ILLUSTRATION DU PROBLÈME ET CONTRIBUTIONS

2.1 Approche illustrée

2.1.1 Système à l'étude (SAE)

En considérant nos hypothèses de départ, nous évaluons notre approche sur un système à l'étude (SAE) présenté dans la thèse de F. Obeid [OD17]. F.Obeid cherche à sécuriser, conformément à une politique de sécurité donnée, une architecture SCADA en s'appuyant sur la réutilisation de patrons de sécurité. Pour ce faire, il cherche à prouver que les modèles d'architecture SCADA générés dans un langage formel (Fiacre) et dotés de mécanismes basés sur les patrons de sécurité, respectent les exigences décrites dans la politique de sécurité choisie au regard d'attaques. Si les exigences de sécurité ne sont pas respectées par le modèle, alors des traces sont produites, indiquant soit que l'application du patron est incorrecte, soit que les choix des patrons ou leurs combinaisons sont incorrects vis-à-vis de l'architecture SCADA et de la politique de sécurité attendue.

Les systèmes d'acquisition et de contrôle de données (SCADA) nous intéressent particulièrement, car ils sont fortement impactés par les enjeux de complexité et de sécurité. Ces systèmes sont mis en œuvre dans les installations industrielles modernes (centrales électriques, gestions de l'eau, raffineries de pétrole, installations nucléaires, etc), et sont composés de moyens informatiques, hautement concurrents, qui contrôlent et pilotent les procédés industriels.

Un SCADA regroupe un ensemble de composants, des capteurs ou actionneurs, des contrôleurs logiques programmables (PLC) ou des machines de contrôle et coordination (GC), connectés ensemble par différents réseaux (Network). Comme le montre la figure 2.1, notre SAE est un système SCADA composé de 4 entités, deux contrôleurs locaux (Plc_1 et Plc_2), un contrôleur global (Gc) connectés par un réseau ($Network$). Ce système est intégré dans un environnement (Env), via une interface au niveau du contrôleur global. Les liens entre les entités sont des canaux (channels) du type FIFO unidirectionnels permettant l'échange de messages de manière asynchrone.

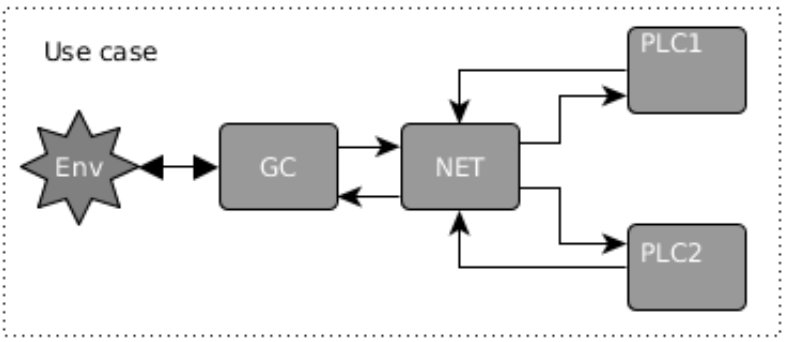

FIGURE 2.1 – Architecture SCADA abstraite

2.1.2 Cas d'utilisations

Considérons les trois exemples de diagnostic sur deux architectures différentes, illustrés sur la figure 2.2.

Dans le premier cas (1), un ingénieur vérifie le comportement d'un système conçu par un tiers, illustré sur la figure 2.3. L'ingénieur a une pratique de la vérification par model checking, mais peu de connaissances quant au domaine du système. Dans le second cas, un concepteur reprend ce même système, et y ajoute un mécanisme d'exclusion mutuelle, comme illustré sur la figure 2.4, puis le vérifie. Il bénéficie d'une expérience dans le domaine de l'application qu'il conçoit. Dans le dernier cas, un architecte sécurise le système, comme illustré sur la figure 2.5, puis le vérifie. Il bénéficie d'une base de connaissances externe. Rappelons que la question générale qui nous concerne est de faciliter la phase de diagnostic qui survient lorsqu'une vérification a échouée.

Premier cas : Vérification du modèle construit par un tiers

Dans le premier cas, un ingénieur que nous appellerons Luka, doit vérifier un modèle conçu par un tiers. Luka n'est pas expert dans le domaine des SCADA, mais il a une bonne connaissance en model checking. Il est familier avec les processus communicants décrits dans des langages formels, familier avec les propriétés formelles décrites en logique temporelle, et familier avec la technique d'exploration des LTS. Le modèle qu'il doit vérifier est un modèle formel exprimé en langage Fiacre généré à partir d'un modèle UML (figure 2.3). En parallèle, il lui a été fourni un ensemble de propriétés à vérifier, décrites en LTL. Il dispose également d'un ensemble de scenarii d'utilisations du modèle formalisé sous la forme de contextes CDL. Luka suit l'approche décrite par C.Baier [BK08]. À l'aide d'un model checker (OBP), il génère le LTS correspondant au modèle, et l'explore.

Dans un des scenarii, le client envoie plusieurs messages au PLC_1 et attend un message

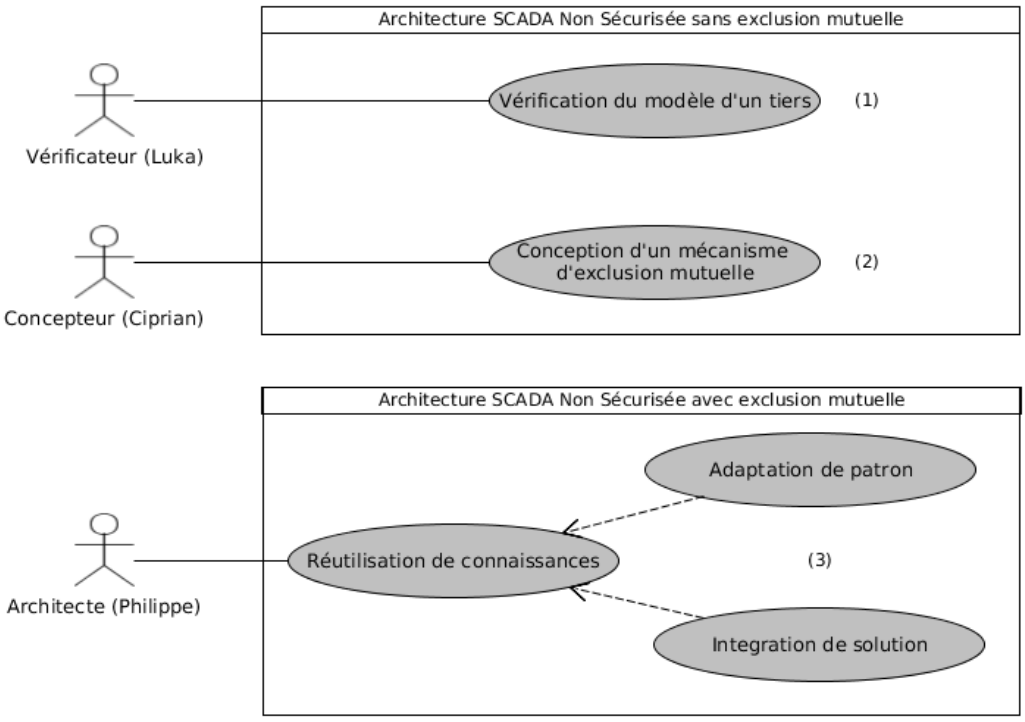

FIGURE 2.2 – Cas d'utilisations

d'acquittement (*ACK*) de la part du système. L'exploration montre que ce scénario ne fonctionne pas car une propriété vérifiant ce comportement est violée (*CLT1SendAck*), déclenchant ainsi une phase de diagnostic. Cette propriété énonce que *si un Client envoie une donnée au* PLC_1*, alors celui-ci recevra fatalement un ACK venant de* PLC_1. OBP génère un contre-exemple exposant à Luka les configurations amenant à cette conclusion. Grâce à sa connaissance en model checking, Luka peut rejouer le contre-exemple pour en comprendre la cause.

Mais la trace produite par le model checker est trop détaillée, constituant un frein à l'analyse manuelle. Il existe un ensemble de techniques d'aide au diagnostic, comme comparer des traces positives et négatives afin d'isoler des parties suspectes de la trace, ou bien réduire la trace par slicing. Luka opte pour l'utilisation d'un outil de visualisation de trace sous forme d'un diagramme de séquence. Le diagramme de séquence permet d'afficher uniquement les interactions entre les différents processus. Cet outil aide Luka à identifier qu'un message n'est pas transmis entre deux processus, PLC_1 et *Network*. Le message ne parvient pas au *Network* car celui-ci n'est pas transmis par le bon canal. Luka propose un diagnostic, le processus PLC_1 est mal connecté.

Une fois la correction apportée au modèle, Luka s'assure que celui-ci est correct en effectuant une autre vérification. Mais la propriété est une nouvelle fois violée. À travers

le diagramme de séquence, Luka s'aperçoit pourtant que sa correction a fonctionné, le message est correctement transmis. Ici, le diagramme de séquence ne permet pas d'aller plus loin dans l'analyse, il faut changer de stratégie. Luka énonce de nouvelles propriétés lui permettant d'observer des comportements particuliers du modèle, en y ajoutant au besoin quelques annotations. Par exemple, une propriété énonce que *quand un channel est lu il doit être dépilé*. Ces propriétés de contrôle sont inhérentes aux processus communicants. Le model checker indique maintenant qu'une propriété est violée. Un channel est lu sans être dépilé. Conclusion, le channel doit être dépilé après avoir été lu.

Le modèle est une nouvelle fois corrigé, mais la propriété est toujours violée. Bien que Luka ait connaissance des spécifications du problème et de connaissances toujours applicables en matière de model checking, celles-ci sont parfois insuffisantes pour effectuer le diagnostic, le rendant long et fastidieux. De nouvelles connaissances sont alors nécessaires pour comprendre la solution.

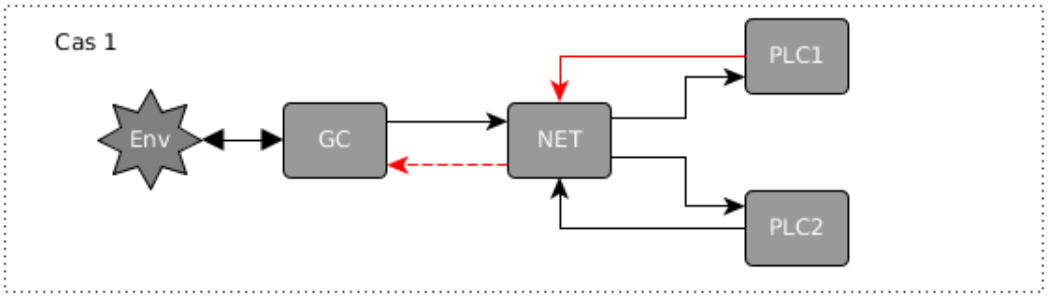

FIGURE 2.3 – Cas d'utilisation 1, en rouge les erreurs identifiées par Luka

Second cas : Conception d'un mécanisme d'exclusion mutuelle

Ciprian est le concepteur du modèle, et il le maîtrise. Il souhaite ajouter une nouvelle fonctionnalité. PLC_1 et PLC_2 partagent tous deux une ressource, lue ou écrite lorsque PLC_1 ou PLC_2 sont dans leur état *Access*, soit *PLC1@Access ou PLC2@Access*[1]. Le modèle doit vérifier l'exigence suivante : *il n'est pas possible que* PLC_1 *et* PLC_2 *soient en même temps dans leur état* Access, exigence traduite sous la forme d'une propriété *PSingleAccess*. Pour répondre à ce nouveau problème il opte pour un mécanisme de drapeaux. Le principe est le suivant, PLC_1 et PLC_2 possèdent tous deux un drapeau. PLC_1 ou PLC_2 lève son drapeau, pour informer qu'il accède à la ressource. Avant d'accéder à cette ressource, un processus doit vérifier que le drapeau de l'autre processus n'est pas levé, sinon, il doit attendre qu'il soit baissé. Lorsqu'un processus a fini d'utiliser la ressource, il baisse son drapeau et libère la ressource.

1. Nous exprimons qu'un processus *Processus* est dans son état *Etat* par la notation *Processus@Etat*

Après vérification, il s'avère que la propriété *PSingleAccess* est violée, signalant l'existence d'un contre exemple où PLC_1 et PLC_2 utilisent en même temps la ressource. Comme précédemment, Ciprian commence par vérifier des propriétés liées à la modélisation des processus communicants (comme dans le cas précédent). Au bout d'un certain temps, sans succès, il en conclut que le problème vient de la manière dont il a conçu le mécanisme. Pour identifier parmi les milliers de configurations de la trace le comportement du mécanisme implémenté, il doit exprimer des propriétés liées à ce mécanisme. Ciprian s'aperçoit alors que le drapeau est levé après le test d'accès à la ressource. Il doit donc corriger le modèle et repartir sur un cycle de vérification.

Si Ciprian connaît la spécification du problème et est capable de le résoudre, il n'a pas de connaissances de solutions existantes. Il va devoir concevoir lui-même un mécanisme d'exclusion mutuelle et suivra un cycle itératif de conception-vérification, pouvant être bloqué s'il ne sait pas le résoudre entièrement. Celui-ci pourrait être évité par la réutilisation de solutions du domaine.

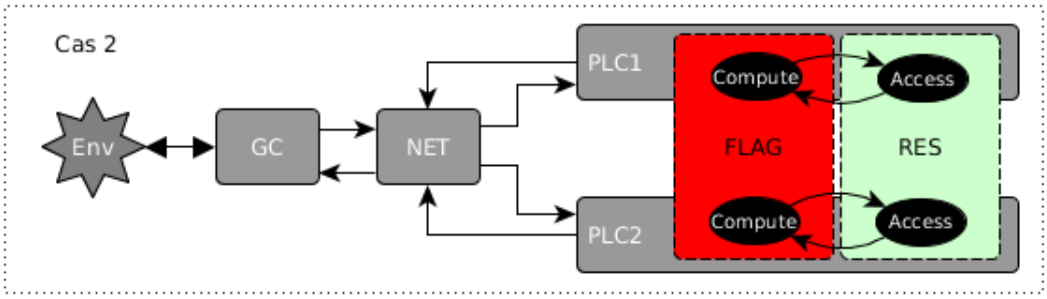

FIGURE 2.4 – Cas d'utilisation 2, Ciprian conçoit une exclusion mutuelle

Troisième cas : Réutilisation de connaissances

Après avoir assuré un mécanisme d'exclusion mutuelle pour les deux *Plc* envers la ressource, un autre ingénieur, Philippe, doit sécuriser l'architecture face à des cyberattaques. Pour ce faire, il prévoit d'implémenter un mécanisme de Single Access Point (*SAP*), réduisant l'accès à la ressource en un point unique. Philippe choisit de réutiliser une solution existante, mettant en œuvre le *SAP*, qui est accompagnée de propriétés et d'une architecture abstraite. Pour l'utiliser, il doit réaliser un ensemble de modifications dans son modèle, et pour le vérifier, il doit faire correspondre les propriétés exprimées par le patron avec les éléments de son modèle. Après vérification, il s'avère qu'une des propriétés est violée. Grâce au *SAP*, Philippe bénéficie d'un ensemble de propriétés qui peuvent l'aider à localiser le problème. La connaissance du domaine, c'est-à-dire la connaissance de solutions, permet alors de le guider dans le diagnostic. Finalement, il s'avère que la

solution construite n'est pas conforme. Bien que des propriétés aient pu aider au diagnostic, la solution conçue de manière ad hoc, reste difficile à diagnostiquer. Philippe doit donc comprendre comment il a mis en œuvre la solution pour comprendre l'origine du problème.

Une nouvelle exigence, traduite sous la forme d'une propriété formelle, exige que si le client a les droits suffisants, il doit être autorisé à accéder à la ressource. Le mécanisme de *SAP* n'est plus suffisant pour protéger l'architecture *SCADA*. Un mécanisme d'autorisation (*AUTH*) doit être ajouté. Ainsi, seuls les clients autorisés pourront accéder à la ressource. Cette fois-ci, il existe un patron d'autorisation incluant une solution adaptable. Philippe applique ce patron à son modèle. La propriété est tout de même violée, préfigurant deux problèmes. Soit l'adaptation du patron a été mal réalisée, c'est-à-dire par exemple que le patron n'est pas adapté au modèle de la bonne manière, soit c'est un mauvais choix de patron. Une fois corrigée, la solution est capturée sous la forme d'un composant réutilisable. Ici c'est la méthode employée qui est au cœur du problème. La solution choisie amène avec elle un lot de propriétés liées aux composants intégrés. Le modèle est ainsi constitué d'un ensemble de composants, chacun associé à un ensemble de propriétés axiomatiques dont certaines permettent de contrôler l'intégration. Cette couverture permet d'isoler plus précisément le composant qui porte le problème, réduisant ainsi l'effort de diagnostic.

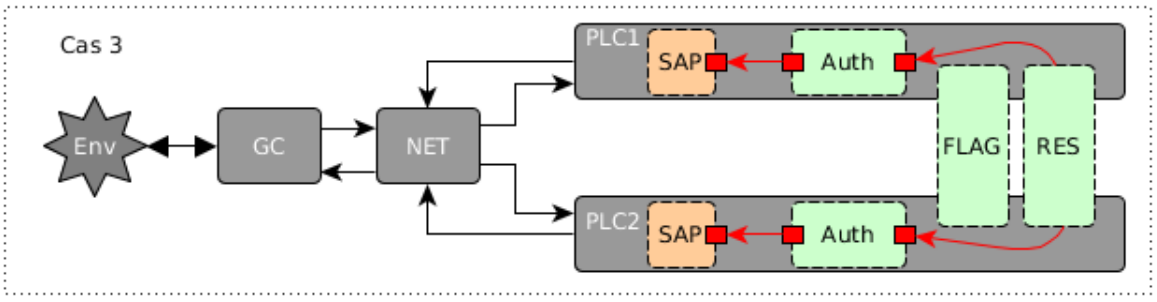

FIGURE 2.5 – Cas d'utilisation 3, Philippe sécurise l'architecture

2.2 Contributions

2.2.1 Cadre

Comme nous cherchons à fournir des aides au diagnostic, nous devons définir ce qu'est un diagnostic. Il est défini par [Mer04] comme une *investigation ou une analyse des causes ou natures d'une condition, situation ou problème.* Dans le cas du model checking, il est

déclenché par la découverte d'un contre-exemple. Il est traditionnellement facilité par différentes techniques, comme par exemple :

— *Simuler* en rejouant la trace, *stimuler* par l'altération légère du modèle ou une modification de son contexte d'exécution, ou *générer* de nouvelles propriétés afin d'acquérir de nouvelles connaissances.

— *Séparer*, *isoler* ou *réduire* la trace, afin d'éliminer ou simplifier les connaissances.

— *Comparer* des traces valides ou violées, ou *corréler* les traces et modèles avec de nouvelles informations (pretty-printing, annotations, propriétés) pour changer de point de vue ou permettre de nouvelles analyses.

Dans la définition de [Mer04], le diagnostic représente un processus cognitif (investigation ou analyse). De même, *simuler*, *stimuler*, *générer*, *isoler*, *réduire*, *séparer*, *corréler* ou *comparer* sont des opérations cognitives que nous utilisons dans le contexte de la vérification par model checking.

Processus cognitifs et diagnostic

La taxonomie de Bloom [Blo+64] est une classification éprouvée et mature qui permet de classer les activités cognitives. Destinée à l'évaluation des connaissances des étudiants, elle est suffisamment générique pour être extrapolée à la connaissance sur tout type de problème, dont celui de la conception de systèmes logiciels [SR04][BE03a]. Dans [XR04] les auteurs associent les activités cognitives requises lors du debug un programme logiciel avec les activités cognitives de Bloom, montrant qu'il était possible d'exprimer une partie de ce processus à l'aide d'activités de la taxonomie.

La taxonomie regroupe des comportements intellectuels, initialement les résultats attendus par le processus éducatif, en six catégories. *Se souvenir* représente le fait de mémoriser des informations, et de pouvoir se les rappeler, sans pour autant les comprendre. Les opérations cognitives associées sont entre autres *rappeler* ou *reconnaître*. *Comprendre* est la capacité à expliquer le sens de l'information, c'est-à-dire être capable de traduire un contenu dans un autre langage, ou bien l'extrapoler pour en faire des prédictions. Les opérations cognitives associées sont par exemple *interpréter*, *exemplifier*, *classifier*, *résumer*, *inférer*, *comparer* ou *expliquer*. *Appliquer* est l'usage d'abstractions dans des situations concrètes, autrement dit, le fait de sélectionner des données pour répondre à un problème nouveau. Parmi les opérations cognitives associées on retrouve *exécuter* ou *développer*. *Analyser* est la décomposition d'un tout en composants, autrement dit, rendre explicite des liens entre ces éléments pour déterminer des interactions, ou bien reconnaître des arrangements ou des structures qui maintiennent ces éléments entre eux. Cette catégorie inclut les opérations cognitives *différencier*, *organiser* ou *attribuer*. *Créer* est la mise ensemble de parties afin de réaliser un nouveau tout, les opérations cognitives sont *générer*,

1.0 Remember – Retrieving relevant knowledge from long-term memory.
- ***1.1 Recognizing***
- ***1.2 Recalling***

2.0 Understand – Determining the meaning of instructional messages, including oral, written, and graphic communication.
- ***2.1 Interpreting***
- ***2.2 Exemplifying***
- ***2.3 Classifying***
- ***2.4 Summarizing***
- ***2.5 Inferring***
- ***2.6 Comparing***
- ***2.7 Explaining***

3.0 Apply – Carrying out or using a procedure in a given situation.
- ***3.1 Executing***
- ***3.2 Implementing***

4.0 Analyze – Breaking material into its constituent parts and detecting how the parts relate to one another and to an overall structure or purpose.
- ***4.1 Differentiating***
- ***4.2 Organizing***
- ***4.3 Attributing***

5.0 Evaluate – Making judgments based on criteria and standards.
- ***5.1 Checking***
- ***5.2 Critiquing***

6.0 Create – Putting elements together to form a novel, coherent whole or make an original product.
- ***6.1 Generating***
- ***6.2 Planning***
- ***6.3 Producing***

FIGURE 2.6 – Taxonomie de Bloom, processus cognitifs d'après D.Krathwohl [Kra02]

A. Factual Knowledge – The basic elements that students must know to be acquainted with a discipline or solve problems in it.
- **Aa. Knowledge of terminology**
- **Ab. Knowledge of specific details and elements**

B. Conceptual Knowledge – The interrelationships among the basic elements within a larger structure that enable them to function together.
- **Ba. Knowledge of classifications and categories**
- **Bb. Knowledge of principles and generalizations**
- **Bc. Knowledge of theories, models, and structures**

C. Procedural Knowledge – How to do something; methods of inquiry, and criteria for using skills, algorithms, techniques, and methods.
- **Ca. Knowledge of subject-specific skills and algorithms**
- **Cb. Knowledge of subject-specific techniques and methods**
- **Cc. Knowledge of criteria for determining when to use appropriate procedures**

D. Metacognitive Knowledge – Knowledge of cognition in general as well as awareness and knowledge of one's own cognition.
- **Da. Strategic knowledge**
- **Db. Knowledge about cognitive tasks, including appropriate contextual and conditional knowledge**
- **Dc. Self-knowledge**

FIGURE 2.7 – Taxonomie de Bloom révisée, connaissances d'après D.Krathwohl [Kra02]

produire ou *planifier*. Enfin *évaluer* est le fait de produire des jugements à propos des idées ou des phénomènes. On peut considérer que le diagnostic est une opération cognitive d'évaluation, pour autant celle-ci a recours à l'ensemble des opérations cognitives de la taxonomie, par exemple le diagnostic *le processus GC ne transmet pas le message ACK* inclut d'*isoler* les communications entre les processus.

La figure 2.6 présente la taxonomie de Bloom du point de vue de la dimension des *processus cognitifs*.

Connaissances et diagnostic

On décrit généralement les objectifs d'apprentissage en termes de contenu (exprimé sous la forme d'un nom) et de descriptions de ce qui doit être fait avec ce contenu (exprimé

sous la forme d'un verbe). Cette distinction s'applique aussi au diagnostic car d'après la définition, le processus de diagnostic (verbe, ex : isoler) s'établit à partir de conditions, situations ou problèmes (nom, ex :Plc1). D.Krathwohl [Kra02] a proposé de faire apparaître dans la taxonomie de Bloom une nouvelle dimension, celle des connaissances. La figure 2.7 présente cette nouvelle dimension. En reprenant cette idée d'une dimension de connaissance indépendante des processus cognitifs, nous structurons les connaissances en trois catégories, les connaissances liées au model checking, les connaissances du domaine et les connaissances de gestion.

Les connaissances liées au model checking regroupent un ensemble d'informations qui sont inhérentes au contexte de vérification et de diagnostic. De manière non exhaustive, il s'agit des connaissances relatives aux processus concurrents (processus, canaux...), aux LTS (états, transitions, configurations, variables...), aux propriétés formelles (LTL, vivacité, sûreté ...) ainsi que des connaissances générales en matière de vérification par model checking (scénarios, model checkers, exploration exhaustive, BFS, DFS, cycles, deadlocks, contre-exemples...).

Les connaissances du domaine ne sont pas toujours des connaissances applicables dans la vérification par model checking, elles sont liées au domaine de l'application. Un domaine d'application peut être décrit par tous les phénomènes observables du domaine (entités, fonctions, événements, comportements...) et les relations qui les unissent. Il exprime le périmètre de l'application, sans aucune référence à ses exigences, ni à ses implémentations [Bjø06]. Une représentation d'un domaine n'est pas simplement de la connaissance contenue dans le cerveau d'un expert du domaine ; c'est une abstraction rigoureusement organisée et sélective de cette connaissance. Les connaissances du domaine sont par exemple une architecture SCADA (GC, PLC, Network), un mécanisme d'exclusion mutuelle (façonné en trois états : Entrée, Section Critique, Sortie) un mécanisme d'autorisation, ou encore le mécanisme de drapeau répondant à l'exclusion mutuelle. Ces différentes connaissances sont exprimables dans un patron, c'est-à-dire une solution ou un ensemble de bonnes pratiques éprouvées pour une classe de problèmes récurrents. Il existe différentes formes de patrons, de conception, d'architecture, ou bien encore donc, de sécurité. Ainsi, dans notre troisième cas d'utilisation, chaque patron de sécurité est décrit par une description formelle de sa structure et de son comportement, ainsi qu'une description formelle des propriétés de sécurité associées à ce patron.

Les connaissances de gestion, par exemple celles relatives à la résolution de problème ou au résultat de diagnostic, sont transversales aux autres connaissances. Tout d'abord, nous avons vu dans la définition du diagnostic que les processus cognitifs manipulaient des connaissances afin d'atteindre un objectif, le diagnostic. Ce diagnostic ici, est le résultat du processus de diagnostic, et représente un type de connaissance que l'on peut vouloir conserver. Il existe aussi des connaissances transversales aux domaines liées à la méthode

de résolution, souvent implicites et qui sont généralement assimilées à de l'expérience. Un exemple est la méthode d'application d'un patron. D'un côté, intégrer un patron sous la forme d'un nouveau processus dans son design est un gain du point de vue de la lisibilité de design, mais peut être un frein du point de vue de d'explosion combinatoire. D'un autre côté, mélanger le comportement du patron dans le modèle, par exemple en ajoutant des nouveaux états dans un processus, peut-être un choix judicieux pour diminuer l'impact de l'explosion combinatoire, mais rendra aussi le diagnostic plus complexe.

La décomposition de la dimension processus de Bloom est utilisée dans cette thèse. Sans avoir repris la décomposition de la dimension connaissance de Bloom, nous utilisons son pouvoir structurant pour organiser la thèse grâce aux trois catégories de connaissances définies ci-dessus.

2.2.2 Comprendre et analyser le Système A l'Étude

Parmi les enjeux identifiés au premier chapitre, les deux premiers sont inhérents aux connaissances de model checking.

Comment structurer et outiller l'analyse de la trace ? Pour analyser le contre-exemple, le diagnosticien doit réaliser un ensemble d'activités cognitives qu'il est difficile d'appréhender. En plaçant ces activités dans la taxonomie de Bloom, nous permettons de comprendre ces activités cognitives menées lors d'un diagnostic, chacune pouvant être facilitée par un outil ou une technique.

Si l'on porte l'effort sur les traces, l'ingénieur a à sa disposition au moins deux approches, soit simplifier (réduire ou isoler les parties suspectes amenant à la trace, séparer pour appréhender le système par parties), soit offrir un nouveau point de vue sur la trace à travers un outil de visualisation par exemple.

Comment réduire le fossé sémantique ? Lors du diagnostic, les connaissances liées au model checking ne permettant pas toujours de résoudre le problème, il faut souvent raisonner sur une représentation abstraite tel qu'un algorithme. Ce changement de point de vue contraint le diagnosticien à résoudre un fossé sémantique entre les spécifications abstraites et le contre-exemple exprimé par des connaissances liées au model checking. Il faut pour cela définir des corrélations entre les éléments de trace qui doivent être annotés, et le problème formulé. Nous proposons un cadre pour corréler les éléments entre les différents niveaux de connaissances, et ce à travers un mécanisme de fédération de modèles.

2.2.3 Formaliser, partager et réutiliser la connaissance

Comment peut-on faciliter le partage et la réutilisation de la connaissance ? Pour partager des connaissances, il faut qu'il y ait un consensus ontologique et une structure.

Nous proposons une formalisation des différentes connaissances, de la vérification par model checking, à la méthode et au diagnostic en passant par le domaine. Cette formalisation permet de capitaliser la connaissance et de la préparer au partage d'expérience grâce à des *problem cases*.

Comment aider à la résolution, c'est-à-dire à la construction et à la vérification de la solution ?. Dans une approche par réutilisation, comme dans le cas 3, le diagnostic est grandement simplifié. Premièrement, les solutions sont déjà éprouvées, par conséquent les erreurs sont liées aux choix de ces patrons, ou à leur composition. Deuxièmement, ces patrons ramènent un ensemble de propriétés qui, une fois assemblées, vont pouvoir être exploitées dans le modèle à vérifier. Troisièmement, les patrons constituent une documentation quant au fonctionnement du système, aidant à la compréhension. Nous proposons de formaliser cette démarche sous la forme d'une méthode de résolution de problèmes.

2.2.4 Organiser la méthode, le domaine

Comment organiser toutes ces connaissances et les interactions qu'elles supportent ? Quelle infrastructure peut supporter la méthode ? C.Baier [BK08] souligne que, de manière transversale aux autres phases, le processus de vérification doit être planifié, administré et organisé, à travers une activité appelée *l'organisation de la vérification*. Une plateforme logicielle supportant l'organisation de la vérification serait aussi en mesure de capitaliser les expériences passées, et d'en permettre la réutilisation. Tout comme le pense [Bou09], la mise en œuvre d'un support ou d'une méthodologie outillée est primordiale pour faciliter à la fois la détection de problèmes, la localisation des fautes, et l'analyse des causes de ces dysfonctionnements. Ainsi nous proposons d'outiller l'ensemble des points précédents en s'appuyant sur une infrastructure organisant leurs artefacts et les interactions qu'ils supportent.