ORGANISER LES CONNAISSANCES ET LES INTERACTIONS

7.1 Introduction

7.1.1 Une profusion d'informations

Informations hétérogènes

Un processus de conception et vérification par model checking produit une profusion d'informations hétérogènes (spécifications, modèles structurels ou comportementaux, propriétés, traces ...), et ce à différents niveaux d'abstractions et de détails, comme argumenté par [RB03]. Dans le chapitre 4, nous avons vu par exemple qu'une des techniques de diagnostic consistait à écrire un ensemble de propriétés liées à des connaissances en model checking. Cette technique produisait 30 propriétés pour un système très simple. Aussi pour faciliter leur mise en œuvre et limiter les erreurs, ces propriétés doivent être générées (conformément à des normes de nommage) ou instanciées à partir d'un modèle ou d'un patron défini par un expert, et dont les concepts doivent être formalisés. Ces modèles peuvent être stockés dans une bibliothèque.

Relations

Des relations de tout ordre existent entre ces informations. Dans le chapitre 5, par exemple, nous avons vu que pour résoudre le fossé sémantique il fallait rapprocher deux types de connaissances, les connaissances du niveau domaine et les connaissances liées au model checking. Pour cela il s'agit de définir des corrélations entre ces éléments. Pour pouvoir outiller ces corrélations, celles-ci doivent être capturées et formalisées.

Temporalité

Ces informations et leurs relations évoluent au cours du temps. Elles sont modifiées, enrichies ou abandonnées tout au long du processus de conception et de vérification. Dans le chapitre 5, nous avons vu par exemple qu'un processus de résolution de problèmes

implique une multitude de cycles de conception-vérification. Lors de ces cycles, des solutions sont produites, conservées, améliorées ou bien abandonnées. Le système devient au fil du temps un enchevêtrement de solutions (associées à leurs problèmes) garantissant des propriétés. Si l'on ne conserve pas l'historique de ces solutions, ont perd alors les raisonnements qui nous ont conduits à ces solutions.

7.1.2 Besoin de gérer le processus de vérification

Constat actuel sur la gestion du processus de vérification

Actuellement on constate que les processus de vérification font face aux problèmes suivants :

- (1) Les connaissances ne sont pas bien gérées, et donc indisponibles.
- (2) Les activités manipulant ces connaissances ne sont pas bien contrôlées, ni enregistrées.
- (3) Il n'y a pas de consensus sur un formalisme de ces connaissances.
- (4) Il n'y a pas d'accord sur une manière de les capturer.

Pour capturer et utiliser les connaissances et les relations au fil du temps, il faut une gestion du processus de vérification qui n'est aujourd'hui pas pratiquée [RB03].

Vers un système organisationnel

Baier 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* [BK08]. Un système supportant l'organisation de la vérification serait aussi en mesure de capitaliser les expériences passées, et d'en permettre la réutilisation, notamment pour contribuer au processus de diagnostic. L'enjeu de ce chapitre est de proposer une infrastructure pour organiser ces connaissances et les interactions qu'elles supportent. Pour cela, il faut répondre à différentes questions :

- (1) Où et sous quelle forme capturer ces éléments ?
- (2) Comment les retrouver et quelles sont les interactions possibles ?

Système organisationnel

Organiser, c'est créer des capacités en imposant intentionnellement un ordre et une structure [Glu12]. C'est une activité tellement commune que nous le faisons souvent inconsciemment (organiser les jouets dans des boîtes, organiser les vêtements dans les placards...). Ces tâches organisatrices ne sont souvent pas réfléchies ou exprimées. Nous prenons pour acquis les concepts et les méthodes utilisés dans le système d'organisation

avec lequel nous travaillons. Dans [Glu12], Glushko introduit le concept de système organisationnel qu'il définit comme "un ensemble de ressources organisé intentionnellement et les interactions qu'elles supportent". La figure 7.1 décrit un modèle conceptuel d'un tel système découpé en trois parties :

- (1) Les ressources disposées intentionnellement

- (2) Les différentes interactions (différents types de flèches)

- (3) Les entités (humaines ou machines) qui interagissent avec les ressources dans différents contextes.

Une ressource est une chose physique (un fichier de trace) ou non physique (le concept de configuration), ou une information sur cette chose (ce fichier de trace représente un contre-exemple). Une collection est une ressource qui regroupe des ressources sélectionnées dans un but (ce run de vérification vérifie la propriété PRT_1 sur le modèle $System_1$). Un arrangement intentionnel met l'accent sur l'organisation explicite ou implicite commis par des personnes (ou processus), se différenciant des disposition naturelles (créés par des processus physiques ou temporels). Une interaction est une action, une fonction, un service ou une capacité qui utilise les ressources d'une collection ou de la collection dans son ensemble.

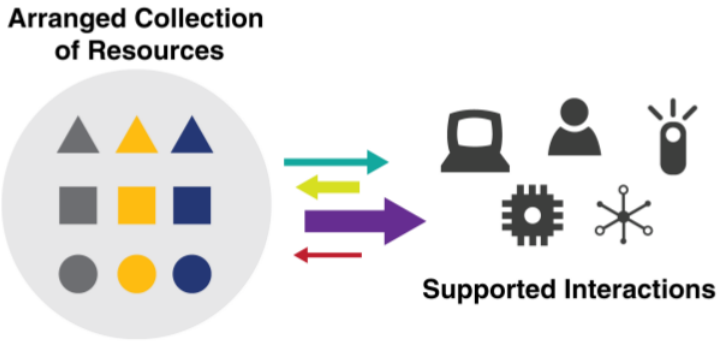

FIGURE 7.1 – Organizing System de Glusko

Il est généralement admis par les architectes et concepteurs de logiciels de séparer le stockage des données de la logique métier utilisant ces données et de l'interface utilisateur (ou les composants d'interactions). Cette architecture est modulaire, et permet à chacun des trois niveaux d'être modifié indépendamment pour répondre aux nouvelles exigences ou utiliser de nouvelles technologies.

Notre proposition est de définir un système organisationnel pour la conception et vérification par model checking axé sur les interactions de diagnostic.

7.2 Verification Organizing System

7.2.1 Description du système

Nous appelons *VOS* pour Verification Organizing System, le support organisationnel permettant à la fois de gérer les informations relatives aux activités de conception et de vérification, et de servir de base aux outils et méthodes de diagnostic. Le système est composé de 3 niveaux : - un niveau physique où sont stockées les données réelles induites par les processus de conception et de vérification ; - un niveau ontologique évolutif, au sein duquel les données physiques sont réinterprétées et mises à disposition d'interactions ; - et enfin un niveau d'accès où les interactions sont mises à disposition d'outils ou d'humain extérieurs.

Établir un tel système implique des décisions qui sont intrinsèquement liées, mais qu'il est plus facile d'introduire indépendamment. Glusko préconise cinq groupes de questions indépendantes pour décider de la conception : qu'est ce qui est organisé, pourquoi, sous quelle forme, quand et comment ?

Qu'est-ce qui est organisé ?

Il faut organiser les connaissances et les interactions. D'un côté nous disposons de trois types de connaissances :

- (1) Les connaissances liées au model checking incluent les modèles de système, les propriétés, et les vérifications

- (2) Les connaissances de domaine incluent les *problem cases*, les *sample*, *pattern* et *component cases*

- (3) Les connaissances de gestion incluent les éléments de la méthode, les cycles de vérification, les activités ou résultats de diagnostic.

D'un autre côté, les interactions sont des blocs élémentaires d'activité, par exemple "explorer la trace", "réduire la trace", "comparer deux traces". Le processus de diagnostic est un ensemble ordonné de ces activités (interactions) s'appuyant sur ces connaissances.

Toutes sont des ressources numériques, mais nous pouvons faire la distinction entre les ressources primaires et les descriptions de ressources à propos des ressources primaires. Tout utilisateur du VOS peut également organiser ses propres activités de vérification dans des collections ou des sous collections de ressources.

Pourquoi est-ce organisé ?

Les utilisateurs du VOS sont des ingénieurs en conception et vérification travaillant seuls ou en équipe qui doivent "gérer l'énorme quantité des données au fil du temps et qui est produite au cours de la phase de conception et de vérification" [RB03]. Le VOS

partage et organise des informations qualitatives et quantitatives permettant la création de connaissances et le raisonnement basé sur ces connaissances. Les utilisateurs du VOS partagent leurs connaissances sans être contraints d'épouser un formalisme donné. Les utilisateurs du VOS doivent naviguer efficacement dans cet espace de ressources.

Sous quelle forme est-ce organisé ?

Le VOS le plus simple consiste en un système de gestion de configuration (comme un GIT ou un SVN) contrôlant les changements de chaque ressource numérique. À l'opposé, le VOS peut être une ontologie complète où toute relation entre des éléments d'information est soigneusement définie et contrôlée. De notre point de vue, le VOS gère essentiellement des documents (modèles, résultats, traces). Chaque document organise sa structure de connaissances et son contenu, en fonction de son type, et l'ingénieur écrit et lit les informations en fonction de cette structure. La réification de la structure de connaissances sous-jacente est effectuée automatiquement par le VOS.

Quand est-il organisé ?

Le VOS est conçu pour assister l'ingénieur dans ses tâches quotidiennes de conception et de vérification. Les ressources sont donc organisées en permanence. Cependant, le VOS doit offrir une fonctionnalité d'ingestion permettant de saisir de nouvelles entrées. Cette fonctionnalité lui permet d'accepter des tentatives de vérification complexes ou des benchmarks et de préparer le contenu pour le stockage et la gestion au sein du VOS. À l'inverse, une fonctionnalité d'accès fournit les services et les fonctions qui aident les utilisateurs à interagir avec les informations stockées dans le VOS.

Comment ou par qui, ou par quels processus informatiques, est-il organisé ?

Même si un seul ingénieur en vérification bénéficie de l'utilisation du VOS, le VOS est conçu pour prendre en charge le travail en équipe et partager les connaissances sur les modèles et les tentatives de vérification. Des processus automatisés doivent extraire autant de connaissances que possible de la structure interne des documents et de l'organisation des collections. En tant que travail d'équipe collaboratif, l'organisation est effectuée de manière ascendante et distribuée.

7.2.2 Architecture

Vue conceptuelle

L'architecture conceptuelle du VOS est présentée sur la figure 7.2.

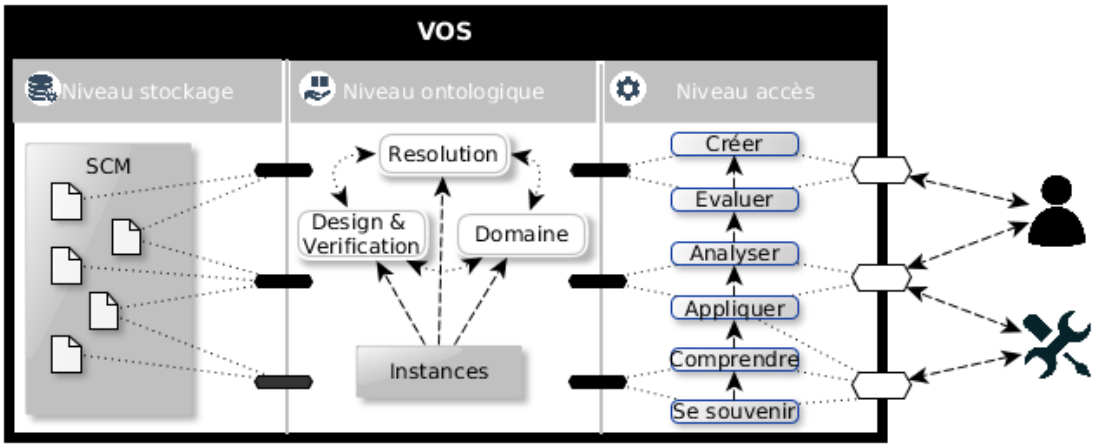

FIGURE 7.2 – Architecture conceptuelle du VOS

Le niveau de stockage. Une difficulté pratique liée à la conception et vérification basée sur du model checking réside dans la gestion de toutes les données (générées) au cours des opérations de conception et de vérification. Un enregistrement rigoureux des informations est nécessaire, et pour cela nous proposons d'utiliser un système de gestion de configuration logicielle (SCM) pour contrôler les artefacts versionnés produits lors des phases de conception et de vérification.

Niveau ontologique. D'un point de vue conceptuel, la couche ontologique est divisée en deux parties, le réseau sémantique de types et le réseau sémantique d'objets. Le réseau sémantique de types consiste en types sémantiques liés par des types de relations sémantiques, équivalents à un modèle relation-entité ou à un diagramme de classes UML. Il est lui-même constitué de trois parties, les connaissances de résolution, de domaine et de conception et vérification. Le réseau sémantique d'objets quant à lui, contient un nœud pour chaque objet à grain fin plus des nœuds pour les objets composites, chacun étant affecté à un ou plusieurs types sémantiques et lié à d'autres objets par des relations sémantiques. Le niveau ontologique évolue constamment au fil du temps, car de nouveaux types de ressources et de nouvelles ressources peuvent être ajoutés à tout moment. Il est irréaliste de s'attendre à ce que toutes les personnes et organisations développant des systèmes d'application de connaissances utilisent une ontologie commune partagée [HPS04]. Nous devons vivre avec différentes ontologies et il sera nécessaire de réconcilier ces ontologies avec une ontologie supérieure commune. Parce que nous sommes habitués à l'ontologie CIDOC CRM, une structure normalisée (ISO 21127 : 2014) permettant de décrire les concepts et relations implicites et explicites utilisés pour décrire le patrimoine culturel, nous utilisons CIDOC CRM comme une ontologie de haut niveau [1].

1. http ://www.cidoc-crm.or/official_release_cidoc.html

Le niveau d'accès L'un des principaux objectifs d'un système d'organisation est l'aide à la conception et à la mise en œuvre d'actions, fonctions ou services qui utilisent les ressources. Dans une architecture classique à 3 niveaux, le niveau de présentation est le niveau dans lequel les utilisateurs interagissent avec une application. Les interactions utilisateur les plus primitives sont les ingestions (importation de nouvelles ressources dans le système d'exploitation), les recherches, la navigation, les annotations ou l'extraction d'informations. Sur cette base d'interactions primitives se trouvent des interactions plus complexes, comme des comparaisons, des isolations, des générations. Ainsi, nous classons les interactions suivant les six catégories cognitives de Bloom [Blo+64], *se souvenir, comprendre, appliquer, analyser, évaluer et créer*. De façon générale, un système organisationnel est également destiné à interagir avec d'autres applications et doit fournir des fonctionnalités d'échange d'informations avec ou sans transformations sémantiques.

Vue technique

L'architecture technique du VOS est présentée dans la figure 7.3. Il convient de réaliser un système facile d'utilisation, accessible et adaptable. Nos choix techniques se sont portés sur un environnement basé sur la technique de fédération de modèles pour l'adaptabilité, des technologies serveur/client web pour l'accessibilité, et du framework CSS Material Design pour l'ergonomie et la facilité d'utilisation.

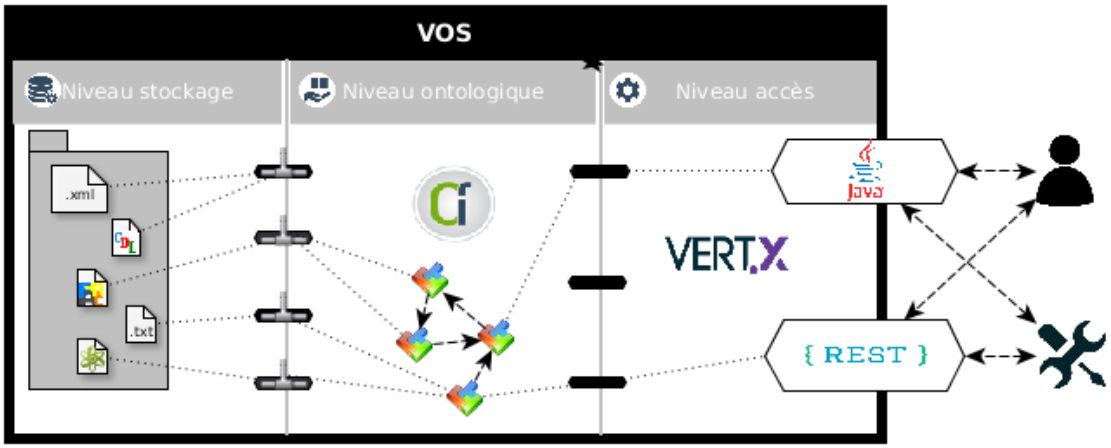

FIGURE 7.3 – Architecture technique du VOS

Niveau stockage. Car chaque système et chaque équipe est différente, nous optons pour une infrastructure flexible, supportant l'implémentation adaptateurs techniques. Ainsi, il n'y a pas de limitations sur le type de ressources qui peut être stocké dans le VOS. Nous n'imposons pas non plus d'arrangement de ressources, uniquement un répertoire regroupant les tentatives de vérification, avec une totale liberté pour y organiser

les informations internes. Les objets complexes tels que les ensembles de propriétés ou la décomposition de designs sont gérés de la même manière, avec un répertoire racine et une liberté d'organisation. Afin de faciliter l'intégration de chaque objet au niveau logique, un fichier de description XML stocke des informations sur les objets. La structure du XML (son schéma) est utilisée par les composants logiciels (fournissant des fonctionnalités d'acquisition et d'accès) pour maintenir un réseau ontologique à jour au niveau logique. Pour éviter la construction d'un silo d'information, c'est-à-dire un système de gestion insulaire qui ne peut fonctionner avec d'autres systèmes, fichiers, répertoires, descriptions XML, garantissent un accès indépendant de tout système de gestion. Comme la description XML est inspirée de la structure des modèles du BEEM [Pel07] (benchmark pour la vérification par model checking), la base de cas peut être remplie par ces éléments.

Niveau ontologique. Parmi les solutions possibles, nous avons choisi une approche pragmatique, appelée la fédération de modèles [Guy+13] supportée par un outil open source [2].

La fédération de modèles est une approche qui vise à créer un mapping bidirectionnel entre des modèles hétérogènes. Elle se distingue des autres approches d'interopérabilité comme l'intégration (grouper tous les concepts dans un métamodèle unique), ou l'unification (transformations vers un métamodèle pivot). La fédération offre un moyen pour franchir les frontières technologiques entre les modèles et favoriser les échanges dans les deux sens.

Openflexo est un outil permettant de réaliser la fédération. Il offre un moyen pour définir des modèles fédérés grâce à la notion de modèles virtuels (*virtual model*). Un *virtual model* est constitué d'un ensemble de concepts fédérés *flexoconcepts*. Tout *flexoconcept* comporte une partie structurelle qui définit ses caractéristiques propres ainsi que les relations (*roles*) vers d'autres concepts ou ressources provenant de divers espaces techniques, et comporte une partie comportementale supportant la création d'*actions* paramétrables. Il existe de nombreux espaces techniques actuellement supportés (fiacre, cdl, excel, word, powerpoint, owl, emf...). Ces *virtual models* peuvent être instanciés en *virtual model instances*, et les *flexo concept* en *flexo concept instances*.

Notre VOS repose sur cette base technique, comme l'illustre l'exemple de la figure 7.4. Un *virtual model* fédére les éléments appartenant au niveau de stockage aux connaissances exprimées dans l'infrastructure Openflexo grâces aux connecteurs techniques. Ces connaissances (model checking, domaine, résolution) sont exprimées par des *flexoconcepts*, par exemple, le concept de *SampleCase* est porté par un *flexoconcept* appartenant au *virtual model* de *domaine*.

Au niveau des instances, une instance de *samplecase* (aussi appelée *flexoconceptinstance*) est liée à plusieurs ressources (cdl, fiacre...) exprimant le *samplecase* dans le niveau

2. http ://research.openflexo.org

de stockage.

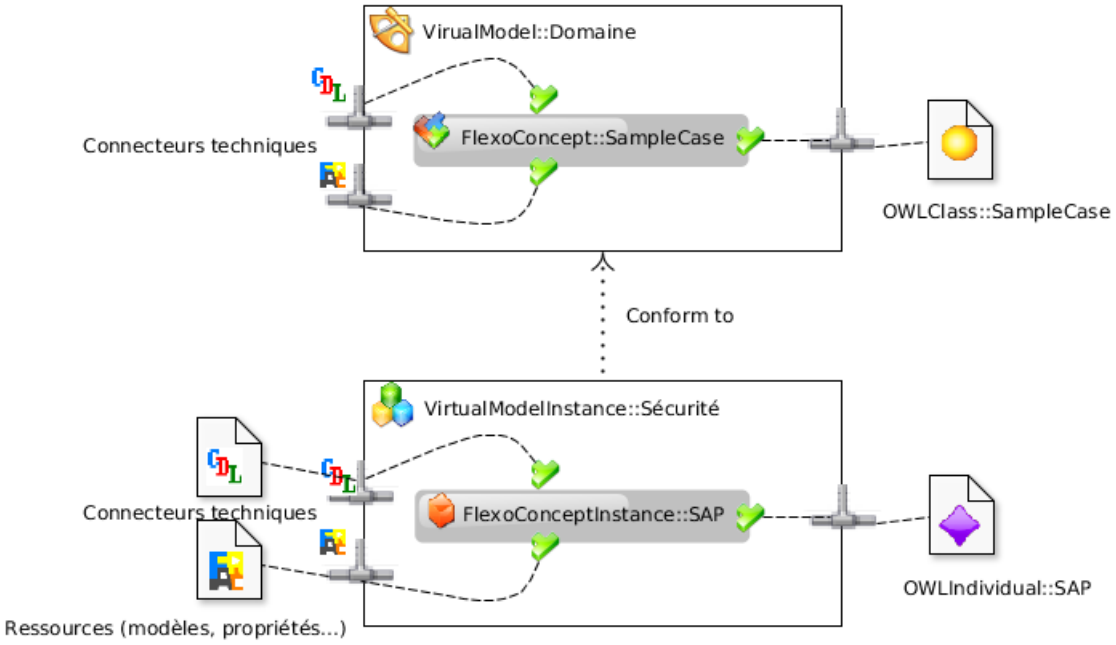

FIGURE 7.4 – Principe de la fédération pour le VOS

Un connecteur technique permet de fédérer l'ensemble à une ontologie, permettant ainsi de l'inférence et du partage de connaissances. Ainsi, chaque *flexoconcept* est lié à une classe ontologique et chaque *flexoconceptinstance* est associé à un *individual* via le connecteur technique OWL. Il existe donc un isomorphisme entre un regroupement d'objets physiques du niveau de stockage et les objets sémantiques de la couche ontologique.

Niveau d'accès. L'infrastructure Openflexo fournit une couche d'accès de différentes manières. D'abord un atelier de modélisation flexible et modulable peut être utilisé pour accéder aux diverses informations, mais aussi pour déclencher les comportements associés aux *flexoconcepts*. Pour permettre aux non initiés à Openflexo d'accéder aux informations, un serveur HTTP Vert.x[3] a été développé. Il propose un ensemble de services *REST*, utilisés par un front end de type single page application réalisé en Typescript/CSS(MDL)/HTML, ergonomique et plus adapté aux humains.

7.3 Illustrations

Le VOS organise donc les connaissances et les interactions qu'elles supportent. Dans cette section nous présentons plus précisément la structure des connaissances au moyen

3. https ://vertx.io/

d'exemples, ainsi que les interactions supportées et les moyens mis en œuvre pour la capture des connaissances.

La structure des connaissances est décrite par des classes et des propriétés (au sens ontologique). Cette structure est représentée en haut de chaque illustration par des rectangles gris (les classes) et des flèches (les propriétés). Pour simplifier les illustrations, nous représentons les sous-classes soit avec une flèche blanche pointillée, soit en positionnant une sous-classe à l'intérieur de sa superclasse. Conformément à la structure définie par ces classes, un ensemble de connaissances est instancié sous la forme d'un réseau d'individus. Chaque individu est représenté par un ovale au centre de la figure, et peut être lié à d'autres individus (flèche noire entre individus). Le lien d'instanciation entre un individu et sa classe est symbolisé par la flèche grise pointillée. Un ensemble d'interactions supportées par ces connaissances est présenté en bas de chaque figure, ces interactions sont extraites des chapitres précédents.

7.3.1 Connaissances liées au model checking

L'illustration 7.5 ci-dessous est un extrait de l'ontologie du point de vue des connaissances liées au model checking. En haut figurent les différentes classes qui ont été présentées au chapitre 5, au centre quelques individus et en bas des interactions propres à ces individus. Plus précisément il s'agit d'une vue sur l'application en bas à gauche, et une trace en bas à droite.

Structure

Les éléments techniques (*technical elements*) sont partagés en trois sous-ensembles, les éléments de spécification (*specification element*), les éléments de conception (*design element*) et les éléments de vérification (*verification elements*). Les éléments de spécification sont essentiellement des propriétés formelles (*formal property*), par exemple la propriété $PNotPlc_1Plc_2Access$. Chaque propriété est associée à une formule logique (*formula*), ici $PNotPlc_1Plc_2Access$ est associée à la formule $\Box\neg(plc_1@Access \wedge plc_2@Access)$. Une formule logique est un agrégat de propositions atomiques *(atomic proposition*), comme par exemple $plc_1@Access$, qui sont liées au moyen d'opérateurs (*operator*) par exemple $\wedge$. Une proposition atomique est liée à un ensemble d'éléments du modèle (*model element*), comme ici, la proposition $plc_1@Access$ est liée à l'état *Access* du processus plc_1. Le modèle (*model*), par exemple *Scada*, contient un ensemble de processus (*process*), ici plc_1 et plc_2, ou des variables (*variables*) partagées entre processus, comme $flag(boolean, boolean)$. Tout processus contient des variables locales, des états (*state*) et des transitions (*transition*), comme ici les états *Access*.

Une tentative de vérification (*run*) est réalisée à partir d'un modèle et de spécifications.

Chacune des spécifications, exprimée par une propriété, est vérifiée dans l'élément *propertyverfication*. Ici nous avons une vérification de propriété, celle de $PNotPlc_1Plc_2Access$. Une vérification de propriété nous fournit des résultats concernant l'évaluation de la propriété (sauf si l'exploration n'est pas possible). Un espace d'état (LTS) est produit ainsi qu'un contre-exemple, ici la trace $T12345A$. Une trace est un ensemble de configurations ordonné (*configuration*) possédant des informations (*configurationElement*) à propos d'éléments de modèles ou de spécifications. Par exemple nous avons ici deux configurations *C45Z76* et *C458A6*, chacune disposant d'informations sur l'état du *flag* et l'état courant des processus.

Présentations

Ces connaissances et leurs relations sont complexes, mais nous sont utiles pour définir différentes interactions facilitant la construction de la solution. En bas de l'illustration, ces connaissances liées au model checking ont permis de produire différentes vues déjà présentées dans le mémoire au chapitre 5.5.1. La première vue est un affichage synthétique d'un sample case. Celle-ci permet de naviguer rapidement dans les éléments pertinents d'une application et des tentatives de vérifications. La seconde vue est la trace présentée au chapitre 4.3.2, elle met en évidence les éléments de configurations modifiés entre deux configurations d'une trace. D'autres vues peuvent être produites, comme des vues matricielles permettant de corréler les propriétés par rapport aux processus ou aux scénarios.

Ingestions

Une des difficultés de ce type de système est la capture des informations. Il est bien sûr possible de les expliciter manuellement à travers un formulaire d'ingestion, qui est fourni par le VOS, mais il est moins laborieux d'automatiser leur acquisition. Le VOS facilite cette capacité d'ingestion de deux manières. D'une part il propose un ensemble de connecteurs techniques (incluant des parseurs) réifiant les ressources physiques (par exemple un fichier Fiacre) dans la structure du VOS de façon automatique. S'ils ne satisfont pas à l'utilisateur, le VOS offre la capacité d'ajouter de nouveaux connecteurs techniques. D'autre part, il joue le rôle d'une interface de commande en proposant des interactions. Par exemple, il est possible de déclencher l'exploration exhaustive avec OBP à partir d'un ensemble de propriétés (extraites directement d'un fichier cdl via le connecteur technique cdl), d'un modèle (connecteur technique fiacre), et d'une configuration du model checker. Après analyse, les résultats sont automatiquement rangés : un fichier de trace devient un individu trace ; le fichier sera parsé et des configurations crées pour cet individu trace.

Avec seulement les connaissances sur le SAE, ces vues ne permettent pas de construire

des vues de plus haut niveau. Il faut pour ça corréler ces connaissances avec d'autres connaissances.

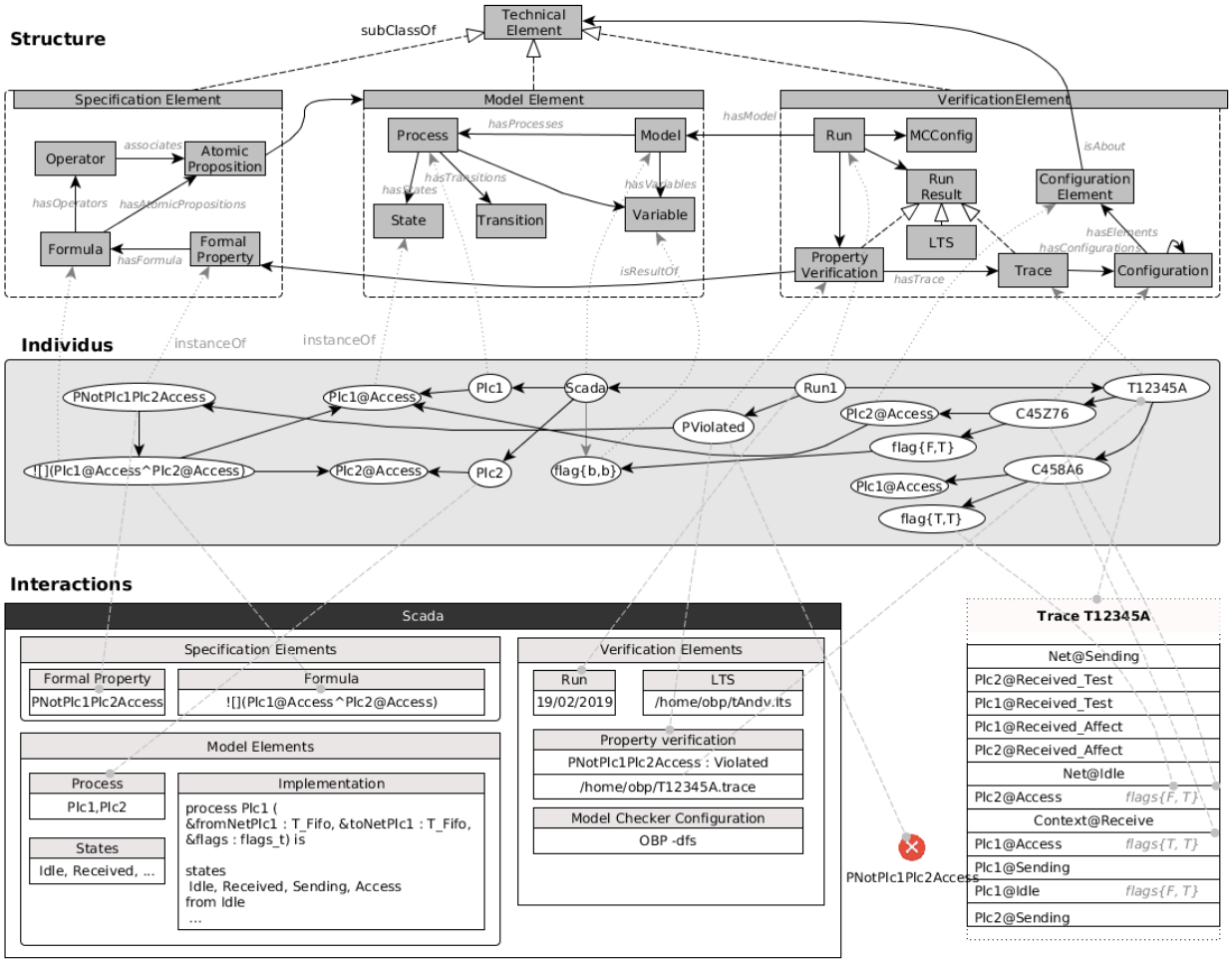

FIGURE 7.5 – Connaissances liées au model checking

7.3.2 Utilisation du domaine pour le diagnostic

Structure

Dans le cas précèdent le domaine n'est pas capturé dans le VOS. Dans l'illustration 7.6, on complète les connaissances précédentes avec des connaissances sur les problèmes du domaine. Dans la partie du haut, la structure des éléments du SAE a déjà été capturée (*design elements*, *specification elements* et *verification elements*). Un problème du domaine (*problem case*), par exemple le problème de l'exclusion mutuelle *PMutex*, est caractérisé

par un ensemble d'énoncés (*problem statement*) composés de propriétés (*abstract property*) ou d'architectures abstraites (*abstract architecture*) sur lesquelles vont s'appliquer les propriétés. Une propriété abstraite, comme ($PMutexPriPrj$), regroupe un ensemble de propositions atomiques abstraites (*abstract atomic proposition*) comme Pr_iCS, à implémenter pour la solution choisie, comme par exemple $Plc_1@Access$. Une architecture abstraite est composée d'éléments structurels (*abstract structure*) comme par exemple Pr_i, ou comportementaux (*abstract behavior*) comme Pr_iCS. De même que pour les propriétés abstraites, ils doivent correspondre à des éléments concrets dans le modèle. Ainsi, Pr_i est concrétisé par Plc_1.

Présentations

A partir des connaissances de domaine corrélées aux connaissances du SAE, par exemple il existe un chemin entre la configuration $C45Z76$ et le concept de domaine Pr_jCS, directement lié à Plc_2, il est possible de générer le point de vue de la trace par rapport au domaine comme le montre la figure en bas de l'illustration, extraite du chapitre 5.2.3. D'autres interactions sont possibles comme des visualisations propres au domaine, ou bien générer des propriétés propres au domaine.

Ingestions

Les connaissances de domaine sont abstraites. Elles sont utilisées dans divers contextes, en associant les éléments de problème abstraits aux éléments concrets de la solution construite. Il devient alors possible de générer les vues de domaine. Comme dans le cas précédent, les liens peuvent être définis à la main via des formulaires, ou bien partiellement automatisés par des interactions du VOS (sélection du problème de domaine, association à des éléments de solution via un système de nommage par exemple).

7.3.3 Application d'un pattern case

Dans cette nouvelle illustration nous allons présenter la réutilisation d'une solution au *problem case* définit précédemment.

Structure

Une solution à ce problème est illustrée sur la figure 7.7. Nous appelons les solutions à un problème du domaine les *solution cases*, par exemple $Peterson$ est une solution au problème de l'exclusion mutuelle. Une solution vise à résoudre un *problem case* grâce à un ensemble d'éléments de solutions (*solution statements*). Un *solution statement* concrétise une partie de problème. Par exemple, le processus Pr_1 est une concrétisation de l'élément Pr_i

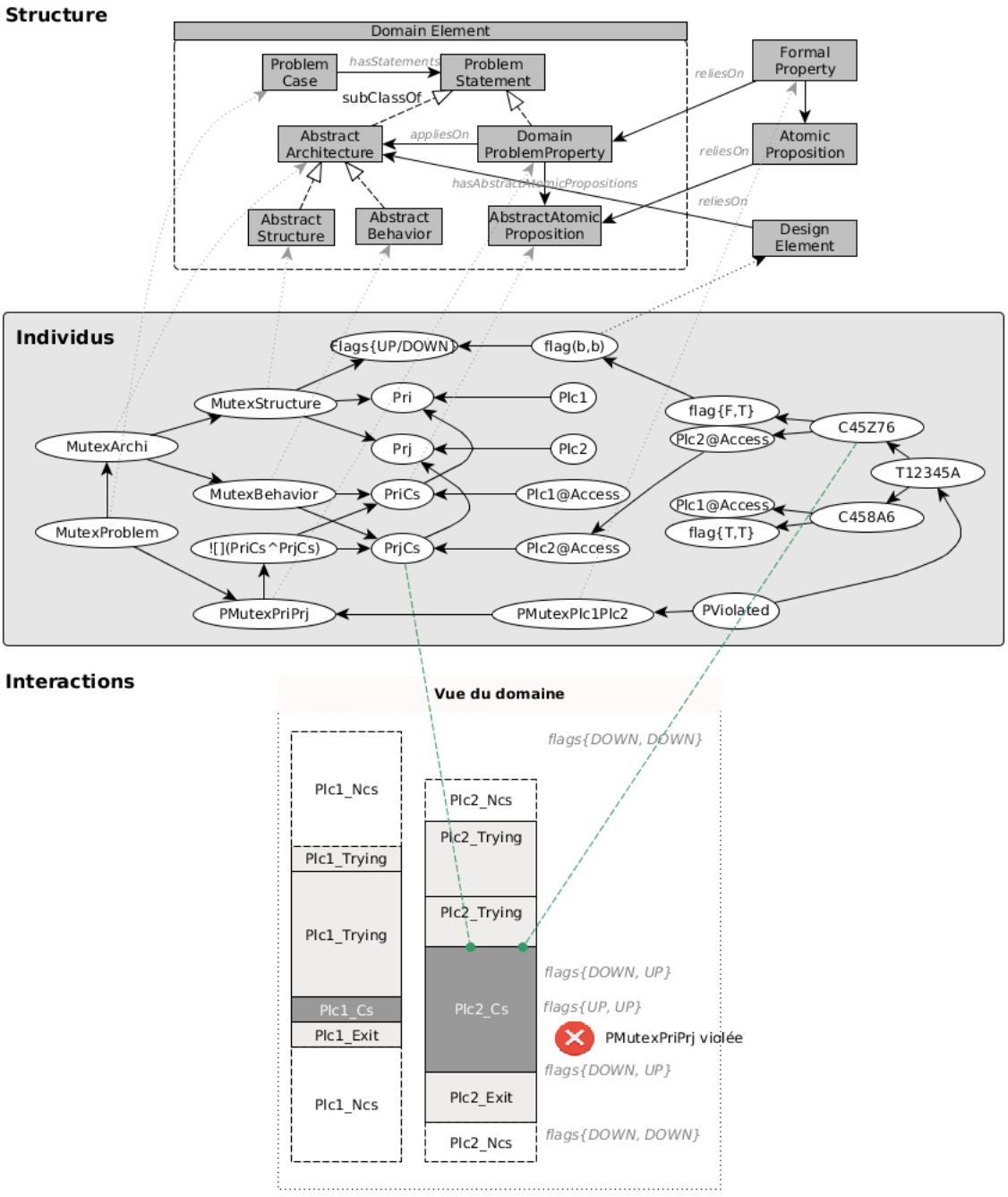

FIGURE 7.6 – Connaissances de problèmes domaine

de l'architecture abstraite du problème de l'exclusion mutuelle. De même, une propriété de solution de domaine (*domain solution property*) concrétise une propriété de problème de domaine. Par exemple $\Box\neg(Pr_1Cs \wedge Pr_2Cs)$ est la concrétisation de la propriété de problème du domaine $\Box\neg(Pr_iCs \wedge Pr_jCs)$. Il existe différentes solutions, le *sample case* est une solution capturée dans sa forme brute, le *component case* est réutilisable dans de nouveaux contextes par le biais de contrats à réaliser avec les éléments de l'application, et enfin le *pattern case* s'utilise par le biais d'un ensemble de règles de transformations (*transformation rule*). Une *transformation rule* prend en arguments un ensemble d'éléments de l'application, et un ensemble d'éléments de la solution du domaine, autrement dit $transfo(designElements, domainElements) \rightarrow solutionElements$. Ces transformations peuvent être des copies (*copy*), dans ce cas elles rajoutent du contenu dans l'application, ou bien des adaptations plus complexes (*adaptation*), lesquelles nécessiteront l'usage d'un langage de transformation (le VOS fournit un langage par l'intermédiaire d'Openflexo). Pour appliquer la solution Peterson à notre application il faut exprimer des règles. La règle $t(Pr_1, Plc)$ adapte un processus Pr_1 en un processus Plc, il s'agit en clair d'un changement de nom. La règle $t(Wait, Receive)$ adapte l'opération de *trying* à l'état *receive* en la sectionnant en deux nouveaux états $receive\_affect$ et $receive\_test$. La règle $t(flag)$ ajoute la déclaration d'une variable partagée (un drapeau) à l'application.

Pour pouvoir définir ces règles, il faut avoir en tête l'architecture de l'application. Supposons que l'on ait hérité d'une situation antérieure, schématisée par C_i dans la figure 7.8. Celle-ci est une solution S_i répondant aux propriétés P_i. À l'itération suivante, nous souhaitons appliquer un *pattern case*. Pour l'appliquer, toute la solution précédente (C_i) va jouer le rôle de problème ($P_i + 1$). Pour appliquer un patron il faut s'appuyer sur une architecture, celle-ci vient donc de la solution produite à l'itération précédente.

Présentations

Dans le cas d'un *pattern case*, la solution de l'application peut être générée à partir de règles entre les éléments de l'application définis lors de l'itération précédente et les éléments du domaine. Cette génération peut se faire pour l'architecture, dans ce cas le principal bénéfice sera un gain de temps ainsi qu'une diminution des risques d'erreurs lors de la conception de la solution. Elle peut aussi se faire sur les propriétés, dans ce cas les propriétés générées peuvent aider à isoler le problème lors de la phase de diagnostic. Le bas de la figure 7.8 montre à gauche le code de l'application et à droite le code de la solution produite. En vert figurent des éléments de solutions copiés et en jaune des éléments de solutions générés grâce au patron. Cette vue permet d'isoler les informations liées au domaine, des informations spécifiques à l'application.

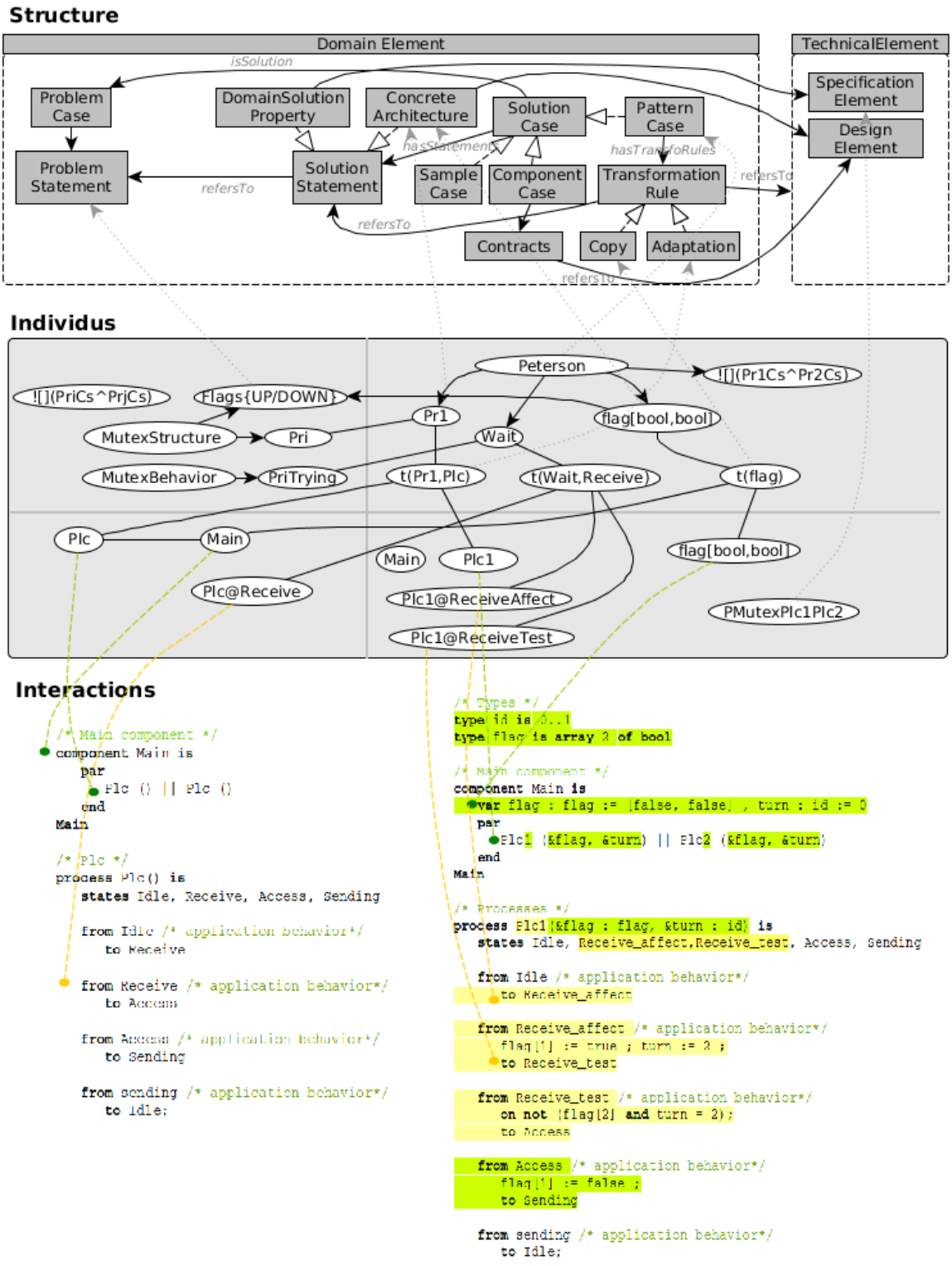

FIGURE 7.7 – Connaissances de solution domaine

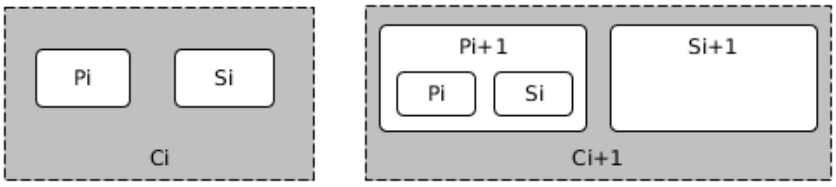

FIGURE 7.8 – Application du pattern case, contexte

Ingestion

Disposer de morceaux de solutions réutilisables ou bien de règles de génération permet de progresser plus rapidement dans la solution, ainsi que d'éviter de produire des liens inutiles dans le VOS. Prenons un système Scada avec une centaine de PLC à vérifier, tous redondants. Les instancier tous dans le VOS va nécessiter de créer beaucoup d'individus et de liens. S'ils sont construits sur la base d'un même modèle abstrait, il est possible de générer directement ces individus ou liens au moment où ils sont nécessaires.

7.3.4 Construction d'une solution et connaissances de gestion

Structure

L'illustration 7.9 présente les connaissances de gestion et un exemple d'interaction. Dans les connaissances de gestion nous avons regroupé les processus cognitifs (*reasoningProcess*) qui sont une succession d'activités cognitives plus simple (*cognitiveActivity*), comme la recherche et la récupération de problèmes ou solutions de domaine (*retrieve*), la conception de la solution de l'application (*design*), la vérification de la solution (*verify*) ou encore la capture d'une solution ou problème (*retain*). Une activité cognitive adresse une révision particulière de la solution (*revision*). Cette révision est composée d'éléments techniques et d'éléments de domaine. Par exemple la révision $revision_1$ s'applique sur la solution $solutionflag_3$, le problème $MutexProblem$ et au problème de l'application *Scada*. Une révision peut être construite à partir d'une révision précédente. Ce mécanisme est laissé volontairement simple, le rôle du VOS n'étant pas de suppléer au système de gestion de version gérant déjà les ressources physiques ingérées par le VOS.

Présentation

À partir de cette structure, il est possible de générer la vue du chapitre 5.2.7. Celle-ci met en avant les successions d'essais de construction de l'application. L'intérêt est double, faciliter la maintenance de la solution, imaginons par exemple que la solution soit reprise par un ingénieur n'ayant pas la connaissance de l'historique de sa construction, et

faciliter le diagnostic, car au cours de la construction et des multiples essais, l'ingénieur a sélectionné ou désélectionné des propriétés. Lors de la vérification l'ingénieur a besoin des propriétés qui caractérisent le mieux la solution.

La capture dans le temps des solutions construites et leurs évaluations permet des analyses basées sur l'historique (statistiques, probabilités, algorithmes de machine learning ...). Il est possible de proposer au reuse les solutions les plus pertinentes à l'ingénieur (en prenant en compte son domaine ou ses habitudes).

Enfin, en capturant la trace des activités cognitives réalisées par l'ingénieur, des outils pourraient permettre de mieux comprendre le fonctionnement du processus mental de résolution de problèmes. Se souvenir du processus passé permet de ne pas refaire les mêmes erreurs ou ne pas repartir de zéro.

Ingestion

Certaines connaissances de gestion peuvent être capturées automatiquement grâce à des interactions du VOS, comme lancer la vérification, sélectionner des problèmes ou des solutions pertinentes etc...

7.4 Conclusion

Le processus de conception et vérification par model checking produit une multitude d'informations corrélées et hétérogènes. Pour permettre leur organisation et fournir des interactions, il faut les organiser au sein d'une structure. Le VOS est une spécification simple et adaptable d'une structure répondant à ce besoin. Le VOS permet de réaliser les différentes interactions présentées dans la thèse (réutilisation de connaissances, vues...). Chaque organisation doit l'adapter à ses contraintes et usages métiers, et proposer ses propres interactions.

Pour qu'une telle structure soit efficace, elle doit contenir des connaissances, et fournir des interactions. Dans le chapitre suivant, nous remplissons la structure avec des connaissances, et spécifions différentes interactions.

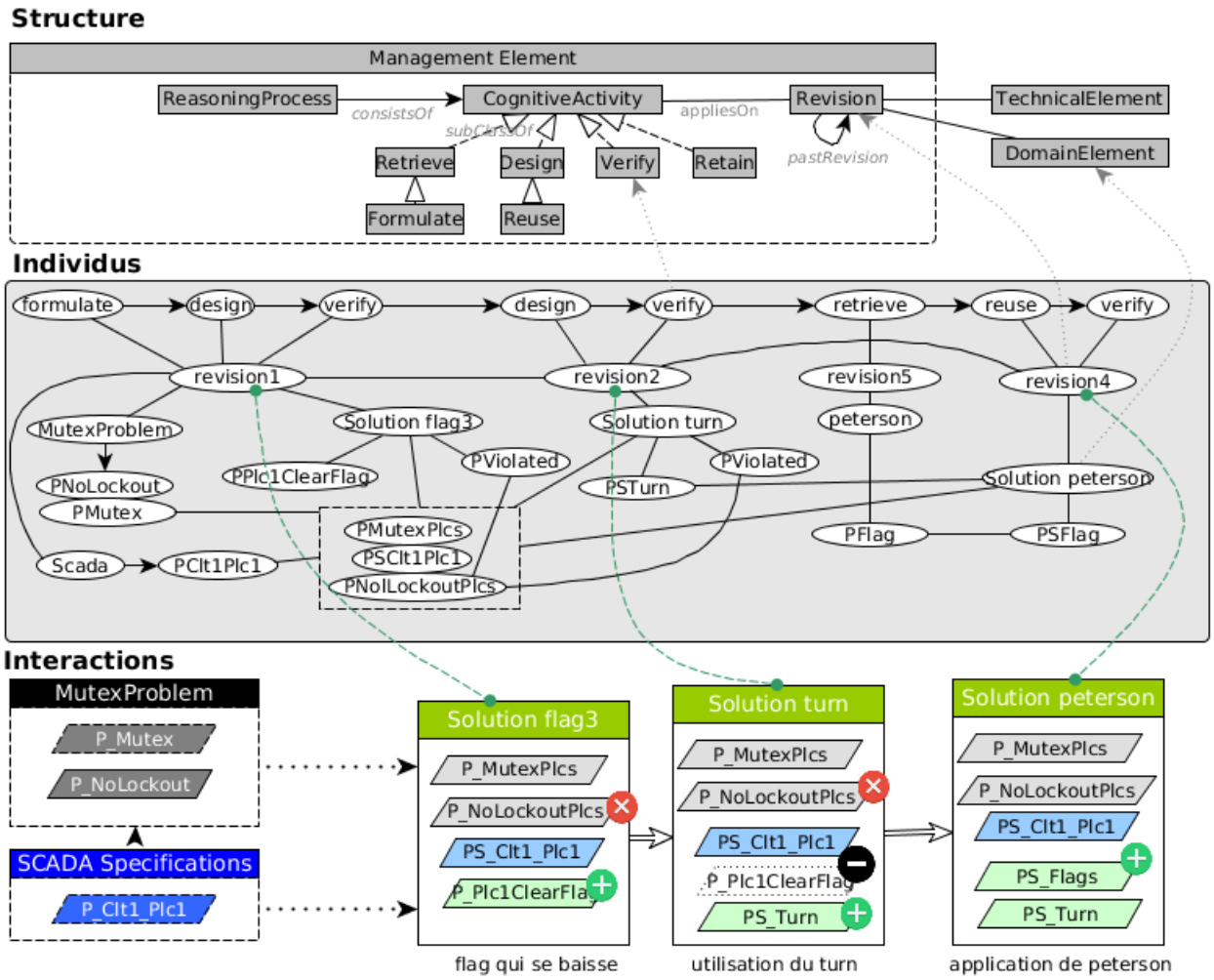

FIGURE 7.9 – Connaissances de gestion

Chapitre 8

Exemple d'utilisation du VOS

8.1 Ingestion de connaissances

8.1.1 Beem

Présentation du benchmark

Pour constituer un ensemble de problèmes et de solutions réutilisables, il est nécessaire de s'appuyer sur l'ingénierie du domaine, une pratique non répandue dans le domaine du model checking. Heureusement, il existe dans cette communauté des benchmarks dont ont peut partir pour disposer du domaine. Le BEEM [Pel07] est un benchmark qui est utilisé pour évaluer les outils et les algorithmes des model checkers explicites. Cet ensemble de référence comprend plus de 50 modèles paramétrés (soit 300 exemples concrets), associés à leurs propriétés de sûreté ou de vivacité, des informations détaillées sur tous les modèles et sur les espaces d'états.

Les modèles du BEEM sont initialement implémentés dans un langage de modélisation de bas niveau basé sur des machines à états finis communicantes appelé DVE. Des transformations permettent de passer automatiquement de modèles DVE en modèles Fiacre ou Promela. Ces modèles sont pour la plupart des exemples bien connus couvrant divers domaines d'applications. Les domaines sont les suivants : - les algorithmes d'exclusion mutuelle ; - les protocoles de communication ; - les contrôleurs ; - les algorithmes d'élection de leader ; - les ordonnancements ; - les énigmes. Chaque modèle est classé selon sa complexité, allant d'un exemple très simple spécifié en quelques lignes de code (*toy*), à un étude de cas complexe dont la description comporte plus de 100 lignes de code (*complex*).

Ces modèles sont accompagnés de propriétés. Deux types de propriétés sont prises en charges, les propriétés d'atteignabilité et les propriétés de logique temporelle linéaire (LTL). Les propriétés sont exprimées par des propositions atomiques définies à travers d'expressions sur des variables de modèle. Les modèles ne sont pas tous corrects et certains contiennent des erreurs.

Les domaines d'applications

Les algorithmes d'exclusion mutuelle garantissent un accès exclusif à une ressource partagée. Les modèles de ces algorithmes se composent généralement de plusieurs processus presque identiques qui communiquent via des variables partagées. Les algorithmes appartenant à cette catégorie sont par exemple l'algorithme du boulanger ou de Peterson.

Les protocoles de communication ont pour objectif de garantir la communication sur un support peu fiable. Un modèle de protocole de communication comprend généralement un processus émetteur, un processus récepteur et un bus ou un support. Les processus communiquent par poignée de main et les variables partagées ne sont pas utilisées. Les protocoles sont par exemple le bounded retransmission ou le collision avoidance.

Les algorithmes d'élection de leaders ont pour objectif de choisir un leader unique parmi un ensemble de nœuds. Ces modèles consistent en un ensemble de processus connectés dans un anneau, un arbre ou un graphe ; la communication s'effectue via des canaux bufferisés. Parmi ces modèles ont trouve par exemple les algorithmes basés sur l'extinction et sur les filtres, le Firewire (IEEE 1394) tree identification protocol ou le Lann leader election algorithm for token ring.

Les contrôleurs sont généralement des modèles disposant d'une architecture centralisée : un processus de contrôle communique avec différents processus représentant des parties du système. La communication peut se faire à la fois par des variables partagées et par poignée de main. Les modèles sont par exemple le contrôleur de puissance audio / vidéo, le contrôleur d'un ascenseur, le contrôleur de vitesse ou le contrôleur de porte de train.

Les énigmes, les ordonnanceurs, et autres incluent divers modèles dont par exemple un modèle d'ordonnancement de machines de production, différentes énigmes (Bridge puzzle ou Peg solitaire puzzle), un protocole de service de télécommunication, des algorithmes d'authentification (Needham-Schroeder) etc....

8.1.2 Intégration au VOS

Structure du Beem

Le BEEM représente une base de cas de grande taille reconnue en model checking, il est donc naturel de s'inspirer du BEEM pour structurer les connaissances du VOS, aussi la structure des éléments du BEEM et du VOS est très proche. Chaque modèle du BEEM est décrit par un dossier dans lequel se trouve le fichier contenant le modèle (.mdve et .fiacre) pouvant être paramétrés. Pour Peterson par exemple, le nombre de processus participant à l'exclusion mutuelle est paramétrable (N). Le modèle est accompagné d'une description (fichier .xml) dans lequel se trouve son nom accompagné de descriptions, du