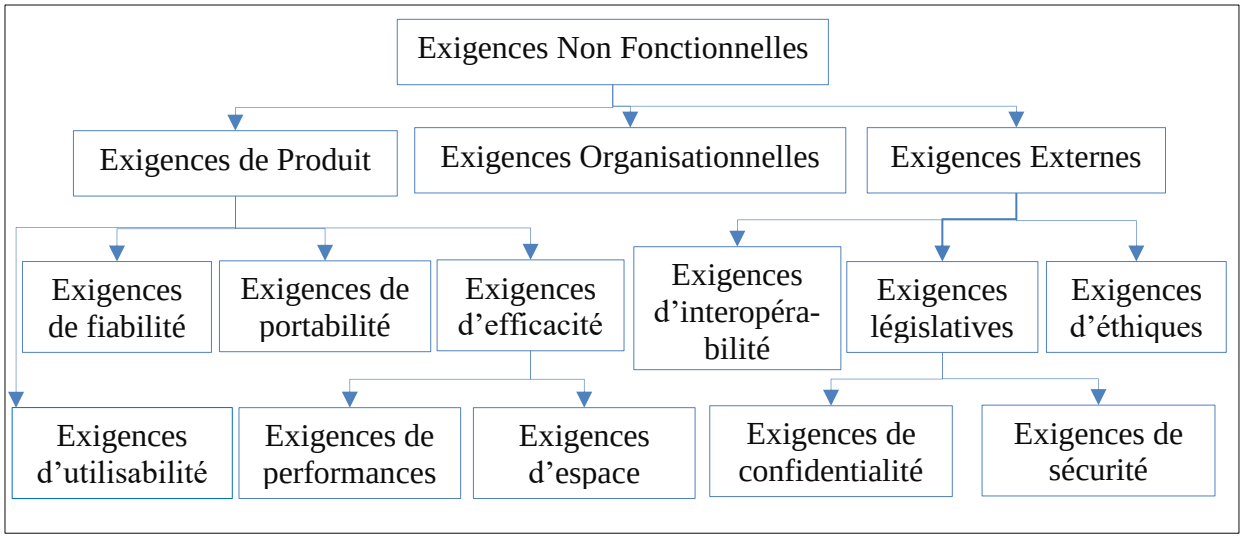


Figure 2-1 : Classification des exigences non fonctionnelles **[75]**.

3. Processus d'ingénierie des exigences

Le processus d'ingénierie des exigences se compose de deux processus principaux, le développement des exigences et la gestion des exigences, comme illustré dans la figure suivante. Donc, Le processus vise à produire un document d'exigences convenu qui spécifie un système répondant aux exigences des parties prenantes. Notons que, les parties prenantes (intervenants ; ou stakeholders) d'un système sont les suivants : clients, utilisateur final du système, Expert du domaine, développeurs, gestionnaires de projet ou bien tout autre personne qui apporte une valeur ajoutée au futur système.

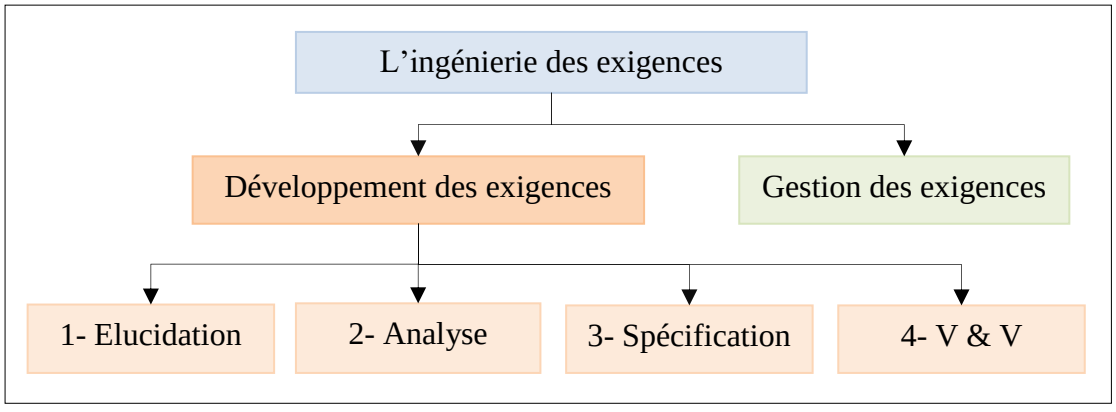


Figure 2-2 : Processus d'ingénierie des exigences de haut niveau.

Le cycle de vie du processus d'ingénierie des exigences couvre cinq activités distinctes mais liées. Le développement des exigences comprend l'élucidation des exigences, l'analyse, la

spécification et la validation des exigences. Tandis que, la gestion des exigences couvre la gestion des documents d'exigences et les éventuelles demandes de modification pour ceux-ci après avoir passé toutes les phases de développement des exigences. En pratique, l'ingénierie des exigences est un processus itératif dans lequel les activités sont imbriquées. Chaque itération du cycle est déclenchée par un besoin d'adapter, de réviser ou d'atteindre le document de spécification.

Les connexions des activités du cycle de vie sont décrites dans la Figure suivante.

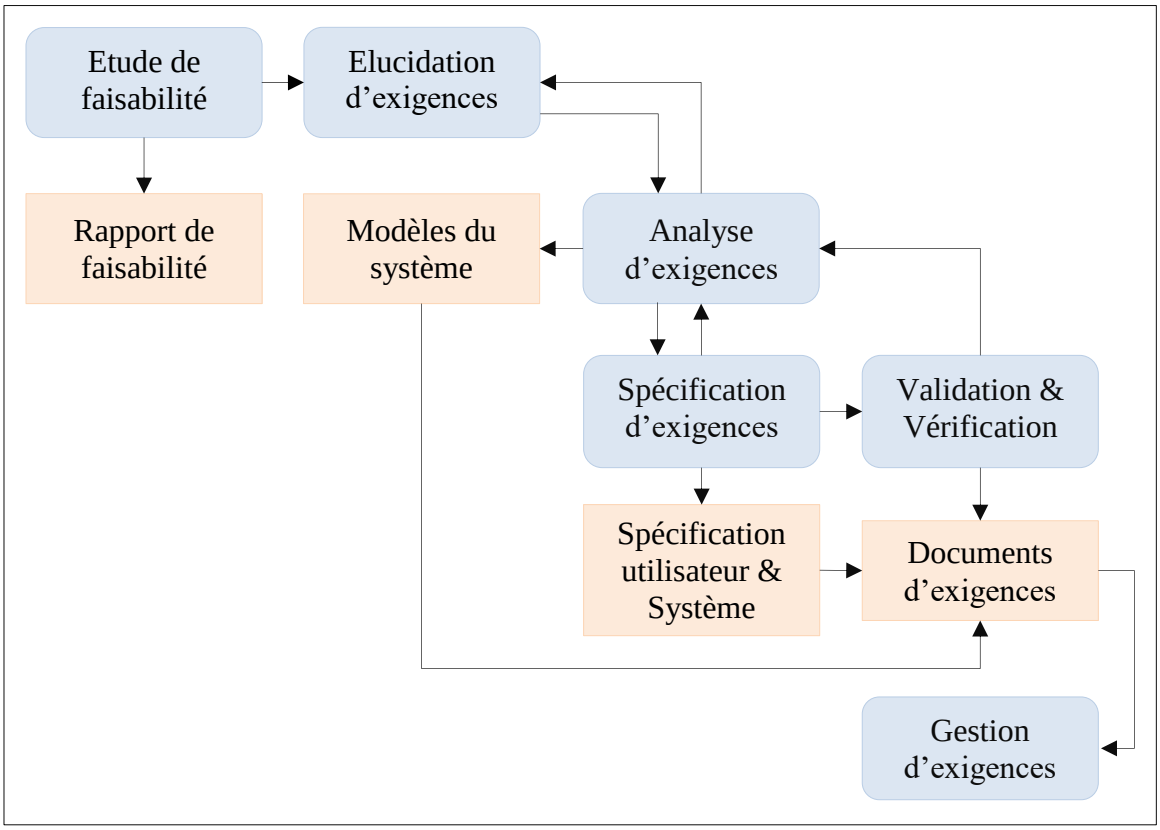


Figure 2-3 : Processus d'ingénierie des exigences.

3.1.Etude de faisabilité

L'étude de faisabilité est une étude courte et ciblée qui doit avoir lieu au début du processus d'ingénierie des exigences. Il doit répondre à trois questions clés :

a. le système contribue-t-il aux objectifs généraux de l'organisation ?
b. le système peut-il être mis en œuvre dans les délais et le budget disponible en utilisant la technologie actuelle ? et
c. le système peut-il être intégré à d'autres systèmes utilisés ?

Si la réponse à l'une de ces questions est non, vous ne devriez probablement pas poursuivre le projet.

3.2. L'élucidation

Dans cette activité, les ingénieurs système communiquent avec les clients et les utilisateurs finaux du système pour découvrir le domaine d'application, les services que le système doit fournir, les performances requises du système, les contraintes matérielles et logicielles, etc.

Obtenir et comprendre les exigences des parties prenantes du système est un processus difficile pour plusieurs raisons :

a. Les parties prenantes ne savent souvent pas ce qu'elles attendent d'un système embarqué, sauf dans les termes les plus généraux ; ils peuvent trouver difficile d'articuler ce qu'ils veulent que le système fasse ; ils peuvent faire des demandes irréalistes parce qu'ils ne savent pas ce qui est et ce qui n'est pas faisable.
b. Les parties prenantes d'un système expriment naturellement des exigences dans leurs propres termes et avec une connaissance implicite de leur propre travail. Les ingénieurs des exigences, sans expérience dans le domaine du client, peuvent ne pas comprendre ces exigences.
c. Différentes parties prenantes ont des exigences différentes et peuvent les exprimer de différentes manières. Les ingénieurs des exigences doivent découvrir toutes les sources potentielles d'exigences et découvrir les points communs et les conflits.
d. L'environnement économique et commercial dans lequel se déroule l'analyse est dynamique. Il change inévitablement au cours du processus d'analyse. L'importance d'exigences particulières peut changer. De nouvelles exigences peuvent émerger de nouvelles parties prenantes qui n'ont pas été consultées à l'origine.

La figure suivante montre les différentes techniques d'élucidation des exigences.

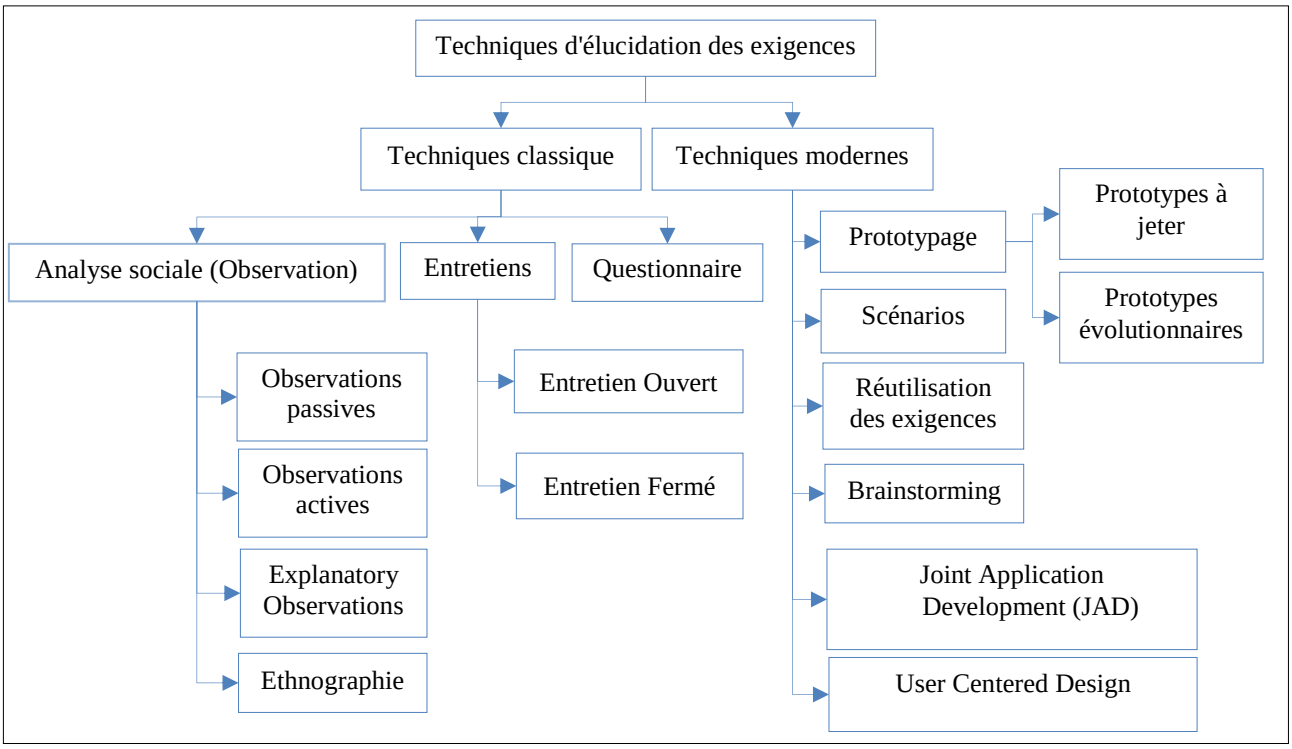


Figure 2-4 : Les techniques d'élucidation.

3.3. L'analyse

Inévitablement, les différentes parties prenantes ont des points de vue différents sur l'importance et la priorité des exigences et, parfois, ces points de vue sont contradictoires. Au cours du processus, les analystes devront organiser des négociations régulières avec les parties prenantes afin de parvenir à des compromis. Il est impossible de satisfaire complètement toutes les parties prenantes, mais si certaines parties prenantes estiment que leurs points de vue n'ont pas été correctement pris en compte, elles peuvent délibérément tenter de saper le processus d'ingénierie des exigences.

3.4. La spécification

L'imprécision dans la spécification des exigences est la cause de nombreux problèmes. Pour cela, le défi de l'ingénierie des exigences est d'atteindre une spécification d'exigences complète et non ambiguë.

La diversité des utilisateurs possibles signifie que le document d'exigences doit être un compromis entre la communication des exigences aux clients, la définition précise des exigences pour les développeurs et les testeurs, et l'inclusion d'informations sur l'évolution

possible du système.

Le niveau de détail devant être inclus dans un document d'exigences dépend du type de système en cours de développement et du processus de développement utilisé [75]. Les systèmes critiques doivent avoir des exigences détaillées car la sûreté et la sécurité doivent être analysées en détail. Lorsque le système doit être développé par une société distincte, les spécifications du système doivent être détaillées et précises. Si un processus de développement itératif interne est utilisé, le document d'exigences peut être beaucoup moins détaillé et toute ambiguïté peut être résolue pendant le développement du système.

La spécification des exigences est le processus consistant à écrire les exigences de l'utilisateur et du système dans un document d'exigences. Idéalement, les exigences de l'utilisateur et du système doivent être claires, sans ambiguïté, faciles à comprendre, complètes et cohérentes. Dans la pratique, cela est difficile à réaliser car les parties prenantes interprètent les exigences de différentes manières et il existe souvent des conflits et des incohérences inhérents aux exigences.

Les exigences des utilisateurs pour un système doivent décrire les exigences fonctionnelles et non fonctionnelles afin qu'elles soient compréhensibles par les utilisateurs du système qui n'ont pas de connaissances techniques détaillées. Idéalement, ils ne devraient spécifier que le comportement externe du système. Le document d'exigences ne doit pas inclure de détails sur l'architecture ou la conception du système. Par conséquent, si vous rédigez des exigences utilisateur, vous ne devez pas utiliser de jargon matériel ou logiciel, de notations structurées ou de notations formelles. Vous devez rédiger les exigences des utilisateurs en langage naturel, avec des tableaux simples, des formulaires et des diagrammes intuitifs.

Les exigences système sont des versions étendues des exigences utilisateur qui sont utilisées par les ingénieurs logiciels/Matériel comme point de départ pour la conception du système. Ils ajoutent des détails et expliquent comment les besoins des utilisateurs doivent être fournis par le système. Ils peuvent être utilisés dans le cadre du contrat de mise en œuvre du système et doivent donc constituer une spécification complète et détaillée de l'ensemble du système.

En principe, la spécification des exigences d'un système doit être à la fois complète et cohérente. La complétude signifie que tous les services requis par l'utilisateur doivent être définis. La cohérence signifie que les exigences ne doivent pas avoir de définitions contradictoires. En pratique, pour les grands systèmes complexes, il est pratiquement

impossible d'atteindre la cohérence et l'exhaustivité des exigences. L'une des raisons en est qu'il est facile de faire des erreurs et des omissions lors de la rédaction de spécifications pour des systèmes complexes. Une autre raison est qu'il y a beaucoup de parties prenantes dans un grand système. Une partie prenante est une personne ou un rôle qui est affecté par le système d'une manière ou d'une autre. Les parties prenantes ont des besoins différents et souvent incohérents. Ces incohérences peuvent ne pas être évidentes lorsque les exigences sont spécifiées pour la première fois, de sorte que des exigences incohérentes sont incluses dans la spécification. Les problèmes peuvent n'apparaître qu'après une analyse plus approfondie ou après la livraison du système au client.

Idéalement, les exigences du système devraient simplement décrire le comportement externe du système et ses contraintes opérationnelles. Ils ne devraient pas se préoccuper de la façon dont le système devrait être conçu ou mis en œuvre. Cependant, au niveau de détail requis pour spécifier complètement un système complexe, il est pratiquement impossible d'exclure toutes les informations de conception. Il y a plusieurs raisons à cela:

a. On doit peut-être concevoir une architecture initiale du système pour aider à structurer la spécification des exigences. Les exigences du système sont organisées selon les différents sous-systèmes qui composent le système. Cette définition architecturale est essentielle si vous souhaitez réutiliser des composants logiciels ou matériels lors de la mise en œuvre du système.
b. Dans la plupart des cas, les systèmes doivent interagir avec les systèmes existants, ce qui limite la conception et impose des exigences au nouveau système.
c. L'utilisation d'une architecture spécifique pour satisfaire des exigences non fonctionnelles peut être nécessaire. Un régulateur externe qui doit certifier que le système est sûr peut spécifier qu'une conception architecturale déjà certifiée doit être utilisée.

Le référentiel d'exigences doit être :

- *Complet :* Toutes les exigences existent et sont complètes,
- *Cohérent :* Les exigences ne se contredisent pas,
- *Doté d'une structure claire,*
- *Modifiable :* Modification du produit possible, et
- *Extensible :* Evolution du produit possible.

Un certain nombre de grandes organisations, telles que le département américain de la Défense et l'IEEE, ont défini des normes pour les documents d'exigences, comme la norme ISO/IEC/IEEE 29148:2018. Celles-ci sont généralement très génériques mais sont néanmoins utiles comme base pour développer des normes plus détaillées.

Les exigences des utilisateurs sont presque toujours écrites en langage naturel complétées par des diagrammes et des tableaux appropriés dans le document des exigences. Les exigences système peuvent également être écrites en langage naturel, mais d'autres notations basées sur des formulaires, des modèles de système graphiques ou des modèles de système mathématiques peuvent également être utilisées. Le tableau suivant résume les notations possibles qui pourraient être utilisées pour écrire les exigences du système.

Tableau 2-2 : Façons d'écrire une spécification d'exigences système.

Notation	Description
1- Phrases en langage naturel	Les exigences sont rédigées à l'aide de phrases numérotées en langage naturel. Chaque phrase doit exprimer une exigence.
2- Langage naturel structuré	Les exigences sont écrites en langage naturel sur une forme standard ou un modèle. Chaque champ fournit des informations sur un aspect de l'exigence.
3- Notations graphiques	Des modèles graphiques, complétés par des annotations textuelles, sont utilisés pour définir les exigences fonctionnelles du système ; Les diagrammes de cas d'utilisation et de séquence UML sont couramment utilisés.

4- Spécifications mathématiques	Ces notations sont basées sur des concepts mathématiques tels que les machines à états finis ou les ensembles. Bien que ces spécifications non ambiguës puissent réduire l'ambiguïté d'un document d'exigences, la plupart des clients ne comprennent pas une spécification formelle. Ils ne peuvent pas vérifier qu'il représente ce qu'ils veulent et sont réticents à l'accepter comme un contrat système. Donc, les spécifications mathématiques formelles sont généralement utilisées pour décrire les exigences des systèmes critiques pour la sûreté ou la sécurité, mais sont rarement utilisées dans d'autres circonstances.

3.5. La validation

Le référentiel d'exigences est l'entrée principale de la phase de réalisation du nouveau système embarqué. Pour cela, il doit être validé par les parties prenantes afin de repérer les insuffisances par rapport aux besoins réels (Les exigences doivent être testables sinon elles ne sont que des buts). Donc, la validation des exigences est importante car les erreurs dans le référentiel d'exigences peuvent entraîner des coûts importants lorsque ces problèmes sont découverts pendant le développement ou après la mise en service du système.

Le résultat final de cette activité est un référentiel d'exigences approuvé.

Au cours du processus de validation des exigences, différents types de contrôles doivent être effectués sur les exigences du référentiel d'exigences. Ces contrôles comprennent :

a. **Contrôles de validité** : Un utilisateur peut penser qu'un système est nécessaire pour exécuter certaines fonctions. Cependant, une réflexion et une analyse plus approfondies peuvent identifier des fonctions supplémentaires ou différentes qui sont nécessaires.
b. **Contrôles de cohérence** *:* Les exigences du référentiel ne doivent pas entrer en conflit. Autrement dit, il ne devrait pas y avoir de contraintes contradictoires ou de descriptions différentes de la même fonction système.
c. **Contrôles d'exhaustivité** *:* Le référentiel d'exigences doit inclure des exigences qui

définissent toutes les fonctions et les contraintes prévues par l'utilisateur du système.

d. **Contrôles de réalisme** *:* à base des connaissances des technologies existantes, les exigences doivent être vérifiées pour s'assurer qu'elles peuvent effectivement être mises en œuvre. Ces vérifications doivent également tenir compte du budget et du calendrier de développement du système.
e. **Vérifiabilité** *:* Pour réduire le risque de litige entre le client et l'entrepreneur, les exigences du système doivent toujours être écrites de manière à être vérifiables. Cela signifie que vous devez être en mesure d'écrire un ensemble de tests qui peuvent démontrer que le système livré répond à chaque exigence spécifiée.

Finalement, Il existe un certain nombre de techniques de validation des exigences qui peuvent être utilisées individuellement ou conjointement les unes avec les autres :

a. **Revues d'exigences** *:* Les exigences sont analysées systématiquement par une équipe de relecteurs qui vérifient les erreurs et les incohérences.
b. **Prototypage** : Dans cette approche de validation, un modèle exécutable du système en question est présenté aux utilisateurs finaux et aux clients. Ils peuvent expérimenter ce modèle pour voir s'il répond à leurs besoins réels.
c. **Génération de cas de test** : Les exigences doivent pouvoir être testées. Si les tests des exigences sont conçus dans le cadre du processus de validation, cela révèle souvent des problèmes d'exigences. Si un test est difficile ou impossible à concevoir, cela signifie généralement que les exigences seront difficiles à mettre en œuvre et doivent être reconsidérées.

3.6. La gestion des exigences

Les besoins, les priorités ou les circonstances peuvent modifier les exigences au fil du temps. Une procédure de gestion des changements doit être mise en place pour évaluer, approuver ou rejeter les modifications proposées tout en réduisant les risques pour le projet.

Pour une bonne gestion, la traçabilité est nécessaire.

4. Défis pour l'ingénierie des exigences des systèmes embarqués

Voici une liste des défis de l'ingénierie des exigences des systèmes embarqués :

- **Complexité** : Les systèmes embarqués sont souvent très complexes, avec de

nombreuses interactions entre les différents composants. Cela rend l'ingénierie des exigences plus difficile, car il est plus difficile de comprendre les exigences et de s'assurer qu'elles sont correctement spécifiées.

- **Contraintes matérielles** : Les systèmes embarqués sont souvent soumis à des contraintes matérielles strictes, telles que la taille, la consommation d'énergie et la puissance de traitement limitée. Cela peut rendre difficile la satisfaction de toutes les exigences du système.
- **Évolution des exigences** : Les exigences des systèmes embarqués peuvent évoluer rapidement en raison de l'évolution des technologies, des besoins des utilisateurs ou des contraintes du marché. Cela peut rendre difficile la gestion des exigences et la garantie que toutes les parties prenantes sont satisfaites.
- **Intégration** : Les systèmes embarqués sont souvent intégrés dans des environnements complexes, tels que des véhicules, des avions ou des équipements industriels. Cela peut rendre difficile la spécification des exigences et la garantie que le système fonctionnera correctement dans son environnement.
- **Sécurité** : Les systèmes embarqués sont souvent utilisés dans des applications critiques, telles que les systèmes de contrôle de vol ou les systèmes de freinage. Cela signifie que la sécurité est une préoccupation majeure, et que les exigences de sécurité doivent être correctement spécifiées et vérifiées.

En résumé, les défis de l'ingénierie des exigences des systèmes embarqués incluent la complexité, les contraintes matérielles, l'évolution des exigences, l'intégration et la sécurité. Ces défis doivent être pris en compte lors de la spécification, de la vérification et de la validation des exigences des systèmes embarqués.

5. Les approches de l'ingénierie des exigences

Dans la littérature, il existe plusieurs approches d'ingénierie des exigences. Chacune d'entre elles se concentrent sur des activités spécifiques du processus d'ingénierie des exigences. Dans ce qui suit, nous présentons brièvement quelques approches.

5.1. L'approche orientées buts (GORE)

L'approche orientée but (GORE[1]) est une méthode de l'ingénierie des exigences qui se concentre sur la définition des objectifs ou des buts d'un système plutôt que sur les fonctionnalités spécifiques. Donc, la source des exigences fonctionnelles et non fonctionnelles sont les objectifs (buts) des acteurs.

Avant de travailler à l'élucidation des exigences, il convient de se concentrer sur l'élucidation des buts. Donc, L'approche orientée but modélise les buts. Ces derniers peuvent être vus comme les propriétés désirées du système qui ont été exprimées par les utilisateurs.

Les objectifs peuvent être exprimés à différents niveaux d'abstraction, allant des objectifs stratégiques de haut niveau à des objectifs opérationnels. Les buts de haut niveau peuvent être progressivement raffinés en buts plus concrets et finalement opérationnels en établissant des relations reliant un but parent à plusieurs buts fils, avec des conditions de satisfaction différentes, soit « ET » (tous les fils sont nécessaires) soit « OU » (un fil est suffisant : des alternatives sont possibles).

Il existe un certain nombre d'avantages revendiqués dans la littérature concernant cette approche[76]. En voici un résumé :

a. **Exhaustivité et pertinence des exigences** : Les objectifs permettent d'assurer l'exhaustivité et la pertinence d'une spécification d'exigences.
b. **Justification des exigences** : Une exigence existe parce qu'elle satisfait ses objectifs supérieurs. Toute exigence qui ne contribue à aucun objectif ne sera pas prise en compte. Pour cette raison, chaque exigence aura une justification. Expliquer les exigences aux parties prenantes est une autre question importante. Les objectifs fournissent la justification des exigences.
c. **Traçabilité** : Les graphiques d'objectifs fournissent des liens de traçabilité allant des exigences de bas niveau aux objectifs de haut niveau.
d. **Gestion des conflits** : Les contributions entre les objectifs (positives ou négatives) peuvent être modélisées et gérées. De cette manière, les conflits peuvent être identifiés et résolus.

[1] Goal Oriented Requirements Engineering.

e. **Gestion de l'évolution des exigences** : Plus un objectif est de haut niveau, plus il a de chances d'être stable. Les objectifs sont donc des éléments essentiels pour gérer l'évolution des exigences.

Parmi les variantes, on peut citer : NFR Framework (qui se concentre sur les exigences non fonctionnelles), GBRAM[1], AGORA[2], I* (I Star) ou la méthode KAOS[3].

5.2.L'ingénierie des exigences a base des scénarios

Les acteurs de l'organisation ont parfois des difficultés à énoncer à priori les objectifs attendus du système; ceux-ci ne sont facilement explicités que lorsque les acteurs ont bien compris ce que doit faire le système.

Selon [77], « *Un scénario est une séquence d'interactions entre le système envisagé et son environnement énoncée dans le contexte restreint d'un propos particulier. Il décrit une histoire, c'est un écrit concret de la façon dont un acteur imagine d'interagir avec le système* ».

Cependant, selon les méthodologies, le terme scénario porte des sémantiques bien différentes[78]. On associe parfois les scénarios à des traces d'événements internes et/ou externes, des échanges de messages entre composants, des séquences d'interactions entre un système et l'utilisateur, ou encore des ensembles plus ou moins génériques de ces séquences. Donc, à travers les scénarios on peut capturer des exemples, des scènes, des descriptions narratives du contexte et du fonctionnement du système, des cas d'utilisation ou bien des exemples de comportement d'utilisateur.

Pour exprimer les scénarios, on peut utiliser les notations informelles, semi-formelles ou bien les notations formelles. Les scénarios informels sont décrits à l'aide du langage naturel[79]. Les scénarios semi-formels utilisent des notations structurées comme des tableaux [80] ou des scripts [81] et les scénarios formels sont décrits généralement à l'aide de langages basés sur des grammaires régulières [82] .

5.3. L'approche orientée-aspects (AORE)

Parmi les paradigmes de programmation, il existe la programmation orientée aspect qui a été défini par la société Xerox. L'avantage de ce paradigme est la capacité de séparer

[1] Goal-Based Requirements Analysis Method.
[2] Attributed Goal Oriented Requirements Analysis.
[3] Knowledge Acquisition in autOmated Specification, Keep All Objectives Satisfied.

l'implémentation de toutes les exigences (fonctionnelles ou non fonctionnelles) d'un logiciel. Le principe est donc de coder chaque problématique séparément et de définir leurs règles d'intégration pour les combiner en vue de former le système final. Donc, l'utilisation de ce paradigme dans la phase d'implémentation du cycle de développement des applications logicielles apporte des solutions aux difficultés du paradigme de l'objet qui sont la dispersion et l'enchevêtrement du code. Il existe plusieurs langages de programmation orientés aspect comme AspectJ, HyperJ, AspectC, JAC,.....etc.

L'ingénierie des exigences orientée aspect qui trouve son origine dans la programmation orientée aspect, consiste à séparer chaque exigence transversale en un aspect [83]. Donc, au lieu d'intégrer et de résoudre l'exigence transversale pour toutes les exigences qu'elle traverse, l'aspect est résolu de manière isolée. Par conséquent, l'orientation vers l'aspect conduit à une séparation claire des préoccupations. Pour combiner un aspect avec une exigence, un aspect définit un pointcut (ensemble de points de jointure), qui décrit comment l'aspect et une exigence peuvent être combinés. Malheureusement, L'utilisation du paradigme aspect, lors des premières phases du cycle de développement notamment dans la phase de l'ingénierie des exigences n'a pas atteint encore un stade de maturité suffisant.

Il existe quelques approches particulières dites « Early Aspects Approaches » comme Arcade [84] , ARGM[1] [85] et l'approche AOSD/UC[2].

5.4. Les approches par points de vue

Dans le domaine de l'IE, les vues ou points de vue [86] reflètent les compétences, objectifs et rôles de chaque participant dans le processus d'IE. Les points de vue multiples permettent certainement une meilleure élicitation des exigences, et ne peuvent être que bénéfiques au processus d'élicitation des exigences. Les points de vue sont utilisés pour guider la découverte, l'analyse et la gestion des exigences. La méthode PREview[3] est l'une des méthodes les plus connues utilisant la notion de point de vue.

5.5.L'approche schéma de problème (Problem Frames)

L'approche schéma de problèmes [87] suit généralement le principe de séparation des préoccupations. Il décompose le problème global de la construction du futur système en petits

[1] Aspects in Requirements Goal Models.
[2] Aspect Oriented Software Development with Use Cases.
[3] Process and Requirements Engineering VIEWpoint.

sous-problèmes qui s'inscrivent dans un cadre de problème. Chaque sous-problème est résolu par une machine, qui doit être spécifiée en utilisant les connaissances du domaine donné. Toutes les machines doivent être composées pour former la machine globale.

5.6. Les approches hybrides

Il existe des approches couplant les buts et les scénarios comme la méthode CREWS-l'Ecritoire [88] (CREWS pour Cooperative Requirements Engineering With Scenario). Elle est développée dans le cadre du projet ESPRIT[1] pour permettre la découverte semi-automatique des exigences à partir de scénarios textuels au moyen d'un couplage bidirectionnel but/scénario. L'Ecritoire est l'application logicielle de l'approche CREWS.

Pour gérer la complexité des exigeants du système et fournir une description cohérente et complète du système, [83] propose une méthode appelée AORE4PF[2] qui combine l'ingénierie des exigences orientée aspects et l'approche schéma de problème. AORE4PF fournit des conseils pour la classification des besoins, la séparation des préoccupations, la modélisation des exigences et la réalisation d'analyses de complétude et d'interaction.

6. Outils pour l'ingénierie des exigences

Pour une activité aussi complexe que l'ingénierie des exigences, le support d'outils puissants est primordial. Cependant, il a été constaté que la pratique courante dans ce domaine repose souvent sur des outils bureautiques classiques, tels que les éditeurs de texte et les tableurs, plutôt que sur des outils spécifiquement dédiés à l'ingénierie des exigences.

Parmi les outils disponibles, RE-Tools [89] se distingue comme une boîte à outils open source implémentée à l'aide de StarUML. Elle prend en charge de nombreuses notations de modélisation d'exigences, notamment les frameworks NFR, i*, KAOS, Problem Frames et UML. Un autre exemple est ReqView, un outil gratuit, simple et efficace pour la gestion des exigences. Bien qu'il permette de créer, structurer des exigences et générer des matrices de traçabilité, il offre cependant moins de fonctionnalités avancées comparé à des outils plus robustes.

Dans le domaine des systèmes embarqués, plusieurs outils sont utilisés spécifiquement lors de la phase d'ingénierie des exigences. Parmi eux, Zaki tool [90], un outil conçu pour soutenir les

[1] European Reactive Research Project.
[2] Aspect-Oriented Requirements Engineering for Problem Frames.

activités de GERSE, est particulièrement notable. Utilisé durant la phase d'élucidation des exigences, il a été développé avec la plateforme .NET, le langage C#, et SQL Server. Toutefois, cet outil n'est malheureusement pas disponible en téléchargement. L'outil DODT constitue un autre exemple pertinent, permettant de transformer de manière semi-automatique les exigences.

Enfin, IBM DOORS[1] est un autre outil robuste et bien établi pour la gestion des exigences. Il permet de créer, gérer et tracer les exigences tout au long du cycle de vie du projet. Bien qu'il soit particulièrement adapté aux projets complexes nécessitant une traçabilité rigoureuse, sa complexité et son coût peuvent constituer un obstacle pour les petites équipes.

7. Travaux voisins sur l'ingénierie des exigences des systèmes embarqués

Selon la littérature, il existe peu de recherches sur l'ingénierie des exigences des systèmes embarqués. Le tableau suivant présente une synthèse des travaux les plus connus.

1 fait référence à la phase d'élucidation, 2 fait référence à l'analyse, 3 fait référence à la spécification et 4 fait référence à la validation et à la vérification des exigences.

Tableau 2-3 : Synthèse des travaux les plus pertinents en ingénierie des exigences pour les systèmes embarqués.

N°	Le Travail	Réf.	Année	Le processus					Pris en charge par un outil	Evalué dans l'industrie	Evaluer par des experts
				Développement				Gestion			
				1	2	3	4				
01	The mapping of SysML-AD into an ETPN	[91]	2009	✕	✕	✓	✓	✕	✕	✕	✕
	Domaine D'application	Les SEs temps réel.									
02	TERASE	[92]	2010	✕	✕	✓	✕	✕	✕	✕	✕
	Domaine D'application	Les systèmes embarqués.									

[1] Dynamic Object-Oriented Requirements System.

03	CAMA	[90]	2010	✕	✕	✓	✕	✕	✕	✕	✕
	Domaine D'application	les systèmes embarqués dans l'automobile qui utilisent le protocole CAN pour échanger des informations.									
04	GERSE	[93]	2012	✓	✕	✕	✕	✕	ZAKI Tool	✕	✓
	Domaine D'application	Les systèmes embarqués.									
05	SysML-Sec	[94]	2013	✕	✕	✓	✕	✕	TTool	✕	✕
	Domaine D'application	Exigences de sécurité pour les systèmes embarqués.									
06	Requirements elicitation process through gamified approach	[95]	2016	✓	✕	✕	✕	✕	✕	✓	✕
	Domaine D'application	Cyber Physical Product Service Systems (CPS based PSS).									
07	REPES	[96]	2018	✓	✕	✕	✕	✕	✕	✕	✓
	Domaine D'application	Les systèmes embarqués.									
08	Model-based approach for requirement validation	[97]	2020	✕	✕	✕	✓	✕	scade tool	✓	✕
	Domaine D'application	Les systèmes embarqués dans l'industrie automobile.									

- SysML-AD into an ETPN: SysML Activity Diagram to Time Petri Net with Energy constraint.
- TERASE: Acronyme portugais pour "Template para Especificação de Requisitos de Ambiente em Sistemas Embarcados".
- CAMA: CAN, Martins and Almudi.
- GERSE: Acronyme portugais pour "Guia de Elicitação de Requisitos para Sistemas Embarcados".
- SysML-Sec: SysML Security requirement.
- REPES: Requirements Engineering Process for Embedded Systems.

8. L'ingénierie des exigences des SEAAs

Dans les systèmes auto-adaptatifs (SAAs), il est difficile de connaître toutes les modifications apportées aux exigences au moment de la conception et, en particulier, il peut être impossible d'énumérer toutes les alternatives possibles. A cet effet, l'ingénierie des exigences pour les SAAs est un processus complexe visant à identifier, analyser, spécifier et valider les exigences d'un système capable de réagir automatiquement à des changements internes ou externes imprévisibles. Par conséquent, par rapport aux approches traditionnelles de spécification des exigences système, les langages et notations utilisés pour ces systèmes doivent explicitement prendre en compte les incertitudes inhérentes. Donc, pour atténuer cette incertitude, [7], [8]propose le langage RELAX. Il est basé sur un langage naturel structuré, permet d'assouplir les exigences liées au comportement adaptatif et il prend en charge deux types d'incertitudes : les conditions changeantes de l'environnement d'exécution, qui peuvent être difficiles à prévoir , telles que les pannes de capteurs, les réseaux bruyants, les menaces malveillantes et les interventions (humaines) inattendues (*incertitude environnementale*), et les variations de comportement du système lors de l'exécution, en réponse soit à un manque d'informations sur le comportement attendu, soit à l'incertitude environnementale (*incertitude comportementale*). A cet effet, RELAX identifie deux types d'exigences : les exigences invariantes et les exigences RELAXées (variantes)[8]. Les exigences RELAXées peuvent être fonctionnelles ou non fonctionnelles et représentent une version améliorée des exigences d'origine[98]. Donc, une fois que l'ingénieur des exigences a déterminé qu'un certain niveau de flexibilité peut être toléré dans les exigences, il appartient alors aux développeurs en aval, y compris les concepteurs et les développeurs, d'incorporer les mécanismes adaptatifs les plus appropriés pour prendre en charge la fonctionnalité requise.

Par ailleurs, il est important d'indiquer quels sont les facteurs d'incertitude qui justifient un assouplissement de ces exigences, ce qui nécessite un comportement adaptatif. Chaque facteur d'incertitude est spécifié en utilisant les mots clés suivant :

- **MON** (Monitor) : définit un ensemble de propriétés qui peuvent être surveillées par le système,
- **ENV** (Environment) : définit un ensemble de propriétés qui définissent l'environnement du système,
- **REL** (Relation): indique le lien entre les deux précédents, il est utilisé pour spécifier

de quelle manière les observables (MON) peuvent être utilisés pour dériver des informations sur l'environnement (ENV). A titre d'exemple [7], un avion doté uniquement d'un équipement de radiogoniométrie ne peut pas déterminer sa position directement. Il peut cependant mesurer sa distance par rapport à un ensemble de points de repère connus et doit déduire sa position à partir de ces mesures. Dans le langage RELAX, la position de l'avion est une propriété de l'environnement, tandis que les distances aux points de repère sont des éléments observables. REL serait utilisé pour définir comment calculer la position à partir des mesures de distance.

- et **DEP** (Dependency) : Pour préciser les liens entre les exigences, ceux-ci peuvent être positifs ou négatifs.

Aussi, pour gérer les différentes incertitudes lors de la spécification des exigences et pour donner plus de souplesse dans la façon dont la contrainte doit être prise en compte, le langage propose trois types d'opérateurs de relaxation : modal, temporel et ordinal. Chacun de ces opérateurs définit des contraintes sur la manière dont une exigence peut être assouplie au moment de l'exécution.

La figure suivante présente un aperçu global du processus de RELAXation des exigences. Il convient de noter qu'avant l'application de ce processus, nous présumons qu'un processus traditionnel de découverte des exigences a été mené afin de produire un ensemble d'énoncés SHALL.

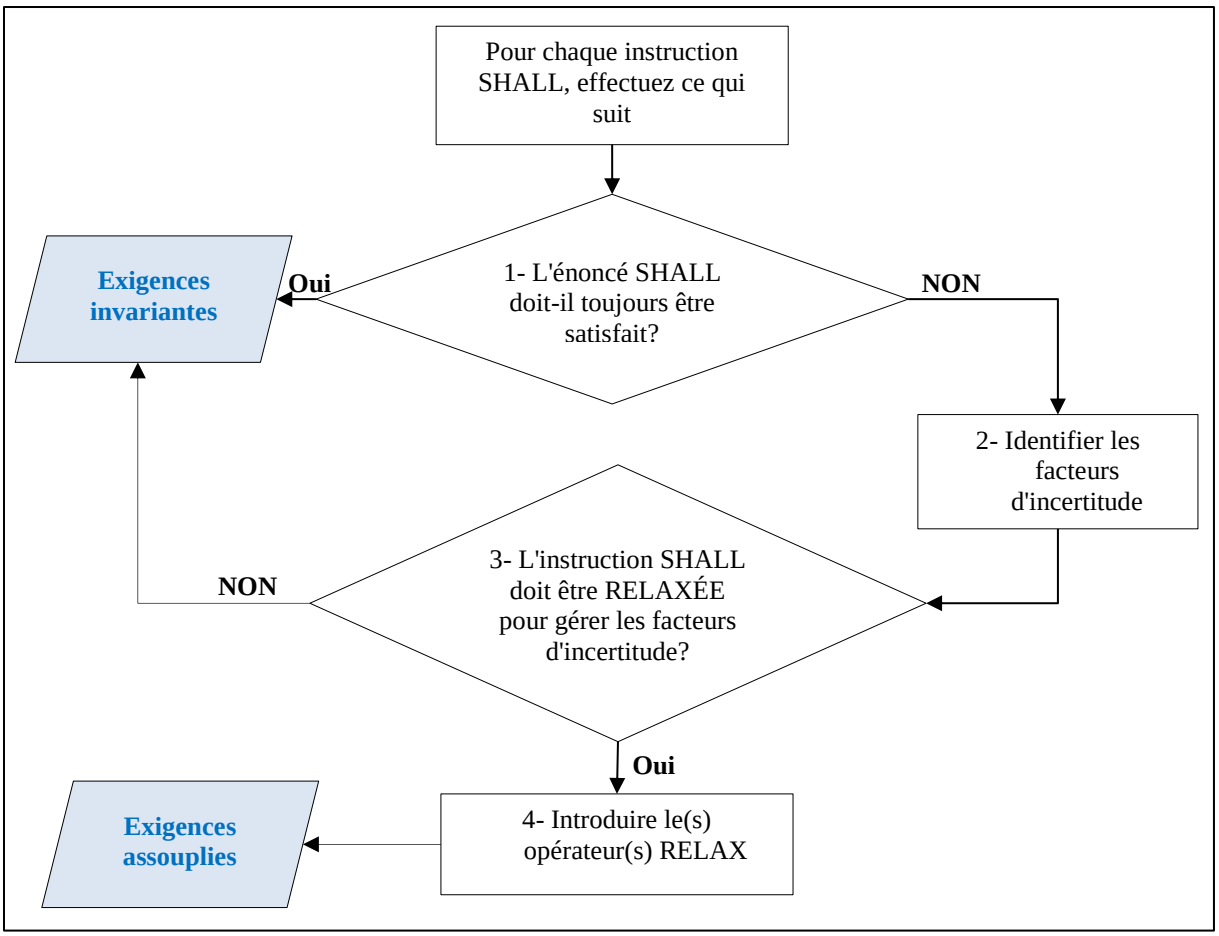


Figure 2-5 : Le processus RELAX **[7]**.

Enfin, pour dériver les exigences sous forme graphique à partir d'exigences textuelles, représentées par des diagrammes d'exigences SysML, [99] combine trois approches : l'approche orientée objectifs KAOS, SysML et le langage RELAX.

D'un autre coté, les SAAs qui sont basée sur la boucle de rétroaction (feedback loop) doivent non seulement répondre aux exigences traditionnelles (fonctionnelles et non fonctionnelles), mais aussi il faut intégrer des exigences spécifiques à l'auto-adaptation. Pour cela, une autre approche de l'ingénierie des exigences des systèmes auto-adaptatifs part de la question suivante:

Quel problème lié aux exigences une boucle de rétroaction pour l'auto-adaptation cherche-t-elle à résoudre ?

Donc, En réponse à la question, la deuxième approche prend en charge le problème des exigences à résoudre par une boucle de rétroaction du point de vue de l'état succès/échec d'exécution des autres exigences du système, c'est-à-dire les exigences du système géré.

Par conséquent, les exigences d'un système de gestion sont des méta-exigences[55] (*Meta-Requirements*) des exigences du système géré. Ces méta-exigences représentent les préoccupations du système de gestion dans le modèle conceptuel des systèmes auto-adaptatifs. Cette approche est basée sur les notions d'exigences de conscience[100] (*AwReqs* pour *Awareness Requirements*) et d'exigences d'évolution [101] (*EvoReqs* pour *Evolution Requirements*), qui sont des concepts de modélisation bien connus pour spécifier les méta-exigences des systèmes auto-adaptatifs. Les deux types d'exigences se complètent : les exigences de conscience indiquent les situations qui nécessitent une adaptation et les exigences d'évolution prescrivent ce qu'il faut faire dans ces situations [101].

Le tableau suivant présente les différents types d'exigences de conscience [55].

Tableau 2-4 : Différents types d'exigences de conscience.

Type	Brève description
Regular	Une exigence qui ne doit jamais échouer.
Aggregate	Impose une contrainte sur le taux de réussite/échec d'une autre exigence, qui peut faire référence à une fréquence, un intervalle ou une limite sur la réussite/échec.
Trend	Permet de comparer les taux de réussite/échec sur un certain nombre de périodes afin de spécifier comment les taux de réussite/échec d'une exigence évoluent au fil du temps.
Delta	Permet de spécifier des seuils acceptables pour la satisfaction d'une exigence.
Meta	Une exigence de conscience à propos d'une autre exigence de conscience.

En combinant les deux, on peut spécifier les exigences pour une boucle de rétroaction qui opérationnalise l'adaptation au moment de l'exécution. Enfin, il faut aussi déterminer les exigences fonctionnelles des boucles de rétroaction qui sont essentielles pour le fonctionnement efficace des systèmes auto-adaptatifs.

Voici une liste d'exemples d'exigences fonctionnelles de la boucle de rétroaction.

- **Précision et Fiabilité de la Surveillance :** Les mécanismes de surveillance doivent être précis et fiables pour fournir des données exactes et en temps réel sur l'état du système.
- **Capacité d'Analyse :** Les mécanismes d'analyse doivent être capables de traiter les données collectées, de détecter les problèmes potentiels, et de comprendre les implications des changements dans l'environnement.

- **Flexibilité de la Planification**: Les mécanismes de planification doivent être suffisamment flexibles pour générer des plans d'action adaptés à une variété de situations et de conditions.
- **Efficacité de l'Exécution:** Les mécanismes d'exécution doivent être capables de mettre en œuvre les adaptations rapidement et efficacement, en minimisant les interruptions et les impacts négatifs sur le système.

Donc, le processus d'ingénierie des exigences des systèmes embarqués auto-adaptatifs SEAAs doit prendre en compte à la fois les contraintes des systèmes embarqués, souvent limités en termes de ressources (mémoire, puissance de traitement, énergie), et celles des systèmes auto-adaptatifs, telles que la gestion des incertitudes.

Malheureusement, la littérature présente une pénurie d'études, méthodes, techniques et outils d'ingénierie des exigences spécifiquement développés pour le domaine des systèmes embarqués auto-adaptatifs (SEAAs). À titre d'exemple, dans le cadre d'un projet de développement de systèmes d'assistance à la vie autonome (AAL), [102] a étudié l'adéquation de trois langages de spécification d'exigences pour les SAAs. À cet effet, [102] a comparé les langages RELAX, Tropos4AS et GODA. L'étude, réalisée auprès de 14 personnes, conclut que la compréhension des spécifications est similaire pour les trois langages, mais qu'elle présente certaines difficultés pour les utilisateurs. Plus précisément, ces derniers estiment que la spécification RELAX demande moins d'efforts de compréhension et les rend moins frustrés.

Pour construire un système de transport intelligent ayant la capacité d'auto-adaptation, Afin de concevoir un système de transport intelligent (ITS[1]) doté de capacités d'auto-adaptation, [103] propose une approche pour identifier les exigences auto-adaptatives. Il appuie sur un modèle basé sur des scénarios et un modèle orienté but, en suivant les étapes du processus d'ingénierie des exigences avec MAPE-K. L'approche KAOS est utilisée pour créer un modèle d'objectifs à l'aide de l'outil Objectiver, tandis que la syntaxe RELAX est employée pour formuler les exigences adaptatives.

[1] Intelligent Transportation System.

9. Conclusion

Ce chapitre met en lumière l'importance cruciale de l'ingénierie des exigences pour garantir le bon développement des systèmes embarqués, notamment ceux dotés de capacités d'auto-adaptation. À cet effet, il récapitule les éléments clés relatifs à l'ingénierie des exigences dans le contexte des systèmes embarqués. Après avoir introduit la catégorisation des exigences selon leur niveau d'abstraction, une distinction a été établie entre les exigences fonctionnelles et non fonctionnelles. Le processus d'ingénierie des exigences a ensuite été décomposé en étapes cruciales, allant de l'élucidation à la gestion des exigences.

Les défis spécifiques à l'ingénierie des exigences des systèmes embarqués ont été mis en lumière, suivis d'une revue des différentes approches méthodologiques en ingénierie des exigences. Ces méthodes offrent diverses perspectives pour mieux appréhender la complexité de ces systèmes. Les outils dédiés à cette discipline et les travaux de recherche connexes montrent qu'il existe une pénurie d'études, de méthodes, de techniques et d'outils d'ingénierie des exigences spécifiquement développés pour ce domaine.

Enfin, les travaux de recherche sur l'ingénierie des exigences des systèmes auto-adaptatifs sont présentés.

Contributions

CHAPITRE 3 : Métamodèle pour les systèmes embarqués auto-adaptatifs basé sur MAPE-K

« Mieux vaut allumer une bougie que maudire les ténèbres. »

Lao-Tseu

SOMMAIRE:

1. Introduction

Avant d'essayer de mise à jour/proposer un nouveau processus pour l'ingénierie des exigences des systèmes embarqués auto-adaptatifs (SEAAs), il est essentiel de déterminer s'il existe déjà un processus complet pour l'ingénierie des exigences des systèmes embarqués (SEs). Ainsi, notre travail débute par une revue systématique de la littérature (SLR) sur ce sujet. Malheureusement, le rapport de notre revue systématique, prouvé l'absence d'un processus complet pour l'ingénierie des exigences dans le domaine des SEs. Notre contribution dans ce domaine consistera donc à proposer un processus détaillé pour le développement des exigences des SEAAs, basé sur le modèle MAPE-K. Ce processus couvre les étapes de l'ingénierie des exigences, depuis l'élucidation jusqu'à la validation et la vérification.

Cependant, avant d'aller plus loin, une question clé se pose : Quels éléments faut-il prendre en compte pour élaborer un tel processus dans le cadre des SEAAs ?

L'un des principaux problèmes dans le domaine des SEAAs est le manque d'une terminologie commune. Ce problème vient du fait que la recherche dans ce domaine est encore dans une phase exploratoire. Par conséquent, un métamodèle bien défini qui capture les concepts des SEAAs avec les relations qui lient plusieurs de ces concepts entre eux constituerait une avancée significative pour améliorer la qualité des exigences dans ce domaine. Pour cette raison, notre contribution centrale est l'introduction du métamodèle qui permet d'améliorer d'une manière significative le processus d'ingénierie des exigences des SEAAs.

Dans ce chapitre, nous présentons notre première contribution. Nous commençons par expliquer la démarche de réalisation de notre revue systématique de la littérature, couvrant les études de 1980 à 2024. Ensuite, nous décrivons la méthodologie de développement de notre processus. Puis, nous détaillons le processus de développement de notre métamodèle. Enfin, une étude comparative avec d'autres métamodèles met en évidence l'importance de notre proposition.

2. Une revue systématique de la littérature sur l'IE des SEs

La majorité des recherches en ingénierie des exigences des SEs commencent par une revue de littérature quelconque (voir [104], [105], [106], [107]). Cependant, à moins qu'une revue de la littérature ne soit approfondie et juste, elle n'a que peu de valeur scientifique. C'est la principale raison d'entreprendre des revues systématiques. Donc, Afin d'identifier les lacunes dans les recherches actuelles en ingénierie des exigences des SEs, nous avons entrepris une revue systématique de la littérature sur ce sujet. [108].

Une revue systématique de la littérature (souvent appelée revue systématique) est un moyen d'identifier, d'évaluer et d'interpréter toutes les recherches disponibles pertinentes à une question de recherche, un domaine ou un phénomène d'intérêt particulier. Les études individuelles contribuant à une revue systématique sont appelées études primaires ; une revue systématique est une forme d'étude secondaire[109]. Donc, Une revue systématique synthétise les travaux existants d'une manière juste et perçue comme telle.

Pour mener à bien notre revue systématique de la littérature (SLR), nous nous sommes appuyés sur les directives et le modèle de protocole proposés par Kitchenham et Charters[109], qui fournissent des recommandations adaptées aux besoins des chercheurs en génie logiciel pour la réalisation de revues systématiques. Ainsi, la méthodologie de réalisation d'une SLR comprend trois phases : la planification, la réalisation et la rédaction du rapport de la revue. Les étapes peuvent sembler séquentielles, mais il est important de reconnaître que bon nombre d'entre elles impliquent une itération.

2.1. La démarche de réalisation de notre SLR

Le résumé de notre démarche de réalisation de la revue systématique est illustré dans la figure suivante (Figure 3-1).

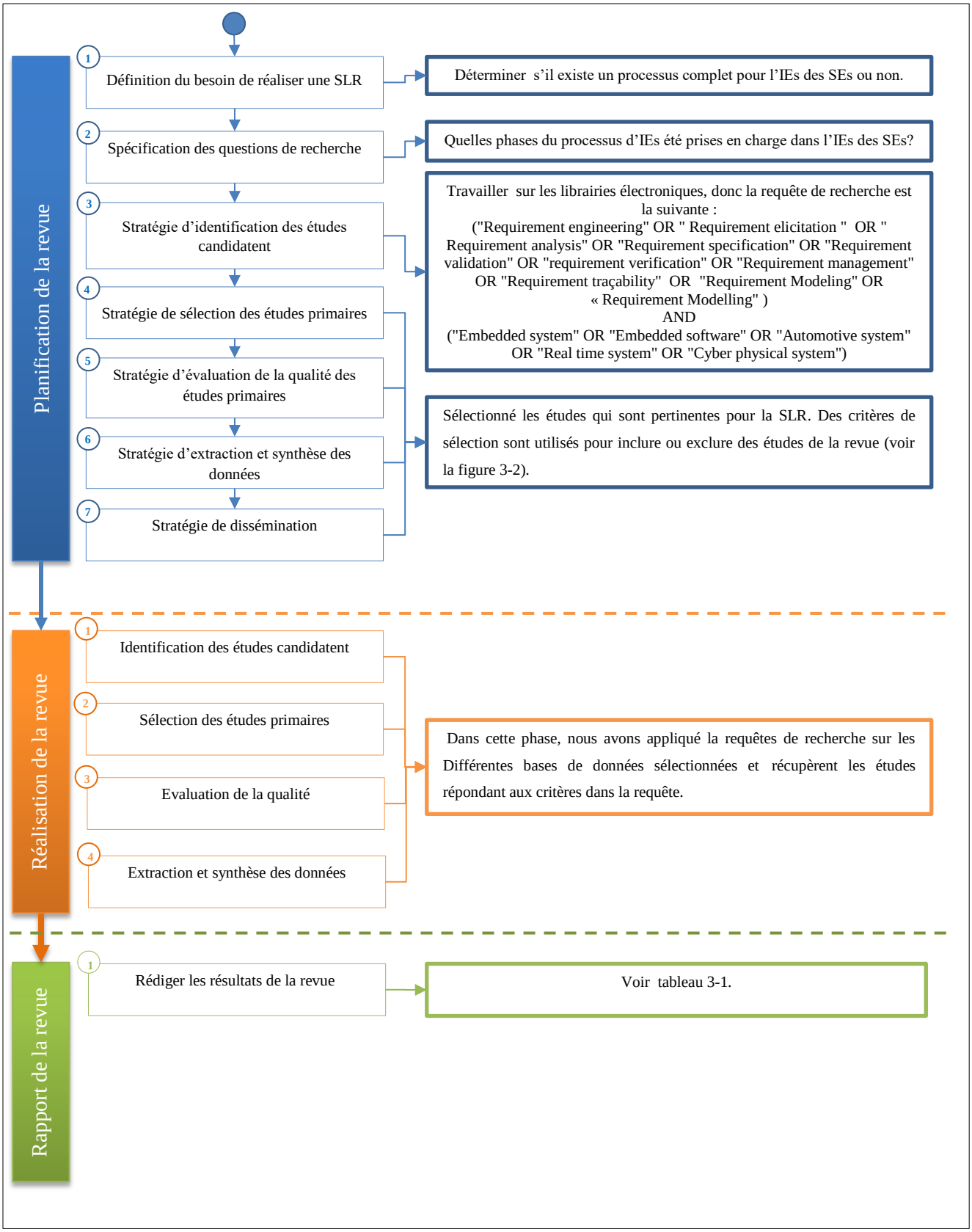


Figure 3-1 : Processus de réalisation de notre SLR.

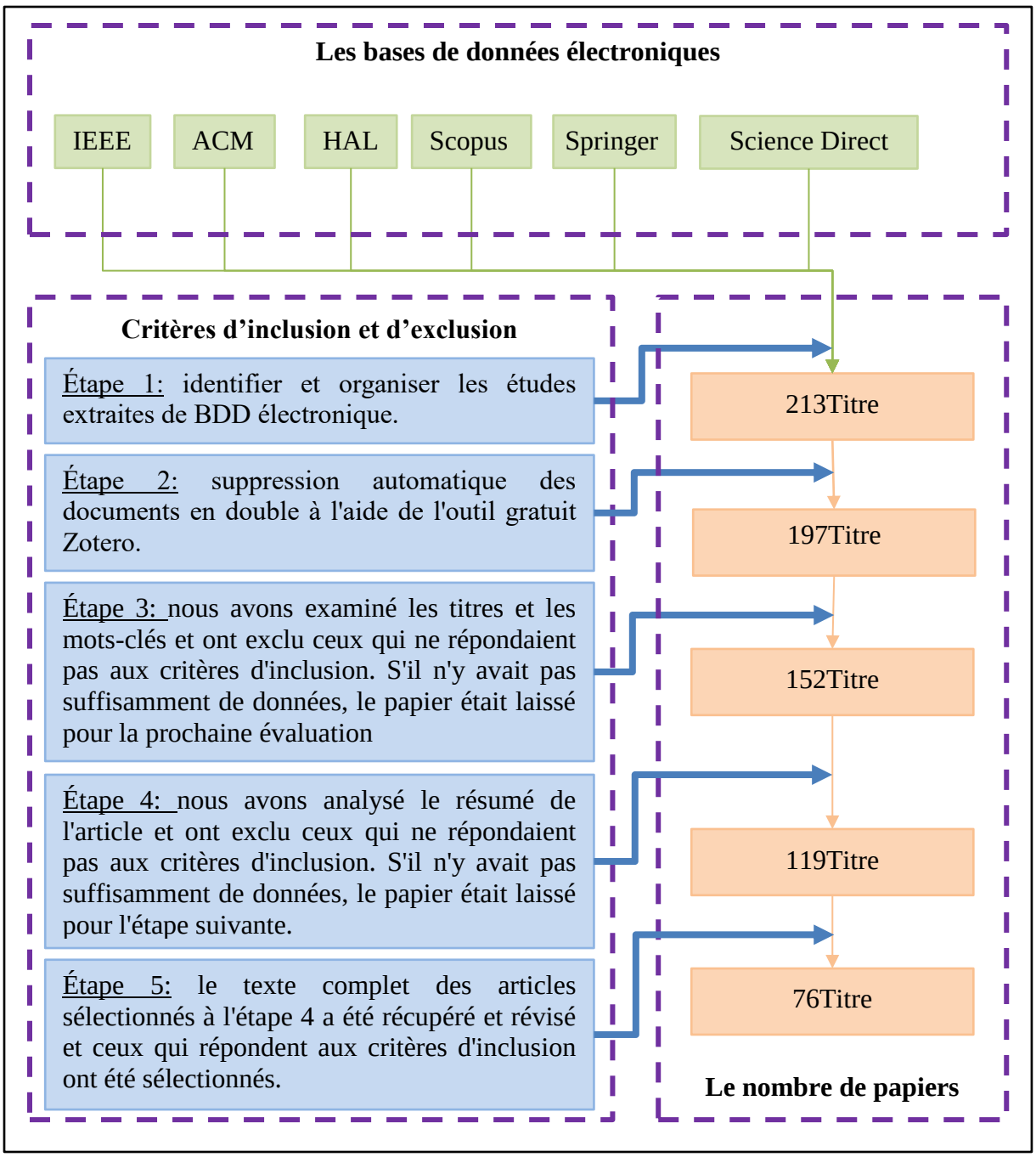


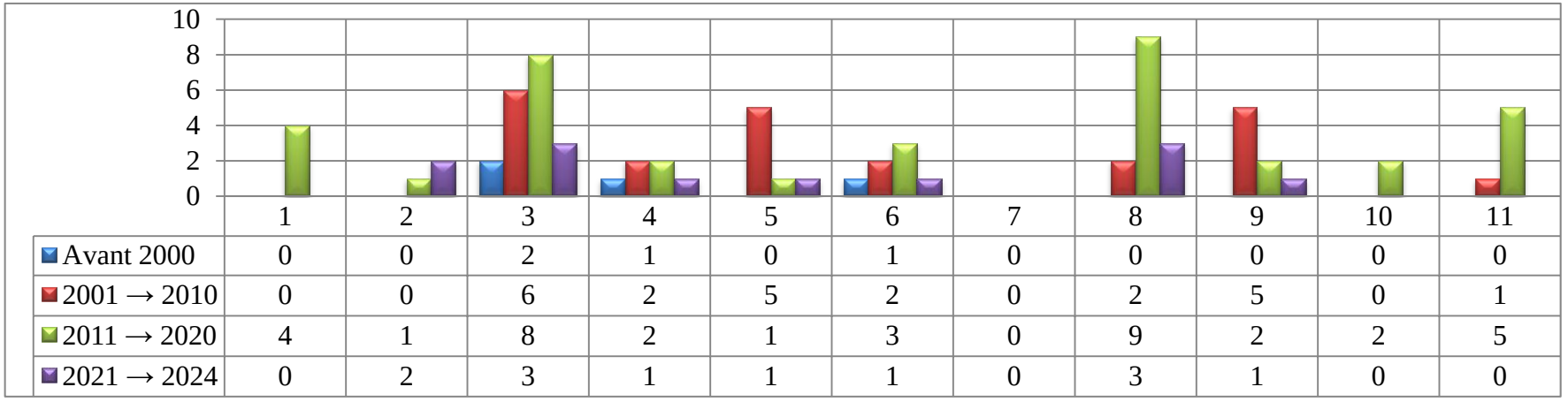
Figure 3-2 : Liste des critères de sélection et d'élimination.

Tableau 3-1 : Le rapport de la revue.

1 (Elucidation)-2 (Analyse), 3(Specification), 4 (Validation & Vérification), 5(Gestion des exigences), 6(Plus d'une phase), 7 (toutes les phases de l'élucidation à la gestion), 8 (Articles d'enquête & revue de la littérature), 9(IE pour Logiciel Embarqué), 10 (Outils pour l'IE) et 11(Exigences de sécurité)

	1	2	3	4	5	6	7	8	9	10	11
1980											
1981			[110]								
1982			[111]								
83-89											
1990						[112]					
1991											
1992											
1993											
1994											
1995											
1996											
1997											
1998											
1999				[113]							
2000											
2001						[114]					
2002				[115]	[116]	[117]			[118]		
2003								[104]			
2004									[119], [120]		
2005									[121], [122]		
2006											
2007			[123], [124]		[125]			[105]			
2008					[126]						[127]
2009			[128], [129]	[91]							
2010			[90], [92]		[130], [131]						
2011					[132]			[106]		[133]	[134]
2012	[93]		[89]			[135]		[136]			[137]

2013	[138]		[139], [140], [141], [142]					[107], [143], [144]			[94]
2014						[145]		[146]	[147]		
2015				[148]				[149]			
2016	[95]		[150]			[151]		[152]			
2017		[153]	[154]					[155]			[156]
2018	[96]									[157]	[158]
2019			[159]						[160]		
2020				[97]							
2021								[161]			
2022		[162]	[163]			[164]		[165]			
2023		[166]	[167] [168]		[169]				[170]		
2024				[171]				[172]			
TOTAL	**04**	**03**	**19**	**06**	**07**	**07**	**00**	**14**	**08**	**02**	**06**



	1	2	3	4	5	6	7	8	9	10	11
Avant 2000	0	0	2	1	0	1	0	0	0	0	0
2001 → 2010	0	0	6	2	5	2	0	2	5	0	1
2011 → 2020	4	1	8	2	1	3	0	9	2	2	5
2021 → 2024	0	2	3	1	1	1	0	3	1	0	0

Figure 3-3 : Nombre d'études réalisées sur l'IE des SEs entre 1980 et 2024.

2.2. Discussion

Au cours des dernières années, un intérêt croissant a été observé pour l'ingénierie des exigences des systèmes embarqués. Cependant, selon le rapport de notre revue systématique qui comprend des études de 1980 à 2024, aboutissant initialement à 76 articles, il n'y a rien qui puisse être utilisé comme référence pour l'IE des SEs. En outre, on note un manque notable d'outils. Ce constat nous a conduits à proposer un processus dédié à l'IE des systèmes embarqués auto-adaptatifs (SEAAs).

Un des principaux défis dans le domaine des SEAAs est le manque d'une terminologie commune. Ce problème vient du fait que la recherche dans ce domaine est encore dans une phase exploratoire. Par conséquent, l'absence de consensus sur les termes clés dans le domaine implique que dans la phase de recherche, nous ne pouvons pas couvrir toutes les études pertinentes sur l'auto-adaptation. En outre, la confusion qui en découle entre les ingénieurs des exigences et les parties prenantes (stakeholders) illustre la nécessité d'un métamodèle bien défini. Celui-ci permettrait de capturer les concepts des SEAAs ainsi que leurs interrelations, et constituerait un atout majeur pour améliorer la qualité des exigences des SEAAs, notamment en s'appuyant sur le modèle MAPE-K.

3. La méthodologie de développement de notre processus

Pour proposer un processus détaillé d'ingénierie des exigences des SEAAs, basé sur le modèle MAPE-K, nous couvrons toutes les étapes de développement des exigences, de l'élucidation jusqu'à la validation et la vérification. La méthodologie de développement que nous avons adoptée est la suivante:

a. La première étape consiste à proposer un métamodèle (MM4SAES) qui intègre les différents concepts utilisés lors de la phase d'ingénierie des exigences d'un SEAA, basé sur le modèle MAPE-K, ainsi que les diverses relations existant entre ces concepts.
b. Ensuite, il s'agit d'enrichir le métamodèle par un ensemble de contraintes OCL, permettant d'exprimer les contraintes complexes qui ne peuvent être facilement représentées graphiquement.
c. Par la suite, proposer un processus détaillé (REP4SAES) structuré en phases principales, sous-phases et activités.
d. Sur la base du métamodèle, générer des plugins Eclipse pour soutenir notre processus. Notons qu'un plug-in, dans le contexte d'Eclipse, correspond à un ou plusieurs fichiers

JAR permettant d'étendre les fonctionnalités de la plateforme en ajoutant de nouvelles perspectives, vues, outils, ou autres capacités.

e. Dans la phase de spécification, nous avons proposé un nouveau profil SysML appelé SysML4SAS ainsi qu'un outil qui le supporte.

f. Enfin, à la fin de notre processus, nous offrons la possibilité de générer automatiquement le référentiel d'exigences, incluant à la fois le document des exigences utilisateurs et le document des exigences système.

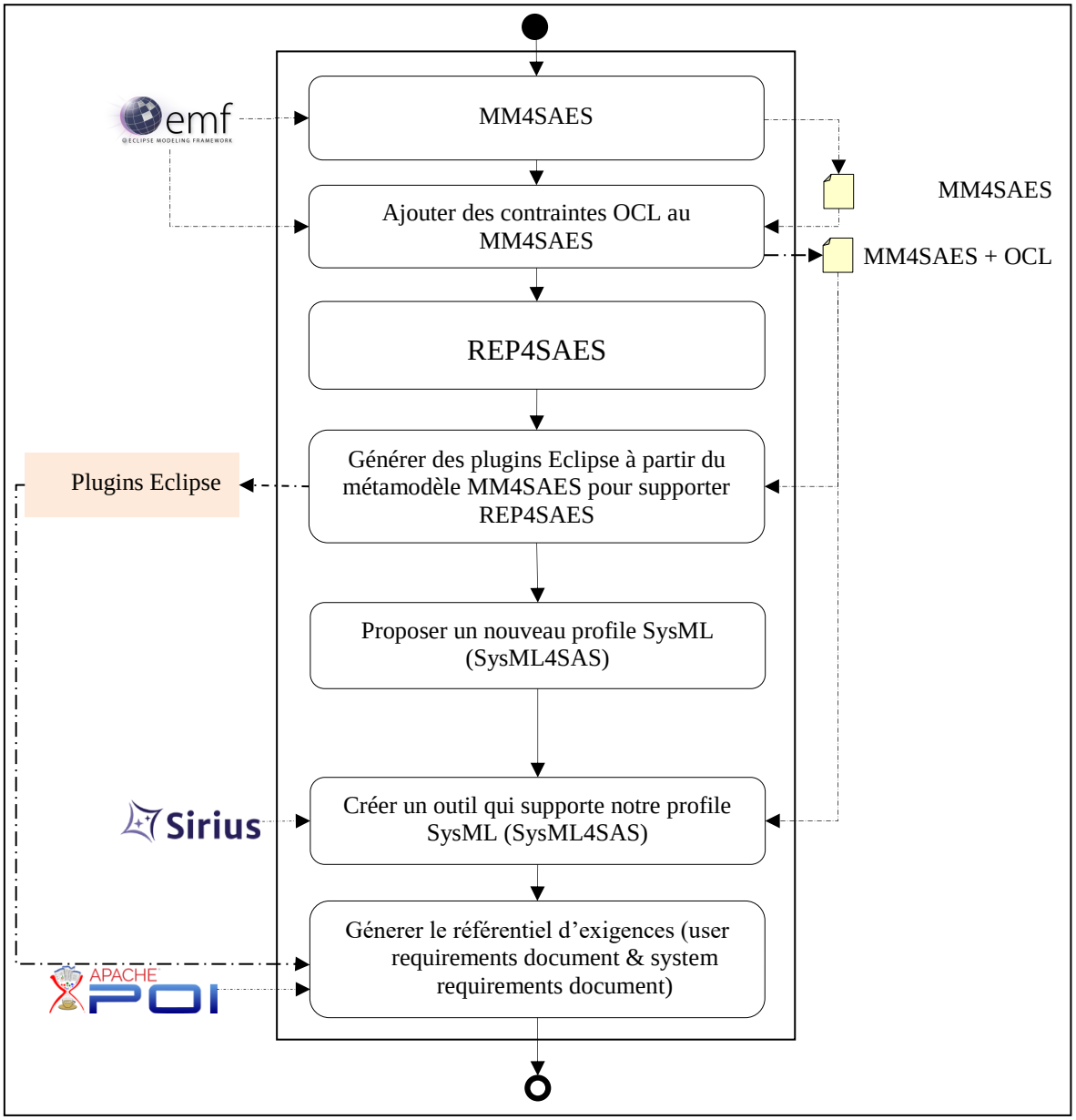


Figure 3-4 : La méthodologie de developement de notre processus.

4. Processus de développement du MM4SAES

Dans le domaine des systèmes complexes tels que les systèmes embarqués (SEs), la métamodélisation joue un rôle crucial. Elle permet de créer des modèles abstraits qui capturent les caractéristiques essentielles d'un système, facilitant ainsi la compréhension et la communication en représentant différents niveaux de détail. De plus, la métamodélisation encourage la réutilisabilité en définissant des structures et schémas applicables à divers contextes, ce qui optimise l'utilisation des ressources. Elle offre également un cadre solide pour l'interprétation et l'analyse des modèles, permettant de déduire des informations essentielles sur les systèmes qu'ils représentent. En garantissant la consistance et la cohérence grâce à des règles de modélisation strictes, la métamodélisation assure que les modèles développés sont fiables et complets, essentiels pour la prise de décision. Par ailleurs, elle favorise l'évolutivité en facilitant l'adaptation des modèles aux changements d'exigences ou de spécifications. Enfin, elle constitue la base pour l'automatisation, en permettant la génération de code, la validation et la transformation des modèles, améliorant ainsi l'efficacité du processus de développement. En somme, la métamodélisation fournit un cadre conceptuel et des outils indispensables pour la création, la manipulation et l'analyse de modèles dans divers domaines, contribuant à une meilleure gestion des systèmes complexes.

Le processus de développement d'un métamodèle peut varier en fonction du contexte spécifique et des objectifs visés. Dans notre thèse, nous avons suivi des étapes similaires à celles adoptées par [173] .

Dans cette section, nous présentons le processus de développement de notre métamodèle (voir la figure 3-5). Chaque étape est brièvement décrite ci-après.

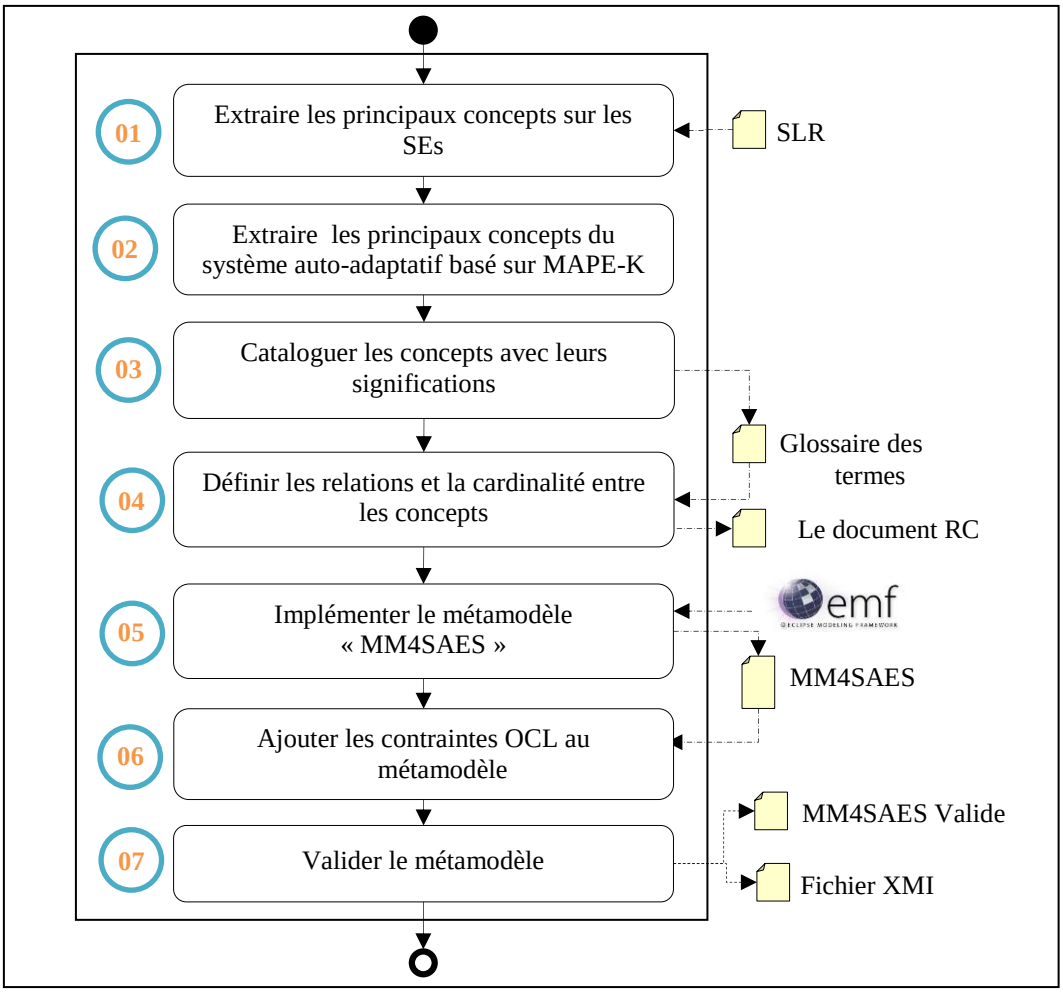


Figure 3-5 : Processus de développement de notre métamodèle.

4.1.Etape N°01, 02&03

Dans la première phase (lecture et extraction des concepts principaux des SE), nous avons utilisé les différents articles issus de notre SLR. Nous les avons analysés en profondeur pour identifier les concepts clés liés au domaine des SE. Ensuite, lors de la deuxième phase, nous avons recherché et structuré les concepts spécifiques aux systèmes auto-adaptatifs basés sur le modèle MAPE-K. Enfin, nous avons organisé les connaissances recueillies dans un glossaire de termes (voir tableau ci-dessous).

Tableau 3-2 : Glossaire des termes.

Nom de la classe	Désignation	Brève description
AbstractFR	Abstract Functional Requirement	Une exigence qui décrit de manière générale et non détaillée une fonctionnalité ou un service que le système doit fournir, sans spécifier les détails techniques ou les conditions précises de mise en œuvre.
AbstractNFR	Abstract Non Functional Requirement	Une exigence qui décrit de manière générale un critère ou une qualité que le système doit posséder, sans entrer dans des détails spécifiques ou des mesures précises.
Actuator	-	Un actionneur est un composant qui convertit des signaux de commande (généralement électriques) en actions physiques comme un mouvement. Cela permet au système d'interagir activement avec son environnement, en modifiant son état ou celui d'autres objets. L'intégration d'un actionneur n'est pas indispensable dans un SE conçu exclusivement pour des tâches de surveillance.
AdaptationGoal	Adaptation Goal	Fait référence à l'objectif à atteindre lors du processus d'adaptation du système
Analysis	Analyze Entity	Est un composant du modèle MAPE-K responsable de l'analyse des données recueillies lors de la phase de surveillance.
ApplicationSoft	Application Software	Il désigne les programmes spécifiques qui permettent au SE d'accomplir des tâches ciblées, en fonction de l'objectif pour lequel il a été conçu, tout en interagissant avec le matériel et en optimisant l'utilisation des ressources limitées.
AwReq	Awareness Requirement	Les exigences de conscience (AwReq) indiquent les situations qui nécessitent une adaptation.

Battery	-	La batterie désigne une source d'alimentation électrique autonome utilisée pour alimenter l'ensemble des composants du SE.
CommunicationInterface	Communication Interface	Une interface de communication matérielle est un point de connexion physique qui permet l'échange de données entre des dispositifs matériels, comme des capteurs, des actionneurs et des microcontrôleurs. Elle inclut des normes et des protocoles spécifiques pour garantir une communication efficace et fiable.
Constraint	-	Ce sont les limites imposées sur la manière dont les objectifs peuvent être atteints. Elles peuvent concerner des aspects techniques, légaux, financiers, etc.
DebuggingSoft	Debugging Software	Le débogage logiciel (debugging software) est un type de logiciel utilisé pendant le développement pour localiser et corriger les erreurs dans le code.
Device_Driver	Dvice Driver	C'est un logiciel écrit pour un matériel particulier.
ElementaryFR	Elementary Functional Requirement	une exigence qui décrit précisément une fonctionnalité ou une opération spécifique que le système doit accomplir.
ElementaryNFR	Elementary Non Functional Requirement	une exigence qui décrit précisément une contrainte ou une qualité spécifique du système qui est clairement définie et mesurable.
Env_Var	environment variable	Désigne des facteurs externes au système qui peuvent influencer son fonctionnement et son comportement. Ces variables peuvent inclure des conditions telles que la température, l'humidité, la charge du réseau, ou d'autres paramètres externes. Les SEAAs s'adaptent souvent en réponse à ces variables d'environnement, car elles sont susceptibles de changer au fil du temps.

ErrorHandlingSoft	Error Handling Software	Désigne généralement des composants ou des modules logiciels dédiés à la gestion des erreurs pendant l'exécution d'un programme ou d'un système. ils peuvent détecter, signaler, et gérer des erreurs afin d'assurer la stabilité, la sécurité et la continuité de l'exécution du système, notamment dans les SEs ou critiques.
EvoReq	Evolution Requirement	Les exigences d'évolution prescrivent ce qu'il faut faire dans les situations qui nécessitent une adaptation.
Execution	Execute Entity	Est un composant du modèle MAPE-K, chargé de mettre en œuvre le plan élaboré lors de la phase d'analyse, en utilisant un effecteur pour effectuer des ajustements spécifiques.
ExternalCommunInt	External Communication Interface	L'interface de communication externe est un mécanisme qui permet à un SE de communiquer avec des dispositifs externes, tels que des ordinateurs, des réseaux ou d'autres systèmes. Elle facilite l'échange de données et la synchronisation entre le système embarqué et son environnement, utilisant souvent des protocoles standard tels qu'USB, Ethernet ou Bluetooth
FeedBackLoop	FeedBack Loop	Est un mécanisme structuré permettant à un système d'auto-surveiller son fonctionnement et de s'adapter en conséquence. Elle consiste en un flux cyclique d'informations où les données collectées sur l'état actuel du système et son environnement sont analysées, comparées à des objectifs ou des conditions souhaitées, puis utilisées pour prendre des décisions ou effectuer des ajustements.
FR	Functional Requirement	Une exigence fonctionnelle décrit toute exigence qui spécifie ce que le système doit faire.

Goal	-	Un objectif (Goal) désigne une cible ou un résultat souhaité que le système doit atteindre. Il représente une fonction, un service ou un comportement que le système embarqué doit réaliser, souvent en réponse aux besoins des parties prenantes. Les objectifs guident la définition des exigences du système et orientent les choix de conception et d'implémentation.
HardAdapter	Hardware Adapter	L'adaptateur matériel dans un SE permettant la connectivité et la communication entre les différentes composantes matérielles du système, ainsi qu'avec des périphériques externes, en adaptant les formats, les protocoles et les signaux selon les besoins du système.
HardDevice	Hardware Device	Il existe différents composants matériels comme le processeur, la mémoire… etc.
HardRequirement	Hardware Requirement	Les exigences matérielles dépendent du type et de la complexité du projet. Par exemple, déterminez le capteur (capteur d'humidité, capteur de température… etc.), clarifiez l'interaction de l'utilisateur (graphique LCD, buzzer… etc.) et définissez les ports de communication externes (ports USB).
InputDevice	Input Device	Les dispositifs d'entrée dans les SEs facilitent l'interaction entre le système et son environnement en permettant aux utilisateurs ou à d'autres systèmes de fournir des données ou des commandes au SE, jouant ainsi un rôle clé dans la collecte d'informations ou l'initiation d'actions en fonction des besoins du système. Exemples de dispositifs d'entrée : capteurs, écrans tactiles, commandes vocales, interfaces de communication, etc.

Knowledge	Knowledge Entity	est un composant du modèle MAPE-K qui comprend un ensemble d'informations, telles que des données contextuelles, des modèles, des règles et des historiques de performance. Cette connaissance est utilisée pour améliorer la précision et l'efficacité des processus de surveillance, d'analyse, de planification et d'exécution, en permettant au système d'apprendre de ses expériences passées et d'ajuster ses comportements en fonction des évolutions observées.
ManagedSystem	Managed System	Le système géré c'est le système qui fait l'objet d'une adaptation. Il comprend à la fois le logiciel embarqué et le matériel, qui ensemble réalisent les fonctions nécessaires pour répondre aux besoins des utilisateurs du système
ManagingSystem	Managing System	Est l'entité chargée de coordonner les adaptations du système géré. Il intègre à la fois la boucle de rétroaction et les objectifs d'adaptation, permettant au système de s'ajuster de façon autonome en fonction des données recueillies sur son état interne et son environnement externe.
Memory	-	La mémoire, comme la RAM, la ROM et la mémoire Flash, est essentielle pour stocker des informations critiques dans le système embarqué, qu'il s'agisse de données temporaires ou permanentes.
MetaRequirement	Meta Requirement	Les méta-exigences (Meta requirements) représentent les préoccupations du système de gestion dans le modèle conceptuel des systèmes auto-adaptatifs.

Monitor	Monitor Entity	Est un composant du modèle MAPE-K, responsable de la collecte en temps réel des données relatives à l'état du système et à son environnement. Il surveille les différentes variables et paramètres définis, permettant de détecter tout changement, toute anomalie ou tout écart par rapport aux attentes.
Need	-	un besoin (need) est une condition ou une capacité essentielle que le système doit satisfaire pour répondre aux attentes des parties prenantes
NFR	Non Functional Requirement	Une exigence non fonctionnelle définit les critères de qualité ou les contraintes d'un système, sans concerner directement les fonctionnalités qu'il doit offrir.
OnBoardCommunInt	On Board Communication Interface	Une interface de communication embarquée (on-board communication interface) est un système qui permet l'échange de données entre les différents composants d'un SE. Elle assure la connectivité interne entre les processeurs, les capteurs, les actionneurs et d'autres dispositifs, en utilisant des protocoles de communication spécifiques pour garantir une communication efficace et fiable au sein du système.
OutputDevice	Output Device	Un dispositif de sortie (output device) dans les SEs est un composant qui reçoit des données traitées par le système et les transforme en un format compréhensible par l'utilisateur ou d'autres systèmes. Permettant ainsi au système d'interagir avec son environnement et de fournir des résultats ou des informations.
Plan	Plan Entity	Est un composant du modèle MAPE-K chargé de formuler des actions ou des stratégies à suivre en réponse aux résultats de l'analyse.

ProcessingUnit	Processing Unit	Une unité de traitement (processing unit) est le composant central d'un SE, chargé d'exécuter des instructions et de traiter des données
Refinement / RefinementNFR	Refinement / Refinement Non Functional Requirement	Le raffinement des exigences consiste à décomposer une exigence complexe en sous-exigences plus spécifiques ou détaillées, en utilisant les opérateurs AND ou OR pour exprimer les relations entre ces sous-exigences.
Requirement	-	Les exigences décrivent ce qui doit être accompli par le système et définissent comment ses fonctions doivent être réalisées. Les parties prenantes peuvent justifier ces exigences en expliquant pourquoi elles doivent être satisfaites dans le cadre du futur système.
RTOS	Real Time Operating System	Un RTOS est un système d'exploitation conçu pour gérer les exigences de temps réel dans les systèmes embarqués. Contrairement aux systèmes d'exploitation traditionnels, un RTOS garantit que les tâches critiques sont exécutées à des moments spécifiques et dans des délais stricts, ce qui est essentiel pour les applications telles que le contrôle industriel, les systèmes de contrôle de vol, et d'autres applications où la réactivité et la prédictibilité sont cruciales.
SAES	Self-Adaptive Embedded System	Revenir à chapitre 1 (système embarqué auto adaptatif)
Sensor	-	Un capteur (sensor) est un dispositif qui mesure une grandeur physique et la convertit en un signal électrique ou numérique exploitable, parfois avec un stockage temporaire des données mesurées.

SoftRequirement	Software Requirement	Les exigences logicielles sont des exigences détaillées pour les logiciels dérivées de la configuration du système requise.
Software	-	Est un programme conçu pour fonctionner sur un matériel spécifique afin de remplir une tâche précise, souvent avec des contraintes de ressources et de temps réel.
SoftwareInterface	Software Interface	L'interface logicielle est le logiciel chargé de connecter un ou plusieurs logiciels.
Stakeholder	-	Dans le domaine des SEs, un stakeholder (ou partie prenante) fait référence à toute personne ou entité qui est directement ou indirectement affectée par le développement, le déploiement ou l'utilisation d'un SE. Cela peut inclure des développeurs de logiciels, des ingénieurs matériels, des utilisateurs finaux, des clients, des gestionnaires de projet, des investisseurs, et même des régulateurs ou des organismes de normalisation. Chaque partie prenante a des intérêts ou des préoccupations spécifiques concernant le système.
Sub_Goal	Sub Goal	Un sous-objectif (sub-goal) est un objectif intermédiaire dérivé d'un objectif principal, qui aide à décomposer un problème complexe en étapes plus petites et plus gérables. Il contribue à atteindre l'objectif global en se concentrant sur des aspects spécifiques ou des tâches partielles nécessaires pour accomplir la finalité du système.

4.2.Etape N°04 : Définir les relations et la cardinalité entre les concepts

Outre la définition des concepts, le métamodèle décrit également les relations entre ces concepts. De telles relations peuvent être utilisées pour améliorer l'élucidation des exigences. Par exemple, l'utilisation de relations de spécialisation/généralisation, de relations de

composition et de relation d'association entre concepts réduit évidemment l'ambiguïté et augmente l'exactitude d'une spécification d'exigences.

Donc, dans cette étape, nous avons effectué l'identification et la description des concepts et des relations dans un document de cardinalité-relation (RC[1]). Il se compose du nom des classes, des relations et de la cardinalité entre elles. Le tableau suivant montre les résultats de cette étape. Par exemple, voir dans la figure ci-dessous qui représente une partie de notre métamodèle, la classe *DevideDriver* a une relation appelée *Control* avec la classe *HardDevice*. La classe *HardDevice* a une relation de composition appelée *Defines* avec la classe *HardRequirement*, et la classe *HardAdapter* a une relation appelée *Connects* avec la classe *HardDevice*.

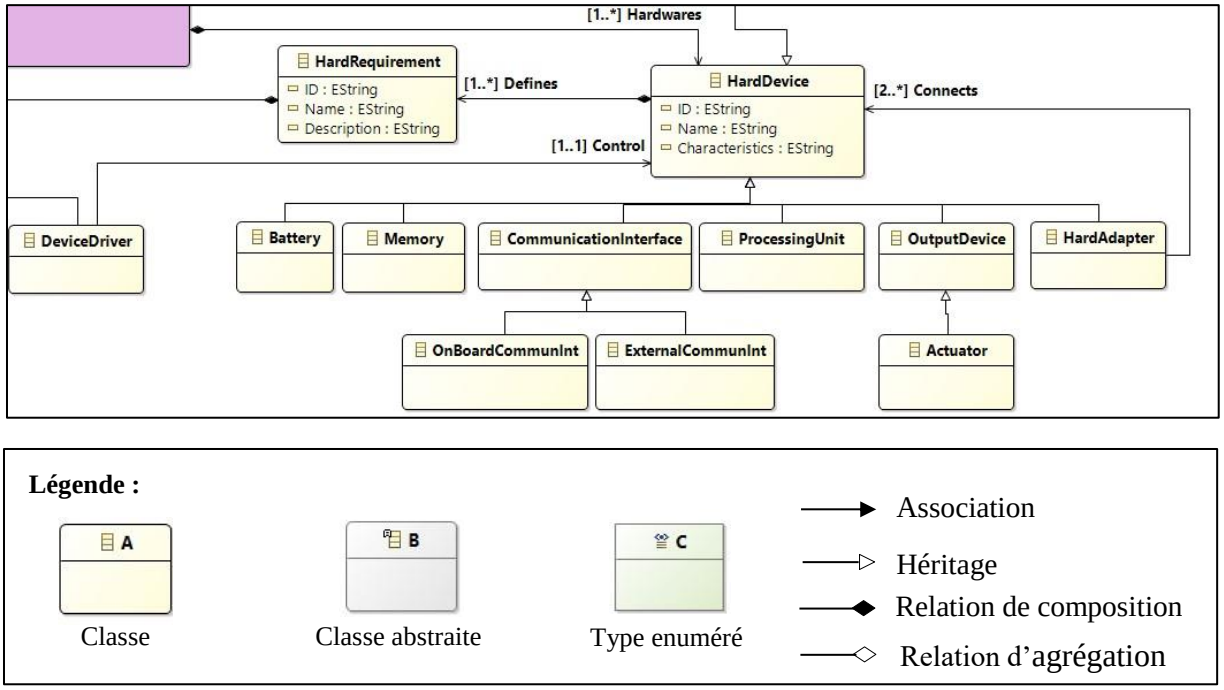


Figure 3-6 : Une partie de notre métamodèle.

Nous avons utilisé le document de cardinalité-relation pour implémenter le métamodèle. Chaque ligne du document a été mappée à une classe avec ses relations.

Tableau 3-3 : Relation –Cardinalité.

Nom de classe	Relation	Type relation	Car	Nom de classe
AbstractFR	refinement	Composition	1…1	Refinement
AbstractNFR	refinementnfr	Composition	1…1	RefinementNFR

[1] Relation-Cardinality.

Analysis	fr	Composition	1…*	ElementaryFR
AwReq	awreq	Composition	1…*	AwReq
CommunicationInterface	-	Héritage	-	OnBoardCommunInt
CommunicationInterface	-	Héritage	-	ExternalCommunInt
Constraint	stakeholder	Association	1…*	Stakeholder
Constraint	Requirements	Composition	1…*	Requirement
DeviceDriver	Control	Association	1…1	HardDevice
ElementaryFR	Refine	Association	0…1	ElementaryFR
ElementaryFR	metarequirement	Composition	0…*	MetaRequirement
ElementaryFR	MON	Association	0…*	Monitor
ElementaryFR	VAR	Association	0…*	Env_Var
ElementaryNFR	Refine	Association	0…1	ElementaryNFR
ElementaryNFR	metarequirement	Composition	0…*	MetaRequirement
ElementaryNFR	MON	Association	0…*	Monitor
ElementaryNFR	VAR	Association	0…*	Env_Var
Execution	fr	Composition	1…*	ElementaryFR
FeedBackLoop	execution	Composition	1…*	Execution
FeedBackLoop	plan	Composition	1…*	Plan
FeedBackLoop	knowledge	Composition	1…*	Knowledge
FeedBackLoop	analysis	Composition	1…*	Analysis
FeedBackLoop	monitor	Composition	1…*	Monitor
FR	contributesTo	Association	0…1	NFR
FR	-	Héritage	-	ElementaryFR
FR	-	Héritage	-	AbstractFR
Goal	Sub_Goals	Composition	1…*	Sub_Goal
HardAdapter	Connects	Association	2…*	HardDevice
HardDevice	Defines	Composition	1…*	HardRequirement
HardDevice	-	Héritage	-	Battery
HardDevice	-	Héritage	-	Memory
HardDevice	-	Héritage	-	CommunicationInterface
HardDevice	-	Héritage	-	ProcessingUnit

HardDevice	-	Héritage	-	OutputDevice
HardDevice	-	Héritage	-	HardAdapter
HardDevice	-	Héritage	-	InputDevice
HardRequirement	Requirements	Composition	1…*	Requirement
InputDevice	-	Héritage	-	Sensor
Knowledge	fr	Composition	1…*	ElementaryFR
ManagingSystem	feedbackloop	Composition	1…1	FeedBackLoop
ManagingSystem	adaptationgoal	Composition	1…1	AdaptationGoal
MetaRequirement	AwReqs	Composition	1…*	AwReq
MetaRequirement	EvoReqs	Composition	1…*	EvoReq
Monitor	REL	Association	1…*	Env_Var
Monitor	Receive_Data	Association	1…*	Sensor
Monitor	fr	Composition	1…*	ElementaryFR
Need	stakeholder	Association	1…*	Stakeholder
Need	Goals	Composition	1…*	Goal
NFR	Generate	Association	0…*	FR
NFR	-	Héritage	-	ElementaryNFR
NFR	-	Héritage	-	AbstractNFR
OutputDevice	-	Héritage	-	Actuator
Plan	fr	Composition	1…*	ElementaryFR
Refinement	fr	Composition	2…*	FR
RefinementNFR	nfr	Composition	2…*	NFR
Requirement	-	Héritage	-	FR
Requirement	-	Héritage	-	NFR
Requirement	PositiveDEP	Association	0…*	Requirement
Requirement	NegativeDEP	Association	0…*	Requirement
Requirement	Operationalized ByManagingSys	Association	0…1	ManagingSystem
Requirement	Operationalized ByManagedSys	Association	0…1	ManagedSystem
SAES	Constraints	Composition	1…*	Constraint
SAES	Stakeholders	Composition	1…*	Stakeholder

SAES	Needs	Composition	1…*	Need
SAES	Softwares	Composition	0…*	Software
SAES	Hardwares	Composition	0…*	HardDevice
SAES	managedsystem	Composition	1…1	ManagedSystem
SAES	managingsystem	Composition	0…1	ManagingSystem
SAES	Env_vars	Composition	0…*	Env_Var
SoftRequirement	Requirements	Composition	1…*	Requirement
Software	Defines	Composition	1…*	SoftRequirement
Software	-	Héritage	-	SoftwareInterface
Software	-	Héritage	-	ErrorHandlingSoft
Software	-	Héritage	-	DebuggingSoft
Software	-	Héritage	-	ApplicationSoft
Software	-	Héritage	-	RTOS
Software	-	Héritage	-	DeviceDriver
SoftwareInterface	Connects	Association	2…*	Software
Sub_Goal	Requirements	Composition	1…*	Requirement

4.3.Etape N°05 : Implémenter le métamodèle « MM4SAES »

Eclipse, largement utilisé pour la métamodélisation, offre une intégration étroite avec EMF[1], qui est essentiel pour la création de métamodèles. EMF permet de définir et de gérer des modèles complexes grâce à sa structure modulaire, de générer automatiquement du code Java à partir des métamodèle, et propose des éditeurs graphiques pour manipuler les modèles via une interface utilisateur intuitive. Par conséquent, pour implémenter notre métamodèle, nous avons utilisé le document de cardinalité-relation et l'outil Eclipse IDE[2], en intégrant la bibliothèque Ecore d'EMF. Chaque ligne du document a été mappée à une classe et à ses relations correspondantes. La figure 3-7 présente notre métamodèle.

4.4.Etape N°06 : Ajouter les contraintes OCL au métamodèle

L'OCL[3] joue un rôle fondamental dans les métamodèles, en particulier dans les environnements de modélisation basés sur UML[1]. Il permet de spécifier des contraintes non

[1] Eclipse Modeling Framework.
[2] Integrated Development Environment.
[3] Object Constraint Language.

représentables graphiquement, telles que des règles métier spécifiques ou des contraintes d'intégrité, garantissant ainsi la validité des modèles. Grâce à l'OCL, la validation des modèles devient possible en vérifiant que les instances respectent les contraintes définies, ce qui facilite la détection d'erreurs avant l'implémentation. En outre, l'OCL permet de naviguer dans les modèles et de sélectionner des éléments précis selon des critères spécifiques, tout en facilitant la définition de propriétés dérivées calculées à partir d'autres propriétés du modèle, comme la somme des prix dans une commande. Enfin, l'OCL joue un rôle clé dans l'automatisation et la génération de code, en assurant que les contraintes du modèle sont respectées par le code généré. En résumé, l'OCL enrichit les métamodèles en apportant un cadre formel pour la définition de règles, la validation des modèles, et l'automatisation des tâches liées à la modélisation.

Dans notre thèse, nous avons ajouté les contraintes OCL directement dans notre métamodèle Ecore à l'aide de l'éditeur de modèles dans Eclipse. Chaque contrainte OCL est expliquée et détaillée dans le chapitre suivant.

[1] Unified Modeling Language.

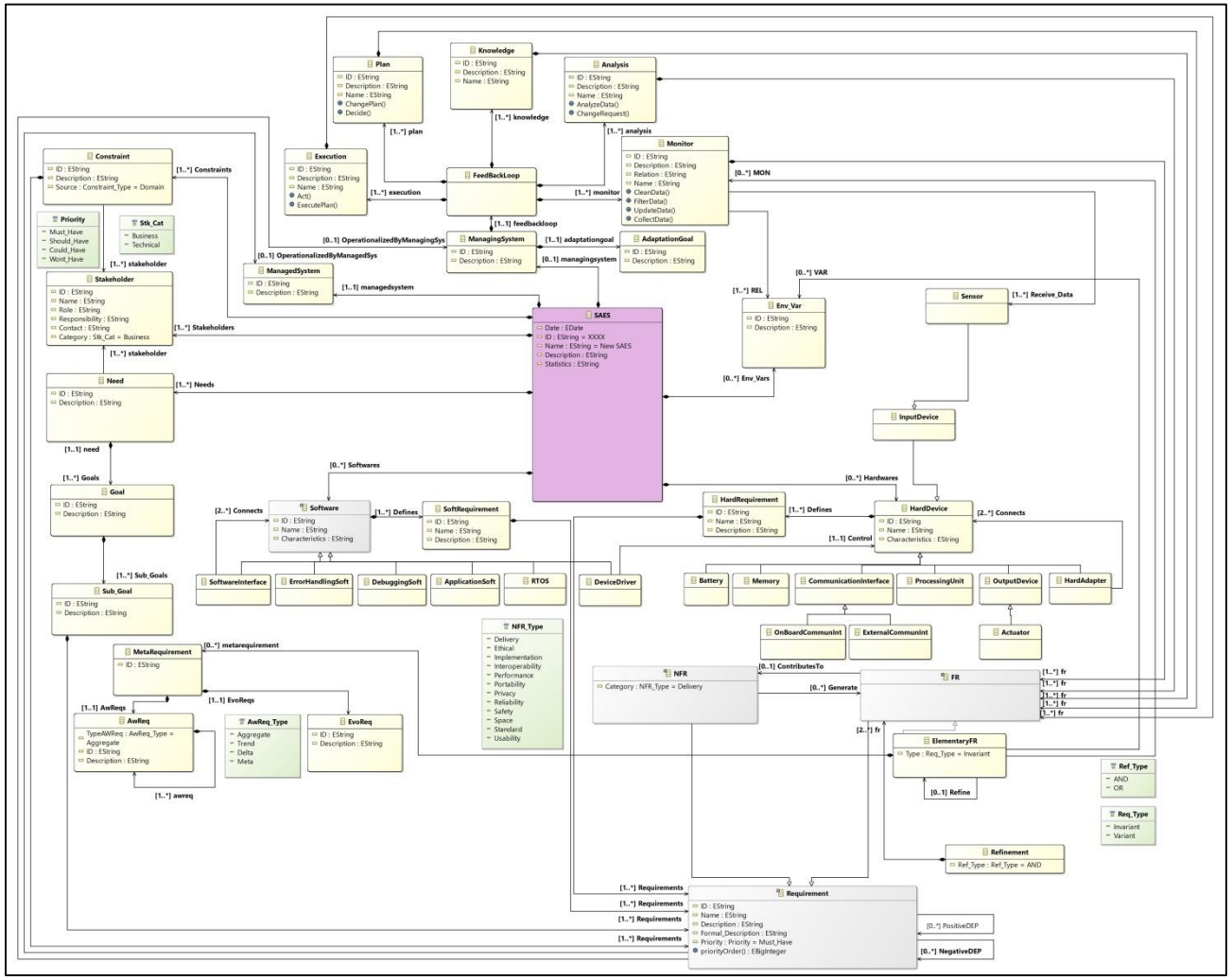


Figure 3-7 : Le métamodèle MM4SAES.

REMARQUE : Pour améliorer la lisibilité de l'image, il suffit de la copier et de la coller dans Paint.

4.5.Etape N°07 : Valider le métamodèle

Au cours de cette étape, nous avons réalisé à la fois une validation structurelle et une validation personnalisée de notre métamodèle. La validation structurelle nous a permis de vérifier que notre métamodèle respecte les règles fondamentales définies par le métamodèle Ecore, telles que les types de données, les relations, les multiplicités et la hiérarchie des classes. En parallèle, nous avons mené une validation personnalisée en utilisant des contraintes OCL, ce qui nous a permis d'ajouter des vérifications supplémentaires plus complexes, adaptées à des exigences spécifiques non couvertes par les règles structurelles standard. Ces deux approches complémentaires nous ont assuré que notre métamodèle est à la fois conforme aux spécifications d'Ecore et capable de répondre à des contraintes spécifiques du domaine de l'IE des SEAAs, basé sur le modèle MAPE-K.

La figure suivante démontre la validation réussie de notre métamodèle.

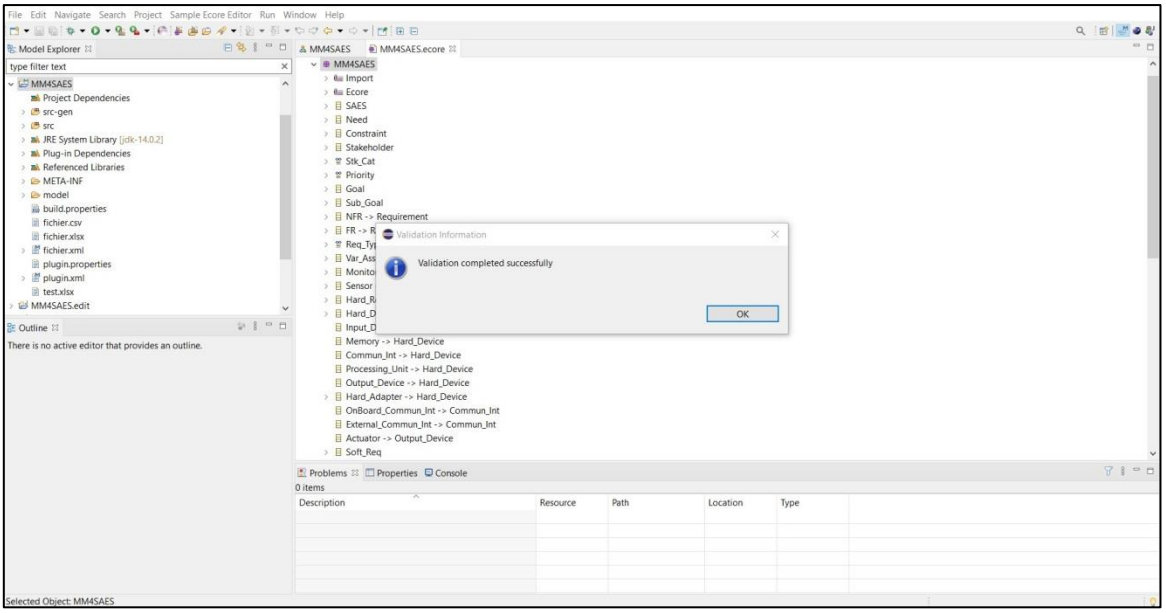


Figure 3-8 : Validation du métamodèle.

Après avoir validé notre métamodèle, nous avons généré son fichier XMI[1], un standard de l'industrie basé sur XML[2] et développé par l'OMG[3]. Ce fichier a été conçu pour permettre l'utilisation des modèles créés avec notre métamodèle dans divers environnements compatibles XMI, favorisant ainsi l'intégration et l'interopérabilité entre plusieurs plateformes. Cette approche garantit non seulement une plus grande flexibilité dans le

[1] XML Metamodel Interchange.
[2] eXtensible Markup Language.
[3] Object Management Group.

développement, mais aussi une compatibilité renforcée avec d'autres outils et technologies. Le fichier XMI complet de notre métamodèle est disponible en annexe.

5. Comparaison avec les autres métamodèle

Dans cette section, nous allons élaborer une étude comparative entre les différents métamodèle qui existent dans le but de bien énumérer les lacunes et limites de ces travaux d'une part et, de mettre en valeur notre contribution.

Donc, afin de comparer de manière systématique et rigoureuse les différents métamodèles, notre comparaison proposée a été effectuée selon plusieurs critères à savoir :

a. **Étapes de l'IE couvertes par le métamodèle** *:* Quelles étapes de l'ingénierie des exigences (IE) sont couvertes par le métamodèle ?
b. **Capacité de modélisation des SEs** : Le métamodèle permet-il de capturer les aspects pertinents des SEs ?
c. **Capacité de modélisation des SAAs** : Le métamodèle permet-il de capturer les aspects pertinents des SAAs?
d. **Capacité de modélisation des SEAAs** : Le métamodèle permet-il de capturer les aspects pertinents des SEAAs, basé sur le modèle MAPE-K?
e. **Support des contraintes et des objectifs non fonctionnels** : Le métamodèle permet-il de spécifier et de gérer des exigences non fonctionnelles ?
f. **Facilité de compréhension et d'utilisation** : le métamodèle est facile à comprendre et à utiliser par les ingénieurs ? Notons qu'il est important d'incorporer les termes d'exigences issus des normes afin de créer une base de connaissances commune et de réduire les éventuels problèmes de communication ainsi que les écarts entre les différentes phases.
g. **Documentation et Support** : Le métamodèle est-il bien documentés?
h. **Exemples et cas d'utilisation** : Existe-t-il des exemples pratiques et des cas d'utilisation démontrant l'efficacité du métamodèle ?
i. **Disponibilité des outils:** Existe-t-il des outils ou des méthodes associées pour valider et vérifier les modèles créés avec le métamodèle ?
j. **Modularité** : Le métamodèle peut-il être facilement étendu ou modifié pour inclure de nouvelles exigences ou aspects non prévus initialement ?

Tableau 3-4 : Comparaison du MM4SAES avec d'autres métamodèles.

Le critère	**Métamodèles étudiés**				
	[173]	[132]	[174]	[175]	**MM4SAES**
Étapes de l'IE couvertes par le métamodèle	**Elucidation des exigences**	**Gestion des exigences**	**Aucune étape**	**Spécification des exigences**	**les phases de développement d'exigences**
Capacité de modélisation des SEs	✓	✓	✕	✓	✓
Capacité de modélisation des SAAs	✕	✕	✓	✕	✓
Capacité de modélisation des SEAAs	✕	✕	✕	✕	✓
Support des contraintes et des objectifs non fonctionnels	✕	✓	✕	✓	✓
Facilité de compréhension et d'utilisation	✓	✓	✓	✕	✓
Documentation et Support	✓	✓	✓	✓	✓
Exemples et cas d'utilisation	✕	✓	✓	✓	✓
Disponibilité des outils	✕	✓	✕	✓	✓
Modularité	✓	✓	✓	✓	✓

D'après le tableau ci-dessus, notre métamodèle se distingue par sa capacité à modéliser les exigences des systèmes embarqués auto-adaptatifs, en intégrant à la fois les exigences fonctionnelles et non fonctionnelles. Il propose également un support complet, tant en matière de documentation que d'outils, tout en offrant une grande modularité et une facilité d'utilisation.

La disponibilité des outils est discutée en détail dans le chapitre suivant.

6. Conclusion

Dans ce chapitre, nous présentons un métamodèle pour les systèmes embarqués auto-adaptatifs (SEAAs), basé sur le modèle MAPE-K, que nous avons nommé MM4SAES. Ce métamodèle vise à soutenir le développement des exigences dans le cadre du processus d'ingénierie des exigences. Il est enrichi d'un ensemble de contraintes OCL et a été développé à l'aide d'Ecore EMF. Il se compose de 48 entités ainsi que des relations entre elles. Par conséquent, les concepts définis dans ce métamodèle seront utilisés dans le chapitre suivant pour élaborer le processus d'ingénierie des exigences des systèmes embarqués auto-adaptatifs, également basé sur le modèle MAPE-K (REP4SAES[1]).

[1] Requirement Engineering Process for Self Adaptive Embedded System.

CHAPITRE4: Un processus d'ingénierie des exigences pour les SEAAs – REP4SAES

« Un voyage de mille lieues commence toujours par un premier pas. »

Lao-Tseu

SOMMAIRE:

1. Introduction

En l'absence d'un processus bien structuré pour l'ingénierie des exigences, les SEAAs sont souvent développés de façon ad hoc. Cela signifie qu'ils risquent d'être inefficaces, mal adaptés ou incapables de répondre correctement aux besoins pour lesquels ils ont été conçus. Donc, pour concevoir des SEAAs efficaces basés sur le modèle MAPE-K, il est essentiel de bien comprendre, bien modéliser et bien spécifier les besoins du nouveau système. Sans cette compréhension, les efforts de conception risquent d'être désordonnés et peu efficace.

Dans cette thèse, notre processus prend en charge trois approches importantes pour l'ingénierie des exigences des systèmes auto-adaptatifs.

1. **Atténuation des incertitudes** : Cette approche repose sur l'assouplissement des exigences spécifiques aux systèmes auto-adaptatifs. Pour cela, nous adaptons le langage RELAX afin de répondre aux besoins et spécificités des systèmes embarqués, aboutissant à une nouvelle version baptisée RELAX_ES.
2. **Représentation des objectifs d'adaptation** : Cette approche modélise les exigences liées au comportement adaptatif « adaptation goals » sous forme de méta-exigences, c'est-à-dire des exigences qui s'appliquent aux exigences régulières du système.
3. **Exigences des boucles de rétroaction** : La dernière approche se focalise sur les exigences associées au comportement fonctionnel des boucles de rétroaction.

La combinaison de ces trois approches s'inspire des travaux de D.Weyns [55], qui a proposé leur intégration. Ainsi, pour soutenir ce processus, nous proposons un nouveau profil SysML adapté à la spécification des exigences des SEAAs. Ce profil enrichit le diagramme des exigences et le diagramme de bloc en introduisant de nouveaux stéréotypes et relations, conçus pour capturer les particularités des systèmes auto-adaptatifs. En s'appuyant sur ce profil, nous formalisons à la fois les exigences classiques et les exigences spécifiques au comportement adaptatif des boucles de rétroaction.

En outre, afin de faciliter l'adoption et l'utilisation de notre processus, nous avons également développé trois outils : un éditeur graphique qui supporte notre profil SysML, permettant de modéliser et de visualiser facilement les exigences et leur évolution dans un environnement visuel intégré. Un outil de validation permettant d'assurer la traçabilité entre les besoins, les contraintes et les exigences, ainsi que de vérifier que la liste des exigences et leurs relations

sont complètes et conformes aux contraintes OCL du métamodèle (En réalité, ce n'est pas un outil, mais un ensemble de plugins Eclipse générés à partir de notre métamodèle). Enfin, un outil de génération de documents qui produit les documents d'exigences utilisateur et système, facilitant ainsi la communication des besoins et des attentes au sein des parties prenante.

Par conséquent, ce chapitre présente une vision globale et structurée de notre processus REP4SAES. Il détaille les étapes clés, les données produites à chaque activité, ainsi que le profil SysML et les outils développés. La figure suivante illustre ce processus, facilitant la compréhension des sorties et des interconnexions entre les étapes. Chaque étape est ensuite décrite en détail, en précisant son rôle, ses objectifs et sa contribution au processus global.

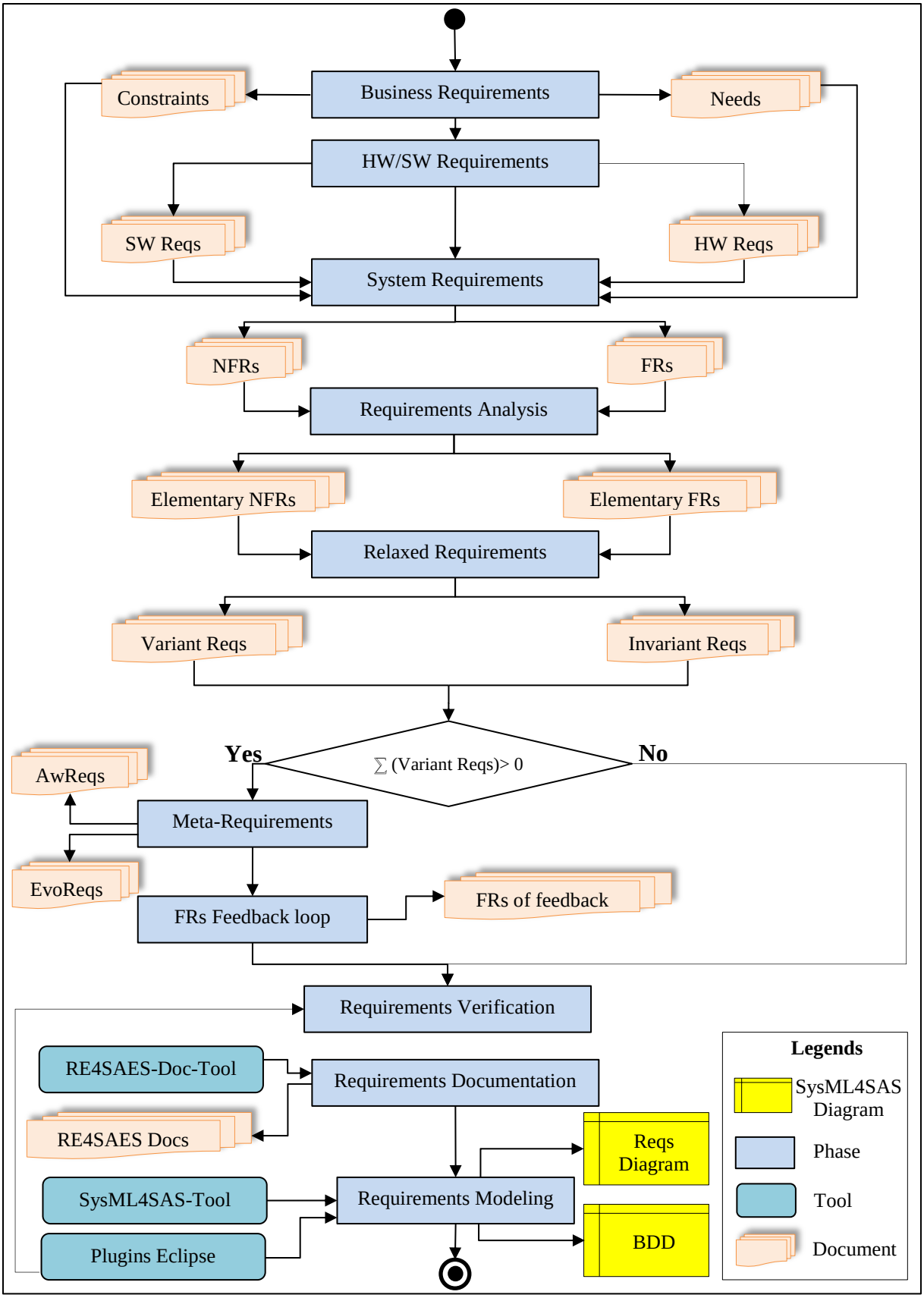


Figure 4-1 : Diagramme du processus REP4SAES.

2. La démarche du processus REP4SAES

Pour la construction de notre processus, nous avons suivi la structure du Uni-REPM[176], [177], une méthode pour évaluer la maturité du processus d'ingénierie des exigences dans les projets logiciels. Par conséquent, notre processus est structuré en trois niveaux : Domaine de Processus Principal (MPA[1]), Sous-Domaine de Processus (SPA[2]) et Action (ACT[3]). Au niveau supérieur, il y a huit Domaines de Processus Principaux (MPA). Chaque MPA est ensuite décomposé en un ou plusieurs Sous-domaines de Processus (SPA), qui, à leur tour, sont subdivisés en une ou plusieurs Actions. Une Action désigne une activité particulière à exécuter ou un élément spécifique qui doit être présent. Les activités sont interconnectées et souvent itératives, nécessitant une communication continue avec les parties prenantes pour s'assurer que les exigences reflètent fidèlement leurs besoins et contraintes. Ainsi, dans cette section, nous présentons notre processus d'ingénierie des exigences pour les systèmes embarqués auto-adaptatifs (REP4SAES, pour Requirements Engineering Process of Self-Adaptive Embedded Systems). Chaque MPA, SPA et ACT est décrit en détail ci-après. Avant cela, le processus est résumé et illustré dans la figure 4-2.

1. BR - Exigences métier « Business Requirements »

1.1. BR.SI - Identification des parties prenantes « Stakeholders Identification »

Les parties prenantes regroupent les personnes, groupes ou organisations ayant un intérêt dans le développement et le cycle de vie du système. Leur identification est cruciale, car une mauvaise prise en compte peut entraîner des omissions d'exigences critiques, générant des problèmes coûteux ultérieurement. Ainsi, l'identification des parties prenantes assure que le système réponde aux attentes de tous les acteurs concernés.

A. **BR.SI.a1** : Analyse contextuelle

- **Le but** : Identifier la liste des parties prenantes impliquées dans le cycle de vie du système.

[1] Main Process Area.
[2] Sub-Process Area.
[3] Action.

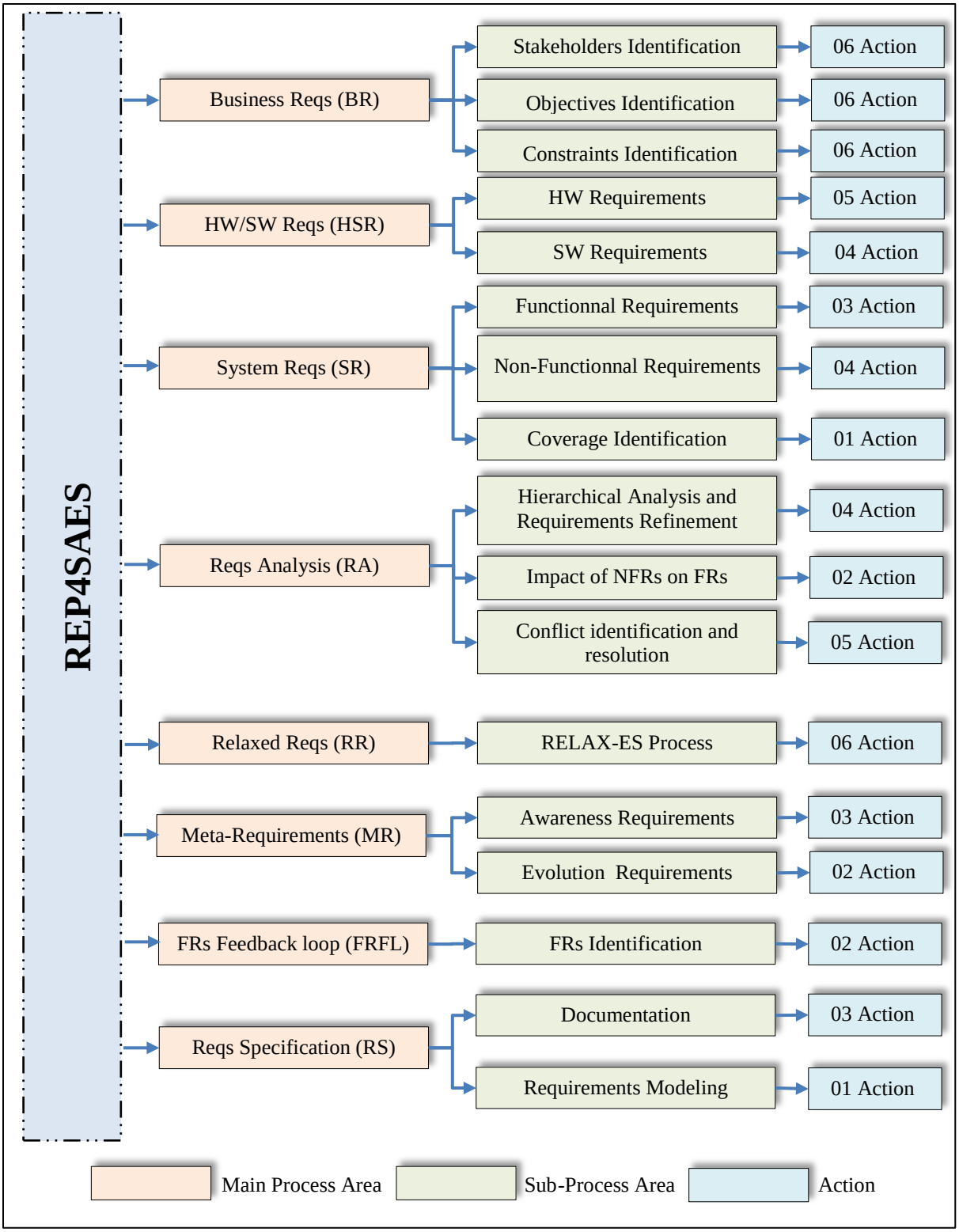


Figure 4-2 : Le processus REP4SAES inspiré du modèle Uni-REPM.

Il faut commencer par une analyse du contexte du système pour déterminer qui est affecté ou impliqué dans son développement. Cela peut inclure des clients, des utilisateurs finaux, des équipes de développement, des gestionnaires de projet, des régulateurs...etc.

B. **BR.SI.a2**: Définition des rôles

- **Le but**: Assurez que chaque partie prenante comprend clairement son rôle et son importance dans le projet.

Dans cette activité, définissez le rôle de chaque partie prenante en fonction de leurs compétences, expertise et contribution attendue au projet. Par exemple, un client peut avoir le rôle de « validateur final des exigences », tandis qu'un ingénieur en électronique peut être le « concepteur des modules matériels ».

C. **BR.SI.a3 :** Attribution des responsabilités

- **Le but** : Garantir que le projet se déroule de manière fluide et que les exigences du système embarqué sont correctement gérées.

Assignez des responsabilités spécifiques à chaque rôle. Par exemple, un gestionnaire de projet peut être responsable de la coordination entre les équipes, tandis qu'un utilisateur final peut être responsable des retours d'expérience sur les prototypes.

D. **BR.SI.a4** : Catégorisation

- **Le but** : Identifier et associer les parties prenantes ayant des compétences différentes à chaque document à produire.

Chaque partie prenante est classée dans l'une des deux catégories suivantes : *Business Stakeholders* ou *Technical Stakeholders*. Les Business Stakeholders regroupent les parties prenantes intéressées par les exigences du point de vue des utilisateurs finaux, des clients et des responsables produits. Ils se concentrent principalement sur les besoins métiers et l'expérience utilisateur, en utilisant le document d'exigences utilisateur (URD[1]). Quant aux Technical Stakeholders, ils incluent les ingénieurs systèmes, les architectes, les développeurs et les testeurs. Leur attention se porte davantage sur les aspects techniques et la mise en œuvre du système, en s'appuyant sur le document d'exigences système (SRD[2]).

[1] User Requirements Document.
[2] System Requirements Document.

E. **BR.SI.a5**: Validation

- **Le but** : Assurer qu'aucun groupe important n'a été oublié et que chaque partie prenante est d'accord et comprend ce qui est attendu d'elle.

La liste des parties prenantes, ainsi que leurs rôles et responsabilités, doit être validée avec les membres de l'équipe de projet afin de s'assurer qu'aucun groupe important n'a été omis et que chaque partie prenante comprend et accepte ce qui est attendu d'elle. En cas d'erreurs, il est nécessaire de revenir à la première activité.

F. **BR.SI.a6** : Documentation

- **Le but** : Les données de chaque partie prenante sont sauvegardées pour une utilisation future dans la génération automatique des documents d'exigences utilisateur et système.

La documentation des différentes informations tout au long des activités de notre processus est effectuée à l'aide de nos plugins Eclipse, générés à partir de notre métamodèle MM4SAES (détaillé dans le chapitre suivant). Cela permet de saisir les informations suivantes pour chaque partie prenante (voir figure 4-3): ID : l'identificateur ; Name : le nom ; Role : le rôle ; Responsibility : la responsabilité ; Contact : numéro de téléphone ou e-mail ; Category : choisir entre Business ou Technical (par défaut, Business).

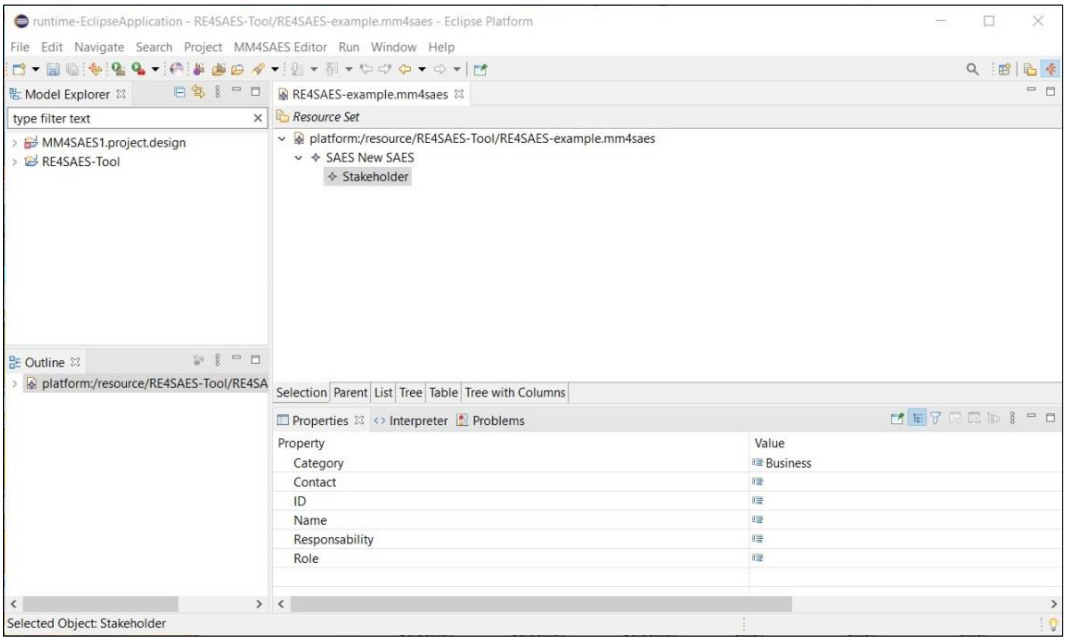


Figure 4-3 : Documentation de la liste des stakeholders à l'aide de notre éditeur graphique.

Il est important de noter qu'avant la génération de nos plugins Eclipse, nous avons enrichi

notre métamodèle avec des contraintes OCL pour la classe *stakeholder* (voir la figure suivante). Nous avons veillé à éliminer les redondances et à garantir une saisie unique des informations pour chaque partie prenante, en s'assurant que tous les champs soient remplis et que chaque identifiant commence par "STK" suivi d'un numéro.

```
class Stakeholder
{
    attribute ID : String[?] = '';
    attribute Name : String[?];
    attribute Role : String[?];
    attribute Responsibility : String[?];
    attribute Contact : String[?];
    attribute Category : Stk_Cat[?] = 'Business';
    invariant NonEmptyID:
        not self.ID->isEmpty();
    invariant NonEmptyName:
        not self.Name->isEmpty();
    invariant NonEmptyRole:
        not self.Role->isEmpty();
    invariant NonEmptyResponsibility:
        not self.Responsibility->isEmpty();
    invariant NonEmptyContact:
        not self.Contact->isEmpty();
    invariant IDUniqueness:
        Stakeholder.allInstances()->isUnique(ID);
    invariant NameUniqueness:
        Stakeholder.allInstances()->isUnique(Name);
    invariant ContactUniqueness:
        Stakeholder.allInstances()->isUnique(Contact);
    invariant IDFormat:
        self.ID.matches('STK[0-9]+');
}
```

Figure 4-4 : Les contraintes OCL concernant la classe stakeholder.

1.2. BR.IO- Identification des objectifs « Identification of Objectives»

A. **BR.IO.a1** : Analyse des Besoins

- **Le but** : Identifiez les besoins.

Pour cerner les besoins et attentes des parties prenantes, il est nécessaire d'effectuer une collecte des besoins. Cette activité peut s'appuyer sur diverses techniques, telles que des entretiens, des ateliers collaboratifs, analyse des scénarios, observations…etc.

B. **BR.IO.a2** : Définition des objectifs stratégiques

- **Le but** : Établissez une vision claire du système en termes de sa mission principale.

Dans cette activité, à partir de la liste des besoins, déterminez les objectifs stratégiques qui reflètent la finalité du nouveau système.

C. **BR.IO.a3** : Décomposition des Objectifs

- **Le but** : Décomposez les objectifs globaux en objectifs spécifiques.

Dans cette activité, chaque objectif global peut être divisé en sous-objectifs afin de fournir des détails supplémentaires. Toutefois, pour une meilleure utilisation de nos plugins Eclipse, il est recommandé que, si un objectif global ne peut pas être divisé en sous-objectifs, il soit considéré comme un sous-objectif.

D. **BR.IO.a4** : Validation des objectifs/Sous-objectifs

- **Le but** : Assurer que les objectifs reflètent fidèlement les attentes des parties prenante et qu'ils sont réalisables.

Les objectifs et sous-objectifs doivent être validés avec les parties prenantes pour garantir qu'ils reflètent fidèlement leurs attentes. Dans le cas où le même besoin ou objectif est déclaré par différentes parties prenantes, il convient de supprimer les redondances.

E. BR.IO.a5 : Documentation

- **Le but** : Les données de chaque besoin, objectif est sous-objectif sont sauvegardées pour une utilisation future dans la génération automatique des documents d'exigences.

Pour chaque besoin, objectif ou sous-objectif, il est nécessaire de saisir les deux informations suivantes : un *ID* en tant qu'identificateur et une *Description* contenant les détails du besoin, de l'objectif ou du sous-objectif. Par ailleurs, pour chaque besoin, il est essentiel d'identifier la *partie prenante* correspondante à partir de la liste des parties prenantes afin de garantir l'origine de chaque besoin (voir la figure 4-5).

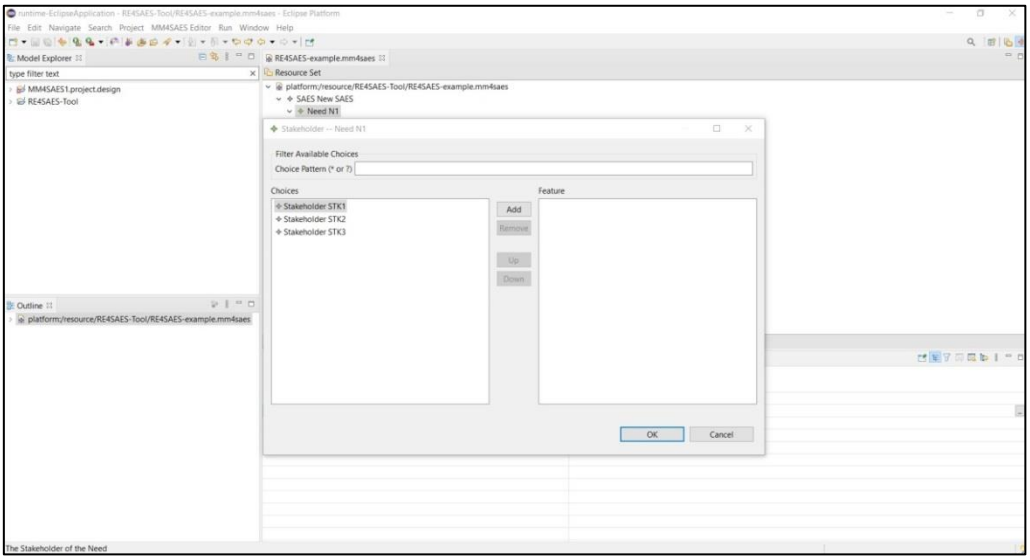


Figure 4-5 : Association des besoins aux parties prenantes.

De la même manière que pour la classe Stakeholder, notre métamodèle a été enrichi avec des contraintes OCL pour les classes Need, Goal et Sub_Goal (voir figure 4-6). Ces contraintes permettent d'éliminer les redondances et de garantir une saisie unique des informations, en veillant à ce que tous les champs soient remplis et que chaque identifiant commence par « N » pour Need, « G » pour Goal, et « S.G » pour Sub_Goal, suivi d'un numéro.

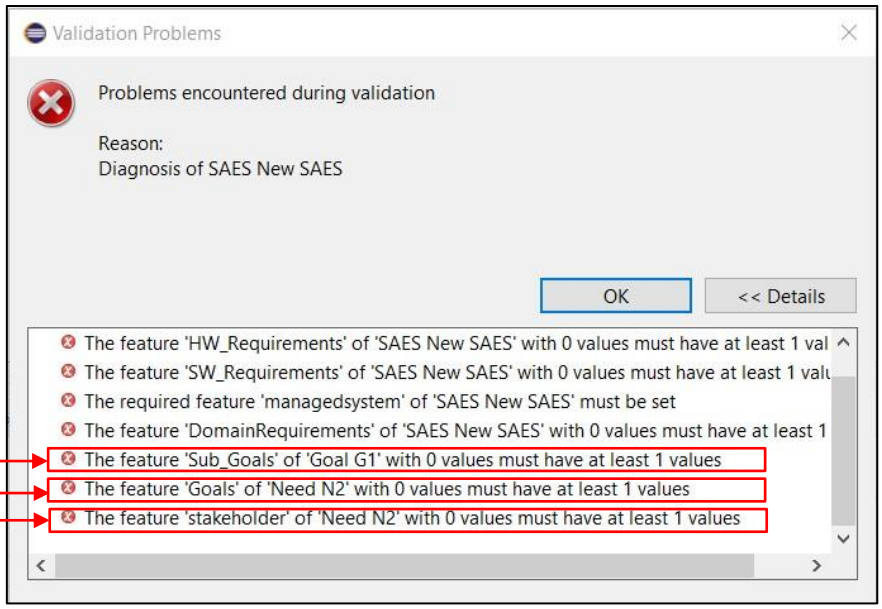
```
class Need
{
    attribute ID : String[?] = '';
    attribute Description : String[?];
    property Goals#need : Goal[+|1] { ordered composes };
    property stakeholder : Stakeholder[+|1] { ordered };

    invariant NonEmptyID:
        not self.ID->isEmpty();
    invariant NonEmptyDescription:
        not self.Description->isEmpty();
    invariant NonEmptyStakeholder:
        not self.stakeholder->isEmpty();

    invariant IDUniqueness:
        Need.allInstances()->isUnique(ID);
    invariant DescriptionUniqueness:
        Need.allInstances()->isUnique(Description);

    invariant IDFormat:
        self.ID.matches('N[0-9]+');

}
```

Figure 4-6 : Les contraintes OCL concernant la classe Need.

F. **BR.IO.a6**: Vérification de la traçabilité

- **Le but** : Assurer que toutes les objectifs/sous-objectifs peuvent être tracées aux besoins initiaux pour faciliter leur gestion tout au long du cycle de vie du projet.

Afin d'assurer une traçabilité optimale, il est essentiel que chaque besoin exprimé par les parties prenantes soit explicitement relié à un ou plusieurs objectifs clairement définis, et que chaque objectif soit également lié à un ou plusieurs sous-objectifs. L'utilisation de nos plugins permet de visualiser et de gérer facilement ces liens. Dans MM4SAES, la cardinalité entre la classe Need et Goal est de 1..*, c'est-à-dire un ou plusieurs, et entre Goal et Sub-Goal, elle est également de 1...* Cela garantit une compréhension claire des relations entre les besoins, les objectifs et les sous-objectifs, tout en évitant les ambiguïtés ou les omissions. En cas d'omission, eclipse affiche un message d'erreur (voir la figure 4-7).



1- L'instance G1 de la classe « Goal », qui ne comporte actuellement aucun sous-objectif « Sub-Goal », doit en inclure au moins un.
2- L'instance N2 de la classe « Need », qui ne comporte actuellement aucun objectif « Goal », doit en inclure au moins un.
3- La propriété "stakeholder" de l'objet N1 de la classe « Need » est vide et doit être remplie afin de déterminer l'origine de chaque besoin.

Figure 4-7 : problème entre les besoins et les objectifs.

1.3. BR.IC - Identification des Contraintes « Identification of Constraints »

Les contraintes définissent les limites dans lesquelles le système doit fonctionner. Si elles ne sont pas identifiées dès le départ, elles peuvent entraîner des erreurs de conception, des retards et des coûts supplémentaires.

A. **BR.IC.a1:** Identification des contraintes du domaine

- **Le but** : Garantit que le système répond aux caractéristiques et contraintes spécifiques du domaine d'application, assurant ainsi son bon fonctionnement.

Dans cette activité, il s'agit d'identifier les exigences de domaine issues du domaine d'application du système et reflétant les caractéristiques et contraintes spécifiques à celui-ci. Contrairement aux besoins des utilisateurs, ces exigences sont dérivées des particularités du domaine d'application et incluent souvent une terminologie spécialisée ou font référence à des concepts propres à ce dernier. Pour les identifier, on peut analyser le domaine d'application, consulter des experts, examiner la documentation existante, et observer les systèmes déjà en place, entre autres.

B. **BR.IC.a2**: Identification des contraintes environnementales

- **Le but** : Prendre en compte les facteurs externes qui influenceront le fonctionnement du système.

Il s'agit d'identifier les contraintes liées à l'environnement où le système sera déployé, comme les températures extrêmes, l'humidité, les vibrations, ou les interférences électromagnétiques. Ces informations sont cruciales pour garantir la robustesse du système.

C. **BR.IC.a3**: Identification des contraintes normatives

- **Le but** : Garantir la conformité du système aux réglementations et normes applicables, facilitant ainsi les processus de certification et d'approbation requis pour sa mise sur le marché.

Cette activité a pour objectif d'identifier et de documenter l'ensemble des exigences imposées par des normes spécifiques, telles que la norme CEI[1]60730-1 pour les dispositifs de commande électrique automatiques à usage domestique et analogue, la norme CEI 61131-3 pour les langages de programmation pour l'automation industrielle, la norme IEC[2] 62443, le standard de la cybersécurité industrielle, ou d'autres normes en fonction du domaine d'application (santé, automobile, aéronautique, etc.). Dans les SEs, en particulier dans les secteurs critiques, ces normes couvrent plusieurs aspects : sécurité, fiabilité, performance, efficacité énergétique, et protection des données. Elles peuvent imposer des contraintes strictes sur le matériel et le logiciel, influençant directement les choix de conception.

D. **BR.IC.a4:** Évaluation des ressources humaines et financières

- **Le but** : Assurer la faisabilité du projet en termes de budget et de personnel.

À cette activité, on analyse les ressources disponibles pour développer et maintenir le système: budget global, compétences requises, nombre de personnes nécessaires, et allocation du temps. Une mauvaise planification peut entraîner des retards ou des dépassements de coûts.

[1] Commission Electrotechnique Internationale - Français.
[2] International Electrotechnical Commission - Anglais.

E. **BR.IC.a5**: Validation avec les parties prenantes

- **Le but** : Assurer que toutes les contraintes sont pleinement comprises et acceptées par l'ensemble des parties prenantes, en obtenant leur accord.

Dans cette activité, il faut valider les contraintes avec les parties prenantes pour garantir que celles-ci sont bien comprises et acceptées par tous, assurant ainsi un alignement complet sur les exigences initiales et évitant tout désaccord ou malentendu ultérieur.

F. **BR.IC.a6**: Documentation

- **Le but** : Les données de chaque contrainte sont sauvegardées pour une utilisation future dans la génération automatique des documents d'exigences utilisateur et système.

Une fois les contraintes identifiées, elles doivent être documentées de manière claire et précise, afin d'être justifiées et compréhensibles par toutes les parties prenantes. À l'aide des plugins Eclipse, il convient de renseigner pour chaque contrainte les informations suivantes : ID, description et source. Pour la source, l'une des options suivantes doit être sélectionnée : *Domain, Environment, Standard and regulation, Human_Resources ou Budget*. De plus, il est essentiel d'associer chaque contrainte à la partie prenante correspondante.

2. HSR - Exigences matériel/logiciel « HW/SW Requirements »

2.1. HSR.HWR - Exigences matériel « HW Requirements »

L'identification des exigences matérielles est cruciale pour le développement des SEs, car elle permet d'assurer que le matériel est adapté aux contraintes spécifiques et aux objectifs de performance du système.

A. **HSR.HWR.a1**: Définition des dispositifs matériels (hardware)

- **Le but** : Identifier le matériel nécessaire.

Pour assurer le bon fonctionnement d'un SE, il est essentiel de définir précisément les composants matériels requis, tels que les capteurs, processeurs, mémoires, et interfaces d'entrée/sortie. Cette activité inclut souvent des entretiens avec les parties prenantes, notamment les utilisateurs finaux, les gestionnaires de projets, et les responsables techniques,

qui fournissent des informations clés sur les attentes en matière de matériel. Par exemple, dans un système médical embarqué, les médecins peuvent indiquer les types de capteurs ou dispositifs nécessaires pour assurer un suivi fiable. De plus, l'élaboration de cas d'usage détaillés aide à identifier des besoins matériels spécifiques, comme des interfaces de communication (Bluetooth, Wi-Fi, ports série) ou des exigences d'autonomie énergétique, particulièrement pour les systèmes alimentés par batterie. En particulier, l'identification des moniteurs est une étape cruciale lors de la définition des dispositifs matériels. Ces moniteurs permettent de suivre en temps réel des paramètres critiques, tels que la température, la consommation d'énergie ou l'état des composants, fournissant ainsi des informations essentielles pour adapter dynamiquement le système aux conditions changeantes.

B. **HSR.HWR.a2**: Évaluation des contraintes physiques

- **Le but** : Optimiser l'efficacité matérielle du système tout en prenant en compte les limitations physiques liées à l'espace, à la taille et à la consommation d'énergie.

Les aspects physiques doivent être soigneusement évalués afin de garantir que le système puisse être déployé dans les conditions d'utilisation prévues, tout en respectant les limites acceptables en termes de volume, de poids et de consommation énergétique. Cette activité peut inclure la simulation de scénarios d'utilisation, permettant d'anticiper les besoins spécifiques du matériel. Par exemple, ces simulations aident à identifier des situations critiques où les performances matérielles, telles que des temps de réponse rapides ou une fiabilité en temps réel, sont essentielles.

C. **HSR.HWR.a3**: Évaluation des compromis

- **Le but** : Prendre des décisions éclairées entre les différents aspects techniques, coûts et performances.

Il peut exister des compromis entre les performances, les coûts et la consommation d'énergie. Une analyse approfondie de ces aspects permet d'identifier les contraintes matérielles les plus adaptées au projet. Il s'agit d'examiner les compromis entre des exigences souvent contradictoires : par exemple, améliorer les performances matérielles peut entraîner une augmentation de la consommation énergétique ou des coûts. Cette activité vise à trouver un équilibre optimal en fonction des priorités du projet, afin de répondre aux objectifs tout en respectant les contraintes techniques et budgétaires.

D. **HSR.HWR.a4**: Validation des contraintes

- **Le but** : Confirmer avec les parties prenantes que les choix matériels respectent les attentes.

Avant de finaliser les choix de matériel, il est essentiel d'obtenir un retour des parties prenantes, notamment sur la conformité des composants choisis par rapport aux exigences de performance, de coût, et de consommation d'énergie.

E. **HSR.HWR.a5**: Documentation

- **Le but** : Les données de chaque dispositif matériel et de chaque exigence matérielle sont sauvegardées pour une utilisation future, facilitant ainsi la génération automatique des documents d'exigences utilisateur et système.

La documentation des exigences matérielles détaille tous les aspects techniques liés au matériel, incluant les composants sélectionnés, leurs caractéristiques, ainsi que les contraintes à respecter. Dans cette activité, nous avons toujours utilisé nos plugins Eclipse pour la documentation. Ainsi, pour chaque dispositif, il est nécessaire de saisir les informations suivantes*: ID, Name, Characteristics*; et pour chaque contrainte : *ID, Name et Description*.

2.2. HSR.SWR: Exigences logicielles « SW Requirements »

A. **HSR.SWR.a1**: Définition des logiciels nécessaires

- **Le but** : Identifier les logiciels requis pour gérer les composants matériels et répondre aux besoins et contraintes du nouveau système.

Cette activité consiste à identifier les différents logiciels requis pour gérer les composants matériels et répondre aux besoins et contraintes du nouveau système comme par exemple la sélection d'un système d'exploitation en temps réel (RTOS), détermination des logiciels applicatifs, logiciels de gestion matérielle (Pilotes)…etc.

B. **HSR.SWR.a2**: Définition des exigences logicielles

- **Le but**: Spécifier en détail les exigences de chaque logiciel identifié afin de guider sa conception et son intégration.

Pour chaque logiciel identifié, les exigences doivent être spécifiées en détail pour guider leur conception et leur intégration, par exemple en définissant les fonctionnalités d'un système d'exploitation en temps réel (RTOS) pour assurer la gestion des tâches critiques ou en précisant les caractéristiques d'un pilote pour interfacer un capteur spécifique.

C. **HSR.SWR.a3**: Validation

- **Le but** : Confirmer avec les parties prenantes que les choix logiciels respectent les attentes ainsi que les différentes contraintes imposées.

Dans cette activité, il est essentiel de valider avec les parties prenantes que les choix logiciels effectués sont conformes aux attentes définies et prennent en compte l'ensemble des contraintes imposées, qu'elles soient environnementales, propres au domaine, matérielles, réglementaires ou budgétaires. Cette étape garantit que les solutions sélectionnées répondent aux besoins exprimés tout en respectant les limitations et exigences spécifiques au projet.

D. **HSR.SWR.a5**: Documentation

- **Le but** : Les données de chaque logiciel et de chaque exigence logiciel sont sauvegardées pour une utilisation future, facilitant ainsi la génération automatique des documents d'exigences utilisateur et système.

Dans cette activité, nous avons toujours utilisé nos plugins Eclipse pour la documentation. Ainsi, pour chaque logiciel, il est nécessaire de saisir les informations suivantes *: ID, Name, Characteristics* ; et pour chaque contrainte logicielle : *ID, name et Description.*

3. SR – Exigences du système « System Requirements »

3.1. SR.FR - Exigences fonctionnelles « Functional Requirements »

A. **SR.FR.a1** : Formuler les Exigences Fonctionnelles

- **Le but** : Déterminer *les fonctionnalités* que le nouveau système embarqué doit réaliser.

Dans cette activité, il s'agit de dériver la liste des exigences fonctionnelles du nouveau système embarqué, qui proviennent de plusieurs sources essentielles. Elles découlent principalement des sous-objectifs, qui précisent les tâches ou fonctionnalités que le système doit accomplir pour atteindre ses objectifs globaux. Elles sont également influencées par les

contraintes règlementaires, qui imposent des normes et des lois, notamment en matière de sécurité et de protection des données. Les contraintes matérielles jouent un rôle clé en limitant les exigences fonctionnelles en fonction des capacités physiques de l'équipement. Enfin, les contraintes logicielles encadrent les exigences en tenant compte des limitations et des capacités des logiciels utilisés, assurant ainsi la compatibilité et l'efficacité optimale du système embarqué.

B. **SR.FR.a2** : Validation

- **Le but** : Réduire les risques de redéveloppement coûteux ou de dysfonctionnements après le déploiement du système.

La validation des exigences fonctionnelles consiste à vérifier que chaque exigence identifiée répond aux besoins et contraintes du nouveau système, en assurant sa conformité avec les attentes des parties prenantes.

C. **SR.FR.a3**: Documentation

- **Le but** : Les données de chaque exigence fonctionnelle sont sauvegardées pour une utilisation future dans la génération automatique des documents d'exigences utilisateur et système.

Dans cette activité, il faut indiquer les informations suivantes pour chaque exigence fonctionnelle : *ID, Name et Description.*

3.2. SR .NFR - Exigences Non-Fonctionnelles « Non-Functional Requirements »

A. **SR.NFR.a1**: Formuler les Exigences Non-Fonctionnelles

- **Le but** : Déterminer *comment* le nouveau système embarqué doit fonctionner, c'est-à-dire définir les caractéristiques et les attributs de qualité qu'il doit posséder.

Les exigences non fonctionnelles d'un système embarqué proviennent de diverses sources essentielles. Pour établir une liste cohérente, il est nécessaire d'adopter une approche systématique tenant compte des sous-objectifs et des contraintes. Cela inclut l'examen des sous-objectifs (par exemple, la réactivité pouvant mener à des exigences de performance), l'analyse des normes de sécurité et de conformité, l'évaluation des contraintes environnementales, matérielles et logicielles. Ces éléments permettent de formuler des

exigences non fonctionnelles précises et mesurables, assurant que le système réponde aux critères de qualité et de performance.

B. **SR.NFR.a2** : Catégorisation

- **Le but** : Structurer et organiser les exigences non fonctionnelles de manière à faciliter leur compréhension, leur gestion et leur priorisation.

Dans cette activité, regrouper les exigences non fonctionnelles en différentes catégories selon leur nature ou leur impact sur le système. Par exemple, les NFRs peuvent être classées en catégories comme la performance (temps de réponse, efficacité), la sécurité (authentification, protection des données), la fiabilité (disponibilité, tolérance aux pannes) ou encore la maintenabilité (facilité de mise à jour, modularité). Cette organisation permet aux équipes de développement d'identifier rapidement les aspects critiques à optimiser et à intégrer dans la conception du système, tout en assurant que chaque exigence non fonctionnelle est traitée de manière adaptée.

C. **SR.NFR.a3** : Validation

- **Le but** : Assurer que le système embarqué fonctionne non seulement comme prévu, mais qu'il respecte également les attentes en matière de performance, de fiabilité, de sécurité, …etc.

Dans cette activité, il faut valider avec les parties prenantes la liste des exigences non fonctionnelles afin de garantir que le système répond à des critères de qualité et de performance qui ne sont pas directement liés à ses fonctionnalités spécifiques, mais qui sont néanmoins essentiels à son succès.

D. **SR.NFR.a4**: Documentation

- **Le but** : Les données de chaque exigence non-fonctionnelle sont sauvegardées pour une utilisation future dans la génération automatique des documents d'exigences utilisateur et système.

Il est important de souligner que la rédaction des exigences non fonctionnelles de manière quantitative est essentielle pour pouvoir les tester objectivement et vérifier si elles sont respectées. En effet, des exigences non fonctionnelles formulées de manière vague ou

subjective, comme "*le système doit être rapide*" ou "*le système doit être sécurisé*", peuvent prêter à des interprétations différentes. À l'inverse, en exprimant ces exigences de manière mesurable, il devient plus facile de valider leur réalisation. Par exemple, au lieu de dire "*le système doit être rapide*", une exigence quantitative serait " *Le système SHALL répondre dans un délai ne dépassant pas 2 secondes pour 95 % des requêtes*". Cette approche permet d'utiliser des critères de test objectifs et précis, facilitant ainsi l'évaluation du système et garantissant qu'il répond aux attentes en matière de qualité, de performance, ou de sécurité.

Dans cette activité, nous avons toujours utilisé Eclipse avec nos plugins pour la documentation. Ainsi, pour chaque exigence non-fonctionnelle, il faut saisir les informations suivantes : *ID, Name, Description et Category*, qui spécifie la catégory de l'exigence non fonctionnelle.

3.3. SR.CV - Vérification de la couverture « Coverage Verification »

A. **SR.FR.a1 :** Vérification de la complétude

- **Le but** : Assurez que toutes les exigences nécessaires pour atteindre les objectifs du système sont présentes et que rien n'a été oublié.

Il est essentiel de s'assurer que chaque exigence est reliée à un sous-objectif ou à une contrainte spécifique, et que chaque sous-objectif/contrainte est bien couvert par une ou plusieurs exigences. Si un sous-objectif n'est pas entièrement pris en compte (voir la figure 4-8), des exigences supplémentaires devront être définies, nécessitant ainsi de revenir à la première activité de ce sous-domaine de processus. De plus, si une exigence ne peut être associée à aucun sous-objectif ou contrainte, elle devra être supprimée. La vérification de cette couverture est assurée par nos plugins Eclipse, garantissant ainsi une traçabilité et une conformité optimales dans le processus de documentation des exigences.

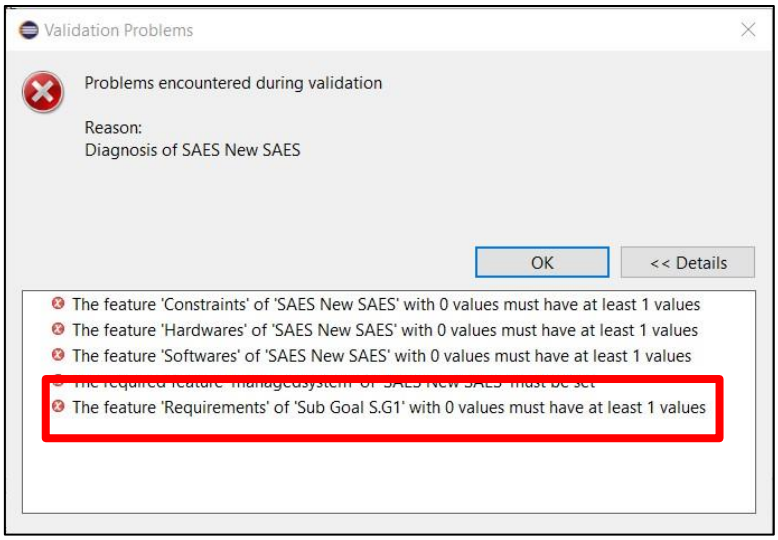


Figure 4-8 : Vérification de la complétude à travers nos plugins Eclipse.

4. RA - Analyse des exigences « Requirements Analysis »

Dans ce domaine du processus, nous avons examiné les exigences recueillies afin de nous assurer qu'elles sont claires, complètes, cohérentes et sans ambiguïté.

4.1. RA.HARR - Analyse hiérarchique et raffinement des exigences «Hierarchical Analysis and Requirements Refinement »

A. **RA. HARR.a1**: Décomposition des exigences

- **Le but** : Clarifier les exigences initiales en les divisant en éléments plus simples et compréhensibles.

Cette activité vise à clarifier les exigences. Une fois les exigences initiales identifiées, elles sont approfondies afin de garantir qu'elles soient bien comprises. Ainsi, dans cette activité, Le processus de décomposition des exigences fonctionnelles et non fonctionnelles utilise des relations de type "AND" et "OR" et se poursuit jusqu'à l'obtention des exigences fondamentales (élémentaires) qui ne peuvent plus être affinées.

B. **RA. HARR.a2**: Affinement des exigences fonctionnelles/non fonctionnelles

- **Le but** : Détailler les exigences élémentaires.

Dans cette activité, chaque exigence élémentaire est raffinée, si possible, afin d'ajouter des précisions, telles que des données quantitatives ou des spécifications supplémentaires.

C. **RA.HARR.a3**: Écrire chaque exigence sous la forme d'une déclaration SHALL

- **Le but** : Validation de la Clarté.

Dans cette activité, nous avons fait l'hypothèse que toutes les exigences sont nécessaires et obligatoires. Par conséquent, il est essentiel d'écrire chaque exigence fondamentale sous la forme d'une déclaration « *SHALL* », qui est une expression semi-formelle afin d'assurer la clarté, la précision et le caractère obligatoire des exigences. En standardisant la formulation des exigences, nous améliorons également la communication entre les équipes et réduisons les risques d'interprétations erronées, ce qui permet de minimiser les retards et les coûts supplémentaires.

D. **RA.HARR.a4**: Documentation

- **Le but** : Les données de chaque exigence sont sauvegardées pour une utilisation future dans la génération automatique des documents d'exigences système.

Dans cette activité, nous avons systématiquement utilisé Eclipse avec nos plugins pour la documentation. Pour chaque exigence fondamentale, il faut saisir les informations suivantes: *ID, Name et Description*. Toutes les exigences sont considérées comme invariantes. Ainsi, pour le champ « type d'exigence », il convient de laisser « Invariant » comme option par défaut. En ce qui concerne la description de l'exigence, elle doit suivre la structure suivante : *« The [sujet] SHALL [action] »*. Notez que nous avons ajouté la contrainte OCL suivante pour les exigences fondamentales « *ElementaryReq* » afin de garantir que la description de chaque exigence respecte notre structure imposée.

```
class ElementaryFR extends FR
{
    property metarequirement : MetaRequirement[*|1] { ordered composes };
    property VAR : Env_Var[*|1] { ordered };
    property Refine : ElementaryFR[?];
    property MON : Monitor[*|1] { ordered };
    attribute Type : Req_Type[?];

    invariant syntaxFormat:
        self.Type = 'Invariant' implies self.Description.matches('^The\\s+\\w+\\s+shall\\s+\\w+.*$');

}
```

Figure 4-9 : Contrainte OCL pour l'atribut Description de la classe ElementaryFR et ElementaryNFR

4.2. RA.INF- Impact des exigences non fonctionnelles sur les exigences fonctionnelles « Impact of NFRs on FRs »

L'extraction des exigences fonctionnelles à partir des exigences non fonctionnelles repose sur une analyse approfondie de ces dernières. Il est cependant important de noter que la distinction entre exigences fonctionnelles et non fonctionnelles n'est pas toujours nette. Par exemple, une exigence utilisateur relative à la sécurité peut, à première vue, sembler être une exigence non fonctionnelle, car elle concerne les caractéristiques globales du système. Cependant, une analyse plus poussée peut révéler que cette exigence génère d'autres exigences fonctionnelles, comme la nécessité d'inclure des dispositifs d'authentification des utilisateurs dans le système. Cette complexité montre que les exigences non fonctionnelles peuvent influencer directement la définition des fonctionnalités du système.

A. **RA.INF.a1**: Analyse approfondie des exigences non fonctionnelles

- **Le but** : La génération des exigences fonctionnelles à partir des exigences non fonctionnelles.

Dans cette activité, il faut analyser chaque exigence non fonctionnelle afin de générer, si possible, des exigences fonctionnelles.

B. **RA.INF.a2:** Décomposition et Affinement des exigences fonctionnelles

- **Le but** : Transformer les exigences fonctionnelles générées à partir des exigences non fonctionnelles en exigences précises, claires et sans ambiguïté, afin qu'elles puissent être correctement comprises et mises en œuvre.

En réalité, cette activité englobe quatre activités, car nous poursuivons les mêmes activités que celles réalisées durant le sous-domaine de processus « RA.HARR Analyse hiérarchique et raffinement des exigences». Ainsi, cette activité inclut la décomposition et l'affinement des exigences fonctionnelles générés à partir des exigences non fonctionnelles, la rédaction de chaque exigence sous la forme d'une déclaration SHALL et la documentation.

4.3. RA.CIR: Identification et résolution des conflits « Conflict identification and resolution »

Les exigences entrent souvent en conflit. Par exemple, il peut être requis que la mémoire

maximale utilisée par un système ne dépasse pas 4 Mo. Les contraintes de mémoire sont courantes pour les systèmes embarqués où l'espace est limité. Une autre exigence pourrait être que le système doit être développé en Ada, un langage de programmation pour le développement de logiciels critiques en temps réel. Cependant, il peut ne pas être possible de compiler un programme Ada avec les fonctionnalités requises en moins de 4 Mo. Il y a donc un compromis à faire entre ces exigences : soit adopter un langage de développement alternatif, soit ajouté de la mémoire au système.

A. **RA.CIR.a1** : Identification des conflits

- **Le but** : Analyser les exigences.

Dans cette activité, il faut examiner chaque exigence pour déterminer si elle entre en conflit avec d'autres exigences. Un conflit peut survenir lorsque deux exigences exigent des résultats opposés ou lorsque leur mise en œuvre entraîne des complications techniques ou fonctionnelles.

B. **RA.CIR.a2**: Négociation

- **Le but** : Propositions de Compromis.

Dans cette activité, les parties prenantes engagent des discussions et des négociations afin d'atteindre un compromis acceptable pour tous. Cela peut inclure la reformulation des exigences pour les rendre plus compatibles.

En cas de reformulation des exigences, il est nécessaire de revenir en arrière et de réanalyser les exigences ainsi modifiées.

C. **RA.CIR.a3**: Priorisation

- **Le but**: Déterminer l'importance relative aux exigences.

Afin de traiter en premier les exigences les plus importantes, il faut classer les exigences par ordre de priorité. Pour ce faire, il faut utiliser des critères de priorisation tels que la valeur pour l'utilisateur, les coûts et les risques associés.

Dans notre thèse, nous avons utilisé la technique *MoSCoW* (Must have, Should have, Could have, Won't have). Cette technique est une méthode de priorisation des exigences largement

utilisée dans la gestion de projets et l'ingénierie des exigences. Elle aide à se concentrer sur les fonctionnalités les plus critiques, permet une allocation efficace des ressources en se concentrant sur les priorités, facilite les ajustements en cours de projet et améliore la communication entre les parties prenantes en clarifiant les priorités et les attentes.

MoSCoW est un acronyme qui signifie :

- **Must have (Doit avoir - Critique)** : Les exigences "Must have" sont essentielles pour le succès du projet. Sans elles, le projet échouera ou le produit ne pourra pas fonctionner correctement. Elles représentent les fonctionnalités de base qui doivent être mises en œuvre pour répondre aux besoins minimaux des utilisateurs ou des parties prenantes.
- **Should have (Devrait avoir - Essentielle)** : Les exigences "Should have" sont importantes mais non critiques. Elles sont souhaitables et ajoutent de la valeur au projet ou au produit, mais leur absence n'entraînerait pas un échec total. Elles sont souvent les suivantes en termes de priorité après les "Must have".
- **Could have (Pourrait avoir - Souhaitable)** : Les exigences "Could have" sont des fonctionnalités qui seraient agréables à avoir si le temps et les ressources le permettent, mais elles ne sont pas essentielles au fonctionnement du produit. Leur absence n'affecterait pas de manière significative la performance ou l'acceptabilité du produit.
- **Won't have (Ne doit pas avoir)** : Les exigences "Won't have" sont celles qui sont explicitement exclues du périmètre de la version actuelle du projet. Elles peuvent être mises de côté pour une future itération ou version, mais elles ne seront pas considérées pour l'instant.

Donc, Chaque exigence est classée dans l'une des quatre catégories :

- **Must have** : Exigences cruciales et non négociables (valeur = 100).
- **Should have** : Exigences importantes mais pouvant être négociées en cas de contraintes de temps ou de ressources (valeur = 66).
- **Could have** : Exigences supplémentaires qui peuvent être incluses si possible (valeur = 33).
- **Won't have** : Exigences qui seront exclues de la version actuelle (valeur = 0).

La priorisation des exigences est dynamique et peut évoluer au fil du temps. Il est essentiel de revoir régulièrement les exigences et d'ajuster les priorités en fonction des nouvelles

informations ou des changements intervenus dans le projet. Toutefois, les exigences classées comme Wont Have restent systématiquement exclues de la version actuelle et ne subissent pas de révision.

D. **RA.CIR.a3**: Validation et Consensus

- **Le but** : Obtenir l'accord de toutes les parties prenantes sur les classifications.

Il est essentiel d'obtenir l'accord de toutes les parties prenantes sur les classifications. Des réunions et des discussions peuvent être nécessaires pour atteindre un consensus.

E. **RA.CIR.a4**: Documentation

- **Le but** : Conserver une trace claire de la priorité de chaque exigence fondamentalle.

Dans le cadre de cette activité, il est essentiel de maintenir une trace précise de la priorité de chaque exigence élémentaire, afin de garantir qu'elles soient traitées selon leur niveau d'importance. Ainsi, pour chaque exigence fonctionnelle ou non fonctionnelle fondamentale, il convient de sélectionner la valeur de la propriété « Priority » parmi les catégories suivantes: *Must_Have*, *Should_Have*, *Could_Have et Won't_Have.*

5. RR –Exigences assouplies «Relaxed Requirements»

Dans cette MPA, chaque exigence fondamentale fait l'objet d'une analyse approfondie. Cette analyse permet de classer les exigences en deux catégories distinctes : les exigences variantes, relaxées ou assouplies, qui sont formulées de manière à permettre une certaine flexibilité dans leur satisfaction, et les exigences invariantes, qui restent constantes et doivent être satisfaites en toutes circonstances. Cela donne la base sur laquelle le système pourra s'adapter et facilite ainsi la gestion de la flexibilité et de la stabilité du système, en mettant en évidence les exigences pouvant être modifiées pour répondre aux besoins d'auto-adaptation, tout en préservant celles qui garantissent les fonctionnalités de base.

5.1. RR.RP: Le processus RELAX_ES «RELAX_ES Process»

Le rapport flexibilité/coût est un aspect crucial à considérer lors de la conception et du développement de systèmes embarqués. Puisqu'ils sont généralement conçus pour des tâches spécifiques avec des ressources limitées, ce qui impose un équilibre délicat entre la flexibilité

du système et les coûts associés. Par conséquent, le compromis entre flexibilité et coût doit être soigneusement évalué en fonction des besoins spécifiques du système. A titre d'exemple, pour un système embarqué à usage unique ou à très faible coût, la flexibilité peut être réduite pour minimiser les dépenses. Par contre, pour des systèmes critiques ou avec des cycles de vie longs, investir dans la flexibilité peut s'avérer rentable.

Une fois que l'ingénieur des exigences détermine qu'un certain niveau de flexibilité peut être toléré, les développeurs en aval, disposent de la latitude nécessaire pour intégrer les mécanismes d'auto adaptation les plus appropriés afin de soutenir les fonctionnalités souhaitées. Nous adaptons ainsi le processus RELAX pour répondre aux besoins et aux spécificités des systèmes embarqués, donnant naissance à une nouvelle version appelée RELAX_ES, illustrée dans la figure 4-10.

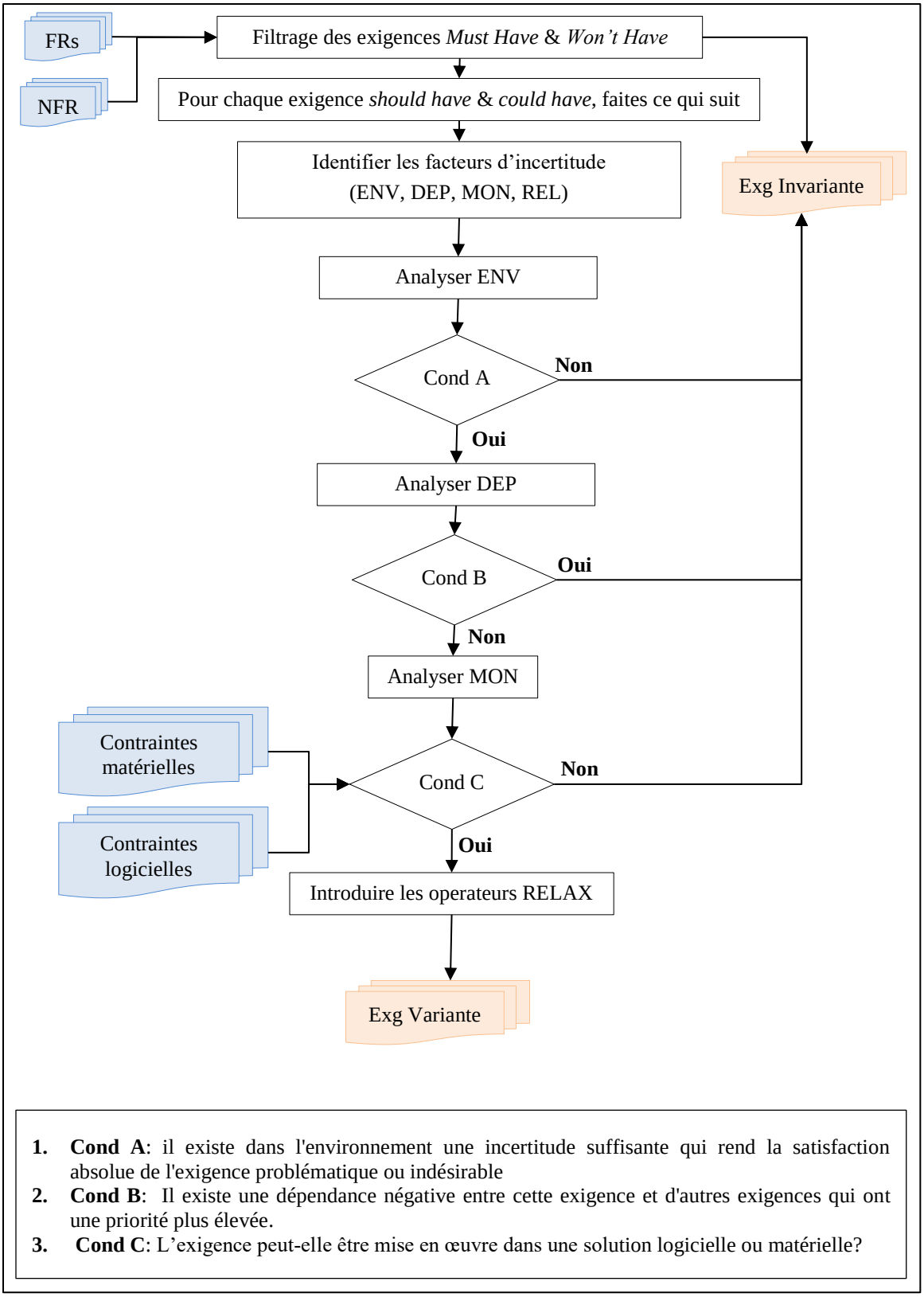


Figure 4-10 : Le processus RELAX_ES.

A. **RR.RP.a1:** Filtrage des exigences

- **Le but:** Filtrage initial des exigences.

Dans cette activité, il est nécessaire de réaliser un filtrage primaire des exigences. Ainsi, pour chaque exigence fondamentale, appliquez les étapes suivantes:

- Chaque exigence ayant la priorité «*Won't Have*», c'est-à-dire possédant la valeur 0, devrait être éliminée, car elle n'est pas essentielle au fonctionnement du système dans le contexte actuel du projet et ne justifie pas l'allocation de ressources limitées. L'élimination de ces exigences permet de concentrer les efforts sur les exigences plus critiques, qui soutiennent la fiabilité et l'auto-adaptabilité du système.
- Chaque exigence ayant la priorité «*MustHave*», c'est-à-dire possédant la valeur 100, devrait être considérée comme invariante, car elle représente une fonctionnalité ou une caractéristique fondamentale du système. Dans le contexte des systèmes auto-adaptatifs, les exigences invariantes sont celles qui ne peuvent ni être modifiées ni ignorées, même en cas d'adaptation. Il est donc essentiel que les processus d'auto-adaptation respectent ces exigences «*MustHave*» afin de prévenir toute dégradation inacceptable des performances du système. Le respect de ces exigences invariantes constitue un pilier de la fiabilité, surtout dans les SEs, où la violation de telles contraintes critiques pourrait engendrer des conséquences graves. donc, les exigences invariantes forment le noyau du système, elles doivent toujours être respectées pour garantir la fonctionnalité essentielle du système.

B. **RR.RP.a2:** Identifier les facteurs d'incertitude

- **Le but:** Identification des facteurs d'incertitude pour les exigences potentiellement relaxable.

Dans notre version du langage RELAX, les exigences potentiellement relaxables sont celles qui ont la priorité «*Should have*» c'est-à-dire possédant la valeur 66, ou «*Could have*» c'est-à-dire possédant la valeur 33. Ainsi, pour chaque exigence potentiellement relaxable, il faut identifier:

- **ENV** : Identifier et décrire la portion de l'environnement associée à cette exigence. L'objectif est d'aider l'ingénieur des exigences à repérer d'éventuelles incertitudes dans l'ENV, susceptibles de compliquer la satisfaction de l'exigence et d'entraîner la nécessité

de son assouplissement (RELAX-ation).

- **MON** : Identifier les propriétés observables de l'environnement qui peuvent être surveillées.
- **REL**: Nous supposons que les attributs ENV (propriétés de l'environnement) et MON (monitorables) coïncident, sauf dans les cas où les propriétés de l'environnement ne peuvent pas être observées directement. Dans ces situations, la relation entre ENV et MON est définie explicitement à l'aide de REL. Prenons l'exemple d'un avion équipé uniquement d'un équipement de recherche de direction. Cet avion ne peut pas estimer directement sa position. Cependant, il peut mesurer sa distance par rapport à un ensemble de points de passage (waypoints) connus et calculer sa position à partir de ces mesures. Dans ce contexte, la position de l'avion est considérée comme une propriété de l'environnement, car elle représente un état ou une caractéristique que l'avion cherche à comprendre ou surveiller. En revanche, les distances par rapport aux points de passage sont des monitorables, c'est-à-dire des données mesurables ou observables servant de base pour effectuer le calcul de la position. REL serait alors utilisé pour formaliser la manière dont la position de l'avion est déduite des distances mesurées. Cette relation explicite permet de lier les données observées (distances) à la propriété de l'environnement (position), en fournissant un cadre rigoureux pour intégrer les observations et les calculs nécessaires
- **DEP** : Dans le langage RELAX, la notion de dépendance positive ou négative entre les exigences permet de décrire comment la satisfaction d'une exigence peut influencer celle d'une autre, en fonction de l'assouplissement (ou "relaxation") de certaines contraintes pour répondre aux changements de contexte. *Une dépendance positive* entre deux exigences signifie que la satisfaction de l'une facilite la satisfaction de l'autre. Par exemple, si une exigence A a une dépendance positive avec une exigence B, alors en assouplissant A, il devient plus facile de satisfaire B. À l'inverse, *une dépendance négative* signifie que l'assouplissement d'une exigence rend la satisfaction de l'autre plus difficile. Cela veut dire qu'en relâchant les contraintes d'une exigence, on complique la satisfaction de l'exigence dépendante.

C. **RR.RP.a3:** Analyser les facteurs d'incertitude

- **Le but:** Classification finale des exigences potentiellement relaxable en exigences variantes ou invariantes.

Dans cette activité, pour chaque exigence potentiellement relaxable, il convient d'analyser les éléments suivants :

- **ENV** : Analysez l'ENV pour déterminer si l'environnement présente une incertitude suffisante pour rendre la satisfaction absolue de l'exigence problématique ou indésirable : « *Existe-t-il une incertitude suffisante dans l'environnement qui empêche la satisfaction totale de cette exigence ?* ». Si tel est le cas, l'exigence doit passer à l'étape suivante pour l'application des opérateurs RELAX. En revanche, si l'analyse ne révèle aucune incertitude dans l'environnement, l'exigence est considérée comme potentiellement toujours satisfaisable et doit être identifiée et maintenue comme invariante.
- **DEP:** Si cette exigence «A» de priorité "could have/should have" présente une dépendance négative avec une exigence «B» de priorité "must have/should have", il est nécessaire d'identifiée et maintenue cette exigence comme invariante, même si sa priorité est moins élevée. Ce choix se justifie par le fait que l'exigence B, ayant une priorité plus élevée, doit être préservée pour assurer des fonctionnalités importantes ou des objectifs critiques du système. Par conséquent, en fixant «A» comme invariant, on évite d'introduire des risques susceptibles de compromettre l'exigence «B», préservant ainsi sa stabilité et sa fiabilité. Toutefois, si l'adaptation de «A» est possible sans altérer significativement «B», une relaxation pourrait être envisagée. En général, lorsqu'une dépendance négative affecte une exigence plus prioritaire, conserver l'exigence moins prioritaire comme invariante constitue une approche de sécurité**.**
- **MON:** Dans un contexte auto-adaptatif, il est crucial de valider les exigences techniques avant de décider d'assouplir une exigence. Cette validation repose sur la vérification des contraintes matérielles et logicielles préalablement définies, afin de s'assurer qu'elles permettent le déploiement de capteurs capables de détecter et quantifier les sources d'incertitude. Si ces conditions sont satisfaites, l'exigence peut être considérée comme assouplie. En revanche, si les contraintes ne sont pas remplies, il devient nécessaire de revenir à la phase d'identification des exigences matérielles et logicielles, et d'interagir avec les parties prenantes pour envisager l'ajout de nouveaux capteurs ou l'ajustement des ressources nécessaires. Si l'ajout de capteurs est possible, l'exigence est considérée comme assouplie ; sinon, elle doit être maintenue comme invariante.

D. **RR.RP.a4:** Introduire le(s) opérateur(s) RELAX

- **Le but:** Transformer les exigences assouplies à partir d'exigences strictes en exigences davantage flexibles et modulables.

Dans cette activité, afin de gérer les différentes incertitudes et pour donner plus de souplesse à chaque exigence assouplie, il est nécessaire d'introduire un ou plusieurs opérateurs RELAX pour chaque exigence assouplie. Ces opérateurs peuvent être Modal, Temporel ou Ordinal (voir le tableau suivant). Chacun d'eux définit des contraintes sur la façon dont une exigence peut être assouplie en moment de l'exécution.

Tableau 4-1 : Les opérateurs du langage RELAX **[7]**.

Opérateur	Explication
Modal Operators	
SHALL	Une exigence doit être respectée.
MAY … OR	Une exigence spécifie une ou plusieurs alternatives.
Temporal Operators	
EVENTUALLY	une exigence doit être respectée finalement.
UNTIL	Une exigence doit être respectée jusqu'à une position future.
BEFORE, AFTER	une exigence doit être respectée avant ou après un événement particulier.
IN	une exigence doit être respectée pendant un intervalle de temps particulier.
AS EARLY, LATE AS POSSIBLE	Une exigence spécifie quelque chose qui devrait être réalisé le plus tôt possible ou qui devrait être retardé le plus longtemps possible.
AS CLOSE AS POSSIBLE TO [frequency]	Une exigence spécifie quelque chose qui se produit de manière répétée, mais la fréquence peut être assouplie.
Ordinal Operators	
AS CLOSE AS POSSIBLE TO [quantity]	Une exigence spécifie une quantité dénombrable, mais le décompte exact peut être assoupli.
AS MANY, FEW AS POSSIBLE	Une exigence spécifie une quantité dénombrable, mais le décompte exact peut être assoupli.

E. **RR.RP.a5:** Rédiger chaque exigence assouplie selon la grammaire RELAX

- **Le but:** Validation de la Clarté.

Dans cette activité, il convient de réécrire la description associée à chaque exigence assouplie en utilisant la grammaire du langage RELAX.

Une expression RELAX valide est toute conjonction d'énoncés s_1; ...; s_m, où chaque s_i est généré par la grammaire suivante.

$$
\begin{aligned}
\phi := \; & true \mid false \mid p \mid SHALL\ \phi \\
& \mid MAY\ \phi_1 \mid OR\ MAY\ \phi_2 \\
& \mid EVENTUALLY\ \phi \mid \phi_1\ UNTIL\ \phi_2 \\
& \mid BEFORE\ e\ \phi \mid AFTER\ e\ \phi \mid IN\ t\ \phi \\
& \mid AS\ CLOSE\ AS\ POSSIBLE\ TO\ f\ \phi \\
& \mid AS\ CLOSE\ AS\ POSSIBLE\ TO\ q\ \phi \\
& \mid AS\ \{EARLY, LATE, MANY, FEW\}\ AS\ POSSIBLE\ \phi
\end{aligned}
$$

Figure 4-11 : La grammaire du langageRELAX**[7]**.

Les paramètres des opérateurs RELAX sont typés comme suit :

- *p* : une proposition atomique;
- *e* : un événement, une occurrence notable qui se produit à un instant donné dans le temps;
- *t* : un intervalle de temps, correspond à toute durée délimitée par deux instants;
- *f* : une fréquence, définit un nombre d'occurrences d'un événement au sein d'un intervalle de temps donné. Si le nombre d'occurrences n'est pas spécifié, il est supposé être égal à un;
- et *q* est une quantité i.e. quelque chose de mesurable (peut être énumérée). En particulier, une expression RELAX Φ est dite quantifiable si et seulement s'il existe une fonction Δ telle que $\Delta(\Phi)$ soit une quantité.

En outre, il faut noter que la sémantique formelle du langage RELAX est définit en associant ce langage avec FBTL[1][178].

F. **RR.RP.a6:** Documentation

- **Le but:** Les données de chaque exigence assouplie, ainsi que leurs facteurs d'incertitude, sont sauvegardées pour une utilisation future dans la génération automatique du document des exigences système.

Dans cette activité, il est nécessaire de mettre à jour à la fois la description et le type des exigences assouplies. La description doit être conforme à la syntaxe de la grammaire RELAX. Le type des exigences doit être modifié de Invariant à Variant. De plus, il est essentiel

[1] Fuzzy Branching Temporal Logic.

d'enregistrer les informations relatives à chaque variable d'environnement, incluant *l'ID* et la *description*. Il convient également de mettre à jour la liste des dispositifs de surveillance et de contrôle des variables environnementales, en précisant pour chacun *l'ID, la description*, ainsi que *la relation* qui définit le lien entre le dispositif de surveillance (Monitor) et la variable d'environnement correspondante. Enfin, il faut ajouter les relations de dépendances positives et négatives entre les exigences.

Nous avons également ajouté deux contraintes OCL :

1. Cont1 : aucune exigence de priorité « Must have » ne doit avoir le type Variant.
2. Cont2 : aucune exigence de priorité inférieure et de type Variant ne doit avoir de relation de dépendance négative avec une exigence de priorité supérieure.

```
abstract class Requirement
{
    attribute ID : String[?];
    attribute Name : String[?];
    attribute Description : String[?];
    attribute Formal_Description : String[?];
    attribute Priority : Priority[?];

    operation priorityOrder() : Integer
    {
    body:
        if self.Priority = 'Must_Have' then 4
            else if self.Priority = 'Should_Have' then 3
                else if self.Priority = 'Could_Have' then 2
                    else if self.Priority = 'Wont_Have' then 1
                        else 0
                        endif
                    endif
                endif
            endif;
    }
    property PositiveDEP : Requirement[*|1] { ordered };
    property NegativeDEP : Requirement[*|1] { ordered };
    property OperationalizedByManagedSys : ManagedSystem[?];
    property OperationalizedByManagingSys : ManagingSystem[?];
    invariant NonEmptyID:
        not self.ID->isEmpty();
    invariant NonEmptyDescription:
        not self.Description->isEmpty();
    invariant NonEmptyFormalDescription:
        not self.Formal_Description->isEmpty();
    invariant IDUniqueness:
        Requirement.allInstances()->isUnique(ID);
    invariant DescriptionUniqueness:
        Requirement.allInstances()->isUnique(Description);
    invariant FormalDescriptionUniqueness:
        Requirement.allInstances()->isUnique(Formal_Description);
}
```

Ajouter une opération pour attribuer un ordre numérique aux priorités

```
class ElementaryFR extends FR
{
    property metarequirement : MetaRequirement[*|1] { ordered composes };
    property VAR : Env_Var[*|1] { ordered };
    property Refine : ElementaryFR[?];
    property MON : Monitor[*|1] { ordered };
    attribute Type : Req_Type[?];

    invariant syntaxFormat:
        self.Type = 'Invariant' implies self.Description.matches('^The\\s+\\w+\\s+shall\\s+\\w+.*$');

    invariant MustHaveNotVariant:
        self.Priority = 'Must have' implies self.Type <> 'Variant';

    invariant NoNegativeDependencyForLowerPriorityVariant:
        self.Type = 'Variant' and self.NegativeDEP->notEmpty()
        implies self.NegativeDEP->forAll(dep |self.priorityOrder() >= dep.priorityOrder());
}
```

Cont1

Cont2

Figure 4-12 : Contraintes OCL pour les exigences assouplies.

6. MR - Méta- Exigences « Meta-Requirements »

Dans cette MPA, il convient de spécifier les exigences liées au comportement auto-adaptatif sous forme de méta-exigences, c'est-à-dire des exigences portant sur les exigences régulières du système. Ainsi, nous nous intéressons à l'étude des exigences qui mènent à la fonctionnalité des boucles de rétroaction. Autrement dit, *dans le modèle MAPE-K, si les boucles de rétroaction constituent une solution architecturale, quel problème d'exigences cette solution est-elle censée résoudre, à savoir les exigences du système géré* ? C'est dans cette perspective que cette MPA propose une approche permettant de spécifier les méta-exigences pour les boucles de rétroaction. Elle repose sur les concepts d'exigences de conscience et d'exigences d'évolution. Ces deux types d'exigences sont complémentaires : les exigences de conscience définissent les situations nécessitant une adaptation, tandis que les exigences d'évolution spécifient les actions à entreprendre dans ces situations. En combinant ces deux éléments, l'ingénieur peut définir les exigences d'une boucle de rétroaction qui permet d'opérationnaliser l'adaptation en temps réel.

6.1. MR.AwReq: Exigences de conscience « Awereness requirements »

Les exigences de conscience (AwReqs[1]) permettent de définir des exigences relatives à la réussite, à l'échec ou à la qualité de service d'autres exigences au moment de l'exécution. Donc, elles font référence à d'autres exigences en spécifiant le degré de réussite acceptable ou le degré d'échec tolérable, permettant ainsi aux parties prenantes et aux ingénieurs des exigences de raisonner sur l'adaptation du système à un niveau d'abstraction plus élevé. En

[1] Awareness Requirements.

effet, les AwReqs abordent les différents états que les exigences peuvent prendre durant leur exécution, ce qui facilite une approche plus flexible dans la gestion des adaptations du système. En utilisant les AwReqs, il est également possible d'exprimer des approximations pour les exigences RELAX-ed, contribuant ainsi à une modélisation plus fine et tolérante aux incertitudes inhérentes aux systèmes auto-adaptatifs.

Il existe cinq types d'exigences de conscience (Voir la section 8 du chapitre 2). Ainsi, chaque sous-domaine du processus se concentre ensuite sur l'extraction et l'identification de l'un de ces types.

A. **MR.AwReq.a1** : Détermination des Exigences NeverFail « Regular Requirement »

- **Le but:** Garantit une fiabilité maximale pour les fonctionnalités essentielles en stipulant que les exigences critiques restent infaillibles, répondant ainsi aux impératifs de fiabilité et de continuité du système

Une exigence de conscience régulière est une exigence imposée à une autre exigence qui ne doit jamais échouer. Dans cette activité, en associant chaque exigence invariante « R » au modèle NeverFail(R), on indique que cette exigence doit être satisfaite de manière continue, quelles que soient les circonstances.

Notons qu'il existe des modèles analogues à NeverFail, tels que «*AlwaysSucceed*» et «*NeverCanceled*». Cependant, dans notre processus, seul le modèle NeverFail a été utilisé.

B. **MR.AwReq.a2** : Association des exigences assouplies aux exigences de conscience

- **Le but:** Association des exigences assouplies aux exigences de conscience.

Dans cette activité, chaque exigence relaxée est associée à une ou plusieurs exigences de conscience.

C. **MR.AwReq.a3**: Documentation

- **Le but:** Les données de chaque exigence de conscience sont sauvegardées pour une utilisation future dans la génération automatique du document des exigences système.

Dans cette activité, Pour chaque exigence de conscience, il faut saisir les informations suivantes: *ID, Description et TypeAWReq*.

6.2. MR.EvoReq : Exigences d'evolution « Evolution requirements »

Les exigences d'évolution spécifient les actions à entreprendre dans les situations nécessitant une adaptation.

A. **MR.EvoReq.a1** : Identification des exigences d'évolution

- **Le but:** Identification des exigences d'évolution.

Une fois qu'une exigence relaxée a été associée à des exigences de conscience, il faut déterminer comment le système évoluera ou s'adaptera en fonction de ces conditions observées. Les exigences d'évolution guideront les actions ou les stratégies spécifiques que le système doit mettre en œuvre pour ajuster son comportement (ex. réduire la charge, activer un mode d'économie d'énergie, reconfigurer le système) afin de répondre aux exigences relaxées tout en respectant les exigences invariantes.

B. **MR.EvoReq.a2** : Documentation

- **Le but:** Les données de chaque exigence d'evolution sont sauvegardées pour une utilisation future dans la génération automatique du document des exigences système.

Dans cette activité, Pour chaque exigence d'evolution, il faut saisir les informations suivantes: *ID et Description*.

7. FRFL - Exigences fonctionnelles de la boucle de rétroaction « Functional Requirements of the Feedback Loop »

Les exigences fonctionnelles des boucles de rétroaction se rapportent au comportement attendu d'une boucle de rétroaction dans un système de contrôle, en particulier dans des systèmes auto-adaptatifs. Dans le cadre du modèle MAPE (Monitor, Analyze, Plan, Execute), ces exigences fonctionnelles doivent être remplies pour assurer que chaque phase de la boucle de rétroaction fonctionne correctement.

7.1. FRFL.FRFL : Exigences fonctionnelles de la boucle de rétroaction « Functional Requirements of the Feedback Loop »

A. **FRFL.FRFL.a1** : Identification des exigences du comportement fonctionnel de la boucle de rétroaction

- **Le but:** Définir comment les différentes phases de la boucle doit interagir et remplir leurs rôles pour garantir une adaptation réactive et efficace.

Les exigences fonctionnelles des boucles de rétroaction définissent comment les différentes phases de la boucle doivent interagir et remplir leurs rôles pour garantir une adaptation réactive et efficace. Par exemple:

- **Durant la phase de surveillance « Monitor »** : La phase doit être capable de collecter des données pertinentes du système en temps réel pour évaluer son état.
- **Durant la phase d'analyse « Analyze »** : L'analyse des données collectées doit permettre d'évaluer si le système fonctionne comme prévu ou s'il est nécessaire d'adopter une nouvelle stratégie.
- **Durant la phase de planification « Plan »** : Sur la base de l'analyse, la phase de planification doit déterminer les actions correctives ou adaptatives nécessaires pour ajuster le système.
- **Durant la phase d'execution « Execute »** : L'exécution doit permettre d'implémenter ces actions de manière effective pour adapter le système.

Il convient de mentionner que durant la phase d'ingénierie des exigences pour les systèmes auto-adaptatifs fondés sur le modèle MAPE-K, il est essentiel d'identifier et de spécifier les exigences fonctionnelles associées aux boucles de rétroaction. Cependant, à ce stade, il ne faut pas déterminer le nombre de boucles de rétroaction ni leur configuration. Cette approche permet de concentrer les efforts sur les exigences fonctionnelles sans présumer d'une architecture spécifique, qu'elle soit centralisée ou décentralisée. En effet, la conception de l'architecture (définition de l'organisation des boucles) intervient généralement dans une phase ultérieure, lorsque les choix d'implémentation doivent être alignés avec les objectifs d'adaptabilité, de scalabilité et de robustesse du système.

B. **FRFL.FRFL.a2** : Décomposition et Affinement des exigences

- **Le but** : Transformer les exigences fonctionnelles en exigences précises, claires et sans ambiguïté, afin qu'elles puissent être correctement comprises et mises en œuvre.

En réalité, cette activité englobe quatre activités, car nous poursuivons les mêmes activités que celles réalisées durant le sous-domaine de processus « RA.HARR Analyse hiérarchique et raffinement des exigences». Ainsi, cette activité inclut la décomposition et l'affinement des exigences fonctionnelles de la boucle MAPE-K, la rédaction de chaque exigence sous la forme d'une déclaration SHALL et la documentation.

8. RS – Spécification des exigences « Requirements Specification »

Dans notre processus, durant la phase de spécification, nous avons adopté la spécification des exigences basée sur la modélisation, une approche qui exploite des représentations graphiques pour clarifier et structurer de manière explicite les besoins du système. Cette méthode repose sur l'idée que « *un modèle graphique vaut mille mots* », jouant ainsi un rôle essentiel en facilitant la compréhension des exigences et réduit les ambiguïtés inhérentes aux descriptions purement textuelles. Elle offre également une base solide pour l'analyse d'impact en cas de modifications ultérieures. Parmi les outils fréquemment utilisés dans ce contexte, le langage SysML se distingue par ses avantages significatifs. SysML permet de modéliser aussi bien les aspects structurels que comportementaux des systèmes, tout en prenant en charge l'intégration des contraintes paramétriques. Sa flexibilité et sa capacité à gérer des systèmes complexes en font un atout majeur, notamment pour les projets nécessitant une forte traçabilité et une collaboration pluridisciplinaire.

8.1. RS.Doc : Documentation

Les exigences peuvent varier entre une déclaration abstraite des services que le système doit fournir ou des contraintes qu'il doit respecter, et une définition formelle/Semi-formelle et détaillée d'une fonctionnalité ou bien qualité du nouveau système. Par conséquent, *les exigences utilisateurs* sont des descriptions générales des services attendus et des contraintes, tandis que *les exigences systèmes* précisent de manière détaillée les fonctions et contraintes du système. Ces niveaux de détail sont utiles pour différents types de lecteurs : les exigences utilisateurs s'adressent souvent à des gestionnaires peu préoccupés par les détails techniques, tandis que les exigences systèmes sont destinées à ceux qui s'occupent de l'implémentation ou

des processus métiers. Donc, il est recommandé de séparer les exigences utilisateur des exigences système plus détaillées. Sinon, les lecteurs non techniques des exigences utilisateur risquent d'être submergés par des détails qui ne concernent que les techniciens.

Dans le but de générer les documents d'une manière automatique, nous avons utilisé la bibliothèque Apache POI de java pour générer un fichier Excel (pour organiser les informations de manière tabulaire et les rendre plus facilement exploitables) à partir d'un fichier XML de notre instance du métamodèle MM4SAES, l'idée est d'extraire les informations du fichier XML, puis de les injecter dans un fichier Excel à l'aide de cette bibliothèque.

Apache POI est une bibliothèque Java utilisée principalement pour la manipulation de documents Microsoft Office, tels que des fichiers Excel (.xls et .xlsx), Word (.doc et .docx), et PowerPoint (.ppt et .pptx). Son principal avantage est de permettre la création, la modification et la lecture de fichiers dans ces formats sans avoir besoin d'installer Office. POI est particulièrement utile pour générer des rapports, extraire des données, ou encore automatiser des traitements sur des fichiers Office.

XmlToExcelConverter.java

```java
package RE4SAES_Documents;

import java.io.File;

public class XmlToExcelConverter {

    public static void main(String[] args) {
        String inputFilePath = "SAES.xml"; // Le chemin du fichier XML

        String outputFilePath1 = "RE4SAES User Requirements Document.xlsx"; //Le chemin du fichier Excel à créer
        XmlToExcelConverter converter1 = new XmlToExcelConverter();
        converter1.convertXmlToExcelUSERGUIDE(inputFilePath, outputFilePath1);

        String outputFilePath2 = "RE4SAES System Requirements Document.xlsx"; //Le chemin du fichier Excel à cré
        XmlToExcelConverter converter2 = new XmlToExcelConverter();
        converter2.convertXmlToExcelSYSTEMGUIDE(inputFilePath, outputFilePath2);
    }

    public void convertXmlToExcelUSERGUIDE(String xmlFilePath, String excelFilePath) {
        try {
                File xmlFile = new File(xmlFilePath);
                DocumentBuilderFactory dbFactory = DocumentBuilderFactory.newInstance();
                DocumentBuilder dBuilder = dbFactory.newDocumentBuilder();
                Document doc = dBuilder.parse(xmlFile);
                doc.getDocumentElement().normalize();
```

Figure 4-13 : Portion du code du générateur de document d'exigences système et utilisateur.

A. **RS.Doc.a1 :** Vérification du système

- **Le but:** Vérifier le nouveau système.

Avant de générer les documents d'exigences utilisateurs et système, le diagramme des

exigences et le diagramme de définition des blocs, il est nécessaire de vérifier le système à l'aide de nos plugins EMF. En cas d'omissions ou d'erreurs, comme illustré à la figure 4.14, celles-ci doivent être complétées ou corrigées jusqu'à validation complète du système, comme montré dans la figure 4.15.

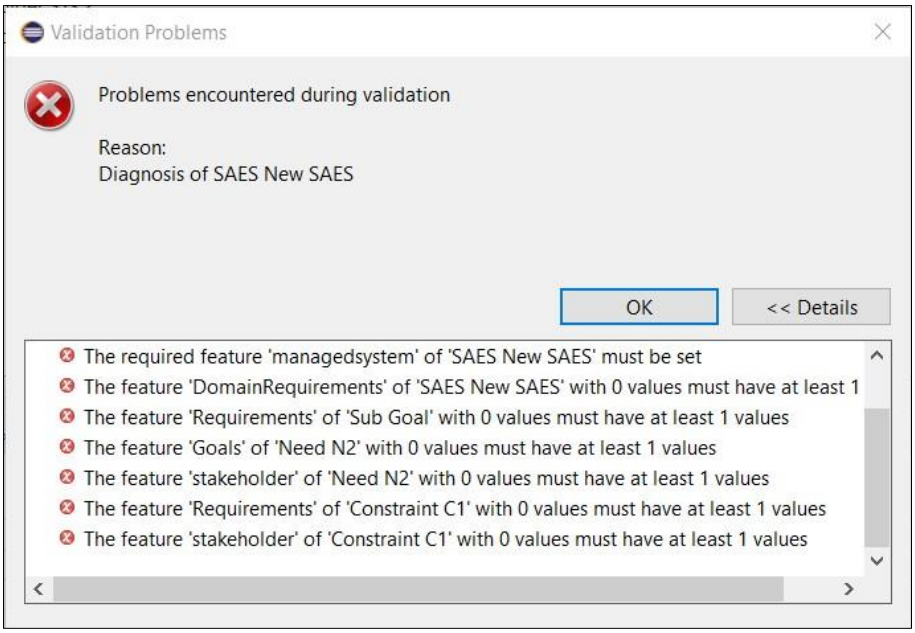


Figure 4-14 : Vérification du nouveau système présente des problèmes.

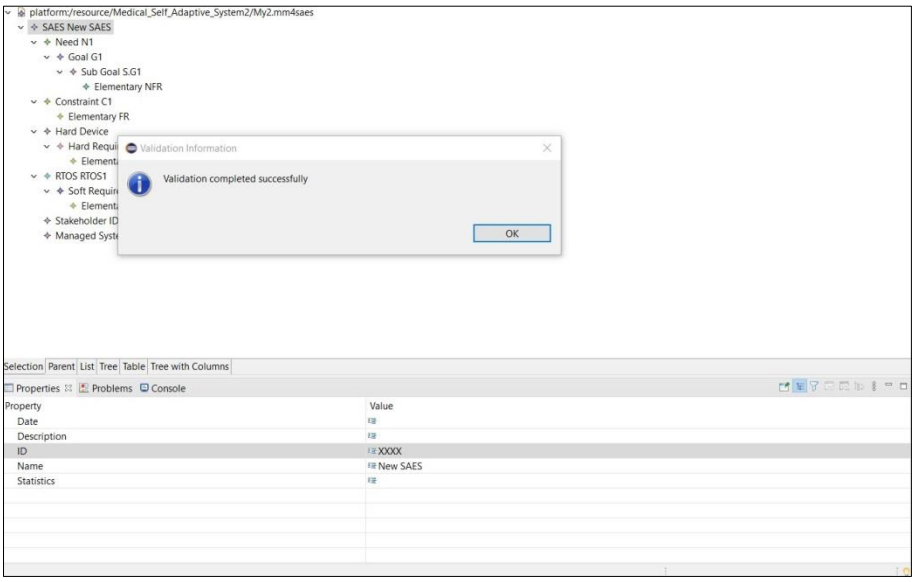


Figure 4-15 : Vérification réussite.

B. RS.Doc.a2 : Générer le document des exigences utilisateur

- **Le but** : Formaliser les besoins des utilisateurs pour guider le développement, assurer la compréhension des attentes et faciliter la validation du système final.

Les exigences utilisateur pour un système doivent décrire les aspects fonctionnels et non fonctionnels de manière compréhensible pour des utilisateurs sans connaissances techniques approfondies. Elles se limitent à spécifier le comportement externe du système, en évitant autant que possible les caractéristiques liées à sa conception. Leur rédaction exclut l'utilisation de jargon logiciel, de notations structurées ou formelles, ainsi que toute description détaillée de l'implémentation. Il est essentiel de formuler ces exigences dans un langage clair et simple. Les informations sont organisées sous forme de tableaux pour faciliter leur compréhension et leur présentation. Ainsi, dans cette activité, le document des exigences utilisateur est généré automatiquement à l'aide de notre outil *RE4SAES-Doc-Tool*.

C. **RS.Doc.a3 :** Générer le document des exigences système

- **Le but** : Traduire les attentes abstraites des utilisateurs en spécifications techniques détaillées.

Les exigences fonctionnelles et non fonctionnelles d'un système passent souvent d'une expression abstraite, en tant qu'exigences utilisateur, à une forme détaillée et précise, en tant qu'exigences système. Ces dernières décrivent avec exactitude les fonctions, qualités ou contraintes spécifiques du système, servant ainsi de base pour la conception et la mise en œuvre technique. En transformant les attentes des utilisateurs en spécifications concrètes, les exigences système permettent une compréhension technique claire et jouent un rôle crucial dans les contrats de développement, en constituant une spécification complète et cohérente. Idéalement, elles se concentrent sur le comportement externe du système et ses contraintes opérationnelles, mais incluent souvent, dans la pratique, des aspects de conception. Contrairement aux exigences utilisateur, formulées dans un langage accessible aux non-spécialistes, les exigences système sont souvent rédigées à l'aide de notations spécialisées telles que des langages naturels structurés, des modèles graphiques ou des spécifications mathématiques formelles. Ainsi, dans cette activité, le document des exigences système est généré automatiquement à l'aide de notre outil *RE4SAES-Doc-Tool*.

8.2. RS.RM : Modélisation des exigences « Requirements modelling »

SysML[1] est un langage de modélisation standardisé par l'OMG[2], largement utilisé pour représenter des systèmes complexes, notamment dans les domaines de l'ingénierie des

[1] Systems Modeling Language.
[2] Object Management Group.

systèmes et des systèmes embarqués. Il s'agit d'une extension de l'UML[1], spécifiquement adaptée pour capturer les spécifications, l'analyse, la conception et la validation des systèmes, en intégrant leurs aspects matériels, logiciels et autres.

Le diagramme des exigences SysML permet de recenser toutes les exigences auxquelles le système doit répondre. Il propose une syntaxe graphique pour représenter les exigences textuelles et les relier entre elles. Il modélise les hiérarchies d'exigences, leurs déclinaisons éventuelles, ainsi que des relations telles que <<satisfy>> et <<verify>>, qui associent une exigence à un élément du modèle. Ainsi, il constitue un pont entre les outils classiques de gestion des exigences et la modélisation des systèmes.

Tableau 4-2 : Les relations dans le diagramme des exigences SysML.

N°	La relation	Description	Notation
01	Contenance	Décomposition d'une macro exigence en exigences plus élémentaires.	
02	Précision	des précisions sont données à propos d'une exigence.	
03	Dérivation	une exigence, non imaginée au départ, est créée à partir d'une autre.	
04	Satisfaction	un bloc (existe dans le diagramme de définition de blocs) satisfait une exigence.	
05	Vérification	un scénario de test vérifie une exigence.	

D'un autre côté, le diagramme de définition des blocs SysML ou BDD[2], est l'un des diagrammes principaux utilisés dans la modélisation des systèmes complexes avec SysML. Il permet de représenter la structure statique d'un système en décrivant ses composants, appelés blocs, ainsi que leurs relations. Chaque bloc peut représenter une partie du système (matérielle, logicielle ou autre) et inclut des propriétés (attributs), des opérations (comportements) et des interfaces. Ce diagramme permet de modéliser la hiérarchie et la décomposition du système en sous-systèmes ou composants, tout en montrant les relations telles que la composition, l'agrégation ou la dépendance. Les blocs peuvent être détaillés davantage à l'aide de diagramme de bloc interne pour illustrer les interactions entre leurs

[1] Unified Modeling Language.
[2] Block Definition Diagram.

parties. Le diagramme de définition de blocs est essentiel pour fournir une vue globale et structurée du système, facilitant ainsi la communication entre les équipes et assurer la traçabilité des exigences vers les composants du système.

Pour soutenir notre processus, nous proposons un nouveau profil SysML, nommé SysML4SAS, spécifiquement adapté à la spécification des exigences des systèmes embarqués auto-adaptatifs. Ce profil enrichit les diagrammes des exigences et de définition de blocs en introduisant de nouveaux stéréotypes et relations, conçus pour capturer les particularités de ces systèmes.

Pour les stéréotypes, nous distinguons les exigences fonctionnelles des exigences non fonctionnelles, ainsi que les exigences invariantes des exigences variantes. De plus, nous avons ajouté deux stéréotypes : l'un pour l'exigence de conscience, « AwReq », et l'autre pour l'exigence d'évolution, « EvoReq ». Notons que, Chaque exigence invariante est associée à une exigence de conscience de type *NeverFail*. Ainsi, pour simplifier le diagramme, nous ne présentons que les exigences de conscience associées aux exigences variantes.

Concernant la propriété Texte (Description), la description de chaque exigence doit être formalisée et respecter strictement la grammaire du langage RELAX. Une description informelle n'est pas admissible car le langage RELAX sera ensuite associé à FBTL pour définir sa sémantique formelle. Cette formalisation garantit une interprétation rigoureuse et exploitable des exigences dans les étapes ultérieures.

Dans le diagramme de définition des blocs, nous présentons la boucle de rétroaction (feedback loop) qui est détaillée en mettant en évidence ses cinq composants principaux : Monitor, Analyze, Plan, Execute et Knowledge. Chaque composant est relié à ses exigences correspondantes, assurant une traçabilité complète entre la structure du système et ses spécifications. Cette approche garantit une meilleure compréhension des liens entre les besoins d'adaptation et leur mise en œuvre structurelle. Par ailleurs, nous avons ajouté un stéréotype pour représenter le matériel (hardware) et un autre pour représenter le logiciel (software). Enfin, un stéréotype supplémentaire a été introduit pour modéliser l'environnement, qui se compose d'un ensemble de variables d'environnement, permettant une prise en compte explicite des facteurs externes influençant le système.

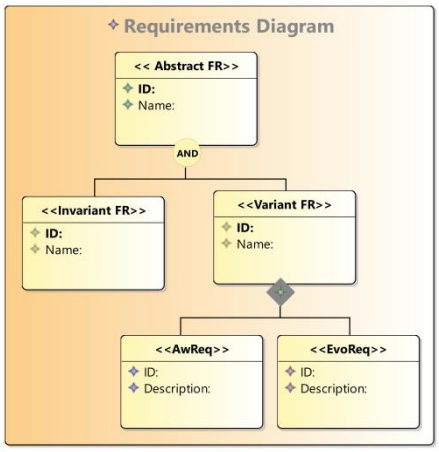


Figure 4-16 : Les stéréotypes SysML4SAS.

Pour les relations, nous avons enrichi SysML avec un ensemble de relations. Premièrement, dans le diagramme des exigences SysML, il existe une relation de contenance permettant la décomposition d'une macro-exigence en exigences plus élémentaires. Dans notre profil, nous avons remplacé la relation de contenance par deux types de décomposition : AND et OR. Cette décomposition est essentielle pour gérer des exigences complexes en les divisant en composants plus petits et plus faciles à gérer. La décomposition AND est utilisée lorsque toutes les sous-exigences doivent être satisfaites pour répondre à la macro-exigence, garantissant une mise en œuvre complète de sa fonctionnalité. À l'inverse, la décomposition OR s'applique lorsque la satisfaction de l'une quelconque des sous-exigences suffit, offrant ainsi une certaine flexibilité pour atteindre la macro-exigence. En définissant ces deux types de décomposition dans notre profil, nous améliorons la clarté et la structure de la modélisation des exigences, ce qui facilite une meilleure traçabilité, une priorisation efficace et une analyse approfondie au cours du processus de développement du système. Deuxièmement, nous avons introduit des relations pour relier chaque Exigence Variante (VariantFR ou VariantNFR) à son ENV et MON correspondants dans le diagramme de définition des blocs (BDD), ainsi que pour associer chaque ENV au MON approprié dans le BDD à l'aide de la relation REL. De plus, pour chaque Exigence Variante, nous établissons des relations de dépendance positive (PositiveDEP) ou négative (NegativeDEP) avec d'autres exigences, lorsque celles-ci existent. Enfin, nous avons ajouté une relation de contenance entre chaque Exigence Variante et ses Exigences de Sensibilisation (AwReq) ainsi que ses Exigences d'Évolution (EvoReq).

Ces améliorations renforcent significativement les capacités de modélisation de SysML, offrant une approche plus structurée et complète pour l'ingénierie des exigences des systèmes embarqués auto-adaptatifs.

Tableau 4-3 : Les relations introduites dans SysML4SAS.

N°	La relation	Description	Notation
01	Décomposition de type « AND »	Pour indiquer qu'une exigence est satisfaite uniquement lorsque toutes ses sous-exigences sont remplies.	
02	Décomposition de type « OR »	Pour indiquer qu'une exigence est satisfaite lorsque l'une des sous-exigences au moins est remplie.	
03	Dépendance positive	La satisfaction d'une exigence variante renforce ou soutient la satisfaction d'une autre exigence.	«*PositiveDEP* »
04	Dépendance négative	La satisfaction d'une exigence variante entre en conflit ou réduit la faisabilité d'une autre exigence.	«*NegativeDEP* »
05	La relation « MON »	Pour relier une exigence variante à ses facteurs d'incertitude	«*MON* »
06	La relation « ENV »		«*ENV* »
07	La relation « REL »		«*REL* »
08	Relation de contenance	Une relation de contenance entre chaque Exigence Variante (Variant Req) et ses exigences de conscience (AwReq) et d'évolution (EvoReq)	

A. **RS.RM.a1**: Génération de diagramme des exigences et le BDD

- **Le but:** Génération de diagramme des exigences et le diagramme de définition de blocs d'une manière automatique.

Dans cette activité, nous avons utilisé Sirius d'Eclipse, pour visualiser automatiquement les deux diagrammes de notre profil à partir d'une instance de notre métamodèle.

Sirius est un environnement de modélisation graphique permettant de créer des outils de modélisation spécifiques à un domaine en s'appuyant sur un modèle EMF. Ce dernier offre la possibilité de spécifier un éditeur de manière entièrement déclarative, tout en permettant une observation en temps réel des résultats, sans nécessiter la génération de code. Il est développé

il y a quelques années par la société Obeo, en partenariat avec Thales, il se distingue par sa capacité à spécifier des éditeurs graphiques sans recourir à l'écriture de code. Cette approche réduit considérablement la complexité, rendant l'outil plus accessible et plus rapide à déployer par rapport à d'autres solutions. En effet, Sirius représente une avancée majeure dans la construction d'éditeurs de diagrammes, notamment en comparaison avec GMF[1], une alternative antérieure beaucoup plus complexe et peu intuitive, dont la prise en main requiert un investissement considérable en termes de temps et d'expertise.

Dans ce cadre, nous avons intégré Sirius dans une instance d'Eclipse comprenant les plugins générés à partir de notre métamodèle EMF, MM4SAES, afin de créer un exemple concret de notre modèle en utilisant les éditeurs par défaut fournis par la plateforme. Cette approche nous permet non seulement de visualiser les diagrammes de manière fluide et interactive, mais aussi d'explorer les possibilités offertes par Sirius pour simplifier et automatiser le processus de modélisation dans notre domaine spécifique.

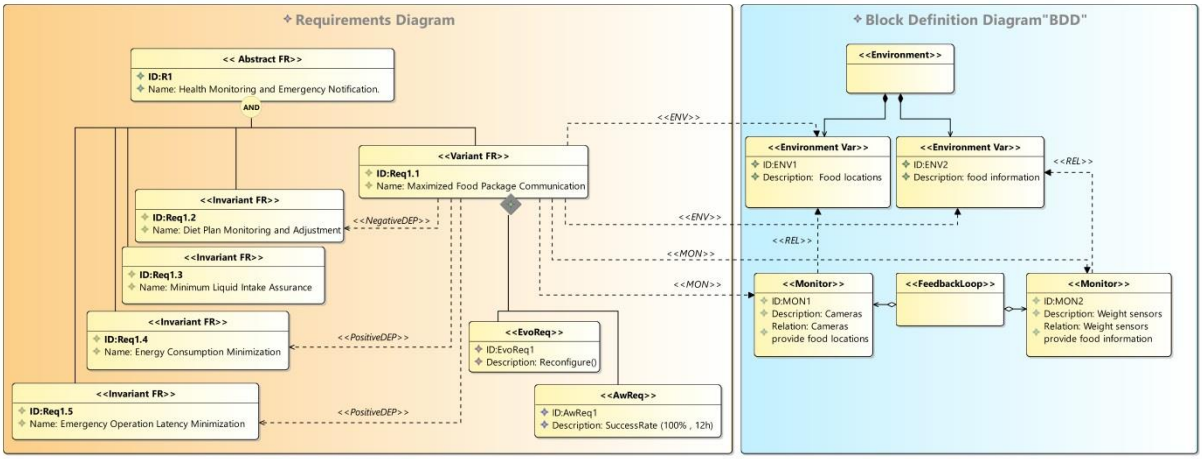


Figure 4-17 : Aperçu du diagramme des exigences et du diagramme de blocs de notre profil.

3. Discussion

Pour lesSEAAs, le développement d'un processus d'ingénierie des exigences dès le début est crucial pour garantir que les spécifications et comportements souhaités sont capturés avec précision. Partir de zéro dans ce type de projet exige une documentation approfondie dès le départ, afin de ne négliger aucun aspect essentiel à la sécurité, à la fiabilité et à l'adaptabilité du système. Ainsi, en démarrant de rien, chaque exigence doit être minutieusement définie et validée pour assurer la robustesse du système dans des conditions dynamiques. Ce processus rigoureux permet de créer une base solide qui garantit une traçabilité optimale des exigences,

[1] Graphical Modeling Framework.

indispensable pour vérifier que chaque adaptation ou mise à jour respecte les contraintes strictes des systèmes embarqués. La traçabilité devient alors un atout majeur, car elle permet de retracer les décisions et ajustements tout au long du développement, contribuant à une validation systématique de la conformité et de la performance du système.

L'approche exhaustive que nous avons adoptée en partant de zéro confère également une certaine autonomie au développement, permettant d'élaborer un système véritablement conçu pour répondre aux besoins spécifiques d'auto-adaptation, sans les contraintes liées à des architectures ou composants hérités. Ainsi, bien qu'un tel processus demande un investissement initial en temps et en effort, il assure que chaque fonctionnalité et exigence soit pleinement adaptée aux objectifs du système final, augmentant sa fiabilité et sa capacité à répondre de manière efficace et sûre aux changements dans l'environnement.

D'une autre part, dans notre processus, les exigences relaxées définissent les marges de flexibilité du système, tandis que les méta-exigences régulent la manière dont ces ajustements peuvent être effectués, garantissant ainsi l'auto-adaptabilité du système tout en respectant des contraintes globales et des objectifs de performance ou de sécurité. C'est pourquoi notre processus peut être considéré comme cohérent : il commence par poser les bases de flexibilité (exigences relaxées) et de rigidité (exigences invariantes), puis utilise les exigences de conscience pour assurer que le système puisse prendre des décisions éclairées sur quand et comment relaxer certaines exigences. Ensuite, les exigences d'évolution sont déterminées pour garantir que le système puisse s'adapter de manière optimale en fonction des conditions détectées. Cette approche systématique permet de maintenir un équilibre entre adaptabilité et robustesse. Ainsi, l'intégration de la validation à la fin de chaque SPA est une approche cruciale pour garantir la qualité et la pertinence des exigences tout au long du processus. Bien que le processus de validation ne soit pas encore détaillé dans notre démarche, le fait de structurer le processus avec des étapes claires pour intégrer la validation à la fine de chaque SPA présente déjà un avantage. Donc, il sera possible d'ajouter les détails techniques de la validation plus tard. Cette démarche offre plusieurs avantages:

1. **Réduction des risques** : En validant chaque SPA avant de passer à la suivante, on s'assure que les exigences sont cohérentes et qu'elles répondent aux attentes des parties prenantes avant d'aller plus loin. Cela réduit le risque de remaniements coûteux ou de retards en cas de malentendus ou de déviations.
2. **Amélioration continue** : La validation itérative permet d'identifier rapidement les erreurs

ou les incohérences. Cela favorise une amélioration continue du modèle d'exigences et une réactivité accrue face aux besoins changeants du projet ou des parties prenantes.

3. **Alignement avec les objectifs du projet** : En validant à la fin de chaque SPA, on garantit que les exigences restent alignées avec les objectifs du projet et les contraintes techniques. Cela renforce la pertinence des exigences vis-à-vis du produit final.
4. **Gain de temps à long terme** : Bien que cette approche demande des efforts de validation supplémentaires tout au long du processus, elle permet d'éviter de grandes révisions en fin de cycle. Ainsi, le temps investi est compensé par la réduction des coûts de correction à un stade avancé.
5. **Engagement des parties prenantes** : En validant les exigences au fur et à mesure, les parties prenantes sont régulièrement impliquées et peuvent donner leur avis à chaque étape. Cela améliore la communication et la satisfaction des parties prenantes, et assure que le produit final répond véritablement à leurs besoins.

4. Conclusion

Dans ce chapitre, nous avons présenté notre processus pour l'ingénierie des exigences des SEAA, fondé sur le modèle MAPE-K. Ce processus comprend 8 domaines principaux, 17 sous-domaines, et 63 actions. Grâce à cette structuration, les ingénieurs de systèmes embarqués peuvent gérer les activités d'élucidation, d'analyse, de spécification, ainsi que de validation et vérification des exigences de manière organisée et efficace, garantissant ainsi l'identification et la documentation complètes de toutes les spécifications nécessaires. Notre processus est soutenu par trois outils complémentaires.

CHAPITRE 5 : Etude de cas

"La patience est un arbre dont la racine est amère, mais dont le fruit est doux"

Proverbe persan

SOMMAIRE :

1. Introduction

Ce chapitre présent une étude de cas visant à démontrer l'application pratique du notre processus d'ingénierie des exigences pour les SEAAs, fondé sur le modèle MAPE-K. À travers cette étude, nous appliquons le processus REP4SAES pour capturer, analyser, modéliser, valider et vérifier les exigences d'un système de santé auto-adaptatif. L'objectif de cette étude est de montrer comment les concepts théoriques présentés dans le chapitre précédent peuvent être appliqués à un cas concret et d'évaluer l'efficacité du processus proposé dans un environnement réel.

Dans un premier temps, nous décrirons le contexte du système étudié ainsi que ses caractéristiques principales. Ensuite, nous présenterons l'application du processus REP4SAES en suivant chaque étape de manière détaillée. Enfin, une conclusion synthétisera les enseignements tirés de cette étude de cas.

2. Le contexte

Le diabète est une maladie chronique qui nécessite une surveillance constante pour prévenir des complications graves, telles que l'hyperglycémie et l'hypoglycémie. Dans ce contexte, un hôpital souhaite développer un système intelligent capable de surveiller en temps réel la glycémie des patients grâce à des capteurs connectés, de détecter les épisodes d'hyperglycémie (taux de glucose élevé) et d'hypoglycémie (taux de glucose bas), d'envoyer des alertes en temps réel aux patients et aux médecins en cas d'anomalies, et d'adapter les recommandations médicales en fonction des données historiques et des tendances de la glycémie. Ce système a pour objectif de fournir des alertes en temps réel aux médecins et aux patients, contribuant ainsi à prévenir les complications graves et à améliorer la qualité des soins.

3. Application du REP4SAES

3.1.BR - Exigences métier

3.1.1. BR.SI - Identification des parties prenantes

A. **BR.SI.a1** : Analyse contextuelle

Après l'analyse contextuelle, la liste des parties prenantes identifiées comprend : *l'architecte système, le développeur logiciel, l'expert en capteurs biomédicaux, le data scientist, le responsable IT, le responsable de l'implémentation du matériel, le médecin endocrinologue, le patient diabétique, l'infirmier(e), le gestionnaire de l'hôpital et le régulateur* représentant l'autorité de santé.

B. **BR.SI.a2**: Définition des rôles

Les parties prenantes identifiées pour le projet et leurs rôles respectifs sont les suivantes : l'architecte système (*STK1*) est chargé de définir l'architecture globale, tandis que le développeur logiciel (*STK2*) s'occupe du développement des applications nécessaires. L'expert en capteurs biomédicaux (*STK3*) est responsable de l'intégration des capteurs, et le data scientist (*STK4*) se concentre sur l'analyse des données collectées. Le responsable IT (*STK5*) assure la gestion de l'infrastructure ainsi que la sécurité, et le responsable de l'implémentation du matériel (*STK6*) supervise l'installation et la maintenance des équipements. Le médecin endocrinologue (*STK7*) joue le rôle de prescripteur, alors que le patient diabétique (*STK8*) est l'utilisateur principal du système. L'infirmier(e) (*STK9*) intervient en tant qu'assistant(e) pour le suivi des patients, le gestionnaire de l'hôpital (*STK10*) prend les décisions stratégiques, et enfin, le régulateur ou l'autorité de santé (*STK11*) s'assure de la conformité aux normes en vigueur.

C. **BR.SI.a3 :** Attribution des responsabilités

Tableau 5-1 : Liste des responsabilités attribuées à chaque partie prenante.

ID	Responsability
STK1	- Concevoir l'architecture globale du système. - Assurer l'intégration des composants matériels et logiciels.

STK2	- Développer les algorithmes d'analyse des données et de détection des anomalies. - Implémenter les fonctionnalités de notification et d'ajustement des paramètres.
STK3	- Sélectionner les capteurs adaptés à la mesure continue de la glycémie. - Assurer leur calibration et leur intégration au système.
STK4	- Analyser les données de glycémie pour identifier des tendances et des corrélations. - Développer des modèles prédictifs pour anticiper les fluctuations glycémiques.
STK5	- Gérer l'infrastructure informatique, y compris les serveurs, le stockage des données et la sécurité des systèmes. - Assurer la disponibilité et la robustesse du système en temps réel.
STK6	- Superviser l'installation et l'intégration des dispositifs matériels. - Garantir leur compatibilité avec l'environnement hospitalier et leur conformité aux normes.
STK7	- Définir les contraintes médicales (seuils, recommandations diététiques, protocoles). - Valider les actions et ajustements proposés par le système.
STK8	- Utiliser les dispositifs et suivre les recommandations personnalisées du système. - Fournir des retours sur l'efficacité et l'utilisabilité du système.
STK9	- Assurer le suivi quotidien des patients et intervenir en cas de besoin. - Participer à la formation des patients à l'utilisation du système.
STK10	- Superviser le déploiement du système à l'échelle institutionnelle. - Garantir que le système répond aux objectifs de qualité des soins et d'efficacité opérationnelle.
STK11	- Assurer la conformité du système aux réglementations en vigueur (sécurité, confidentialité, normes médicales). - Valider les dispositifs et algorithmes avant leur mise en production.

D. **BR.SI.a4** : Catégorisation

Les parties prenantes identifiées se répartissent en deux grandes catégories : **Technical** et **Business**. Dans la catégorie **Technical**, nous retrouvons l'architecte système (STK1), le développeur logiciel (STK2), l'expert en capteurs biomédicaux (STK3), le data scientist (STK4), le responsable IT (STK5), ainsi que le responsable de l'implémentation du matériel (STK6). Dans la catégorie **Business**, nous avons le médecin endocrinologue (STK7), le

patient diabétique (STK8), l'infirmier(e) (STK9), le gestionnaire de l'hôpital (STK10) et le régulateur ou autorité de santé (STK11). Cette distinction permet de mieux structurer les contributions et les attentes de chaque groupe dans le cadre du projet.

E. **BR.SI.a5**: Validation

Pour simplifier notre étude de cas, nous supposons, dans le cadre de cette activité, que la liste des parties prenantes établie lors des activités précédentes est exhaustive. Autrement dit, aucun groupe ou individu essentiel n'a été omis, et tous les rôles ainsi que leurs responsabilités ont été correctement définis.

F. **BR.SI.a6** : Documentation

Dans cette activité, les informations relatives à chaque partie prenante sont documentées en utilisant Eclipse et les plugins EMF générés à partir de notre métamodèle. La première étape consiste à lancer Eclipse et à créer une instance du métamodèle. Ensuite, pour chaque partie prenante, les données suivantes doivent être saisies dans cette instance : *ID, Name, Role, Responsability, Contact et Category.*

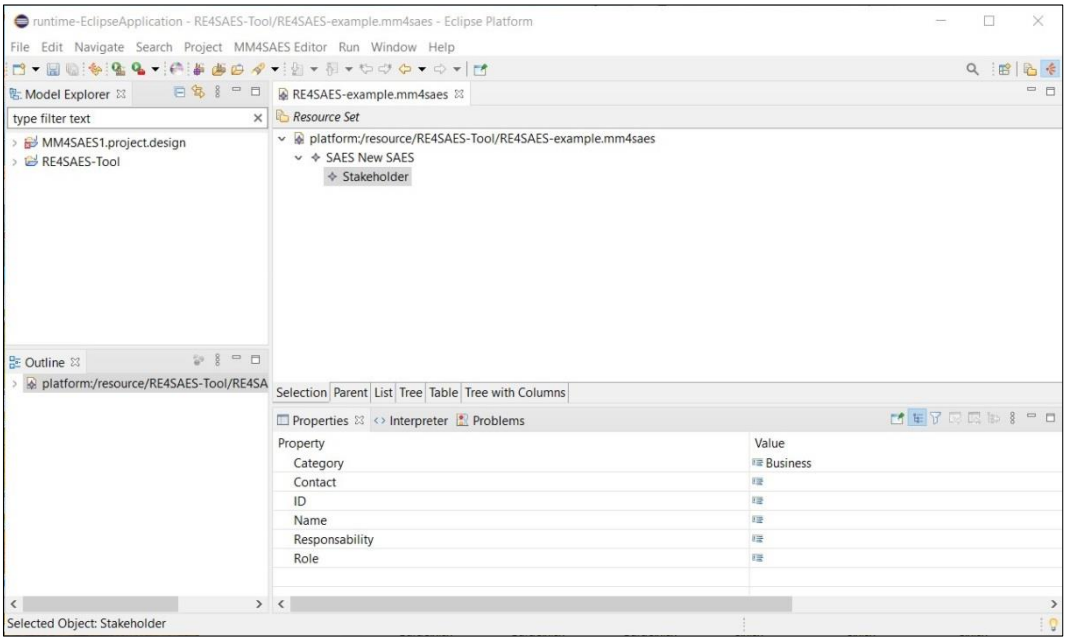


Figure 5-1 : Documentation de la liste des parties prenantes.

3.1.2. BR.IO- Identification des objectifs

A. **BR.IO.a1** : Analyse des Besoins

Pour cette étude de cas, nous avons identifié un ensemble de besoins pour chaque partie prenante :

- Patient diabétique :
  1. **N1** : Le patient souhaite que le système adapte les seuils d'alerte en fonction de son état de santé (par exemple, un seuil plus bas après un repas).
  2. **N2** : Communication vocale avec le système pour les patients malvoyants.
- Médecin endocrinologue :
  1. **N3** : Le médecin souhaite que le système génère automatiquement des recommandations thérapeutiques basées sur l'évolution des tendances glycémiques.
  2. **N4** : Intégration de fonctionnalités prédictives avancées pour d'autres pathologies au-delà du diabète.
- Infirmier(e) :
  1. **N5** : L'infirmier(e) souhaite que le système ajuste automatiquement la fréquence des mesures en cas de détection d'une anomalie.
- Gestionnaire de l'hôpital :
  1. **N6** : Le gestionnaire souhaite que le système optimise les ressources, notamment en réduisant les faux positifs qui mobilisent inutilement le personnel médical.

Cependant, pour simplifier l'étude, nous nous concentrons uniquement sur les besoins du patient (*N1 et N2*).

B. **BR.IO.a2** : Définition des objectifs stratégiques

Dans cette activité, afin de simplifier l'étude de cas, nous avons dérivé un objectif stratégique à partir de chaque besoin. Par conséquent, la liste des objectifs stratégiques dérivés des besoins identifiés est la suivante :

- **G1** : Ajuster les seuils d'alerte de manière dynamique.
- **G2** : Assurer une communication intuitive, accessible et sans dépendance à des interfaces visuelles.

C. **BR.IO.a3** : Décomposition des Objectifs

- **SG.1.1** : Utiliser les données historiques pour détecter les tendances (la détection des tendances aide à établir des seuils personnalisés qui reflètent les besoins spécifiques du patient à partir de son historique médical).
- **SG.1.2** : Détecter les changements dans l'état de santé en temps réel (la détection en temps réel permet d'ajuster les seuils et d'envoyer des alertes appropriées selon l'état actuel du patient, garantissant une prise en charge rapide et adaptée).
- **SG.2.1** : Reconnaissance et compréhension des commandes vocales.
- **SG.2.2 :** Retour vocal adapté et clair.

D. **BR.IO.a4** : Validation des objectifs/Sous_Objectifs

Dans cette activité, il est essentiel de valider la liste des objectifs et des sous-objectifs avec les parties prenantes afin de garantir qu'ils reflètent fidèlement leurs attentes et priorités. Dans notre étude de cas, nous avons limité notre travail à deux besoins spécifiques. Par conséquent, nous pouvons considérer que la liste des objectifs et des sous-objectifs est valide, compte tenu de la simplicité de l'étude de cas.

E. BR.IO.a5 : Documentation

Dans cette activité, les informations suivantes doivent être documentées pour chaque besoin, objectif ou sous-objectif via notre instance du métamodèle : un *ID* et une *Description* contenant les détails du besoin, de l'objectif ou du sous-objectif. Par ailleurs, pour chaque besoin, il est essentiel d'identifier la partie prenante correspondante à partir de la liste des parties prenantes afin de garantir l'origine de chaque besoin (voir la figure suivante).

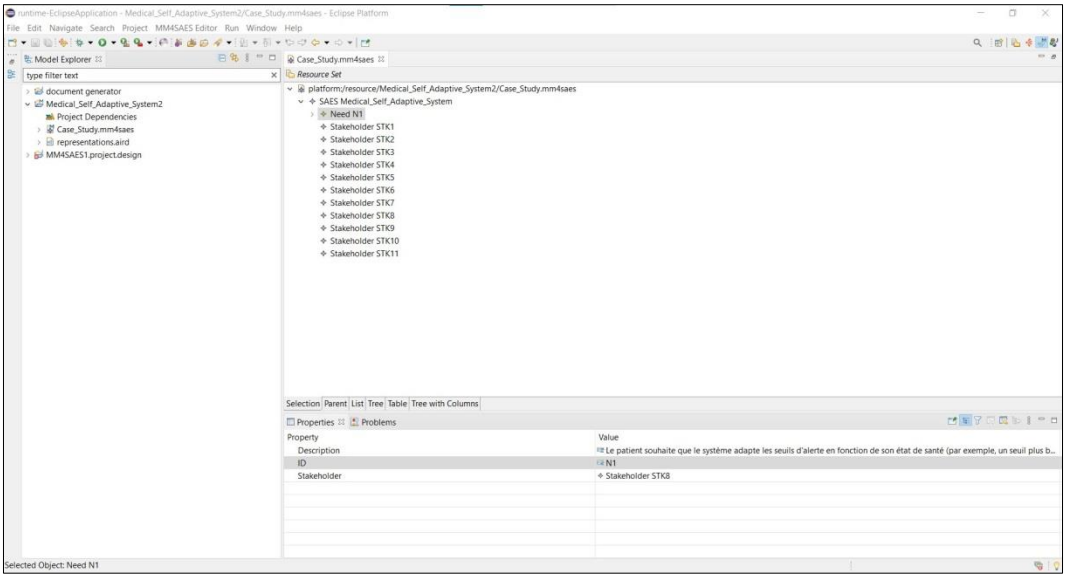


Figure 5-2 : Documentation de la liste des besoins.

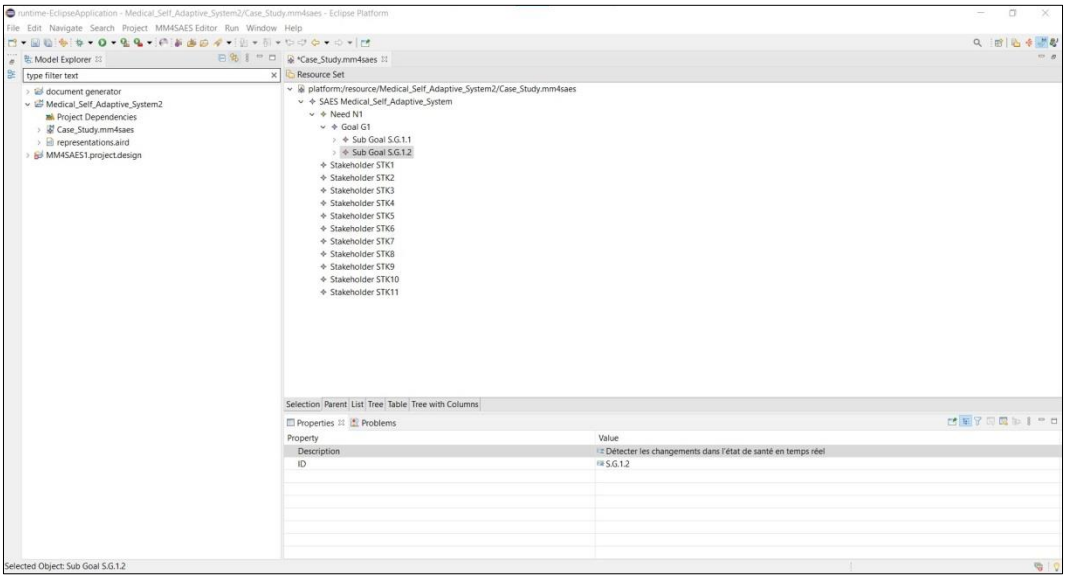


Figure 5-3 : Documentation de la liste des objectifs /sous-objectifs.

F. **BR.IO.a6**: Vérification de la traçabilité

Dans cette activité, il est nécessaire de vérifier notre instance afin de déterminer si des omissions sont présentes ou non. En cas d'omission, il faut compléter les objectifs ou sous-objectifs manquants. Par exemple, dans cette étude de cas, supposons que nous avons omis de documenter l'objectif G2 et les sous-objectifs SG.1.1 et SG.1.2. Dans ce cas, l'outil affiche le message d'erreur suivant (voir la figure ci-dessous).

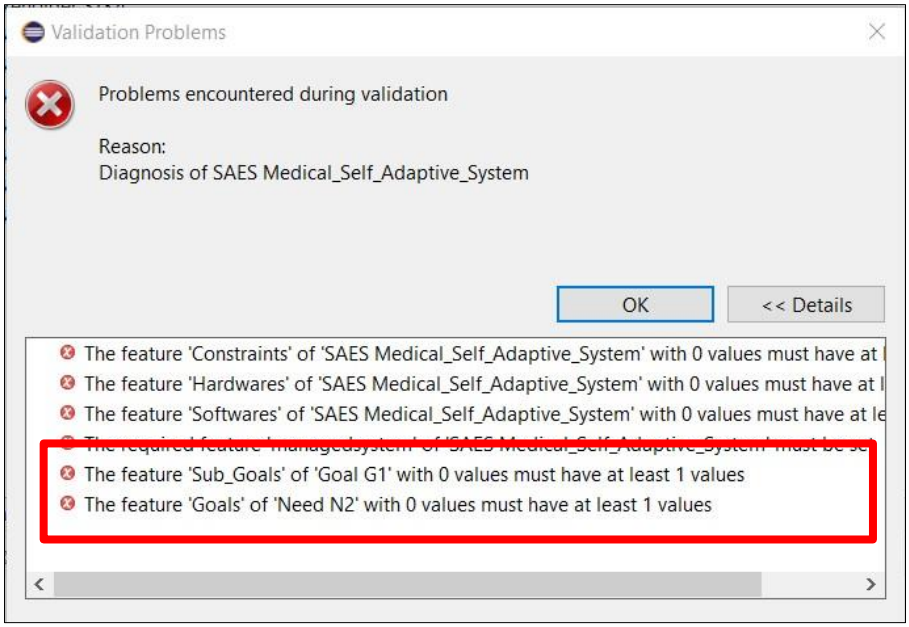


Figure 5-4 : Vérification de la traçabilité.

3.1.3. BR.IC - Identification des Contraintes

A. **BR.IC.a1:** Identification des contraintes du domaine

Pour simplifier notre étude de cas, nous faisons l'hypothèse qu'il n'existe aucune contrainte spécifique au domaine.

B. **BR.IC.a2**: Identification des contraintes environnementales

Pour simplifier notre étude de cas, nous faisons l'hypothèse qu'il n'existe aucune contrainte environnementale.

C. **BR.IC.a3**: Identification des contraintes normatives

Pour simplifier notre étude de cas, nous faisons l'hypothèse qu'il n'existe aucune contrainte normative.

D. **BR.IC.a4:** Évaluation des ressources humaines et financières

Dans cette activité, le gestionnaire de l'hôpital (*STK10*), responsable de la gestion financière et des budgets, a identifié la contrainte suivante afin de garantir la viabilité économique du projet tout en respectant les ressources disponibles : le coût de développement et d'exploitation du système ne doit pas dépasser 5 000 000,00 DA.

E. **BR.IC.a5**: Validation avec les parties prenantes

Durant cette activité, en raison des contraintes budgétaires strictes, il a été décidé, d'exclure le besoin suivant de la version actuelle du système :

- *N2* : Assurer une communication intuitive, accessible et sans dépendance à des interfaces visuelles.

Ce besoin pourrait être pris en compte comme une évolution dans une version future du système.

F. **BR.IC.a6**: Documentation

Dans cette activité, nos plugins EMF ont été utilisés pour renseigner, pour chaque contrainte, les informations suivantes : *ID, description et source*. La source a été sélectionnée parmi les options suivantes : *Domain, Environment, Standards and Regulations, Human Resources ou bien Budget*. Par ailleurs, chaque contrainte a été associée à la partie prenante correspondante. En outre, les informations relatives à B2 ont été supprimées, ce qui a entraîné la suppression de toutes les informations associées, notamment celles concernant G2, SG2.1 et SG2.2.

3.2.HSR - Exigences matériel/logiciel

3.2.1. HSR.HWR - Exigences matériel

A. **HSR.HWR.a1**: Définition des dispositifs matériels (hardware)

Durant cette activité, un dispositif d'entrée a été identifié : **le capteur de glycémie en continu**, un dispositif médical portable permettant de mesurer et de surveiller en temps réel les niveaux de glucose dans le sang. Principalement destiné aux patients diabétiques, il aide à gérer leur état de santé et à ajuster leurs traitements, notamment les doses d'insuline.

B. HSR.HWR.a2: Évaluation des contraintes physiques

Dans cette activité, voici la liste des contraintes physiques pour le capteur :

1. **Précision de mesure** : Le capteur doit être suffisamment précis pour détecter les variations de glycémie dans des limites médicales.
2. **Durée de vie de la batterie** : La batterie du capteur doit durer un certain nombre de jours

(par exemple, 7 jours) avant de devoir être remplacée.

3. **Taille et ergonomie** : Le capteur doit être suffisamment petit et confortable pour être porté par le patient en continu, sans gêner ses activités quotidiennes.

C. **HSR.HWR.a3**: Évaluation des compromis

L'évaluation des compromis pour le capteur de glycémie en continu se concentre sur trois contraintes principales : la précision de mesure, la durée de vie de la batterie et la taille/ergonomie. Ces contraintes sont souvent interdépendantes et nécessitent une analyse approfondie pour trouver un équilibre optimal. Par exemple, améliorer la précision peut exiger une fréquence d'échantillonnage élevée ou des capteurs plus sophistiqués, ce qui augmente la consommation énergétique et réduit l'autonomie. De même, réduire la taille pour améliorer l'ergonomie peut limiter la capacité d'intégrer une batterie durable, compromettant ainsi la durée de vie. Donc, Afin de concilier ces exigences, il est crucial de hiérarchiser les priorités. Un compromis optimal peut être trouvé en utilisant des technologies de capteurs à faible consommation énergétique tout en intégrant des batteries rechargeables dans un design compact et ergonomique. Cette approche garantit un dispositif fiable, confortable et aligné sur les contraintes techniques et budgétaires.

D. **HSR.HWR.a4**: Validation des contraintes

Dans cette activité, nous partons du principe que la liste des contraintes matérielles est considérée comme valide.

E. **HSR.HWR.a5**: Documentation

Pour chaque dispositif, nous avons saisi les informations suivantes*: ID, Name, Characteristics*; et pour chaque contrainte : *ID, Name et Description*.

3.2.2. HSR.SWR: Exigences logicielles

Pour simplifier notre étude de cas, nous avons choisi de ne pas inclure de contraintes logicielles et de nous concentrer exclusivement sur les contraintes matérielles.

3.3.SR – Exigences du système

3.3.1. SR.FR - Exigences fonctionnelles

A. **SR.FR.a1** : Formuler les Exigences Fonctionnelles

Dans cette activité, à partir de la liste des sous-objectifs et des différentes contraintes, nous avons extrait la liste des exigences fonctionnelles suivante:

Tableau 5-2 : Liste des exigences fonctionnelles.

ID	Nom	Description	Source
FR1	Collecte des données historiques	Le système doit collecter les données historiques de glycémie des patients sur une période donnée.	SG1.1
FR2	Analyse des tendances glycémique	Le système doit analyser les tendances des données de glycémie afin de détecter des motifs récurrents comme l'hypoglycémie nocturne.	SG1.1
FR3	Génération d'alertes de tendances	Le système doit générer des alertes lorsqu'une tendance anormale est détectée.	SG1.1
FR4	Détection des anomalies en temps réel	Le système doit détecter automatiquement toute fluctuation anormale de la glycémie (hyperglycémie, hypoglycémie).	SG1.2
FR5	Distinction des anomalies liées au diabète	Le système doit être capable de distinguer les anomalies liées au diabète des fluctuations normales.	SG1.2

B. **SR.FR.a2** : Validation

Dans cette activité, il est nécessaire de valider la liste des exigences fonctionnelles afin de s'assurer que chaque exigence identifiée répond aux besoins et contraintes du nouveau système, tout en garantissant sa conformité avec les attentes des parties prenantes. Nous supposons que la liste des exigences est valide.

C. **SR.FR.a3**: Documentation

Pour chaque exigence fonctionnelle, nous saisissons les informations suivantes *: ID, Name et Description.*

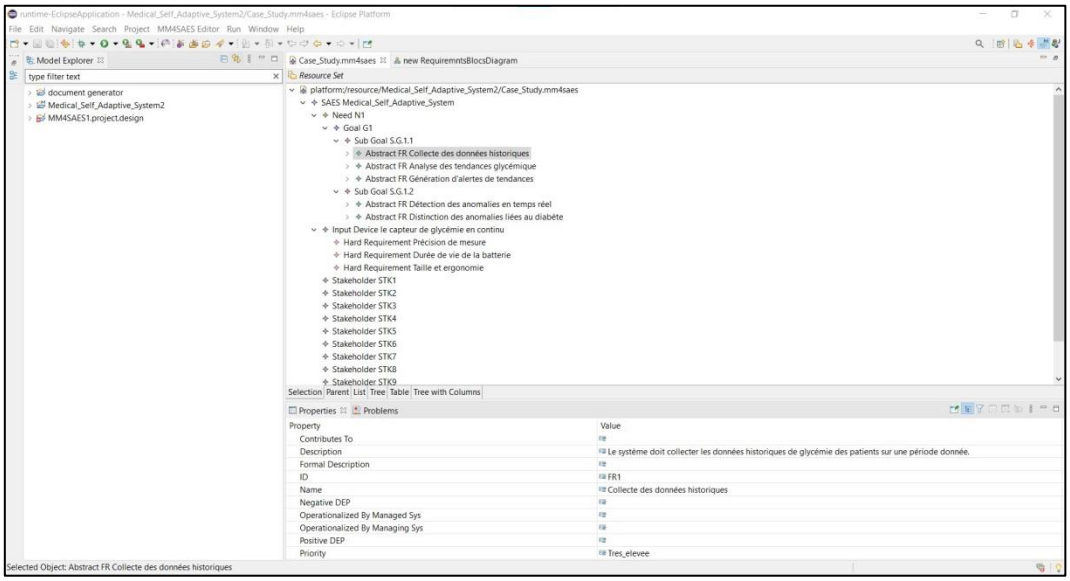


Figure 5-5 : Documentation de la liste des exigences fonctionnelles.

3.3.2. SR .NFR - Exigences Non-Fonctionnelles

A. **SR.NFR.a1**: Formuler les Exigences Non-Fonctionnelles

Dans cette activité, à partir de la liste des sous-objectifs et des différentes contraintes, nous avons extrait la liste des exigences non fonctionnelles suivante:

Tableau 5-3 : Liste des exigences non fonctionnelles.

ID	Nom	Description	Source
NFR1	Stockage sécurisé des données historiques	Les données historiques doivent être conservées de manière sécurisée.	SG1.1
NFR2	Réactivité en temps réel pour la détection	Le système doit être capable de détecter les anomalies en temps réel.	SG1.2
NFR3	Fiabilité de la détection	Le système doit être fiable (un taux d'erreur minimal dans la détection des anomalies).	SG1.2

NFR4	Taille et ergonomie	Le capteur doit être conçu de manière compacte (taille et poids réduits) pour être porté confortablement, permettant une utilisation continue sans gêner les activités quotidiennes du patient.	Contrainte materielle « capteur »
NFR5	Précision de mesure	Le capteur doit mesurer les niveaux de glycémie avec précision, permettant la détection des variations pertinentes pour la santé du patient.	Contrainte materielle « capteur »
NFR6	Durée de vie de la batterie	Le capteur doit fonctionner sans interruption pendant au moins 7 jours avant que la batterie ne nécessite un remplacement ou une recharge.	Contrainte materielle « capteur »

B. **SR.NFR.a2** : Catégorisation

Les exigences non fonctionnelles du système peuvent être classifiées selon plusieurs types spécifiques. Tout d'abord, l'exigence *"Les données historiques doivent être conservées de manière sécurisée"* relève du type **Privacy**, car elle concerne la protection des données personnelles et sensibles. L'exigence *"Le système doit être capable de détecter les anomalies en temps réel"* est classée sous **Performance**, car elle se rapporte à la capacité du système à fonctionner efficacement et rapidement, en réagissant en temps réel aux anomalies. L'exigence *"Le système doit être fiable (un taux d'erreur minimal dans la détection des anomalies)"* concerne la **Reliability**, soulignant l'importance d'un fonctionnement stable et d'un faible taux d'erreur dans la détection des anomalies. L'exigence *"Le capteur doit être conçu de manière compacte (taille et poids réduits) pour être porté confortablement, permettant une utilisation continue sans gêner les activités quotidiennes du patient"* relève du type **Usability**, mettant l'accent sur la facilité d'utilisation et le confort d'usage du capteur. L'exigence *"Le capteur doit mesurer les niveaux de glycémie avec précision, permettant la détection des variations pertinentes pour la santé du patient"* est également classée sous **Performance**, car elle concerne la précision des mesures effectuées par le capteur. Enfin,

l'exigence *"Le capteur doit fonctionner sans interruption pendant au moins 7 jours avant que la batterie ne nécessite un remplacement ou une recharge"* touche à la **Portability**, car elle spécifie l'autonomie du capteur et sa capacité à être utilisé pendant une période prolongée sans nécessiter de maintenance

C. **SR.NFR.a3** : Validation

Dans cette activité, il est nécessaire de valider avec les parties prenantes la liste des exigences non fonctionnelles, afin de s'assurer que le système satisfait des critères de qualité et de performance qui ne sont pas directement liés à ses fonctionnalités spécifiques, mais qui sont néanmoins essentiels à son succès. Nous supposons que la liste des exigences non fonctionnelles est valide.

D. **SR.NFR.a4**: Documentation

Pour chaque exigence non fonctionnelle, nous avons saisi les informations suivantes : *ID, Name, Description et Category*, qui indique le type précis de l'exigence non fonctionnelle.

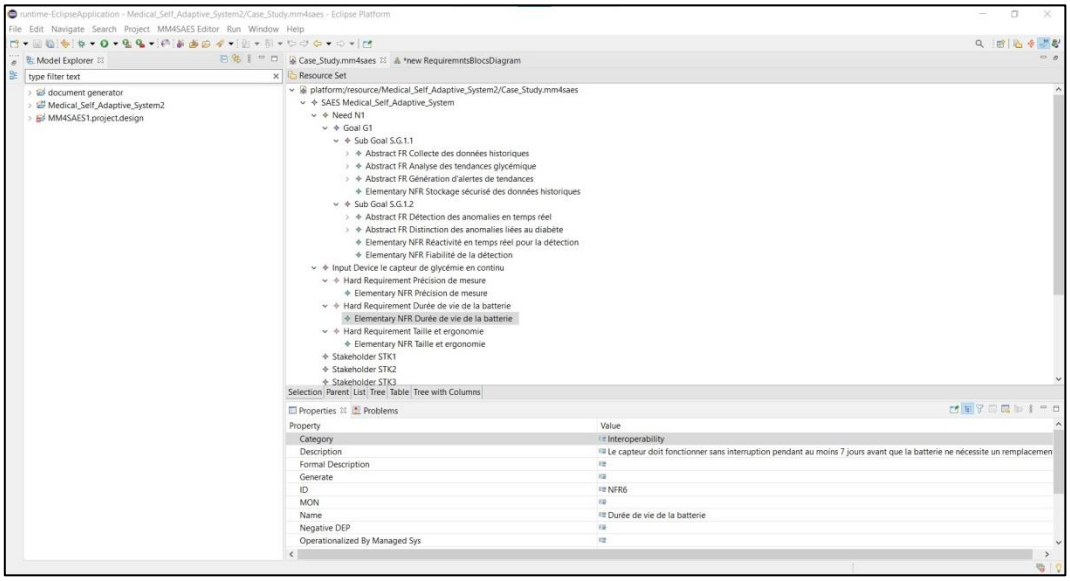


Figure 5-6 : Documentation de la liste des exigences non fonctionnelles.

3.3.3. SR.CV - Vérification de la couverture « Coverage Verification »

A. **SR.FR.a1 :** Vérification de la complétude

Durant cette activité, il est essentiel de s'assurer que chaque exigence soit correctement associée à un sous-objectif ou à une contrainte spécifique, et que chaque sous-objectif soit

entièrement couvert par une ou plusieurs exigences. Dans le cadre de notre étude, toutes ces vérifications ont été effectuées.

3.4.RA - Analyse des exigences

3.4.1. RA.HARR - Analyse hiérarchique et raffinement des exigences

A. **RA. HARR.a1**: Décomposition des exigences

Dans cette activité, nous avons décomposé chaque exigence fonctionnelle ou non fonctionnelle en exigences élémentaires, reliées par des relations logiques AND et OR.

Tableau 5-4 : Décomposition des exigences en exigences élémentaires.

Exg	Rel	ID	Nom	Description
FR1	AND	FR1.1	Collecte Historique Donnees	Le système doit collecter les données historiques de glycémie des patients.
		FR1.2	Specificaion Period Collecte	Le système doit permettre de spécifier la période de collecte.
FR2	AND	FR2.1	Analyse Tendances Donnees	Le système doit analyser les tendances des données de glycémie.
		FR2.2	Detection Patrons Recurrent	Le système doit détecter des motifs récurrents, comme l'hypoglycémie nocturne.
FR3	-	-	-	-
FR4	FR4.1 AND (FR4.2.1 OR FR4.2.2)	FR4.1	Surveillance Anomalies En_Temps_Reel	Le système doit surveiller en temps réel les variations de glycémie.
		FR4.2.1	Detection Hyperglycemie	Le système doit détecter automatiquement des fluctuations anormales, comme l'hyperglycémie.

		FR4.2.2	Detection Hypoglycemie	Le système doit détecter automatiquement des fluctuations anormales, comme l'hypoglycémie.
FR5	FR5.1 AND FR5.2	FR5.1	Identification Anomalies Diabete	Le système doit identifier les anomalies spécifiquement liées au diabète.
		FR5.2	Ignorer Fluctuations Normales	Le système doit ignorer les fluctuations normales non liées à des pathologies.

Concernant les exigences non fonctionnelles, et dans un souci de simplification de l'étude de cas, nous les avons considérées comme des exigences élémentaires.

B. **RA. HARR.a2**: Affinement des exigences fonctionnelles/non fonctionnelles

Dans cette activité, et afin d'ajouter des détails quantitatifs aux exigences pour améliorer la précision des spécifications et leur faisabilité technique, nous avons affiné l'exigence *NFR2*, initialement formulée comme suit : « *Le système doit être capable de détecter les anomalies en temps réel*» La version affinée est désormais : *NFR2-2*, Détection des anomalies avec délai de réponse ≤ 1 minute « *Le système doit détecter les anomalies en temps réel avec un délai de réponse inférieur à un seuil prédéfini, fixé à une minute ou moins*».

C. **RA.HARR.a3**: Écrire chaque exigence sous la forme d'une déclaration SHALL

Dans cette activité, nous rédigeons la description de chaque exigence élémentaire sous la forme d'une déclaration « *SHALL* » :

Tableau 5-5 : Description informelle et structurée de chaque exigence.

ID	Description informelle	Description structurée
FR1.1	Le système doit collecter les données historiques de glycémie des patients.	The system SHALL collect historical blood glucose data from patients.
FR1.2	Le système doit permettre de spécifier la période de collecte.	The system SHALL allow the specification of the collection period.
FR2.1	Le système doit analyser les tendances des données de glycémie.	The system SHALL analyze trends in blood glucose data.

FR2.2	Le système doit détecter des motifs récurrents, comme l'hypoglycémie nocturne.	The system SHALL detect recurring patterns, such as nocturnal hypoglycemia.
FR3	Anomaly Trend Alert Generation	The system SHALL generate alerts when an abnormal trend is detected.
FR4.1	Le système doit surveiller en temps réel les variations de glycémie.	The system SHALL monitor blood glucose variations in real-time.
FR4.2.1	Le système doit détecter automatiquement des fluctuations anormales, comme l'hyperglycémie.	The system SHALL automatically detect abnormal fluctuations, such as hyperglycemia.
FR4.2.2	Le système doit détecter automatiquement des fluctuations anormales, comme l'hypoglycémie.	The system SHALL automatically detect abnormal fluctuations, such as hypoglycemia.
FR5.1	Le système doit identifier les anomalies spécifiquement liées au diabète.	The system SHALL identify anomalies specifically related to diabetes.
FR5.2	Le système doit ignorer les fluctuations normales non liées à des pathologies.	The system SHALL ignore normal fluctuations unrelated to pathological conditions.
NFR1	Les données historiques doivent être conservées de manière sécurisée.	The system SHALL securely store historical data.
NFR2	Le système doit être capable de détecter les anomalies en temps réel.	The system SHALL detect anomalies in real time.
NFR2 – affiné	Le système doit détecter les anomalies en temps réel avec un délai de réponse inférieur à un seuil prédéfini, fixé à une minute ou moins	The system SHALL detect anomalies in real time with a response time below a predefined threshold, set to one minute or less.
NFR3	Le système doit être fiable (un taux d'erreur minimal dans la détection des anomalies).	The system SHALL maintain reliability, ensuring a minimal error rate in anomaly detection.

NFR4	Le capteur doit être conçu de manière compacte (taille et poids réduits) pour être porté confortablement, permettant une utilisation continue sans gêner les activités quotidiennes du patient.	The sensor SHALL be designed in a compact form (reduced size and weight).
NFR5	Le capteur doit mesurer les niveaux de glycémie avec précision, permettant la détection des variations pertinentes pour la santé du patient.	The sensor SHALL accurately measure glucose levels.
NFR6	Le capteur doit fonctionner sans interruption pendant au moins 7 jours avant que la batterie ne nécessite un remplacement ou une recharge.	The sensor SHALL operate continuously for at least 7 days before requiring battery replacement or recharging.

D. **RA.HARR.a4**: Documentation

Pour chaque exigence fondamentale, nous avons saisi les informations suivantes : *ID, Nom et Description*. Toutes les exigences sont considérées comme invariantes. Ainsi, pour le champ type d'exigence, il convient de laisser « Invariant » comme option par défaut.

3.4.2. RA.INF- Impact des exigences non fonctionnelles sur les exigences fonctionnelles

A. **RA.INF.a1**: Analyse approfondie des exigences non fonctionnelles

Après une analyse approfondie de la liste des exigences non fonctionnelles, nous pouvons déduire un ensemble d'exigences fonctionnelles à partir de ces exigences non fonctionnelles. Par exemple, à partir de l'exigence non fonctionnelle NFR6, relative à la durée de vie de la batterie, nous pouvons déduire : « Le système doit notifier l'utilisateur à l'avance lorsque le niveau de batterie atteint un seuil critique, afin d'éviter une interruption du fonctionnement ».

Cependant, pour simplifier l'étude de cas, nous n'avons pas effectué cette déduction des exigences fonctionnelles. Il suffit de travailler avec la liste définie précédemment, sans ajouter de nouvelles exigences fonctionnelles spécifiques.

3.4.3. RA.CIR: Identification et résolution des conflits

A. **RA.CIR.a1** : Identification des conflits

Dans cette activité, nous avons examiné chaque exigence afin de déterminer si elle entre en conflit avec d'autres exigences. Un conflit peut survenir lorsque deux exigences imposent des résultats opposés ou lorsque leur mise en œuvre engendre des complications techniques ou fonctionnelles. Voici la liste des conflits potentiels:

1. Conflits potentiels concernant les périodes de collecte de données et la détection en temps réel: "*The system SHALL collect historical blood glucose data from patients*." et "*The system SHALL monitor blood glucose variations in real-time*." : Pas de conflit direct, mais cela pourrait créer des complications fonctionnelles si les deux opérations (collecte historique et surveillance en temps réel) doivent être effectuées simultanément. Si les ressources du système sont limitées, il pourrait être difficile de gérer ces deux processus efficacement sans interférences.
2. Conflits potentiels liés aux objectifs de détection des tendances et à la spécification de la période de collecte : "*The system SHALL analyze trends in blood glucose data*." et "*The system SHALL allow the specification of the collection period.*" : Pas de conflit direct, mais cela pourrait poser des défis techniques si les périodes de collecte de données spécifiées ne sont pas compatibles avec les algorithmes d'analyse des tendances ou si des périodes de collecte trop courtes ou irrégulières compliquent la détection fiable des tendances.
3. Conflits potentiels entre la gestion des données et la sécurité : "*The system SHALL securely store historical data.*" et "*The system SHALL automatically detect abnormal fluctuations, such as hyperglycemia*." : Pas de conflit direct, mais il pourrait y avoir des complications fonctionnelles si la sécurité des données historiques nécessite un traitement qui ralentit ou empêche la détection en temps réel des anomalies.
4. Conflits potentiels concernant les performances et la détection des anomalies : "*The system SHALL maintain reliability, ensuring a minimal error rate in anomaly detection*." et "*The system SHALL detect anomalies in real time with a response time below a predefined threshold, set to one minute or less*." : L'exigence de fiabilité et de taux d'erreur minimal pourrait entrer en conflit avec l'exigence d'un temps de réponse strict (une minute ou moins), car une détection rapide peut nécessiter des compromis sur la précision et la

fiabilité dans des scénarios complexes.

5. Conflits potentiels entre la durée de vie de la batterie du capteur et son fonctionnement continu : "*The sensor SHALL operate continuously for at least 7 days before requiring battery replacement or recharging*." et "*The sensor SHALL be designed in a compact form (reduced size and weight).*" : Concevoir un capteur compact tout en garantissant une autonomie de 7 jours pourrait être techniquement difficile, car la compacité pourrait limiter la capacité de la batterie à maintenir cette durée de fonctionnement continue.

B. **RA.CIR.a2**: Négociation

Dans cette activité, les parties prenantes engagent des discussions et des négociations afin d'atteindre un compromis acceptable pour tous.

C. **RA.CIR.a3**: Priorisation

Dans cette activité, nous avons utilisé la technique MoSCoW (Must have, Should have, Could have, Won't have) afin de classer les exigences par ordre de priorité.

- **La liste des exigences essentielles (Must Have):**

1. *FR1.1*: The system SHALL collect historical blood glucose data from patients.
2. *FR4.1*: The system SHALL monitor blood glucose variations in real-time.
3. *NFR2*: The system SHALL detect anomalies in real time with a response time below a predefined threshold, set to one minute or less.
4. *NFR5:* The sensor SHALL accurately measure glucose levels.
5. *NFR6*: The sensor SHALL operate continuously for at least 7 days before requiring battery replacement or recharging.

- **La liste des exigences s importantes, mais non essentielles (Should Have)**

1. *FR2.1*: The system SHALL analyze trends in blood glucose data.
2. *FR4.2.1*: The system SHALL automatically detect abnormal fluctuations, such as hyperglycemia.
3. *FR4.2.2*: The system SHALL automatically detect abnormal fluctuations, such as hypoglycemia.
4. *FR3*: The system SHALL generate alerts when an abnormal trend is detected.

5. *NFR1*: The system SHALL securely store historical data.
6. *NFR4*: The sensor SHALL be designed in a compact form (reduced size and weight).

- **La liste des exigences souhaitables, mais pas critiques (Could Have)**

1. *FR1.2*: The system SHALL allow the specification of the collection period.
2. *FR2.2*: The system SHALL detect recurring patterns, such as nocturnal hypoglycemia.
3. *FR5.1*: The system SHALL identify anomalies specifically related to diabetes.
4. *FR5.2:* The system SHALL ignore normal fluctuations unrelated to pathological conditions.
5. *NFR3*: The system SHALL maintain reliability, ensuring a minimal error rate in anomaly detection.

- **La liste des exigences non nécessaires (Won't Have):** aucune exigence n'est complètement exclue.

D. **RA.CIR.a3**: Validation et Consensus

Dans cette activité, il est essentiel d'obtenir l'accord de toutes les parties prenantes sur les classifications. Des réunions et des discussions peuvent être nécessaires pour parvenir à un consensus. Nous partons du principe que les classifications ont été validées.

E. **RA.CIR.a4**: Documentation

Dans cette activité, pour chaque exigence fonctionnelle ou non fonctionnelle fondamentale, nous avons sélectionné la valeur de la propriété « Priority » parmi les catégories suivantes: *Must_Have, Should_Have, Could_Have et Won't_Have.*

3.5.RR –Exigences assouplies

3.5.1. RR.RP: Le processus RELAX_ES

A. **RR.RP.a1:** Filtrage des exigences

- **Exigences éliminées** : Aucune exigence n'a été éliminée, car la liste des exigences non nécessaires est vide.
- **Exigences invariant** : Ces exigences sont essentielles au fonctionnement de base du

système.

1. *FR1.1*: The system SHALL collect historical blood glucose data from patients.
2. *FR4.1*: The system SHALL monitor blood glucose variations in real-time.
3. *NFR2:* The system SHALL detect anomalies in real time with a response time below a predefined threshold, set to one minute or less.
4. *NFR5*: The sensor SHALL accurately measure glucose levels.
5. *NFR6*: The sensor SHALL operate continuously for at least 7 days before requiring battery replacement or recharging.

B. **RR.RP.a2:** Identifier les facteurs d'incertitude

1. Identification des facteurs d'incertitude du FR1.2 :

Dimension	Description
ENV	Période temporelle définie par l'utilisateur pour la collecte des données.
MON	Commandes ou interfaces permettant de configurer la période de collecte.
REL	Les configurations MON influencent directement la période d'observation (ENV).
DEP	Dépendance négative avec : FR4.1"*The system SHALL monitor blood glucose variations in real-time*". Si une période trop courte est spécifiée, cela peut compromettre la surveillance en temps réel.

2. Identification des facteurs d'incertitude du FR2.1 :

Dimension	Description
ENV	Évolution des niveaux de glucose dans le sang (tendances et motifs récurrents).
MON	Données de glucose mesurées sur une période donnée.
REL	Détection de motifs récurrents dans les tendances du glucose.
DEP	- Dépendance négative avec : NFR1 "*The system SHALL securely store historical data*". Si les données historiques sont altérées ou incomplètes, cela compromettra l'analyse des tendances. - Dépend positivement de FR2.2 et FR3 (détection des motifs et alertes facilitent l'analyse des tendances)

3. Identification des facteurs d'incertitude du FR2.2 :

Dimension	Description
ENV	Évolution des niveaux de glucose dans le sang (tendances et motifs récurrents).
MON	Données de glucose mesurées sur une période donnée.
REL	Détection de motifs récurrents dans les tendances du glucose.
DEP	Dépendance négative avec : FR5.2 "*The system SHALL ignore normal fluctuations unrelated to pathological conditions*". En ignorant certaines variations, des motifs récurrents significatifs pourraient ne pas être détectés.

4. Identification des facteurs d'incertitude du FR3 :

Dimension	Description
ENV	Fluctuations anormales du glucose (hyperglycémie, hypoglycémie)
MON	Seuils anormaux détectés dans les variations du glucose.
REL	Détection de fluctuations anormales déclenchant des alertes.
DEP	Dépendance négative avec : NFR1 "*The system SHALL securely store historical data*". Si les anomalies ne sont pas enregistrées correctement, cela pourrait compliquer les alertes futures.

5. Identification des facteurs d'incertitude du FR4.2.1/FR4.2.2 :

Dimension	Description
ENV	Fluctuations anormales du glucose (hyperglycémie, hypoglycémie).
MON	Seuils anormaux détectés dans les variations du glucose.
REL	Détection de fluctuations anormales déclenchant des alertes.
DEP	Dépendance négative avec : NFR1 "*The system SHALL securely store historical data*". Si l'enregistrement des fluctuations anormales est inadéquat, cela peut compromettre les futures analyses ou alarmes.

6. Identification des facteurs d'incertitude du FR5.1 :

Dimension	Description
ENV	Anomalies spécifiques au diabète.
MON	Comportements atypiques des niveaux de glucose.
REL	Identification des anomalies en lien avec des fluctuations anormales.

DEP	Dépendance négative avec : FR5.2 "*The system SHALL ignore normal fluctuations unrelated to pathological conditions*". En excluant des fluctuations normales, des anomalies spécifiques au diabète pourraient être manquées.

7. Identification des facteurs d'incertitude du FR5.2 :

Dimension	Description
ENV	Fluctuations normales des niveaux de glucose, influencées par des facteurs physiologiques (repas, stress, exercice).
MON	Mesures continues de glucose fournies par les capteurs en temps réel.
REL	Les propriétés de l'environnement (ENV) sont filtrées à partir des données mesurées (MON) pour identifier les fluctuations normales.
DEP	- Dependance positive avec : NFR1 « *The system SHALL securely store historical data* ». En éliminant les fluctuations normales, les données stockées seront plus pertinentes. - Dependance négative avec : FR2.2 « *The system SHALL detect recurring patterns, such as nocturnal hypoglycemia* ». Si certaines fluctuations normales sont ignorées, des modèles récurrents pertinents risquent de ne pas être détectés. - Dependance négative avec : FR1.1 « *The system SHALL collect historical blood glucose data from patients* ». En excluant certaines données, l'historique peut devenir incomplet, compromettant certaines analyses.

8. Identification des facteurs d'incertitude du NFR1 :

Dimension	Description
ENV	Infrastructure de stockage sécurisée permettant de conserver les données historiques de glucose (bases de données ou serveurs).
MON	Mécanismes pour vérifier l'intégrité, la disponibilité et la sécurité des données stockées (e.g., journaux, hash).
REL	Les données environnementales (ENV) sont directement dérivées des données mesurées par les capteurs et collectées via MON.

DEP	- Dependance négative avec : FR4.1 « *The system SHALL monitor blood glucose variations in real-time* ». Si les ressources système sont trop concentrées sur le stockage sécurisé, cela peut affecter la surveillance en temps réel. - Dependance négative avec : FR4.2.1 « *The system SHALL automatically detect abnormal fluctuations, such as hyperglycemia* ». Si le système privilégie uniquement la sécurisation des données, les alertes en temps réel peuvent être compromises.

9. Identification des facteurs d'incertitude du NFR3 :

Dimension	Description
ENV	Contexte opérationnel où des anomalies dans les données de glucose doivent être détectées avec une précision élevée. Les incertitudes peuvent inclure des erreurs de mesure ou des bruits environnementaux.
MON	Données issues des capteurs, vérifications automatiques des seuils d'erreur, et tests périodiques du système pour garantir la précision.
REL	Les propriétés environnementales (ENV) liées à la fiabilité sont évaluées à partir des données observables et mesurées (MON).
DEP	- Dependance positive avec : FR2.1 « *The system SHALL analyze trends in blood glucose data* ». Une fiabilité accrue garantit que les tendances identifiées sont exactes. - Dependance positive avec : FR3 « *The system SHALL generate alerts when an abnormal trend is detected* ». Une fiabilité élevée réduit les fausses alertes et améliore la réponse aux anomalies. - Dependance négative avec : FR4.1 « *The system SHALL monitor blood glucose variations in real-time* ». Un focus excessif sur la minimisation des erreurs pourrait ralentir la capacité de réponse en temps réel. - Dependance negative avec : FR1.2 « *The system SHALL allow the specification of the collection period* ». Une période de collecte mal configurée peut réduire la fiabilité globale, car des données insuffisantes ou excessives pourraient affecter l'algorithme de détection.

10. Identification des facteurs d'incertitude du NFR4 :

Dimension	Description
ENV	Dimensions physiques du capteur, y compris sa taille, son poids et ses contraintes ergonomiques liées à son intégration chez l'utilisateur.
MON	Caractéristiques mesurables du capteur : dimensions physiques (mm, g) et tests de conformité aux spécifications de design
REL	Les propriétés physiques (ENV) sont directement observables via des mesures spécifiques (MON).
DEP	- Dependance positive avec : FR4.1 « *The system SHALL monitor blood glucose variations in real-time* ». Un capteur compact et portable favorise une surveillance continue et sans interruption. - Dependance positive avec : NFR6 « *The sensor SHALL operate continuously for at least 7 days before requiring battery replacement or recharging* ». Un design compact peut optimiser la consommation énergétique et améliorer l'autonomie. - Dependance négative avec : NFR5 « *The sensor SHALL accurately measure glucose levels* ». Un capteur plus petit pourrait réduire la précision des mesures en raison de contraintes matérielles. - Dependance négative avec : NFR3 *«The system SHALL maintain reliability, ensuring a minimal error rate in anomaly detection* ». La miniaturisation peut introduire des limitations techniques, augmentant potentiellement le taux d'erreur.

C. **RR.RP.a3:** Analyser les facteurs d'incertitude

1. **Analyser ENV :** Les incertitudes environnementales peuvent affecter la satisfaction de plusieurs exigences du système. Les exigences liées à la collecte et au traitement des données, comme la collecte des données historiques (*FR1.1*) et l'analyse des tendances (*FR2.1*), sont influencées par des interruptions de communication, des données bruitées ou incomplètes. La détection des anomalies (*FR4.2.1 et FR4.2.2*) et des motifs récurrents (*FR2.2*) est compliquée par des variations naturelles ou des interférences dans les capteurs. Les exigences de surveillance en temps réel (*FR4.1*) et de fiabilité (*NFR3*) sont également sujettes à des retards dans le traitement ou des erreurs dues à des variations imprévues dans l'environnement. Cependant, certaines exigences, comme le stockage sécurisé des données (*NFR1*) ou le design compact du capteur (*NFR4*), sont moins

sensibles aux incertitudes environnementales, bien qu'elles puissent être affectées par des limitations techniques ou des contraintes de conception. Ainsi, à cette étape, nous avons décidé de classer NFR1 et NFR4 en tant qu'exigences invariantes.

2. **Analyser DEP :** *FR1.2, FR5.2* et *NFR3* possèdent des dépendances négatives avec des exigences de priorité élevée. Par conséquent, il est nécessaire de classer ces exigences comme des exigences invariantes.
3. **Analyser MON :** L'assouplissement des exigences *FR2.1, FR4.2.1, FR4.2.2, FR3, FR2.2 et FR5.1* est possible parceque le monitoring fournit des données de manière continue, précise, et réactive.

D. **RR.RP.a4:** Introduire le(s) opérateur(s) RELAX

ID	le(s) opérateur(s) RELAX
FR2.1	EVENTUALLY
FR2.2	EVENTUALLY
FR3	UNTIL
FR4.2.1	AS CLOSE AS POSSIBLE TO [frequency]
FR4.2.2	AS CLOSE AS POSSIBLE TO [frequency]
FR5.1	MAY ... OR

E. **RR.RP.a5:** Rédiger chaque exigence assouplie selon la grammaire RELAX

ID	La grammaire RELAX
FR2.1	EVENTUALLY, The system SHALL analyze trends in blood glucose data.
FR2.2	EVENTUALLY, The system SHALL detect recurring patterns, such as nocturnal hypoglycemia.
FR3	The system SHALL generate alerts when an abnormal trend is detected UNTIL a specific threshold [t] is reached.
FR4.2.1	The system SHALL detect abnormal fluctuations AS CLOSE AS POSSIBLE TO a frequency [f], such as hyperglycemia.
FR4.2.2	The system SHALL detect abnormal fluctuations AS CLOSE AS POSSIBLE TO a frequency [f], such as hypoglycemia.
FR5.1	The system MAY identify anomalies specifically related to diabetes OR through alternative clinical patterns.

F. **RR.RP.a6:** Documentation

Dans cette activité, nous avons mis à jour à la fois la description et le type des exigences assouplies. Le type de ces exigences a été modifié, passant de Invariant à Variant. Par ailleurs, les informations relatives à chaque variable d'environnement ont été enregistrées, incluant leur identifiant (ID) et leur description. Nous avons également actualisé la liste des dispositifs de surveillance et de contrôle des variables environnementales, en précisant pour chacun l'ID, la description, ainsi que la relation définissant le lien entre le dispositif de surveillance (Monitor) et la variable d'environnement correspondante. Enfin, nous avons ajouté les relations de dépendance, qu'elles soient positives ou négatives, existant entre les exigences assouplies et les autres exigences.

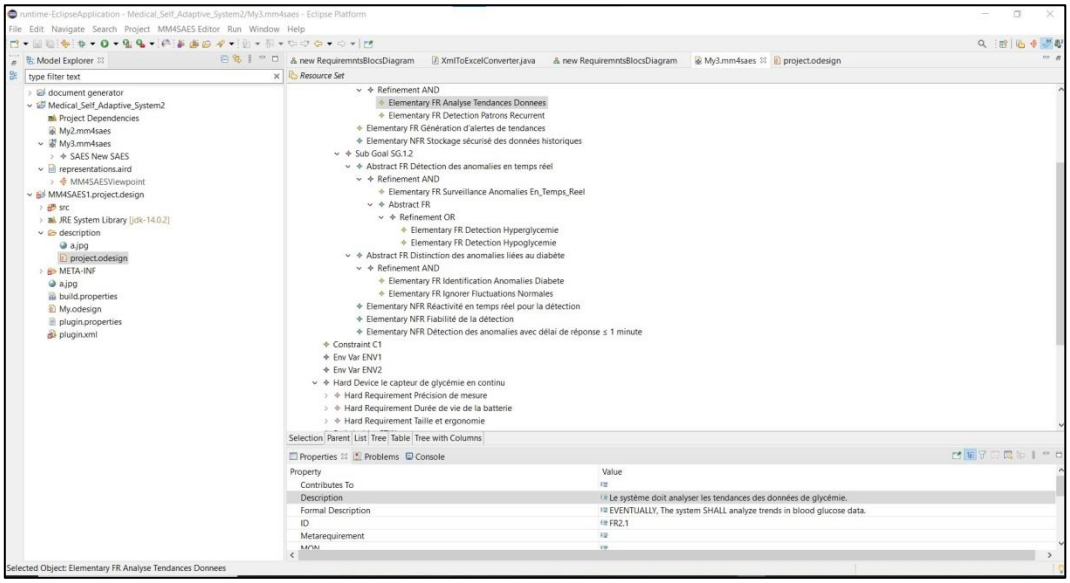


Figure 5-7 : documentation des informations relatives aux exigences variant.

3.6.MR - Méta- Exigences

3.6.1. MR.AwReq: Exigences de conscience

A. **MR.AwReq.a1** : Détermination des Exigences NeverFail « Regular Requirement »

Dans cette activité, nous avons associé chaque exigence invariante à une exigence de conscience de type Regular. Ainsi, chaque exigence 'R' a été associée au modèle NeverFail(R).

B. **MR.AwReq.a2** : Association des exigences assouplies aux exigences de conscience

NFR_ID	ID	Type	Modèle
FR2.1	AwReq1	Aggregate	SuccessRate(AwReq1, 80 %)
FR2.2	AwReq2	Aggregate	SuccessRate(AwReq2, 80 %)
FR3	AwReq3	Aggregate	SuccessRate(AwReq3, 80 %)
FR4.2.1	AwReq4	Aggregate	SuccessRate(AwReq4, 90 %)
FR4.2.2	AwReq5	Aggregate	SuccessRate(AwReq5, 90 %)
FR5.1	AwReq6	Aggregate	SuccessRate(AwReq6, 75 %)

C. **MR.AwReq.3**: Documentation

Dans cette activité, pour chaque exigence de conscience, nous avons saisi les informations suivantes : *ID, Description et TypeAWReq*.

3.6.2. MR.EvoReq : Evolution requirements

A. **MR.EvoReq.a1** : Identification des exigences d'evolution

Tableau 5-6 : Association de chaque exigence de conscience à une exigence d'évolution.

ID	Exigence de conscience	Exigence d'evolution
EvoReq1	AwReq1	Reconfigure()
EvoReq2	AwReq2	Reconfigure()
EvoReq3	AwReq3	Reconfigure()
EvoReq4	AwReq4	Reconfigure()
EvoReq5	AwReq5	Reconfigure()
EvoReq6	AwReq6	Reconfigure()

B. **MR.EvoReq.a3** : Documentation

Dans cette activité, pour chaque exigence d'evolution, nous avons saisi les informations suivantes : *ID etDescription*.

3.7.FRFL - Exigences fonctionnelles de la boucle de rétroaction

3.7.1. FRFL.FRFL : Exigences fonctionnelles de la boucle de rétroaction

A. **FRFL.FRFL.a1** : Identification des exigences du comportement fonctionnel de la boucle de rétroaction

Tableau 5-7 : Exigences fonctionnelles de la boucle de rétroaction.

Composant	ID	Nom	Description informelle	Description sous forme « SHALL »
Analyze	FR7	Analyse en temps réel	Le système doit analyser les données en temps réel, avec un délai maximal de 30 secondes entre la collecte et l'analyse.	The system SHALL analyze the blood glucose data in real-time, with a maximum delay of 30 seconds between data collection and analysis.
Plan	FR8	Prise de décision rapide	Le système doit générer des décisions (alertes et recommandations) en moins de 60 secondes après l'analyse des données.	The system SHALL generate decisions within 10 seconds after analyzing the blood glucose data.
Execution	FR9	Exécution immédiate	Le système doit exécuter les actions recommandées (alerte, ajustement des recommandations médicales) immédiatement après la prise de décision.	The system SHALL execute the recommended actions immediately after decision-making.

B. **FRFL.FRFL.a2** : Décomposition et affinement des exigences

Toutes les exigences étant élémentaires, il n'est pas nécessaire de les décomposer ou de les affiner. Il suffit donc de les documenter en enregistrant, pour chacune, *l'ID, le nom, et une description*, à la fois informelle et formulée sous forme de SHALL.

3.8. RS – Spécification des exigences

3.8.1. RS.Doc : Documentation

A. **RS.Doc.a1 :** Vérification du système

Dans cette activité, nous avons vérifié notre système.

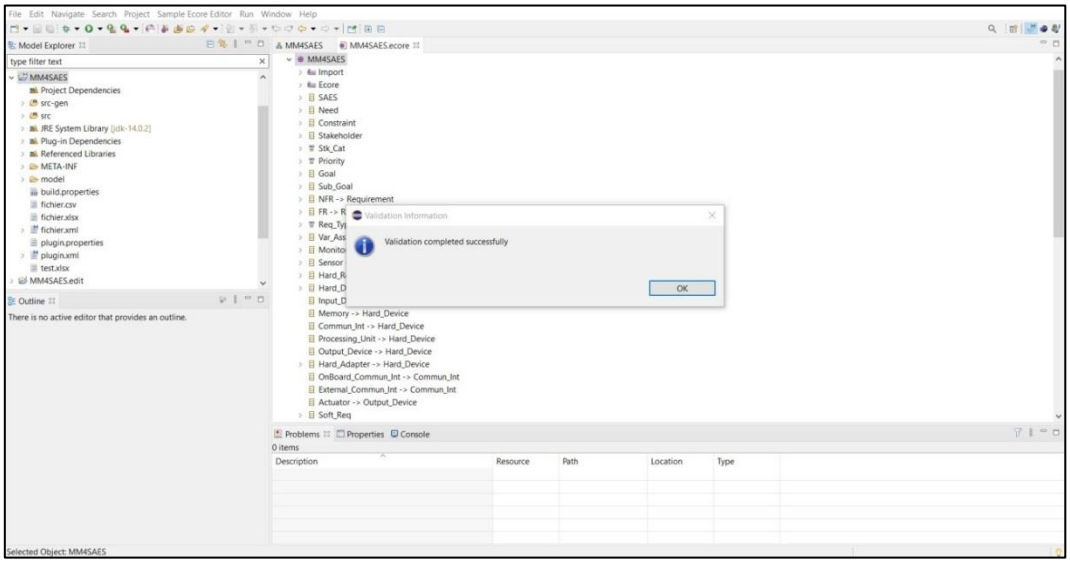


Figure 5-8 : Vérification du système.

B. **RS.Doc.a2**: Générer le document des exigences utilisateur

Dans cette activité, nous avons généré le document des exigences utilisateur.

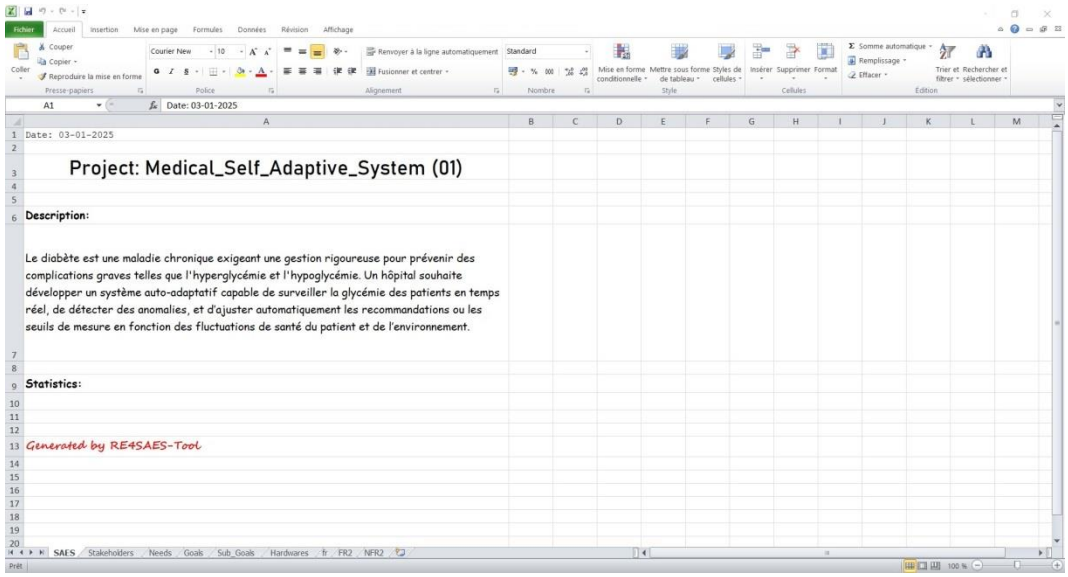
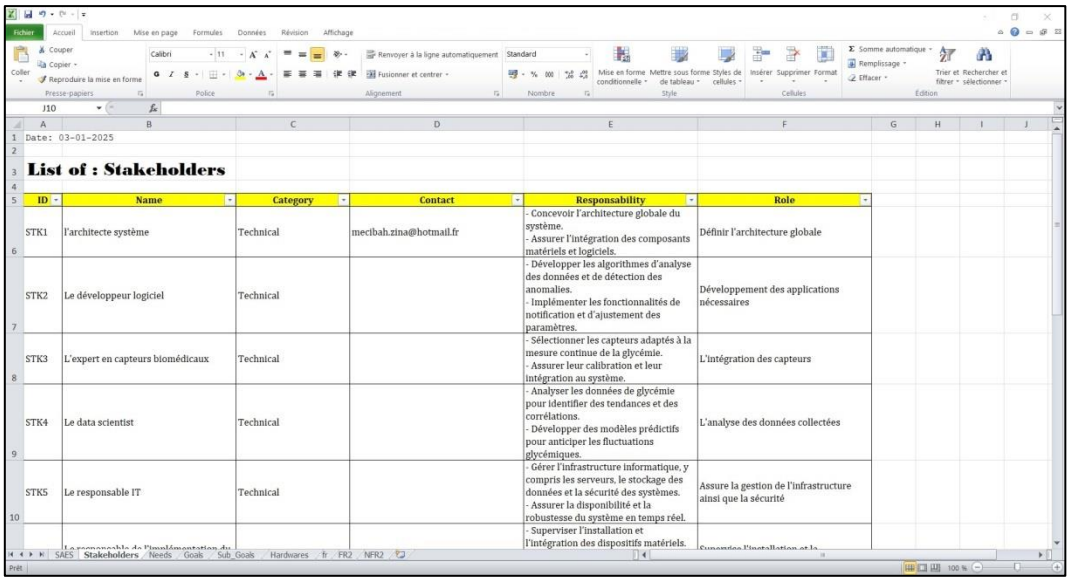


Figure 5-9 : Page de garde du document d'exigence utilisateur.

Pour l'affichage des statistiques, nous avons utilisé les Derived Features (caractéristiques dérivées) d'EMF. Ces propriétés ne stockent pas leurs valeurs directement dans le modèle, mais les calculent dynamiquement à partir d'autres données du modèle. Cela permet de produire des informations dérivées sans duplication ni mise à jour manuelle, ce qui s'avère particulièrement utile pour la génération de statistiques fiables et maintenables.



Date: 03-01-2025

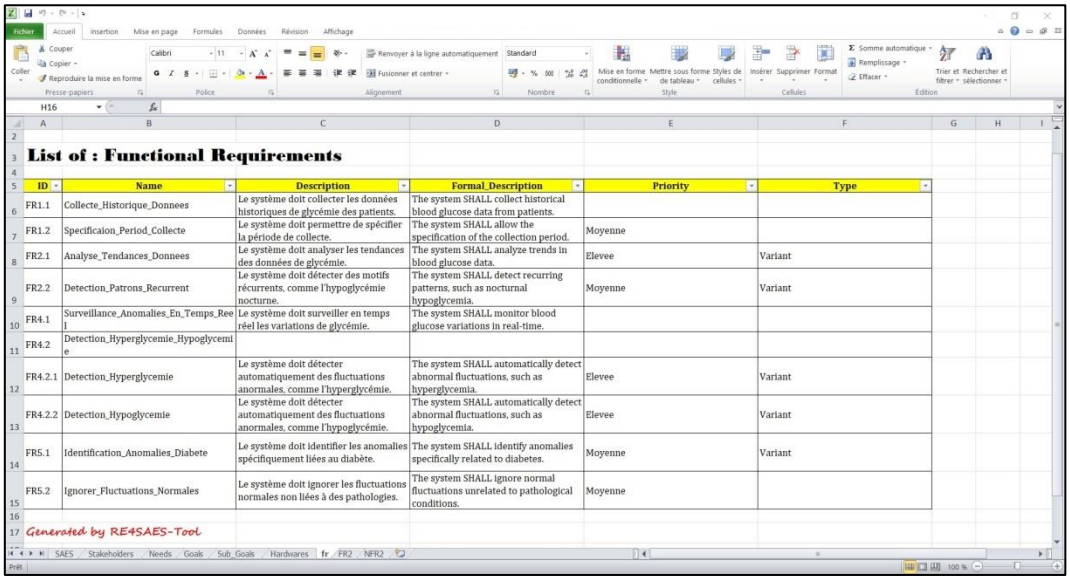
List of : Stakeholders

ID	Name	Category	Contact	Responsability	Role
STK1	l'architecte système	Technical	mecibah.zina@hotmail.fr	- Concevoir l'architecture globale du système. - Assurer l'intégration des composants matériels et logiciels.	Définir l'architecture globale
STK2	Le développeur logiciel	Technical		- Développer les algorithmes d'analyse des données et de détection des anomalies. - Implémenter les fonctionnalités de notification et d'ajustement des paramètres.	Développement des applications nécessaires
STK3	L'expert en capteurs biomédicaux	Technical		- Sélectionner les capteurs adaptés à la mesure continue de la glycémie. - Assurer leur calibration et leur intégration au système.	L'intégration des capteurs
STK4	Le data scientist	Technical		- Analyser les données de glycémie pour identifier des tendances et des corrélations. - Développer des modèles prédictifs pour anticiper les fluctuations glycémiques.	L'analyse des données collectées
STK5	Le responsable IT	Technical		- Gérer l'infrastructure informatique, y compris les serveurs, le stockage des données et la sécurité des systèmes. - Assurer la disponibilité et la robustesse du système en temps réel.	Assure la gestion de l'infrastructure ainsi que la sécurité
				- Superviser l'installation et l'intégration des dispositifs matériels.	

Figure 5-10 : Document des exigences utilisateur.

C. **RS.Doc.a2**: Générer le diagramme des exigences système

Dans cette activité, nous avons généré le document des exigences système.



List of : Functional Requirements

ID	Name	Description	Formal_Description	Priority	Type
FR1.1	Collecte_Historique_Donnees	Le système doit collecter les données historiques de glycémie des patients.	The system SHALL collect historical blood glucose data from patients.		
FR1.2	Specificaion_Period_Collecte	Le système doit permettre de spécifier la période de collecte.	The system SHALL allow the specification of the collection period.	Moyenne	
FR2.1	Analyse_Tendances_Donnees	Le système doit analyser les tendances des données de glycémie.	The system SHALL analyze trends in blood glucose data.	Elevee	Variant
FR2.2	Detection_Patrons_Recurrent	Le système doit détecter des motifs récurrents, comme l'hypoglycémie nocturne.	The system SHALL detect recurring patterns, such as nocturnal hypoglycemia.	Moyenne	Variant
FR4.1	Surveillance_Anomalies_En_Temps_Reel	Le système doit surveiller en temps réel les variations de glycémie.	The system SHALL monitor blood glucose variations in real-time.		
FR4.2	Detection_Hyperglycemie_Hypoglycemie				
FR4.2.1	Detection_Hyperglycemie	Le système doit détecter automatiquement des fluctuations anormales, comme l'hyperglycémie.	The system SHALL automatically detect abnormal fluctuations, such as hyperglycemia.	Elevee	Variant
FR4.2.2	Detection_Hypoglycemie	Le système doit détecter automatiquement des fluctuations anormales, comme l'hypoglycémie.	The system SHALL automatically detect abnormal fluctuations, such as hypoglycemia.	Elevee	Variant
FR5.1	Identification_Anomalies_Diabete	Le système doit identifier les anomalies spécifiquement liées au diabète.	The system SHALL identify anomalies specifically related to diabetes.	Moyenne	Variant
FR5.2	Ignorer_Fluctuations_Normales	Le système doit ignorer les fluctuations normales non liées à des pathologies.	The system SHALL ignore normal fluctuations unrelated to pathological conditions.	Moyenne	

Generated by RE4SAES-Tool

Figure 5-11 : Document des exigences système.

3.8.2. RS.RM: Modélisation des exigences

A. **RS.RM.a1 :** Génération de diagramme des exigences et le BDD

Dans cette activité, nous avons généré le diagramme des exigences et le diagramme de définition de bloc à l'aide de notre outil qui prend en charge notre profil SysML (SysML4SAS).

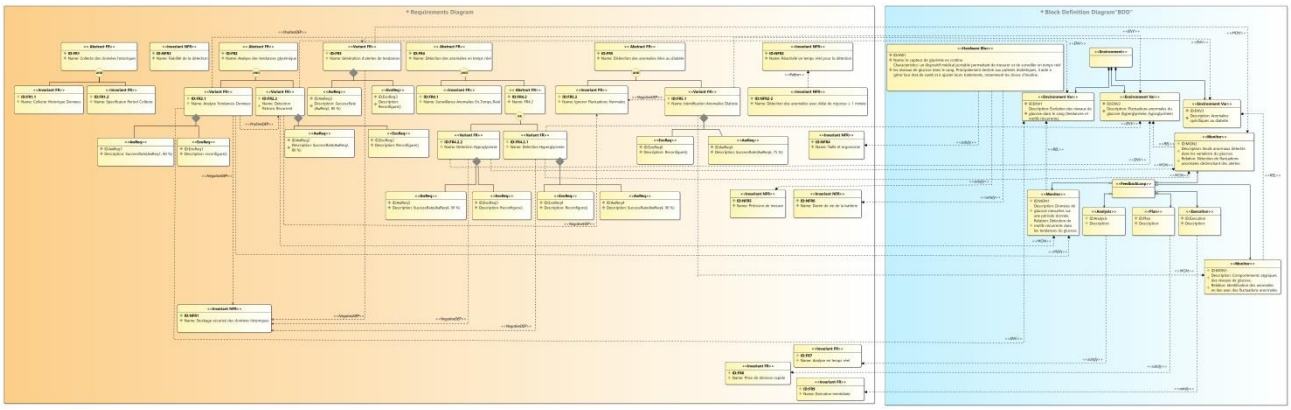


Figure 5-12 : Génération de diagrammes à l'aide notre outil.

4. Conclusion

Ce chapitre a présenté l'application de notre processus d'ingénierie des exigences des Systèmes Embarqué Auto-Adaptatifs à travers une étude de cas tirée du domaine de la santé, mettant en lumière les spécificités et les défis de la gestion des exigences dans un contexte où la précision et la fiabilité sont cruciales.

L'application de notre processus d'ingénierie des exigences (REP4SAES) et de nos outils, a permis de démontrer les différentes étapes du processus. Cela inclut la collecte des exigences, leur analyse, leur documentation et enfin leur modélisation. Nous espérons dans l'avenir proche trouver un terrain d'expérience réel pour notre processus. Un tel terrain nous permettrait de valider notre approche de manière pratique, en confrontant notre processus à des situations réelles et en observant son efficacité dans des conditions concrètes et variées. Cette validation pratique est cruciale pour affiner et améliorer notre méthodologie, et pour assurer qu'elle réponde de manière optimale aux besoins des utilisateurs finaux des SEAAs.

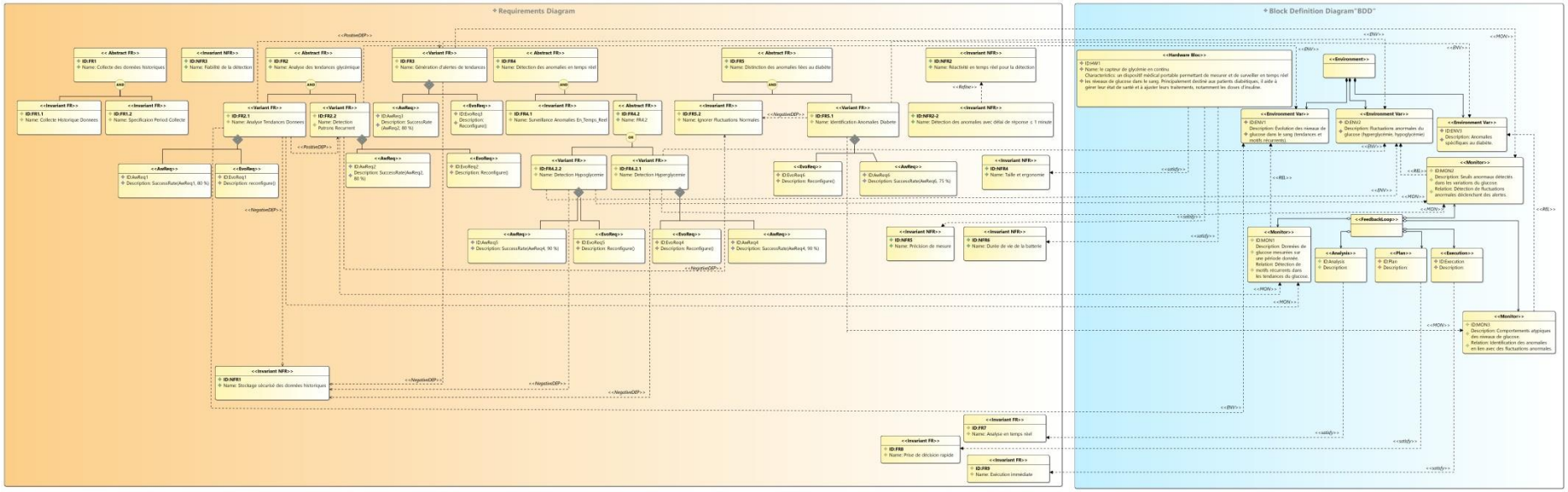


Figure 5-13 : Génération de diagrammes à l'aide notre outil.

REMARQUE : Pour améliorer la lisibilité de l'image, il suffit de la copier et de la coller dans Paint.

CONCLUSION GENERALE ET PERSPECTIVES

"La plus grande gloire n'est pas de ne jamais tomber, mais de se relever à chaque chute"

Confucius

Sommaire :

1. Synthèse

Dans cette thèse, nous avons entamé un domaine de recherche d'actualité qui est l'ingénierie des exigences des systèmes embarqués auto-adaptatifs, basée sur le modèle MAPE-K. À travers le développement d'un métamodèle nommé MM4SAES, nous avons identifié et formalisé les concepts et les relations essentielles à considérer, tout en proposant un processus détaillé pour le développement des exigences. Ce processus aborde trois aspects clés de l'ingénierie des exigences pour les systèmes auto-adaptatifs : la mitigation des incertitudes grâce à des exigences flexibles, la modélisation des exigences pour le comportement adaptatif (objectifs d'adaptation) en tant que méta-exigences (exigences concernant les exigences régulières du système), et l'examen des exigences relatives au comportement fonctionnel des boucles de rétroaction. Ce processus inclut des étapes dédiées à la collecte, à l'analyse, à la spécification et à la validation des exigences, tout en intégrant une suite d'outils permettant de rationaliser et de simplifier le flux de travail de l'ingénierie des exigences.

- Dans le premier chapitre, nous avons présenté les concepts relatifs à la conception des SEs et des SAAs, en mettant en avant les défis et les opportunités liés à leur conception.
- Dans le deuxième chapitre, nous avons exploré l'ingénierie des exigences des systèmes embarqués auto-adaptatifs, en discutant des approches existantes et de leurs limites face aux défis spécifiques des systèmes auto-adaptatifs.
- Au troisième chapitre, nous avons proposé un métamodèle nommé MM4SAES (Metamodel for Self-Adaptive Embedded Systems), qui formalise les concepts essentiels et les relations nécessaires pour décrire les exigences des systèmes embarqués auto-adaptatifs.
- Dans le quatrième chapitre, nous avons présenté un nouveau processus d'ingénierie des exigences pour les systèmes embarqués auto-adaptatifs, nommé REP4SAES (Requirement Engineering Process for Self-Adaptive Embedded Systems). Ce processus a été enrichi par l'introduction d'un nouveau profil SysML conçu pour faciliter la modélisation des exigences, ainsi que par le développement d'une suite d'outils pour soutenir l'application pratique du processus.
- Enfin, dans le dernier chapitre, nous avons validé notre approche à travers une étude de cas concrète, portant sur un système de surveillance intelligent pour les patients atteints de diabète.

Les résultats obtenus confirment que l'approche proposée permet de répondre aux besoins complexes des systèmes embarqués auto-adaptatifs. En alliant un métamodèle structurant, un processus détaillé et des outils adaptés, notre travail contribue à améliorer significativement l'ingénierie des exigences dans ce domaine. Cette thèse ouvre également des perspectives de recherche pour enrichir davantage le métamodèle et étendre l'application de notre processus à d'autres domaines critiques.

2. Perspectives

Notre travail contribue à la compréhension et à l'amélioration de l'ingénierie des exigences pour les systèmes embarqués auto-adaptatifs, tout en ouvrant la voie à des recherches futures prometteuses dans ce domaine dynamique et crucial. Les travaux présentés dans cette thèse ouvrent ainsi plusieurs perspectives de recherche intéressantes.

À court et à moyen terme, nous prévoyons également :

1. **Études empiriques sur la mise en œuvre** : Des études empiriques sont nécessaires pour évaluer l'impact du processus proposé sur la qualité des systèmes développés et sur la satisfaction des utilisateurs finaux.
2. **Extension du métamodèle MM4SAES** : À l'avenir, il serait pertinent d'élargir notre métamodèle pour inclure un domaine spécifique, car les recherches sur les systèmes embarqués ne se limitent pas à un cadre général. Il est essentiel de préciser le domaine d'application afin de mieux appréhender les enjeux et les défis qui lui sont associés. Chaque secteur présente des exigences uniques qui requièrent une attention particulière lors du développement et de l'implémentation des systèmes embarqués, soulignant ainsi l'importance d'une approche ciblée dans l'ingénierie des exigences.
3. **Support de la validation automatique des exigences** : L'utilisation du mot shall et de la syntaxe proposée par le langage RELAX dans la description des exigences ne suffit pas à la considérer comme formelle. Elle est plutôt perçue comme une description textuelle semi-formelle, laissant ainsi place à une certaine ambiguïté. À l'inverse, une description formelle repose sur une notation mathématique ou logique, permettant une interprétation non ambiguë et facilitant la validation automatique des exigences. Dans cette optique, notre objectif futur sera de développer un outil capable de transformer des descriptions semi-formelles en descriptions formelles en FBTL, en exploitant la bibliothèque Xtext de Java.

4. **Intégration de l'intelligence artificielle** *:* L'intelligence artificielle (IA) a connu une évolution rapide et un essor significatif au cours des dernières années, transformant de nombreux secteurs, y compris celui des systèmes embarqués. L'intégration de l'IA dans les SEs a permis d'aborder des problématiques complexes liées à la conception, l'optimisation et l'adaptabilité de ces systèmes, comme en témoignent les travaux récents. Fateh Boutekkouk a présenté une revue détaillée sur les méthodes basées sur l'IA pour résoudre les défis de l'ordonnancement temps réel dans les SEs, mettant en évidence leur capacité à garantir des performances optimales sous des contraintes strictes[179]. En complément, l'application d'une méthode multicritère de décision floue (Fuzzy MCDM) a été explorée pour choisir le meilleur système d'exploitation dans le cadre d'une conception sécurisée et efficace des systèmes embarqués, démontrant l'importance de l'IA pour des choix stratégiques dans la conception [180]. De plus, une perspective globale sur la co-conception des systèmes embarqués sous l'angle de l'IA a été fournie, soulignant l'impact des algorithmes intelligents sur l'efficacité et la résilience de ces systèmes[181]. Dans le contexte de la consommation d'énergie, Ridha Mehalaine et Fateh Boutekkouk ont proposé des approches pour réduire la consommation énergétique dans les systèmes embarqués multiprocesseurs en temps réel avec des données incertaines, intégrant des solutions intelligentes pour gérer efficacement les ressources [182]. En parallèle, ils ont développé un modèle bio-inspiré intelligent visant à améliorer la tolérance aux fautes dans les systèmes embarqués distribués, renforçant leur robustesse face aux défaillances imprévues [183] . Dans le domaine de l'ingénierie des exigences des SEAAs, l'IA joue également un rôle essentiel pour gérer la complexité, l'incertitude et l'évolution dynamique de ces systèmes. Par conséquent, à l'avenir, nous espérons exploiter des techniques telles que le traitement du langage naturel (NLP) pour extraire automatiquement les exigences. L'apprentissage automatique (ML) peut également être utilisé pour classer et prioriser les exigences en fonction de leur impact, tout en identifiant les relations et les dépendances complexes entre elles. En modélisation, des techniques comme l'algorithmie évolutionnaire et les réseaux de neurones pourraient être appliquées pour adapter automatiquement les modèles d'exigences en fonction des besoins évolutifs du système. Les réseaux bayésiens et la logique floue contribuent à détecter les incohérences ou les conflits entre les exigences et les contraintes du système. Lors de la vérification, les techniques de simulation basées sur l'IA peuvent être employées pour tester différents scénarios d'utilisation et garantir que le système réagit correctement aux exigences dans

un environnement dynamique. Enfin, des approches comme l'apprentissage par renforcement et l'optimisation combinatoire permettent de suivre et d'ajuster les exigences au fil du temps, améliorant ainsi le système de manière continue pour qu'il reste adapté aux évolutions de son environnement, assurant son efficacité et sa pertinence tout au long de son cycle de vie.

5. **L'intégration des ontologies :** Bien que notre métamodèle soit déjà enrichi par des contraintes exprimées en OCL, l'intégration des ontologies représente une perspective prometteuse pour renforcer davantage la sémantique et l'expressivité du modèle. En effet, les ontologies permettent non seulement de formaliser les connaissances liées aux exigences de manière plus explicite et standardisée, mais aussi d'activer des mécanismes de raisonnement automatique pour détecter des incohérences, inférer de nouvelles informations ou vérifier la cohérence globale du système. Cette complémentarité entre métamodélisation UML/OCL et ontologies offre une base solide pour améliorer la traçabilité, la réutilisation, et l'interopérabilité des exigences, en particulier dans le contexte des systèmes adaptatifs, où l'évolution dynamique du comportement impose une gestion fine des connaissances et de leur interprétation.

Annexes

Annexe 01 : Synthèse du processus REP4SAES-Version français

ID	Titre
BR	Exigences métier
BR.SI	Identification des parties prenantes
BR.SI.a1	Analyse contextuelle
BR.SI.a2	Définition des rôles
BR.SI.a3	Attribution des responsabilités
BR.SI.a4	Catégorisation
BR.SI.a5	Validation
BR.SI.a6	Documentation
BR.IO	Identification des Objectifs
BR.IO.a1	Analyse des Besoins
BR.IO.a2	Définition des objectifs stratégiques
BR.IO.a3	Décomposition des Objectifs
BR.IO.a4	Validation des objectifs/Sous-objectifs
BR.IO.a5	Documentation
BR.IO.a6	Vérification de la traçabilité
BR.IC	Identification des Contraintes
BR.IC.a1	Identification des contraintes du domaine
BR.IC.a2	Identification des contraintes environnementales
BR.IC.a3	Identification des contraintes normatives
BR.IC.a4	Évaluation des ressources humaines et financières
BR.IC.a5	Validation avec les parties prenantes
BR.IC.a6	Documentation
HSR	Exigences matériel/logiciel
HSR.HWR	Exigences matériel
HSR.HWR.a1	Définition des dispositifs matériels (hardware)
HSR.HWR.a2	Évaluation des contraintes physiques
HSR.HWR.a3	Évaluation des compromis
HSR.HWR.a4	Validation des contraintes
HSR.HWR.a5	Documentation
HSR.SWR	Exigences logiciel
HSR.SWR.a1	Définition des logiciels nécessaires
HSR.SWR.a2	Définition des exigences logicielles
HSR.SWR.a3	Validation
HSR.SWR.a4	Documentation
SR	Exigences du système
SR-FR	Exigences fonctionnelles
SR.FR.a1	Formuler les Exigences Fonctionnelles
SR.FR.a2	Validation

SR.FR.a3	Documentation
SR-NFR	Exigences Non-Fonctionnelles
SR.NFR.a1	Formuler les Exigences Non-Fonctionnelles
SR.NFR.a2	Catégorisation
SR.NFR.a3	Validation
SR.NFR.a4	Documentation
SR-CV	Vérification de la couverture
SR.CV.a1	Vérification de la complétude
RA	Analyse des exigences
RA.HAAR	Analyse hiérarchique et raffinement des exigences
RA. HARR.a1	Décomposition des exigences
RA. HARR.a2	Affinement des exigences fonctionnelles/non fonctionnelles
RA.HARR.a3	Écrire chaque exigence sous la forme d'une déclaration SHALL
RA.HARR.a4	Documentation
RA.INF	Impact des exigences non fonctionnelles sur les exigences fonctionnelles
RA.INF.a1	Analyse approfondie des exigences non fonctionnelles
RA.INF.a2	Décomposition et Affinement des exigences fonctionnelles
RA.CIR	Identification et résolution des conflits
RA.CIR.a1	Identification des conflits
RA.CIR.a2	Négociation
RA.CIR.a3	Priorisation
RA.CIR.a4	Validation et Consensus
RA.CIR.a5	Documentation
RR	Exigences assouplies
RR.RP	Le processus RELAX_ES
RR.RP.a1	Filtrage des exigences
RR.RP.a2	Identifier les facteurs d'incertitude
RR.RP.a3	Analyser les facteurs d'incertitude
RR.RP.a4	Introduire le(s) opérateur(s) RELAX
RR.RP.a5	Rédiger chaque exigence assouplie selon la grammaire RELAX
RR.RP.a6	Documentation
MR	Méta- Exigences
MR.AwReq	Exigences de conscience
MR.AwReq.a1	Détermination des Exigences NeverFail « Regular Requirement »
MR.AwReq.a2	Association des exigences assouplies aux exigences de conscience
MR.AwReq.a3	Documentation
MR.EvoReq	Exigences d'evolution
MR.EvoReq.a1	Identification des exigences d'évolution
MR.EvoReq.a2	Documentation
FRFL	Exigences fonctionnelles de la boucle de rétroaction
FRFL.FRFL	Exigences fonctionnelles de la boucle de rétroaction
FRFL.FRFL.a1	Identification des exigences du comportement fonctionnel de la boucle de rétroaction
FRFL.FRFL.a2	Décomposition et Affinement des exigences
RS	Spécification des exigences
RS.Doc	Documentation

RS.Doc.a1	Vérification du système
RS.Doc.a2	Générer le document des exigences utilisateur
RS.Doc.a3	Générer le document des exigences système
RS.RM	Modélisation des exigences
RS.RM.a1	Génération de diagramme des exigences et le BDD

Annexe 02 : Le fichier XMI du métamodèle

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE ecore:EPackage [
<!ENTITY _0 "http://www.eclipse.org/emf/2002/Ecore">
]>
<ecore:EPackage     xmi:version="2.0"     xmlns:xmi="http://www.omg.org/XMI"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:ecore="http://www.eclipse.org/emf/2002/Ecore"       name="MM4SAES"
nsURI="http://www.example.org/MM4SAES" nsPrefix="MM4SAES">
  <eAnnotations source="http://www.eclipse.org/OCL/Import">
    <details key="ecore" value="http://www.eclipse.org/emf/2002/Ecore"/>
  </eAnnotations>
  <eAnnotations source="http://www.eclipse.org/emf/2002/Ecore">
    <details                                          key="invocationDelegates"
value="http://www.eclipse.org/emf/2002/Ecore/OCL/Pivot"/>
    <details                                            key="settingDelegates"
value="http://www.eclipse.org/emf/2002/Ecore/OCL/Pivot"/>
    <details                                          key="validationDelegates"
value="http://www.eclipse.org/emf/2002/Ecore/OCL/Pivot"/>
  </eAnnotations>
  <eClassifiers xsi:type="ecore:EClass" name="SAES">
    <eAnnotations source="http://www.eclipse.org/emf/2002/Ecore">
      <details key="constraints" value="NonEmptyID NonEmptyDescription"/>
    </eAnnotations>
    <eStructuralFeatures       xsi:type="ecore:EAttribute"       name="Date"
eType="ecore:EDataType &_0;#//EDate"/>
    <eStructuralFeatures        xsi:type="ecore:EAttribute"        name="ID"
eType="ecore:EDataType &_0;#//EString"
        defaultValueLiteral="XXXX"/>
    <eStructuralFeatures       xsi:type="ecore:EAttribute"       name="Name"
eType="ecore:EDataType &_0;#//EString"
        defaultValueLiteral="New SAES"/>
    <eStructuralFeatures    xsi:type="ecore:EAttribute"    name="Description"
eType="ecore:EDataType &_0;#//EString"/>
    <eStructuralFeatures    xsi:type="ecore:EAttribute"    name="Statistics"
eType="ecore:EDataType &_0;#//EString"/>
    <eStructuralFeatures      xsi:type="ecore:EReference"      name="Needs"
lowerBound="1" upperBound="-1"
        eType="#//Need" containment="true"/>
    <eStructuralFeatures    xsi:type="ecore:EReference"    name="Constraints"
lowerBound="1"
        upperBound="-1" eType="#//Constraint" containment="true"/>
    <eStructuralFeatures     xsi:type="ecore:EReference"     name="Env_Vars"
upperBound="-1"
        eType="#//Env_Var" containment="true"/>
    <eStructuralFeatures    xsi:type="ecore:EReference"    name="Hardwares"
upperBound="-1"
        eType="#//HardDevice" containment="true"/>
    <eStructuralFeatures    xsi:type="ecore:EReference"    name="Softwares"
upperBound="-1"
        eType="#//Software" containment="true"/>
    <eStructuralFeatures   xsi:type="ecore:EReference"   name="Stakeholders"
lowerBound="1"
        upperBound="-1" eType="#//Stakeholder" containment="true"/>
    <eStructuralFeatures  xsi:type="ecore:EReference"  name="managedsystem"
lowerBound="1"
        eType="#//ManagedSystem" containment="true"/>
```

```
    <eStructuralFeatures xsi:type="ecore:EReference" name="managingsystem"
eType="#//ManagingSystem"
        containment="true"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="Need">
    <eAnnotations source="http://www.eclipse.org/emf/2002/Ecore">
      <details key="constraints" value="NonEmptyID NonEmptyDescription
NonEmptyStakeholder IDUniqueness DescriptionUniqueness IDFormat"/>
    </eAnnotations>
    <eAnnotations source="http://www.eclipse.org/emf/2002/Ecore/OCL/Pivot">
      <details key="NonEmptyID" value="&#xA;&#x9;&#x9;&#x9;not self.ID-
>isEmpty()"/>
      <details key="NonEmptyDescription" value="&#xA;&#x9;&#x9;&#x9;not
self.Description->isEmpty()"/>
      <details key="NonEmptyStakeholder" value="&#xA;&#x9;&#x9;&#x9;not
self.stakeholder->isEmpty()"/>
      <details key="IDUniqueness"
value="&#xA;&#x9;&#x9;&#x9;Need.allInstances()->isUnique(ID)"/>
      <details key="DescriptionUniqueness"
value="&#xA;&#x9;&#x9;&#x9;Need.allInstances()->isUnique(Description)"/>
      <details key="IDFormat"
value="&#xA;&#x9;&#x9;&#x9;self.ID.matches('N[0-9]+')"/>
    </eAnnotations>
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="ID"
eType="ecore:EDataType &_0;#//EString"
        defaultValueLiteral=""/>
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="Description"
eType="ecore:EDataType &_0;#//EString"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="Goals"
lowerBound="1" upperBound="-1"
        eType="#//Goal" containment="true" eOpposite="#//Goal/need"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="stakeholder"
lowerBound="1"
        upperBound="-1" eType="#//Stakeholder"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="Constraint">
    <eAnnotations source="http://www.eclipse.org/emf/2002/Ecore">
      <details key="constraints" value="NonEmptyID NonEmptyDescription
NonEmptyJustification IDUniqueness DescriptionUniqueness"/>
    </eAnnotations>
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="ID"
eType="ecore:EDataType &_0;#//EString"
        defaultValueLiteral=""/>
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="Description"
eType="ecore:EDataType &_0;#//EString"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="Requirements"
lowerBound="1"
        upperBound="-1" eType="#//Requirement" containment="true"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="stakeholder"
lowerBound="1"
        upperBound="-1" eType="#//Stakeholder"/>
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="Source"
eType="#//Constraint_Type"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="Stakeholder">
    <eAnnotations source="http://www.eclipse.org/emf/2002/Ecore">
      <details key="constraints" value="NonEmptyID NonEmptyName
NonEmptyRole NonEmptyResponsibility NonEmptyContact IDUniqueness
NameUniqueness ContactUniqueness IDFormat"/>
    </eAnnotations>
    <eAnnotations source="http://www.eclipse.org/emf/2002/Ecore/OCL/Pivot">
```

```
      <details   key="NonEmptyID"   value="&#xA;&#x9;&#x9;&#x9;not   self.ID-
>isEmpty()"/>
      <details key="NonEmptyName" value="&#xA;&#x9;&#x9;&#x9;not self.Name-
>isEmpty()"/>
      <details key="NonEmptyRole" value="&#xA;&#x9;&#x9;&#x9;not self.Role-
>isEmpty()"/>
      <details  key="NonEmptyResponsibility"  value="&#xA;&#x9;&#x9;&#x9;not
self.Responsibility->isEmpty()"/>
      <details     key="NonEmptyContact"     value="&#xA;&#x9;&#x9;&#x9;not
self.Contact->isEmpty()"/>
      <details                                                key="IDUniqueness"
value="&#xA;&#x9;&#x9;&#x9;Stakeholder.allInstances()->isUnique(ID)"/>
      <details                                              key="NameUniqueness"
value="&#xA;&#x9;&#x9;&#x9;Stakeholder.allInstances()->isUnique(Name)"/>
      <details                                           key="ContactUniqueness"
value="&#xA;&#x9;&#x9;&#x9;Stakeholder.allInstances()->isUnique(Contact)"/>
      <details                                                  key="IDFormat"
value="&#xA;&#x9;&#x9;&#x9;self.ID.matches('STK[0-9]+')"/>
    </eAnnotations>
    <eStructuralFeatures        xsi:type="ecore:EAttribute"        name="ID"
eType="ecore:EDataType &_0;#//EString"
        defaultValueLiteral=""/>
    <eStructuralFeatures       xsi:type="ecore:EAttribute"       name="Name"
eType="ecore:EDataType &_0;#//EString"/>
    <eStructuralFeatures       xsi:type="ecore:EAttribute"       name="Role"
eType="ecore:EDataType &_0;#//EString"/>
    <eStructuralFeatures  xsi:type="ecore:EAttribute"  name="Responsibility"
eType="ecore:EDataType &_0;#//EString"/>
    <eStructuralFeatures     xsi:type="ecore:EAttribute"     name="Contact"
eType="ecore:EDataType &_0;#//EString"/>
    <eStructuralFeatures    xsi:type="ecore:EAttribute"    name="Category"
eType="#//Stk_Cat"
        defaultValueLiteral="Business"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EEnum" name="Stk_Cat">
    <eLiterals name="Business"/>
    <eLiterals name="Technical" value="1"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EEnum" name="Priority">
    <eLiterals name="Must_Have"/>
    <eLiterals name="Should_Have" value="1"/>
    <eLiterals name="Could_Have" value="2"/>
    <eLiterals name="Wont_Have" value="3"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="Goal">
    <eAnnotations source="http://www.eclipse.org/emf/2002/Ecore">
      <details key="constraints" value="SubObjective NonEmptyID"/>
    </eAnnotations>
    <eStructuralFeatures        xsi:type="ecore:EAttribute"        name="ID"
eType="ecore:EDataType &_0;#//EString"/>
    <eStructuralFeatures   xsi:type="ecore:EAttribute"   name="Description"
eType="ecore:EDataType &_0;#//EString"/>
    <eStructuralFeatures    xsi:type="ecore:EReference"    name="Sub_Goals"
lowerBound="1"
        upperBound="-1" eType="#//Sub_Goal" containment="true"/>
    <eStructuralFeatures       xsi:type="ecore:EReference"       name="need"
lowerBound="1" eType="#//Need"
        eOpposite="#//Need/Goals"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="Sub_Goal">
    <eAnnotations source="http://www.eclipse.org/emf/2002/Ecore">
```

```
      <details key="constraints" value="NonEmptyID"/>
    </eAnnotations>
    <eStructuralFeatures        xsi:type="ecore:EAttribute"        name="ID"
eType="ecore:EDataType &_0;#//EString"/>
    <eStructuralFeatures   xsi:type="ecore:EAttribute"   name="Description"
eType="ecore:EDataType &_0;#//EString"/>
    <eStructuralFeatures   xsi:type="ecore:EReference"   name="Requirements"
lowerBound="1"
        upperBound="-1" eType="#//Requirement" containment="true"/>
  </eClassifiers>
  <eClassifiers    xsi:type="ecore:EClass"    name="NFR"    abstract="true"
eSuperTypes="#//Requirement">
    <eStructuralFeatures     xsi:type="ecore:EReference"     name="Generate"
upperBound="-1"
        eType="#//FR"/>
    <eStructuralFeatures     xsi:type="ecore:EAttribute"     name="Category"
eType="#//NFR_Type"/>
  </eClassifiers>
  <eClassifiers    xsi:type="ecore:EClass"    name="FR"    abstract="true"
eSuperTypes="#//Requirement">
    <eStructuralFeatures  xsi:type="ecore:EReference"  name="ContributesTo"
eType="#//NFR"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EEnum" name="Req_Type">
    <eLiterals name="Invariant"/>
    <eLiterals name="Variant" value="1"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="Env_Var">
    <eStructuralFeatures        xsi:type="ecore:EAttribute"        name="ID"
eType="ecore:EDataType &_0;#//EString"/>
    <eStructuralFeatures   xsi:type="ecore:EAttribute"   name="Description"
eType="ecore:EDataType &_0;#//EString"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="Monitor">
    <eOperations name="CleanData"/>
    <eOperations name="FilterData"/>
    <eOperations name="UpdateData"/>
    <eOperations name="CollectData"/>
    <eStructuralFeatures       xsi:type="ecore:EReference"       name="REL"
lowerBound="1" upperBound="-1"
        eType="#//Env_Var"/>
    <eStructuralFeatures   xsi:type="ecore:EReference"   name="Receive_Data"
lowerBound="1"
        upperBound="-1" eType="#//Sensor"/>
    <eStructuralFeatures        xsi:type="ecore:EAttribute"        name="ID"
eType="ecore:EDataType &_0;#//EString"/>
    <eStructuralFeatures   xsi:type="ecore:EAttribute"   name="Description"
eType="ecore:EDataType &_0;#//EString"/>
    <eStructuralFeatures     xsi:type="ecore:EAttribute"     name="Relation"
eType="ecore:EDataType &_0;#//EString"/>
    <eStructuralFeatures       xsi:type="ecore:EAttribute"       name="Name"
eType="ecore:EDataType &_0;#//EString"/>
    <eStructuralFeatures        xsi:type="ecore:EReference"        name="fr"
lowerBound="1" upperBound="-1"
        eType="#//FR" containment="true"/>
  </eClassifiers>
  <eClassifiers            xsi:type="ecore:EClass"            name="Sensor"
eSuperTypes="#//InputDevice"/>
  <eClassifiers xsi:type="ecore:EClass" name="HardRequirement">
    <eStructuralFeatures        xsi:type="ecore:EAttribute"        name="ID"
eType="ecore:EDataType &_0;#//EString"/>
```

```
    <eStructuralFeatures       xsi:type="ecore:EAttribute"       name="Name"
eType="ecore:EDataType &_0;#//EString"/>
    <eStructuralFeatures   xsi:type="ecore:EAttribute"   name="Description"
eType="ecore:EDataType &_0;#//EString"/>
    <eStructuralFeatures   xsi:type="ecore:EReference"   name="Requirements"
lowerBound="1"
        upperBound="-1" eType="#//Requirement" containment="true"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="HardDevice">
    <eAnnotations source="http://www.eclipse.org/emf/2002/Ecore">
      <details key="constraints" value="NonEmptyID"/>
    </eAnnotations>
    <eStructuralFeatures        xsi:type="ecore:EAttribute"        name="ID"
eType="ecore:EDataType &_0;#//EString"/>
    <eStructuralFeatures       xsi:type="ecore:EAttribute"       name="Name"
eType="ecore:EDataType &_0;#//EString"/>
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="Characteristics"
eType="ecore:EDataType &_0;#//EString"/>
    <eStructuralFeatures     xsi:type="ecore:EReference"     name="Defines"
lowerBound="1"
        upperBound="-1" eType="#//HardRequirement" containment="true"/>
  </eClassifiers>
  <eClassifiers          xsi:type="ecore:EClass"          name="InputDevice"
eSuperTypes="#//HardDevice"/>
  <eClassifiers            xsi:type="ecore:EClass"            name="Memory"
eSuperTypes="#//HardDevice"/>
  <eClassifiers    xsi:type="ecore:EClass"    name="CommunicationInterface"
eSuperTypes="#//HardDevice"/>
  <eClassifiers        xsi:type="ecore:EClass"        name="ProcessingUnit"
eSuperTypes="#//HardDevice"/>
  <eClassifiers         xsi:type="ecore:EClass"         name="OutputDevice"
eSuperTypes="#//HardDevice"/>
  <eClassifiers          xsi:type="ecore:EClass"          name="HardAdapter"
eSuperTypes="#//HardDevice">
    <eStructuralFeatures     xsi:type="ecore:EReference"     name="Connects"
lowerBound="2"
        upperBound="-1" eType="#//HardDevice"/>
  </eClassifiers>
  <eClassifiers       xsi:type="ecore:EClass"       name="OnBoardCommunInt"
eSuperTypes="#//CommunicationInterface"/>
  <eClassifiers      xsi:type="ecore:EClass"      name="ExternalCommunInt"
eSuperTypes="#//CommunicationInterface"/>
  <eClassifiers           xsi:type="ecore:EClass"           name="Actuator"
eSuperTypes="#//OutputDevice"/>
  <eClassifiers xsi:type="ecore:EClass" name="SoftRequirement">
    <eStructuralFeatures        xsi:type="ecore:EAttribute"        name="ID"
eType="ecore:EDataType &_0;#//EString"/>
    <eStructuralFeatures       xsi:type="ecore:EAttribute"       name="Name"
eType="ecore:EDataType &_0;#//EString"/>
    <eStructuralFeatures   xsi:type="ecore:EAttribute"   name="Description"
eType="ecore:EDataType &_0;#//EString"/>
    <eStructuralFeatures   xsi:type="ecore:EReference"   name="Requirements"
lowerBound="1"
        upperBound="-1" eType="#//Requirement" containment="true"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="Software" abstract="true">
    <eAnnotations source="http://www.eclipse.org/emf/2002/Ecore">
      <details key="constraints" value="NonEmptyID"/>
    </eAnnotations>
    <eStructuralFeatures        xsi:type="ecore:EAttribute"        name="ID"
eType="ecore:EDataType &_0;#//EString"/>
```

```
    <eStructuralFeatures       xsi:type="ecore:EAttribute"       name="Name"
eType="ecore:EDataType &_0;#//EString"/>
    <eStructuralFeatures     xsi:type="ecore:EReference"     name="Defines"
lowerBound="1"
        upperBound="-1" eType="#//SoftRequirement" containment="true"/>
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="Characteristics"
eType="ecore:EDataType &_0;#//EString"/>
  </eClassifiers>
  <eClassifiers             xsi:type="ecore:EClass"             name="RTOS"
eSuperTypes="#//Software"/>
  <eClassifiers          xsi:type="ecore:EClass"          name="DeviceDriver"
eSuperTypes="#//Software">
    <eStructuralFeatures     xsi:type="ecore:EReference"     name="Control"
lowerBound="1"
        eType="#//HardDevice"/>
  </eClassifiers>
  <eClassifiers        xsi:type="ecore:EClass"        name="ApplicationSoft"
eSuperTypes="#//Software"/>
  <eClassifiers         xsi:type="ecore:EClass"         name="DebuggingSoft"
eSuperTypes="#//Software"/>
  <eClassifiers       xsi:type="ecore:EClass"       name="ErrorHandlingSoft"
eSuperTypes="#//Software"/>
  <eClassifiers       xsi:type="ecore:EClass"       name="SoftwareInterface"
eSuperTypes="#//Software">
    <eStructuralFeatures     xsi:type="ecore:EReference"     name="Connects"
lowerBound="2"
        upperBound="-1" eType="#//Software"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="MetaRequirement">
    <eStructuralFeatures        xsi:type="ecore:EAttribute"        name="ID"
eType="ecore:EDataType &_0;#//EString"/>
    <eStructuralFeatures      xsi:type="ecore:EReference"      name="AwReqs"
lowerBound="1"
        eType="#//AwReq" containment="true"/>
    <eStructuralFeatures     xsi:type="ecore:EReference"     name="EvoReqs"
lowerBound="1"
        eType="#//EvoReq" containment="true"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="AwReq">
    <eStructuralFeatures    xsi:type="ecore:EAttribute"    name="TypeAWReq"
eType="#//AwReq_Type"
        defaultValueLiteral="Aggregate"/>
    <eStructuralFeatures      xsi:type="ecore:EReference"      name="awreq"
lowerBound="1" upperBound="-1"
        eType="#//AwReq" containment="true"/>
    <eStructuralFeatures        xsi:type="ecore:EAttribute"        name="ID"
eType="ecore:EDataType &_0;#//EString"/>
    <eStructuralFeatures   xsi:type="ecore:EAttribute"   name="Description"
eType="ecore:EDataType &_0;#//EString"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="EvoReq">
    <eStructuralFeatures        xsi:type="ecore:EAttribute"        name="ID"
eType="ecore:EDataType &_0;#//EString"/>
    <eStructuralFeatures   xsi:type="ecore:EAttribute"   name="Description"
eType="ecore:EDataType &_0;#//EString"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="Requirement" abstract="true">
    <eAnnotations source="http://www.eclipse.org/emf/2002/Ecore">
      <details   key="constraints"   value="NonEmptyID   NonEmptyDescription
NonEmptyFormalDescription         IDUniqueness         DescriptionUniqueness
FormalDescriptionUniqueness"/>
```

```
    </eAnnotations>
    <eAnnotations source="http://www.eclipse.org/emf/2002/Ecore/OCL/Pivot">
      <details   key="NonEmptyID"   value="&#xA;&#x9;&#x9;&#x9;not   self.ID-
>isEmpty()"/>
      <details    key="NonEmptyDescription"    value="&#xA;&#x9;&#x9;&#x9;not
self.Description->isEmpty()"/>
      <details                                  key="NonEmptyFormalDescription"
value="&#xA;&#x9;&#x9;&#x9;not self.Formal_Description->isEmpty()"/>
      <details                                                 key="IDUniqueness"
value="&#xA;&#x9;&#x9;&#x9;Requirement.allInstances()->isUnique(ID)"/>
      <details                                        key="DescriptionUniqueness"
value="&#xA;&#x9;&#x9;&#x9;Requirement.allInstances()-
>isUnique(Description)"/>
      <details                                 key="FormalDescriptionUniqueness"
value="&#xA;&#x9;&#x9;&#x9;Requirement.allInstances()-
>isUnique(Formal_Description)"/>
    </eAnnotations>
    <eOperations         name="priorityOrder"         eType="ecore:EDataType
&_0;#//EBigInteger">
      <eAnnotations
source="http://www.eclipse.org/emf/2002/Ecore/OCL/Pivot">
        <details  key="body"  value="&#xA;           &#x9;if  self.Priority  =
'Must_Have'  then  4&#xA;               &#x9;&#x9;else  if  self.Priority  =
'Should_Have' then 3&#xA;           &#x9;&#x9;&#x9;else if self.Priority =
'Could_Have' then 2&#xA;         &#x9;&#x9;&#x9;&#x9;else if self.Priority =
'Wontave'  then  1&#xA;                &#x9;&#x9;&#x9;&#x9;&#x9;else  0&#xA;
&#x9;&#x9;&#x9;&#x9;&#x9;endif &#xA;         &#x9;&#x9;&#x9;&#x9;endif &#xA;
&#x9;&#x9;&#x9;endif &#xA;         &#x9;&#x9;endif"/>
      </eAnnotations>
    </eOperations>
    <eStructuralFeatures        xsi:type="ecore:EAttribute"        name="ID"
eType="ecore:EDataType &_0;#//EString"/>
    <eStructuralFeatures       xsi:type="ecore:EAttribute"       name="Name"
eType="ecore:EDataType &_0;#//EString"/>
    <eStructuralFeatures   xsi:type="ecore:EAttribute"   name="Description"
eType="ecore:EDataType &_0;#//EString"/>
    <eStructuralFeatures                          xsi:type="ecore:EAttribute"
name="Formal_Description" eType="ecore:EDataType &_0;#//EString"/>
    <eStructuralFeatures     xsi:type="ecore:EAttribute"     name="Priority"
eType="#//Priority"/>
    <eStructuralFeatures   xsi:type="ecore:EReference"   name="PositiveDEP"
upperBound="-1"
        eType="#//Requirement"/>
    <eStructuralFeatures   xsi:type="ecore:EReference"   name="NegativeDEP"
upperBound="-1"
        eType="#//Requirement"/>
    <eStructuralFeatures                          xsi:type="ecore:EReference"
name="OperationalizedByManagedSys"
        eType="#//ManagedSystem"/>
    <eStructuralFeatures                          xsi:type="ecore:EReference"
name="OperationalizedByManagingSys"
        eType="#//ManagingSystem"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="ManagedSystem">
    <eStructuralFeatures        xsi:type="ecore:EAttribute"        name="ID"
eType="ecore:EDataType &_0;#//EString"/>
    <eStructuralFeatures   xsi:type="ecore:EAttribute"   name="Description"
eType="ecore:EDataType &_0;#//EString"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="ManagingSystem">
    <eStructuralFeatures   xsi:type="ecore:EReference"   name="feedbackloop"
```

```
lowerBound="1"
        eType="#//FeedBackLoop" containment="true"/>
    <eStructuralFeatures  xsi:type="ecore:EReference"  name="adaptationgoal"
lowerBound="1"
        eType="#//AdaptationGoal" containment="true"/>
    <eStructuralFeatures        xsi:type="ecore:EAttribute"        name="ID"
eType="ecore:EDataType &_0;#//EString"/>
    <eStructuralFeatures   xsi:type="ecore:EAttribute"   name="Description"
eType="ecore:EDataType &_0;#//EString"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="Analysis">
    <eOperations name="AnalyzeData"/>
    <eOperations name="ChangeRequest"/>
    <eStructuralFeatures        xsi:type="ecore:EAttribute"        name="ID"
eType="ecore:EDataType &_0;#//EString"/>
    <eStructuralFeatures   xsi:type="ecore:EAttribute"   name="Description"
eType="ecore:EDataType &_0;#//EString"/>
    <eStructuralFeatures       xsi:type="ecore:EAttribute"       name="Name"
eType="ecore:EDataType &_0;#//EString"/>
    <eStructuralFeatures        xsi:type="ecore:EReference"        name="fr"
lowerBound="1" upperBound="-1"
        eType="#//FR" containment="true"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="FeedBackLoop">
    <eStructuralFeatures     xsi:type="ecore:EReference"     name="analysis"
lowerBound="1"
        upperBound="-1" eType="#//Analysis" containment="true"/>
    <eStructuralFeatures       xsi:type="ecore:EReference"       name="plan"
lowerBound="1" upperBound="-1"
        eType="#//Plan" containment="true"/>
    <eStructuralFeatures     xsi:type="ecore:EReference"     name="monitor"
lowerBound="1"
        upperBound="-1" eType="#//Monitor" containment="true"/>
    <eStructuralFeatures    xsi:type="ecore:EReference"    name="execution"
lowerBound="1"
        upperBound="-1" eType="#//Execution" containment="true"/>
    <eStructuralFeatures    xsi:type="ecore:EReference"    name="knowledge"
lowerBound="1"
        upperBound="-1" eType="#//Knowledge" containment="true"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="AdaptationGoal">
    <eStructuralFeatures        xsi:type="ecore:EAttribute"        name="ID"
eType="ecore:EDataType &_0;#//EString"/>
    <eStructuralFeatures   xsi:type="ecore:EAttribute"   name="Description"
eType="ecore:EDataType &_0;#//EString"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="Plan">
    <eOperations name="ChangePlan"/>
    <eOperations name="Decide"/>
    <eStructuralFeatures        xsi:type="ecore:EAttribute"        name="ID"
eType="ecore:EDataType &_0;#//EString"/>
    <eStructuralFeatures   xsi:type="ecore:EAttribute"   name="Description"
eType="ecore:EDataType &_0;#//EString"/>
    <eStructuralFeatures       xsi:type="ecore:EAttribute"       name="Name"
eType="ecore:EDataType &_0;#//EString"/>
    <eStructuralFeatures        xsi:type="ecore:EReference"        name="fr"
lowerBound="1" upperBound="-1"
        eType="#//FR" containment="true"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="Execution">
    <eOperations name="Act"/>
```

```
    <eOperations name="ExecutePlan"/>
    <eStructuralFeatures        xsi:type="ecore:EAttribute"        name="ID"
eType="ecore:EDataType &_0;#//EString"/>
    <eStructuralFeatures   xsi:type="ecore:EAttribute"   name="Description"
eType="ecore:EDataType &_0;#//EString"/>
    <eStructuralFeatures       xsi:type="ecore:EAttribute"       name="Name"
eType="ecore:EDataType &_0;#//EString"/>
    <eStructuralFeatures        xsi:type="ecore:EReference"        name="fr"
lowerBound="1" upperBound="-1"
        eType="#//FR" containment="true"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="Knowledge">
    <eStructuralFeatures        xsi:type="ecore:EAttribute"        name="ID"
eType="ecore:EDataType &_0;#//EString"/>
    <eStructuralFeatures   xsi:type="ecore:EAttribute"   name="Description"
eType="ecore:EDataType &_0;#//EString"/>
    <eStructuralFeatures       xsi:type="ecore:EAttribute"       name="Name"
eType="ecore:EDataType &_0;#//EString"/>
    <eStructuralFeatures        xsi:type="ecore:EReference"        name="fr"
lowerBound="1" upperBound="-1"
        eType="#//FR" containment="true"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EEnum" name="AwReq_Type">
    <eLiterals name="Aggregate" value="1"/>
    <eLiterals name="Trend" value="2"/>
    <eLiterals name="Delta" value="3"/>
    <eLiterals name="Meta" value="4"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EEnum" name="NFR_Type">
    <eLiterals name="Delivery" value="5"/>
    <eLiterals name="Ethical" value="9"/>
    <eLiterals name="Implementation" value="6"/>
    <eLiterals name="Interoperability" value="8"/>
    <eLiterals name="Performance" value="1"/>
    <eLiterals name="Portability" value="4"/>
    <eLiterals name="Privacy" value="10"/>
    <eLiterals name="Reliability" value="3"/>
    <eLiterals name="Safety" value="11"/>
    <eLiterals name="Space" value="2"/>
    <eLiterals name="Standard" value="7"/>
    <eLiterals name="Usability"/>
  </eClassifiers>
  <eClassifiers           xsi:type="ecore:EClass"           name="Battery"
eSuperTypes="#//HardDevice"/>
  <eClassifiers xsi:type="ecore:EClass" name="Refinement">
    <eStructuralFeatures     xsi:type="ecore:EAttribute"     name="Ref_Type"
eType="#//Ref_Type"/>
    <eStructuralFeatures        xsi:type="ecore:EReference"        name="fr"
lowerBound="2" upperBound="-1"
        eType="#//FR" containment="true"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EEnum" name="Ref_Type">
    <eLiterals name="AND"/>
    <eLiterals name="OR" value="1"/>
  </eClassifiers>
  <eClassifiers         xsi:type="ecore:EClass"         name="ElementaryFR"
eSuperTypes="#//FR">
    <eAnnotations source="http://www.eclipse.org/emf/2002/Ecore">
      <details   key="constraints"   value="syntaxFormat   MustHaveNotVariant
NoNegativeDependencyForLowerPriorityVariant"/>
    </eAnnotations>
```

```
    <eAnnotations source="http://www.eclipse.org/emf/2002/Ecore/OCL/Pivot">
      <details  key="syntaxFormat"  value="&#xA;&#x9;&#x9;&#x9;self.Type  =
'Invariant'                                                    implies
self.Description.matches('^The\\s+\\w+\\s+shall\\s+\\w+.*$')"/>
      <details            key="MustHaveNotVariant"            value="&#xA;
&#x9;&#x9;self.Priority = 'Must have' implies self.Type &lt;> 'Variant'"/>
      <details            key="NoNegativeDependencyForLowerPriorityVariant"
value="&#xA;       &#x9;&#x9;self.Type  =  'Variant'  and  self.NegativeDEP-
>notEmpty()&#xA;           &#x9;&#x9;implies   self.NegativeDEP->forAll(dep
|self.priorityOrder() >= dep.priorityOrder())"/>
    </eAnnotations>
    <eStructuralFeatures xsi:type="ecore:EReference" name="metarequirement"
upperBound="-1"
        eType="#//MetaRequirement" containment="true"/>
    <eStructuralFeatures       xsi:type="ecore:EReference"       name="VAR"
upperBound="-1" eType="#//Env_Var"/>
    <eStructuralFeatures     xsi:type="ecore:EReference"     name="Refine"
eType="#//ElementaryFR"/>
    <eStructuralFeatures       xsi:type="ecore:EReference"       name="MON"
upperBound="-1" eType="#//Monitor"/>
    <eStructuralFeatures       xsi:type="ecore:EAttribute"       name="Type"
eType="#//Req_Type"/>
  </eClassifiers>
  <eClassifiers          xsi:type="ecore:EClass"          name="AbstractFR"
eSuperTypes="#//FR">
    <eStructuralFeatures    xsi:type="ecore:EReference"    name="refinement"
lowerBound="1"
        eType="#//Refinement" containment="true"/>
  </eClassifiers>
  <eClassifiers         xsi:type="ecore:EClass"         name="ElementaryNFR"
eSuperTypes="#//NFR">
    <eAnnotations source="http://www.eclipse.org/emf/2002/Ecore">
      <details key="constraints" value="syntaxFormat"/>
    </eAnnotations>
    <eAnnotations source="http://www.eclipse.org/emf/2002/Ecore/OCL/Pivot">
      <details  key="syntaxFormat"  value="&#xA;&#x9;&#x9;&#x9;self.Type  =
'Invariant'                                                    implies
self.Description.matches('^The\\s+\\w+\\s+shall\\s+\\w+.*$')"/>
    </eAnnotations>
    <eStructuralFeatures xsi:type="ecore:EReference" name="metarequirement"
upperBound="-1"
        eType="#//MetaRequirement" containment="true"/>
    <eStructuralFeatures     xsi:type="ecore:EReference"     name="Refine"
eType="#//ElementaryNFR"/>
    <eStructuralFeatures       xsi:type="ecore:EReference"       name="VAR"
upperBound="-1" eType="#//Env_Var"/>
    <eStructuralFeatures       xsi:type="ecore:EReference"       name="MON"
upperBound="-1" eType="#//Monitor"/>
    <eStructuralFeatures      xsi:type="ecore:EAttribute"       name="Type"
eType="#//Req_Type"/>
  </eClassifiers>
  <eClassifiers          xsi:type="ecore:EClass"          name="AbstractNFR"
eSuperTypes="#//NFR">
    <eStructuralFeatures  xsi:type="ecore:EReference"  name="refinementnfr"
lowerBound="1"
        eType="#//RefinementNFR" containment="true"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="RefinementNFR">
    <eStructuralFeatures       xsi:type="ecore:EReference"       name="nfr"
lowerBound="2" upperBound="-1"
        eType="#//NFR" containment="true"/>
```

```
    <eStructuralFeatures     xsi:type="ecore:EAttribute"     name="Ref_Type"
eType="#//Ref_Type"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EEnum" name="Constraint_Type">
    <eLiterals name="Domain"/>
    <eLiterals name="Environment" value="1"/>
    <eLiterals name="Standard_and_Regulation" value="2"/>
    <eLiterals name="Humain_Resources" value="3"/>
    <eLiterals name="Budget" value="4"/>
  </eClassifiers>
</ecore:EPackage>
```

Bibliographie

[1] J. DionisioParaiba et L. E. G. Martins, « A Proposal of Requirements Specification Process for Adaptive Systems Based on Fuzzy Logic and NFR-Framework », in The Eighth International Conference on Software Engineering Advances (ICSEA 2013), 2013, p. 101-105.

[2] J. Whittle, P. Sawyer, N. Bencomo, et B. H. Cheng, « A language for self-adaptive system requirements », in International Workshop on Service-Oriented Computing: Consequences for Engineering Requirements, Barcelona, Spain, 2008, p. 24-29. doi: https://doi.org/10.1109/SOCCER.2008.1.

[3] N. A. Qureshi, I. Jureta, et A. Perini, « Adaptive RML: A Requirements Modeling Language for Self-Adaptive Systems », 2011.

[4] H. J. Goldsby, P. Sawyer, N. Bencomo, B. H. C. Cheng, et D. Hughes, « Goal-Based Modeling of Dynamically Adaptive System Requirements », in 15th Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems (ecbs 2008), Belfast, UK, mars 2008, p. 36-45. doi: https://doi.org/10.1109/ECBS.2008.22.

[5] G. Brown, B. H. Cheng, H. Goldsby, et J. Zhang, « Goal-oriented specification of adaptation requirements engineering in adaptive systems », in Proceedings of the 2006 international workshop on Self-adaptation and self-managing systems (SEAMS '06), 2006, p. 23-29. doi: https://doi.org/10.1145/1137677.1137682.

[6] J. D. Paraiba et L. E. G. Martins, « PERSA: A Requirements Specification Process for Self-Adaptive Systems Based on Fuzzy Logic and NFR-Framework », International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems, vol. 25, nº 01, p. 145-178, févr. 2017, doi: https://doi.org/10.1142/S0218488517500064.

[7] J. Whittle, P. Sawyer, N. Bencomo, B. H. C. Cheng, et J.-M. Bruel, « RELAX: a language to address uncertainty in self-adaptive systems requirement », Requirements Engineering, vol. 15, nº 2, p. 177-196, juin 2010, doi: https://doi.org/10.1007/s00766-010-0101-0.

[8] J. Whittle, P. Sawyer, N. Bencomo, B. H. Cheng, et J.-M. Bruel, « Relax: Incorporating uncertainty into the specification of self-adaptive systems », in 17th IEEE International Requirements Engineering Conference, Atlanta, GA, USA, 2009, p. 79-88. doi: https://doi.org/10.1109/RE.2009.36.

[9] S. Jovanovic, « Architecture reconfigurable de système embarqué auto-organisé », Thèse, Université Henri Poincaré - Nancy 1, France, 2009. [En ligne]. Disponible sur: https://theses.hal.science/tel-01748306/

[10] E. C. Hall, Journey to the Moon: The History of the Apollo Guidance Computer. American Institute of Aeronautics and Astronautics, 1996. doi: https://doi.org/10.2514/4.868023.

[11] M. Koudil, « Une approche orientée objet pour le codesign », Thèse d'état, Institut National d'Informatique, Alger, 2002.

[12] J.-P. Jamont, « DIAMOND: Une approche pour la conception de systèmes multi-agents embarqués », Thèse, Institut National Polytechnique de Grenoble - INPG, France, 2005. [En ligne]. Disponible sur: https://theses.hal.science/tel-00189046

[13] J. Teich, « Hardware/Software Codesign: The Past, the Present, and Predicting the Future », Proceedings of the IEEE, vol. 100, nº Special Centennial Issue, p. 1411-1430, mai 2012, doi: https://doi.org/10.1109/JPROC.2011.2182009.

[14] P. R. Schaumont, A Practical Introduction to Hardware/Software Codesign. Springer New York, NY, 2010. doi: https://doi.org/10.1007/978-1-4419-6000-9.

[15] W. Wolf, « A decade of hardware/ software codesign », Computer, vol. 36, nº 4, p. 38-43, avr. 2003, doi: https://doi.org/10.1109/MC.2003.1193227.

[16] R. Ernst, « Codesign of Embedded Systems: Status and Trends », IEEE Design & Test of Computers, vol. 15, nº 2, p. 45-54, 1998, doi: https://doi.org/10.1109/54.679207.

[17] G. F. Marchioro, « Découpage transformationnel pour la conception de systèmes mixtes logiciel/matériel », Institut National Polytechnique de Grenoble - INPG, France, 1998. [En ligne]. Disponible sur: https://theses.hal.science/tel-00002985

[18] F. Balarin et al., Hardware-Software Co-Design of Embedded Systems. Boston, MA: Springer New York, NY, 1997. doi: https://doi.org/10.1007/978-1-4615-6127-9.

[19] S. Kumar, J. H. Aylor, B. W. Johnson, et Wm. A. Wulf, The Codesign of Embedded Systems: A Unified Hardware/Software Representation. Springer New York, NY, 1996. doi: https://doi.org/10.1007/978-1-4613-1293-2.

[20] S. Klaus et S. A. Huss, « A novel specification model for IP-based design », in Euromicro Symposium on Digital System Design, 2003. Proceedings., Belek-Antalya, Turkey: IEEE, 2003, p. 190-196. doi: https://doi.org/10.1109/DSD.2003.1231924.

[21] Z. B. Salem, Y. Med. Wassim, et A. Mohamed, « A fast codesign approach for low cost application-specific-system on programmable chip (SoPC) : Application to sensor network », International Journal of Computer Applications, p. 187-194, 2010.

[22] E. Filippi et al., « Intellectual property re-use in embedded system co-design: an industrial case study », in Proceedings. 11th International Symposium on System Synthesis (Cat. No.98EX210), Hsinchu, Taiwan: IEEE Comput. Soc, 1998, p. 37-42. doi: https://doi.org/10.1109/ISSS.1998.730594.

[23] Z. Xiong, S. Li, J. Chen, et D. Wang, « A platform-based SoC hardware/software co-design environment », in 8th International Conference on Computer Supported Cooperative Work in Design, Xiamen, China, 2004, p. 443-448. doi: https://doi.org/10.1109/CACWD.2004.1349229.

[24] R. Chen, M. Sgroi, L. Lavagno, G. Martin, A. Sangiovanni-Vincentelli, et J. Rabaey, « UML and Platform-based Design », in UML for Real, 2003, p. 107-126. doi: https://doi.org/10.1007/0-306-48738-1_5.

[25] I. Ober, S. Van Baelen, S. Graf, M. Filali, T. Weigert, et S. Gérard, « Model Based Architecting and Construction of Embedded Systems », in Models in Software Engineering, Springer Berlin Heidelberg, 2009, p. 63-67. doi: https://doi.org/10.1007/978-3-642-12261-3_7.

[26] S. Schulz, J. W. Rozenblit, M. Mrva, et K. Buchenriede, « Model-based codesign », Computer, vol. 31, nº 8, p. 60-67, août 1998, doi: https://doi.org/10.1109/2.707618.

[27] A. Allam et W. Deabes, « Model-Based Hardware-Software Codesign of ECT Digital Processing Unit », Modelling and Simulation in Engineering, p. 1-14, mars 2021, doi: https://doi.org/10.1155/2021/4757464.

[28] S. J. Cunning, T. C. Ewing, J. T. Olson, J. W. Rozenblit, et S. Schulz, « Towards an integrated, model-based codesign environment », in Proceedings ECBS'99. IEEE Conference and Workshop on Engineering of Computer-Based Systems, Nashville, TN, USA, 1999, p. 136-143. doi: https://doi.org/10.1109/ECBS.1999.755872.

[29] L. Gomes, J. P. Barros, A. Costa, Rui Pais, et F. Moutinho, « Towards Usage of Formal methods within Embedded Systems Co-design », in 2005 IEEE Conference on Emerging Technologies and Factory Automation, Catania, Italy, 2005, p. 281-284. doi: https://doi.org/10.1109/ETFA.2005.1612535.

[30] V. Carchiolo, M. Malgeri, et G. Mangioni, « Formal Codesign Methodology with Multistep Partitioning », VLSI Design, vol. 7, 1996, doi: https://doi.org/10.1155/1998/18340.

[31] V. Carchiolo, M. Malgeri, et G. Mangioni, « Hardware/software synthesis of formal specifications in codesign of embedded systems », ACM Transactions on Design Automation of Electronic Systems, vol. 5, nº 3, 2000.

[32] M. Chiodo, P. Giusto, H. Hsieh, A. Jurecska, L. Lavagno, et A. Sangiovanni-Vincentelli, « A Formal Methodology for Hardware/Software Co-design of Embedded Systems », 1994.

[33] Soonhoi Ha, Choonseung Lee, Youngmin Yi, Seongnam Kwon, et Young-Pyo Joo, « Hardware-Software Codesign of Multimedia Embedded Systems: the PeaCE », in 12th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA'06), Sydney, NSW, Australia, 2006, p. 207-214. doi: https://doi.org/10.1109/RTCSA.2006.36.

[34] A. Aljer, P. Devienne, et S. Tison, « Component based co-design and refinement », in Proceedings. 2004 International Conference on Information and Communication Technologies: From Theory to Applications, 2004., Damascus, Syria, 2004, p. 575-576. doi: https://doi.org/10.1109/ICTTA.2004.1307893.

[35] P. Arato et A. Orban, « Component-Based Hardware/Software Co-Design », in Organic and Pervasive Computing – ARCS 2004, p. 169-183. doi: https://doi.org/10.1007/978-3-540-24714-2_14.

[36] P. H. Cheung, K. Hao, et F. Xie, « Component-Based Hardware/Software Co-Simulation », in 10th Euromicro Conference on Digital System Design Architectures, Methods and Tools (DSD 2007), Lubeck, Germany, août 2007, p. 265-270. doi: https://doi.org/10.1109/DSD.2007.4341479.

[37] Fei Xie, Guowu Yang, et Xiaoyu Song, « Component-based hardware/software co-verification », in Fourth ACM and IEEE International Conference on Formal Methods and Models for Co-Design, 2006. MEMOCODE '06. Proceedings., Napa, CA, USA, 2006, p. 27-36. doi: https://doi.org/10.1109/MEMCOD.2006.1695897.

[38] P. Arató, Z. Ádám Mann, et A. Orbán, « Extending component-based design with hardware components », Science of Computer Programming, vol. 56, nº 1-2, p. 23-39, avr. 2005, doi: https://doi.org/10.1016/j.scico.2004.11.003.

[39] N. T. Pilkington, J. Li, et F. Xie, « ESIDE: An Integrated Development Environment for Component-Based Embedded Systems », in 2009 33rd Annual IEEE International Computer Software and Applications Conference, Seattle, Washington, USA, 2009, p. 305-314. doi: https://doi.org/10.1109/COMPSAC.2009.48.

[40] Y. Mancer, « Hardware/Software codesign selon une approche d'architecture logicielle », Memoire de magister, Saad dahlab de blida, Blida, Algerie, 2012.

[41] D., Bolsens, Van Rompaey, K, Verkest, , I, et De Man, H, « CoWare-a design environment for heterogeneous hardware/software systems », in Proceedings EURO-DAC '96. European Design Automation Conference with EURO-VHDL '96 and Exhibition., Geneva, Switzerland, 1996, p. 252-257. doi: https://doi.org/10.1109/eurdac.1996.558213.

[42] R. K. Gupta et G. De Micheli, « Hardware-software cosynthesis for digital systems », IEEE Design & Test of Computers, vol. 10, nº 3, p. 29-41. doi: https://doi.org/10.1109/54.232470.

[43] R. Ernst, J. Henkel, et T. Benner, « Hardware-software cosynthesis for microcontrollers », IEEE Design & Test of Computers, vol. 10, nº 4, p. 64-75, 1993, doi: https://doi.org/10.1109/54.245964.

[44] R. Boumaza et F. Boutekkouk, « Intellectual Properties Integration: The Past, the Present, and the Future », International Journal of Technology Diffusion, vol. 12, nº 2, p. 32-45, avr. 2021, doi: https://doi.org/10.4018/IJTD.2021040103.

[45] B. Menhorn et F. Slomka, « Confirming the Design Gap », in Third International Conference on Computational Science, Engineering and Information Technology (CCSEIT-2013), D. Nagamalai, A. Kumar, et A. Annamalai, Éd., in Advances in Intelligent Systems and Computing, vol. 1. Konya, Turkey: Springer International Publishing, 2013, p. 281-292. doi: https://doi.org/10.1007/978-3-319-00951-3_27.

[46] L. P. Carloni, F. D. Bernardinis, C. Pinello, A. L. Sangiovanni-Vincentelli, et M. Sgroi, « Platform-Based Design for Embedded Systems », Embedded Systems Handbook, 1st ed, 2005.

[47] A. Ferrari et A. Sangiovanni-Vincentelli, « System Design: Traditional Concepts and New Paradigms », in International Conference on Computer Design: VLSI in Computers and Processors (Cat. No.99CB37040), Austin, TX, USA, 1999, p. 2-12. doi: https://doi.org/10.1109/iccd.1999.808256.

[48] M. A. Wehrmeister, L. B. Becker, F. R. Wagner, et C. E. Pereira, « An Object-Oriented Platform-based Design Process for Embedded Real-Time Systems », in Eighth IEEE

International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC'05), Seattle, WA, USA, 2005, p. 125-128. doi: https://doi.org/10.1109/ISORC.2005.13.

[49] F. Boutekkouk, M. Benmohammed, S. Bilavarn, et M. Auguin, « UML2.0 Profiles for Embedded Systems and Systems On a Chip (SOCs) », The Journal of Object Technology (JOT), vol. 8, nº 1, p. 135-157, 2009, doi: https://doi.org/10.5381/jot.2009.8.1.a1.

[50] V. Cortellessa, L. Pomante, et V. Stoico, « From UML/MARTE Specifications to ESL HW/SW Co-Design: Early Functional Verification and Timing Validation », in Companion of the 2023 ACM/SPEC International Conference on Performance Engineering, avr. 2023, p. 373-380. doi: https://doi.org/10.1145/3578245.3584850.

[51] A. Meliouh, « UML et Model-Checking pour la Modélisation et la Vérification des Systèmes Embarqués », Thèse, MOHAMED KHIDER BISKRA, Biskra, Algerie, 2021.

[52] S. M. Hezavehi, D. Weyns, P. Avgeriou, R. Calinescu, R. Mirandola, et D. Perez-Palacin, « Uncertainty in Self-adaptive Systems: A Research Community Perspective », ACM Transactions on Autonomous and Adaptive Systems (TAAS), vol. 15, nº 4, p. 1-36, 2021, doi: https://doi.org/10.1145/3487921.

[53] S. Mahdavi-Hezavehi, P. Avgeriou, et D. Weyns, « A Protocol for a Classification Framework of Uncertainty in Architecture-based Self-Adaptive Systems with Multiple Quality Requirements », Elsevier, 2015, [En ligne]. Disponible sur: https://research.rug.nl/en/publications/a-protocol-for-a-classification-framework-of-uncertainty-in-archi

[54] F. Boutekkouk, « Architecture Description Languages Taxonomies Review: A Special Focus on Self-Adaptive Distributed Embedded Systems », International Journal of Technology Diffusion (IJTD), vol. 12, nº 1, p. 53-74, janv. 2021, doi: https://doi.org/10.4018/IJTD.2021010103.

[55] D. Weyns, An Introduction to Self-Adaptive Systems: A Contemporary Software Engineering Perspective. John Wiley & Sons Ltd, 2021. doi: https://doi.org/10.1002/9781119574910.

[56] M. Zina et B. Fateh, « Comparative study between Multi Agents Systems methodologies according to intelligent embedded systems requirements », in 4th International Conference on Automation, Control Engineering and Computer Science (ACECS-2017), in Proceedings of Engineering and Technology–PET, vol. 20. Tangier, Morocco, 2017, p. 29-34. [En ligne]. Disponible sur: http://ipco-co.com/PET_Journal/Volume31_ACECS_2017.html

[57] R. Calinescu, C. Ghezzi, M. Kwiatkowska, et R. Mirandola, « Self-adaptive software needs quantitative verification at runtime », Communications of the ACM, vol. 55, nº 9, p. 69-77, sept. 2012, doi: https://doi.org/10.1145/2330667.2330686.

[58] D. Weyns et M. U. Iftikhar, « ActivFORMS: A Formally-Founded Model-Based Approach to Engineer Self-Adaptive Systems », 4 mars 2022, arXiv: arXiv:1908.11179. doi: https://doi.org/10.48550/arXiv.1908.11179.

[59] C. Trabelsi, « Contrôle matériel des systèmes partiellement reconfigurables sur

FPGA: de la modélisation à l'implémentation », Thèse, Université des Sciences et Technologie de Lille - Lille I, France, 2013. [En ligne]. Disponible sur: https://theses.hal.science/tel-00852361/

[60] E. Monmasson et M. N. Cirstea, « FPGA Design Methodology for Industrial Control Systems—A Review », IEEE Transactions on Industrial Electronics, vol. 54, nº 4, p. 1824-1842, août 2007, doi: https://doi.org/10.1109/TIE.2007.898281.

[61] X. ZHANG, « Contribution aux architectures adaptatives : étude de l'efficacité énergétique dans le cas des applications à parallélisme de données », Thèse, Nancy Universit2- Nancy 1, 2009. [En ligne]. Disponible sur: https://hal.univ-lorraine.fr/tel-01748353v1

[62] D. Weyns, « Software Engineering of Self-Adaptive Systems: An Organised Tour and Future Challenges », in Handbook of Software Engineering, 2017, p. 1-43.

[63] J. O. Kephart et D. M. Chess, « The vision of autonomic computing », Computer, vol. 36, nº 1, p. 41-50, janv. 2003, doi: https://doi.org/10.1109/MC.2003.1160055.

[64] D. Weyns et al., « Towards Better Adaptive Systems by Combining MAPE, Control Theory, and Machine Learning », in 2021 International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS), Madrid, Spain, mars 2021, p. 217-223. doi: https://doi.org/10.1109/SEAMS51251.2021.00036.

[65] M. Salehie et L. Tahvildari, « Self-adaptive software: Landscape and research challenges », ACM Transactions on Autonomous and Adaptive Systems (TAAS), vol. 4, nº 2, p. 1-42, mai 2009, doi: https://doi.org/10.1145/1516533.1516538.

[66] M. Wagner, A. Meroth, et D. Zöbel, « Developing self-adaptive automotive systems: On the integration of service-orientation into automotive development processes », Design Automation for Embedded Systems, vol. 18, p. 199-221, sept. 2014, doi: https://doi.org/10.1007/s10617-013-9124-3.

[67] I. Ghribi, R. B. Abdallah, M. Khalgui, Z. Li, K. Alnowibet, et M. Platzner, « R-Codesign: Codesign Methodology for Real-Time Reconfigurable Embedded Systems Under Energy Constraints », IEEE Access, vol. 6, p. 14078-14092, 2018, doi: https://doi.org/10.1109/ACCESS.2018.2799852.

[68] A. Burger, C. Cichiwskyj, S. Schmeißer, et G. Schiele, « The Elastic Internet of Things - A platform for self-integrating and self-adaptive IoT-systems with support for embedded adaptive hardware », Future Generation Computer Systems, vol. 113, p. 607-619, 2020, doi: https://doi.org/10.1016/j.future.2020.07.035.

[69] J.-P. Diguet, Y. Eustache, et G. Gogniat, « Closed-loop--based self-adaptive Hardware/Software-Embedded systems: Design methodology and smart cam case study », ACM Transactions on Embedded Computing Systems (TECS), vol. 10, nº 3, p. 1-28, avr. 2011, doi: https://doi.org/10.1145/1952522.1952531.

[70] D. M. R. Babu et Y. M. Roopa, « Component-based Self-adaptive Middleware Architecture for Networked Embedded Systems », International Journal of Applied Engineering Research, vol. 12, nº 12, p. 3029-3034, 2017.

[71] H. Nakagawa, S. Tsuchida, E. Tramontana, A. Fornaia, et T. Tsuchiya, « Embedded

System Evolution in IoT System Development Based on MAPE-K Loop Mechanism », 26 mai 2022, arXiv: arXiv:2205.13375. doi: https://doi.org/10.48550/arXiv.2205.13375.

[72] L. Fiack, B. Miramond, A. Upegui, et F. Vannel, « Dynamic parallel reconfiguration for self-adaptive hardware architectures », in 2014 NASA/ESA Conference on Adaptive Hardware and Systems (AHS), Leicester, United Kingdom, juill. 2014, p. 218-224. doi: https://doi.org/10.1109/AHS.2014.6880180.

[73] I. Sommerville, « Integrated requirements engineering: a tutorial », IEEE Software, vol. 22, nº 1, p. 16-23, janv. 2005, doi: https://doi.org/10.1109/MS.2005.13.

[74] IEEE Standard Glossary of Software Engineering Terminology -IEEE Std 610.12-1990, 1 décembre 1990. doi: https://doi.org/10.1109/IEEESTD.1990.101064.

[75] I. Sommerville, Software engineering, 9th ed. America: Pearson Education Inc, 2011.

[76] S. Anwer et N. Ikram, « Goal Oriented Requirement Engineering: A Critical Study of Techniques », in 2006 13th Asia Pacific Software Engineering Conference (APSEC'06), Bangalore, India, 2006, p. 121-130. doi: https://doi.org/10.1109/APSEC.2006.38.

[77] M. Amroune, « Vers une approche orientée aspect d'ingénierie des besoins dans les organisations multi-entreprises », Thèse, Université Toulouse le Mirail-Toulouse II, France, 2014. [En ligne]. Disponible sur: https://theses.hal.science/tel-01176448/

[78] J. Horkoff et al., « Goal-oriented requirements engineering: an extended systematic mapping study », Requirements Engineering, p. 133-160, 2019. doi: https://doi.org/10.1007/s00766-017-0280-z.

[79] C. Rolland, C. Souveyet, et C. B. Achour, « Guiding goal modeling using scenarios », IEEE Transactions on Software Engineering, vol. 24, nº 12, p. 1055-1071, déc. 1998, doi: https://doi.org/10.1109/32.738339.

[80] C. Potts, K. Takahashi, et A. I. Anton, « Inquiry-based requirements analysis », IEEE Software, vol. 11, nº 2, p. 21-32, mars 1994, doi: https://doi.org/10.1109/52.268952.

[81] K. S. Rubin et A. Goldberg, « Object behavior analysis », Communications of the ACM, vol. 35, nº 9, p. 48-62, sept. 1992, doi: https://doi.org/10.1145/130994.130996.

[82] M. Glinz, « An integrated formal model of scenarios based on statecharts », in Software Engineering — ESEC '95, vol. 989, W. Schäfer et P. Botella, Éd., Springer Berlin Heidelberg, 1995, p. 254-271. doi: https://doi.org/10.1007/3-540-60406-5_19.

[83] S. Faßbender, M. Heisel, et R. Meis, « Aspect-oriented Requirements Engineering with Problem Frames », in Proceedings of the 9th International Conference on Software Paradigm Trends, Vienna, Austria, 2014, p. 145-156. doi: https://doi.org/10.5220/0005001801450156.

[84] A. Rashid, P. Sawyer, A. Moreira, et J. Araujo, « Early aspects: a model for aspect-

oriented requirements engineering », in Proceedings IEEE Joint International Conference on Requirements Engineering, Essen, Germany, 2002, p. 199-202. doi: https://doi.org/10.1109/ICRE.2002.1048526.

[85] Yijun Yu, J. C. Sampai Do Prado Leite, et J. Mylopoulos, « From goals to aspects: discovering aspects from requirements goal models », in Proceedings. 12th IEEE International Requirements Engineering Conference, 2004., Kyoto, Japan: IEEE, 2004, p. 38-47. doi: https://doi.org/10.1109/ICRE.2004.1335662.

[86] G. Kotonya et I. Sommerville, « Requirements engineering with viewpoints », Software Engineering Journal, vol. 11, nº 1, 1996, doi: https://doi.org/10.1049/sej.1996.0002.

[87] R. Laney, L. Barroca, M. Jackson, et B. Nuseibeh, « Composing requirements using problem frames », in Proceedings. 12th IEEE International Requirements Engineering Conference, 2004., Kyoto, Japan, 2004, p. 122-131. doi: https://doi.org/10.1109/ICRE.2004.1335670.

[88] C. Rolland, « Ingénierie des Besoins: L'Approche L'Ecritoire », Techniques de l'Ingenieur, p. 1-45, 2003.

[89] S. Supakkul et L. Chung, « The RE-Tools: A multi-notational requirements modeling toolkit », in 2012 20th IEEE International Requirements Engineering Conference (RE), Chicago, IL, USA, sept. 2012, p. 333-334. doi: https://doi.org/10.1109/RE.2012.6345831.

[90] D. A. Neto, « A Requirements Specification Template of a Communication Network Based on CAN Protocol to Automotive Embedded Systems », Journal of Computer Science and Technology, vol. 10, nº 3, p. 143-149, 2010.

[91] E. Andrade, P. Maciel, G. Callou, et B. Nogueira, « A Methodology for Mapping SysML Activity Diagram to Time Petri Net for Requirement Validation of Embedded Real-Time Systems with Energy Constraints », in 2009 Third International Conference on Digital Society, Cancun, Mexico, 2009, p. 266-271. doi: https://doi.org/10.1109/ICDS.2009.19.

[92] L. E. G. Martins et R. de S. Júnior, « TERASE: Template para Especificação de Requisitos de Ambiente em Sistemas Embarcados », in Workshop em Engenharia de Requisitos (WER'2010), 2010.

[93] J. C. Ossada, L. E. G. Martins, B. S. Ranieri, et A. Belgamo, « GERSE: Guia de Elicitação de Requisitos para Sistemas Embarcados », in Workshop em Engenharia de Requisitos (WER'2012), 2012.

[94] Y. Roudier, M. S. Idrees, et L. Apvrille, « Towards the model-driven engineering of security requirements for embedded systems », in 2013 3rd International Workshop on Model-Driven Requirements Engineering (MoDRE), Rio de Janeiro, Brazil, juill. 2013, p. 55-64. doi: https://doi.org/10.1109/MoDRE.2013.6597264.

[95] S. Wiesner, J. B. Hauge, F. Haase, et K.-D. Thoben, « Supporting the Requirements Elicitation Process for Cyber-Physical Product-Service Systems Through a Gamified Approach », in Initiatives for a Sustainable World: IFIP WG 5.7 International Conference, APMS 2016, I. Nääs, O. Vendrametto, J. Mendes Reis, R. F. Gonçalves, M. T. Silva, G. von Cieminski, et D. Kiritsis, Éd., in Advances in

Production Management Systems. Initiatives for a Sustainable World., vol. 488. 2016, p. 687-694. doi: https://doi.org/10.1007/978-3-319-51133-7_81.

[96] T. Pereira, A. Sousa, R. Oliveira, D. Albuquerque, F. Alencar, et J. Castro, « A Metamodel to Guide a Requirements Elicitation Process for Embedded Systems », in 2018 11th International Conference on the Quality of Information and Communications Technology (QUATIC), Coimbra, Portugal, 2018, p. 101-109. doi: https://doi.org/10.1109/QUATIC.2018.00023.

[97] D. Iqbal, A. Abbas, M. Ali, M. U. S. Khan, et R. Nawaz, « Requirement Validation for Embedded Systems in Automotive Industry Through Modeling », IEEE Access, vol. 8, p. 8697-8719, 2020, doi: https://doi.org/10.1109/ACCESS.2019.2963774.

[98] A. Manzoor et B. Jean-Michel, « A Comparative Study of RELAX and SysMLKAOS », Rapport de recherche, : Institut de Recherche Informatique de Toulouse, France, 2014. [En ligne]. Disponible sur: https://hal.science/hal-03260606v1

[99] M. Ahmad, J.-M. Bruel, R. Laleau, et C. Gnaho, « Using RELAX, SysML and KAOS for Ambient Systems Requirements Modeling », Procedia Computer Science, vol. 10, p. 474-481, 2012, doi: https://doi.org/10.1016/j.procs.2012.06.061.

[100] V. E. Silva Souza, A. Lapouchnian, W. N. Robinson, et J. Mylopoulos, « Awareness requirements for adaptive systems », in Proceedings of the 6th International Symposium on Software Engineering for Adaptive and Self-Managing Systems, Waikiki, Honolulu HI USA: ACM, mai 2011, p. 60-69. doi: https://doi.org/10.1145/1988008.1988018.

[101] V. E. S. Souza, A. Lapouchnian, K. Angelopoulos, et J. Mylopoulos, « Requirements-driven software evolution », Computer Science - Research and Development, vol. 28, p. 311-329, 2013, doi: https://doi.org/10.1007/s00450-012-0232-2.

[102] I. Ayala, M. Amor, et L. Fuentes, « Analysing Requirements Specification Languages for Self-adaptive AAL Systems », in Proceedings of the 15th International Conference on Ubiquitous Computing & Ambient Intelligence (UCAmI 2023), vol. 842, J. Bravo et G. Urzáiz, Éd., in Lecture Notes in Networks and Systems, vol. 842. , Springer, Cham, 2023, p. 143-154. doi: https://doi.org/10.1007/978-3-031-48642-5_14.

[103] S. Irish et L. Seok-Won, « Self-adaptive requirements for intelligent transportation system: A case study », in 2017 International Conference on Information and Communication Technology Convergence (ICTC), Jeju, Korea (South), 2017, p. 520-526. doi: https://doi.org/10.1109/ICTC.2017.8191032.

[104] C. J. Neill et P. A. Laplante, « Requirements engineering: The state of the practice », IEEE Software, vol. 20, nº 6, p. 40-45, nov. 2003, doi: https://doi.org/10.1109/MS.2003.1241365.

[105] P. Parviainen et M. Tihinen, « A SURVEY OF EXISTING REQUIREMENTS ENGINEERING TECHNOLOGIES AND THEIR COVERAGE », Int. J. Soft. Eng. Knowl. Eng., vol. 17, nº 06, p. 827-850, déc. 2007, doi: https://doi.org/10.1142/S0218194007003513.

[106] E. Sikora, B. Tenbergen, et K. Pohl, « Requirements Engineering for Embedded Systems: An Investigation of Industry Needs », in Requirements Engineering:

Foundation for Software Quality (REFSQ 2011), vol. 6606, D. Berry et X. Franch, Éd., in Lecture Notes in Computer Science, vol. 6606. , Springer Berlin Heidelberg, 2011, p. 151-165. doi: https://doi.org/10.1007/978-3-642-19858-8_16.

[107] L. E. G. Martins, J. C. Ossada, et A. Belgamo, « Towards Requirements Engineering Process for Embedded Systems », in ER@ BR, 2013.

[108] Z. Mecibah et F. Boutekkouk, « Requirements Metamodeling for Self-Adaptive Embedded Systems », International Journal of Software Innovation (IJSI), vol. 10, nº 1, p. 1-24, oct. 2022, doi: https://doi.org/10.4018/IJSI.311508.

[109] B. Kitchenham et S. Charters, « Guidelines for performing systematic literature reviews in software engineering », EBSE Technical Report-Version 2.3, School of Computer Science and Mathematics Keele University, 2007.

[110] Z. Pamela et T. Y. Raymond, « Executable requirements for embedded systems », in In Proceedings of the 5th international conference on Software engineering (ICSE '81), 1981, p. 295-304.

[111] P. Zave, « An Operational Approach to Requirements Specification for Embedded Systems », IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, vol. SE-8, nº 3, p. 250-269, 1982, doi: https://doi.org/10.1109/TSE.1982.235254.

[112] M. Winokur, J. Z.Lavi, I. Lavi, et R. Oz, « Requirements Analysis and Specification of Embedded Systems using ECSAM - a Case Study », in COMPEURO'90: Procedings of the IEEE International Conference on Computer Systems and Software Engineering-Systems Engineering Aspects of Complex Computerized Systems, 1990, p. 80-89. doi: https://doi.org/10.1109/CMPEUR.1990.113611.

[113] S. J. Cunning et J. W. Rozenblit, « Automatic test case generation from requirements specifications for real-time embedded systems », in IEEE SMC'99 Conference Proceedings. 1999 IEEE International Conference on Systems, Man, and Cybernetics (Cat. No.99CH37028), Tokyo, Japan, 1999, p. 784-789. doi: https://doi.org/10.1109/ICSMC.1999.815651.

[114] B. Manfred et S. Oscar, « From Requirements to Validated Embedded Systems », in the First International Workshop on Embedded Software, in Lecture Notes in Computer Science, vol. 2211. Tahoe City, CA, USA: Springer, Berlin, Heidelberg, 2001, p. 51-65. doi: https://doi.org/10.1007/3-540-45449-7_5.

[115] N. Kececi, W. A. Halang, et A. Abran, « A Semi-Formal Method to Verify Correctness of Functional Requirements Specifications of Complex Systems », in Design and Analysis of Distributed Embedded Systems. DIPES 2002, vol. 91, B. Kleinjohann, K. H. Kim, L. Kleinjohann, et A. Rettberg, Éd., Springer, Boston, MA, 2002, p. 61-69. doi: https://doi.org/10.1007/978-0-387-35599-3_7.

[116] A. von Knethen, « Change-oriented requirements traceability. Support for evolution of embedded systems », in International Conference on Software Maintenance, 2002. Proceedings., Montreal, QC, Canada, 2002, p. 482-485. doi: https://doi.org/10.1109/ICSM.2002.1167808.

[117] E. Nasr, J. McDermid, et G. Bernat, « Eliciting and specifying requirements with use cases for embedded systems », in Proceedings of the Seventh IEEE International Workshop on Object-Oriented Real-Time Dependable Systems. (WORDS 2002),

San Diego, CA, USA: IEEE Comput. Soc, 2002, p. 350-357. doi: https://doi.org/10.1109/WORDS.2002.1000073.

[118] M. von der Beeck, P. Braun, M. Rappl, et C. Schroder, « Model based requirements engineering for embedded software », in Proceedings IEEE Joint International Conference on Requirements Engineering, Essen, Germany, 2002, p. Requirements engineering in the development of innovative automotive embedded software systems. doi: https://doi.org/10.1109/ICRE.2002.1048510.

[119] A. Puschnig et R. Tavakoli Kolagari, « Requirements engineering in the development of innovative automotive embedded software systems », in Proceedings. 12th IEEE International Requirements Engineering Conference, 2004., Kyoto, Japan, 2004, p. 328-333. doi: https://doi.org/10.1109/ICRE.2004.1335691.

[120] J. Jaalinoja, « Requirements Implementation in Embedded Software development », in VIT Publications: Espoo, Finland, 2004.

[121] A. Fleischmann, J. Hartmann, C. Pfaller, M. Rappl, S. Rittmann, et D. Wild, « Concretization and Formalization of Requirements for Automotive Embedded Software Systems Development », in Proceedings of the Tenth Australian Workshop on Requirememements Engineering (AWRE'05), 2005, p. 60-65.

[122] L. M. Jose et S. Andrés, Requirements Engineering For Sociotechnical Systems. IGI Global, 2005. doi: https://doi.org/10.4018/978-1-59140-506-1.

[123] I. Krüger, C. Farcas, E. Farcas, et M. Menarini, « 7 Requirements Modeling for Embedded Realtime Systems », in Model-Based Engineering of Embedded Real-Time Systems MBEERTS 2007, vol. 6100, H. Giese, G. Karsai, E. Lee, B. Rumpe, et B. Schätz, Éd., in Lecture Notes in Computer Science, vol. 6100. , Springer Berlin Heidelberg, 2010, p. 155-199. doi: https://doi.org/10.1007/978-3-642-16277-0_7.

[124] E. P. Freitas, M. A. Wehrmeister, C. E. Pereira, F. R. Wagner, E. T. Silva, et F. C. Carvalho, « Using Aspect-Oriented Concepts in the Requirements Analysis of Distributed Real-Time Embedded Systems », in Embedded System Design: Topics, Techniques and Trends. IFIP – The International Federation for Information Processing, vol. 231, A. Rettberg, M. C. Zanella, R. Dömer, A. Gerstlauer, et F. J. Rammig, Éd., Springer, Boston, MA, 2007, p. 221-230. doi: https://doi.org/10.1007/978-0-387-72258-0_19.

[125] A. Albinet, J.-L. Boulanger, H. Dubois, M.-A. Peraldi-Frati, Y. Sorel, et Q.-D. Van, « Model-Based Methodology for Requirements Traceability in Embedded Systems », in Proceedings of 3rd European Conference on Model Driven Architecture® Foundations and Applications, ECMDA'07, 2007.

[126] H. Le Dang, H. Dubois, et S. Gérard, « Towards a traceability model in a MARTE-based methodology for real-time embedded systems », Innovations Syst Softw Eng, vol. 4, p. 189-193, 2008, doi: https://doi.org/10.1007/s11334-008-0053-4.

[127] S. Markose, Xiaoqing Liu, et B. McMillin, « A Systematic Framework for Structured Object-Oriented Security Requirements Analysis in Embedded Systems », in 2008 IEEE/IFIP International Conference on Embedded and Ubiquitous Computing,

Shanghai, China, déc. 2008, p. 75-81. doi: https://doi.org/10.1109/EUC.2008.92.

[128] H. Espinoza, D. Cancila, B. Selic, et S. Gérard, « Challenges in Combining SysML and MARTE for Model-Based Design of Embedded Systems », in Model Driven Architecture - Foundations and Applications, vol. 5562, R. F. Paige, A. Hartman, et A. Rensink, Éd., in Lecture Notes in Computer Science, vol. 5562. , Springer, Berlin, Heidelberg, 2009, p. 98-113. doi: https://doi.org/10.1007/978-3-642-02674-4_8.

[129] C. André et F. Mallet, « Specification and verification of time requirements with CCSL and Esterel », ACM SIGPLAN Notices, vol. 44, nº 7, p. 167-176, 2009, doi: https://doi.org/10.1145/1543136.1542475.

[130] D. Bergsjö, L. Almefelt, et J. Malmqvist, « SUPPORTING REQUIREMENTS MANAGEMENT IN EMBEDDED SYSTEMS DEVELOPMENT IN A LEAN-INFLUENCED ORGANIZATION », in DS 60: Proceedings of DESIGN 2010, 11th International Design Conference, Dubrovnik, Croatia., 2010, p. 1025-1034. [En ligne]. Disponible sur: https://human.designsociety.org/publication/29448/SUPPORTING+REQURIEMENTS+MANAGEMENT+IN+EMBEDDED+SYSTEMS+DEVELOPMENT+IN+A+LEAN-INFLUENCED+ORGANIZATION

[131] H. Dubois, M.-A. Peraldi-Frati, et F. Lakhal, « A Model for Requirements Traceability in a Heterogeneous Model-Based Design Process: Application to Automotive Embedded Systems », in 2010 15th IEEE International Conference on Engineering of Complex Computer Systems, Oxford, United Kingdom, mars 2010, p. 233-242. doi: https://doi.org/10.1109/ICECCS.2010.2.

[132] T. Arpinen, T. D. Hämäläinen, et M. Hännikäinen, « Meta-Model and UML Profile for Requirements Management of Software and Embedded Systems », EURASIP Journal on Embedded Systems, p. 1-14, 2011, doi: https://doi.org/10.1155/2011/592168.

[133] S. Farfeleder, T. Moser, A. Krall, T. Stalhane, H. Zojer, et C. Panis, « DODT: Increasing requirements formalism using domain ontologies for improved embedded systems development », in 14th IEEE International Symposium on Design and Diagnostics of Electronic Circuits and Systems, Cottbus, Germany, 2011, p. 271-274. doi: https://doi.org/10.1109/DDECS.2011.5783092.

[134] K. K. Fletcher et X. Liu, « Security Requirements Analysis, Specification, Prioritization and Policy Development in Cyber-Physical Systems », in 2011 Fifth International Conference on Secure Software Integration and Reliability Improvement - Companion, Jeju, Korea (South), 2011, p. 106-113. doi: https://doi.org/10.1109/SSIRI-C.2011.25.

[135] F. Stefan, « Requirements Specification and Analysis for Embedded Systems », the Vienna University of Technology, 2012. [En ligne]. Disponible sur: https://www.researchgate.net/profile/Stefan-Farfeleder/publication/258341008_Requirements_Specification_and_Analysis_for_Embedded_Systems/links/00b49527ea1f7ac43d000000/Requirements-Specification-and-Analysis-for-Embedded-Systems.pdf

[136] E. Sikora, B. Tenbergen, et K. Pohl, « Industry needs and research directions in requirements engineering for embedded systems », Requirements Engineering,

vol. 17, p. 57-78, mars 2012, doi: https://doi.org/10.1007/s00766-011-0144-x.

[137] M. S. Idrees, « A requirement engineering driven approach to security architecture design for distributed embedded systems », Télécom ParisTech, 2012. [En ligne]. Disponible sur: https://theses.hal.science/tel-01251856

[138] L. E. G. Martins, J. C. Ossada, A. Belgamo, et B. S. Ranieri, « Requirements Elicitation Guide for Embedded Systems: An Industry Challenge », in ICSEA 2013 : The Eighth International Conference on Software Engineering Advances, 2013, p. 106-111.

[139] B. Dominique, « Modeling languages for requirements engineering and quantitative analysis of embedded systems », 2013. [En ligne]. Disponible sur: https://api.semanticscholar.org/CorpusID:60154095

[140] M. P. E. Heimdahl, L. Duan, A. Murugesan, et S. Rayadurgam, « Modeling and requirements on the physical side of cyber-physical systems », in 2013 2nd International Workshop on the Twin Peaks of Requirements and Architecture (TwinPeaks), San Francisco, CA, USA, 2013, p. 1-7. doi: https://doi.org/10.1109/TwinPeaks.2013.6614716.

[141] A. Murugesan, S. Rayadurgam, et M. Heimdahl, « Using Models to Address Challenges in Specifying Requirements for Medical Cyber-Physical Systems », in Fourth workshop on Medical Cyber-Physical Systems., 2013.

[142] M. A. Wehrmeister, C. E. Pereira, et F. J. Rammig, « Aspect-Oriented Model-Driven Engineering for Embedded Systems Applied to Automation Systems », IEEE Transactions on Industrial Informatics, vol. 9, nº 4, p. 2373-2386, nov. 2013, doi: https://doi.org/10.1109/TII.2013.2240308.

[143] S. Teufl, M. Khalil, et D. Mou, « Requirements for a Model-based Requirements Engineering Tool for Embedded Systems: Systematic Literature Review and Survey », White paper. fortiss GmbH, 2013.

[144] D. Aceituna, « Survey of Concerns in Embedded Systems Requirements Engineering », SAE International Journal of Passenger Cars - Electronic and Electrical Systems, vol. 7, nº 1, p. 1-13, 2014, doi: https://doi.org/10.4271/2013-01-2403.

[145] Jiale Zhou, « Requirements development and management of embedded real-time systems », in 2014 IEEE 22nd International Requirements Engineering Conference (RE), Karlskrona, Sweden, août 2014, p. 479-484. doi: https://doi.org/10.1109/RE.2014.6912302.

[146] S. Wiesner, C. Gorldt, M. Soeken, K.-D. Thoben, et R. Drechsler, « Requirements Engineering for Cyber-Physical Systems: Challenges in the Context of "Industrie 4.0" », in Advances in Production Management Systems. Innovative and Knowledge-Based Production Management in a Global-Local World: IFIP WG 5.7 International Conference, APMS 2014, Ajaccio, France, 2014, p. 281-288. doi: https://doi.org/10.1007/978-3-662-44739-0_35.

[147] M. R. Sena Marques, E. Siegert, et L. Brisolara, « Integrating UML, MARTE and sysml to improve requirements specification and traceability in the embedded domain », in 2014 12th IEEE International Conference on Industrial Informatics (INDIN), Porto Alegre RS, Brazil, 2014, p. 176-181. doi:

https://doi.org/10.1109/INDIN.2014.6945504.

[148] M. Daun, T. Weyer, et K. Pohl, « Detecting and Correcting Outdated Requirements in Function-Centered Engineering of Embedded Systems », in Requirements Engineering: Foundation for Software Quality: 21st International Working Conference, REFSQ 2015, Essen, Germany, 2015, p. 65-80. doi: https://doi.org/10.1007/978-3-319-16101-3_5.

[149] S. Wiesner, J. B. Hauge, et K.-D. Thoben, « Challenges for Requirements Engineering of Cyber-Physical Systems in Distributed Environments », in Advances in Production Management Systems: Innovative Production Management Towards Sustainable Growth: IFIP WG 5.7 International Conference, APMS 2015, in IFIP Advances in Information and Communication Technology, vol. 460. Tokyo, Japan: Springer International Publishing, 2015, p. 49-58. doi: https://doi.org/10.1007/978-3-319-22759-7_6.

[150] J. Eriksson, « Formal Requirement Models for Automotive Embedded Systems », 2016.

[151] H. Reza, C. Korvald, J. Straub, J. Hubber, N. Alexander, et A. Chawla, « Toward requirements engineering of cyber-physical systems: Modeling CubeSat », in 2016 IEEE Aerospace Conference, Big Sky, MT, USA, mars 2016, p. 1-13. doi: https://doi.org/10.1109/AERO.2016.7500897.

[152] M. Rashid, M. W. Anwar, F. Azam, et M. Kashif, « Model-based requirements and properties specifications trends for early design verification of embedded systems », in 2016 11th System of Systems Engineering Conference (SoSE), Kongsberg, Norway, juin 2016, p. 1-7. doi: https://doi.org/10.1109/SYSOSE.2016.7542917.

[153] M. M. Rahman et N. Nower, « Requirements Model for Cyber-Physical System », 8 mai 2017, arXiv preprint: arXiv:1705.03095.

[154] B. Lebeaupin, « Vers un langage de haut niveau pour une ingénierie des exigences agile dans le domaine des systèmes embarqués avioniques », Paris-Saclay University (COmUE), 2017. [En ligne]. Disponible sur: https://theses.hal.science/tel-01761690/file/74169_LEBEAUPIN_2017_archivage.pdf

[155] T. Pereira, D. Albuquerque, A. Sousa, F. Alencar, et J. Castro, « Retrospective and Trends in Requirements Engineering for Embedded Systems: A Systematic Literature Review », in Workshop em Engenharia de Requisitos (WER'2017), 2017, p. 427-440.

[156] S. ur Rehman, C. Allgaier, et V. Gruhn, « Security Requirements Engineering: A Framework for Cyber-Physical Systems », in 2018 International Conference on Frontiers of Information Technology (FIT), Islamabad, Pakistan, déc. 2018, p. 315-320. doi: https://doi.org/10.1109/FIT.2018.00062.

[157] S. Pandey, S. Pokharel, et H. Reza, « Towards Cyber-Physical Requirement Engineering Elicitation Tool Support », in 2018 World Automation Congress (WAC), Stevenson, WA, USA, 2018, p. 1-5. doi: https://doi.org/10.23919/WAC.2018.8430399.

[158] S. Rehman et V. Gruhn, « An Effective Security Requirements Engineering Framework for Cyber-Physical Systems », Technologies, vol. 6, nº 3, p. 65, 2018, doi: https://doi.org/10.3390/technologies6030065.

[159] S. Park, « Software Requirement Specification Based on a Gray Box for Embedded Systems: A Case Study of a Mobile Phone Camera Sensor Controller », Computers, vol. 8, nº 1, p. 20, 2019, doi: https://doi.org/10.3390/computers8010020.

[160] M. Staron, « Requirements Engineering for Automotive Embedded Systems », in Automotive Systems and Software Engineering, 2019, p. 11-28. doi: https://doi.org/10.1007/978-3-030-12157-0_2.

[161] T. Pereira, Q. Ribeiro, M. Melo, S. Magro, et J. Castro, « Requirements Engineering for Embedded Systems: A Systematic Literature Review », in WER'2021, 2021.

[162] W. Xiaoqi, C. Xiaohong, Y. Xiao, et Y. Bo, « Requirements Patterns for Complex Embedded Systems », in 2022 IEEE 30th International Requirements Engineering Conference Workshops (REW), Melbourne, Australia, 2022, p. 14-17. doi: https://doi.org/10.1109/REW56159.2022.00011.

[163] W. Chunhui, H. Lu, et C. Xiaohong, « Extracting Requirements Models from Natural-Language Document for Embedded Systems », in 2022 IEEE 30th International Requirements Engineering Conference Workshops (REW), Melbourne, Australia, 2022, p. 18-21. doi: https://doi.org/10.1109/REW56159.2022.00012.

[164] D. Bouskela et al., « Formal requirements modeling for cyber-physical systems engineering: an integrated solution based on FORM-L and Modelica », Requirements Engineering, vol. 27, nº 1, p. 1-30, 2022, doi: https://doi.org/10.1007/s00766-021-00359-z.

[165] F. Zahid, A. Tanveer, M. M. Y. Kuo, et R. Sinha, « A systematic mapping of semi-formal and formal methods in requirements engineering of industrial Cyber-Physical systems », Journal of Intelligent Manufacturing, vol. 33, p. 1603-1638, 2022, doi: https://doi.org/10.1007/s10845-021-01753-8.

[166] M. Y. Chow, « Analysis of Embedded System's Functional Requirement using BERT-based Name Entity Recognition for Extracting IO Entities », Journal of Information Processing, vol. 31, p. 143-153, 2023, doi: https://doi.org/10.2197/ipsjjip.31.143.

[167] R. Kun, C. Xiaohong, et J. Zhi, « Requirements Modeling Aided by ChatGPT: An Experience in Embedded Systems », in 2023 IEEE 31st International Requirements Engineering Conference Workshops (REW), Hannover, Germany, 2023, p. 170-177. doi: https://doi.org/10.1109/REW57809.2023.00035.

[168] Y. Xiao, C. Xiaohong, et W. Jiangtao, « A Model Checking Based Software Requirements Specification Approach for Embedded Systems », in 2023 IEEE 31st International Requirements Engineering Conference Workshops (REW), Hannover, Germany, 2023, p. 184-191. doi: https://doi.org/10.1109/REW57809.2023.00037.

[169] R. Fabíola Gonçalves C, R. Achim, P. Carlos E, S. Charles, et S. Michel S, « A Proposal to Trace and Maintain Requirements Constraints of Real-time Embedded Systems

SPRINGER », in Analysis, Estimations, and Applications of Embedded Systems. IESS 2019. IFIP Advances in Information and Communication Technology, Springer, Cham, 2023, p. 15-26. doi: https://doi.org/10.1007/978-3-031-26500-6_2.

[170] F. Asma, A. Sanaa, et A. Akramul, « Towards Requirements Specification Collaboration Forum for Embedded Software Systems », in 023 ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C), Västerås, Sweden, 2023, p. 312-317. doi: https://doi.org/10.1109/MODELS-C59198.2023.00061.

[171] W. Chunhui, Z. Jiaqi, C. Xiaohong, et J. Zhi, « Generating Requirements Documents for Embedded Systems: A Device Knowledge-Guided Approach », in 2024 IEEE 32nd International Requirements Engineering Conference Workshops (REW), Reykjavik, Iceland, 2024, p. 200-205. doi: https://doi.org/10.1109/REW61692.2024.00032.

[172] F. Asma, A. Sanaa, et A. Akramul, « A Systematic Literature Review on Requirements Engineering and Maintenance for Embedded Software », IEEE Access, vol. 12, p. 114263-114279, 2024, doi: https://doi.org/10.1109/ACCESS.2024.3443271.

[173] P. Tarcisio, A. Deivson, S. Aeda, A. Fernanda, et C. Jaelson, « Towards a Metamodel for a Requirements Engineering Process of Embedded Systems », in 2016 VI Brazilian Symposium on computing Systems Engineering (SBESC), João Pessoa, Brazil, 2016, p. 93-100. doi: https://doi.org/10.1109/SBESC.2016.022.

[174] Y. Abuseta et K. Swesi, « Design Patterns for Self Adaptive Systems Engineering », International Journal of Software Engineering & Applications (IJSEA), vol. 6, nº 4, p. 11-28, 2015, doi: https://doi.org/10.5121/ijsea.2015.6402.

[175] M. Li, F. Batmaz, L. Guan, A. Grigg, M. Ingham, et P. Bull, « Model-based systems engineering with requirements variability for embedded real-time systems », in 2015 IEEE International Model-Driven Requirements Engineering Workshop (MoDRE), Ottawa, ON, Canada, 2015, p. 1-10. doi: https://doi.org/10.1109/MoDRE.2015.7343874.

[176] M. Nguyen, N. Thi, T. Loan, A. Pettersson, A. Gomes, et K. Tejle, « Requirements Engineering Process Maturity Model Uni--REPM », Blekinge Institue of Technology, Karlskrona, Sweden, 2011.

[177] T. Gorschek et K. Tejle, « A Method for Assessing Requirements Engineering Process Maturity in Software Projects », 2002.

[178] S. Moon, K. H. Lee, et D. Lee, « Fuzzy Branching Temporal Logic », IEEE Trans. Syst., Man, Cybern. B, vol. 34, nº 2, p. 1045-1055, avr. 2004, doi: https://doi.org/10.1109/TSMCB.2003.819485.

[179] F. Boutekkouk, « AI-Based Methods to Resolve Real-Time Scheduling for Embedded Systems: A Review », International Journal of Cognitive Informatics and Natural Intelligence, vol. 15, nº 4, p. 1-44, 2022, doi: https://doi.org/10.4018/IJCINI.290308.

[180] F. Boutekkouk, « Application of a Fuzzy MCDM Method to Select the Best

Operating System for an Efficient Security-Aware Design of Embedded Systems »:, International Journal of Applied Evolutionary Computation, vol. 12, nº 3, p. 1-20, juill. 2021, doi: https://doi.org/10.4018/IJAEC.2021070101.

[181] F. Boutekkouk, « Embedded systems codesign under artificial intelligence perspective: a review », International Journal of Ad Hoc and Ubiquitous Computing, vol. 32, nº 4, p. 257, 2019, doi: https://doi.org/10.1504/IJAHUC.2019.103265.

[182] R. Mehalaine et F. Boutekkouk, « Energy Consumption Reduction in Real Time Multiprocessor Embedded Systems with Uncertain Data », in Computer Science On-line Conference (CSOC), 2020, p. 46-55. doi: https://doi.org/10.1007/978-3-030-51971-1_4.

[183] R. Mehalaine et F. Boutekkouk, « A New Intelligent Biologically-Inspired Model for Fault Tolerance in Distributed Embedded Systems », International Journal of Embedded and Real-Time Communication Systems, vol. 11, nº 3, p. 22-47, juill. 2020, doi: https://doi.org/10.4018/IJERTCS.2020070102.

Communications & Publications

1. **MECIBAH Zina**, BOUTEKKOUK Fateh, (2017). *«Comparative study between Multi Agents Systems methodologies according to intelligent embedded systems requirements»*. International conference on Automation, Control Engineering and Computer Science (ACECS), Proceeding of Engineering and Technology PET, Vol.20 pp.29-34, March 2017, tangier, Morocco.

   URL : http://ipco-co.com/PET_Journal/Volume31_ACECS_2017.html

2. **MECIBAH Zina**, BOUTEKKOUK Fateh, (2018). *«Towards Requirements Engineering Process for Self-Adaptive Embedded Systems»*, 7th Computer Science On-line Conference (CSOC), Proceeding of Springer: Advances in Intelligent Systems and Computing - ISSN 2194-5357, vol 763. pp 338–345. Springer, Cham.

   Doi: https://doi.org/10.1007/978-3-319-91186-1_35.

   URL : https://link.springer.com/chapter/10.1007/978-3-319-91186-1_35

3. Fateh Boutekkouk, Ridha Mahalaine, **Zina Mecibah,** Saliha Lakhdari, Ramissa Djouani and Djalila Belkebir, (2018). *«Intelligent Embedded Software: New Perspectives and Challenges»,* Intelligent System Chatchawal Wongchoosuk, IntechOpen.

   Doi: https://doi.org/10.5772/intechopen.72417.

   URL: https://www.intechopen.com/books/intelligent-system/intelligent-embedded-software-new-perspectives-and-challenges

4. **MECIBAH, Zina**, BOUTEKKOUK, Fateh, (2019). *«Metamodel for a Requirements Engineering Process of Self Adaptive Embedded Systems-First Version»,* International training days of Science and Engineering (ITDSE 2019), December 15, 2019-December 17, 2019, ISTA Ain Mlila, Algeria.

5. **MECIBAH, Zina**, BOUTEKKOUK, Fateh, (2022). *«Requirements Metamodeling for Self-Adaptive Embedded Systems»,* International journal of software innovation (IJSI), Volume 10, Issue 1, Article 134.

Doi: https://doi.org/10.4018/IJSI.311508

URL: https://www.igi-global.com/article/requirements-metamodeling-for-self-adaptive-embedded-systems/311508

6. **MECIBAH Zina,** BOUTEKKOUK Fateh, (2024). *« Designing of Self-Adaptive Embedded System: an Overview of Recent Advances and Prospects»*, la 1ère Conférence Nationale (en ligne) sur l'Application d'Intelligence Artificielle et le Développement Durable (1ère CNAIADD'24), 17-18 Avril 2024, Centre Universitaire Nour Bachir El-Bayadh, Institut des sciences & Laboratoire des Systèmes Electroniques, Télécommunications et Energies Renouvelables (LSETER), El-Bayadh, Algérie.

   URL : https://sites.google.com/view/cnaiadd/papers-presented

7. **MECIBAH Zina,** BOUTEKKOUK Fateh, (2024). *«Challenges and Opportunities in Applying Artificial Intelligence Techniques to Requirements Engineering for Self-Adaptive Embedded Systems»*, The 1st National Conference on Artificial Intelligence and Information Technology (NCAIIT'2024), 8-8 Decembre 2024, Université de Relizane, Algérie.

   URL : https://ncaiit-2024.sciencesconf.org/?lang=en

8. **MECIBAH Zina,** BOUTEKKOUK Fateh, (2025). *«Requirements Modeling for Self-Adaptive Embedded Systems: SysML4SAS Profile»,* INTERNATIONAL CONFERENCE ON ARTIFICIAL INTELLIGENCE AND SUSTAINABLE DEVELOPMENT (ICAISD'2025), 12-13 April 2025, Université de Relizane, Algérie.

الحمد لله الذي بنعمته تتم الصالحات