Acknowledgement

«The discovery of nuclear reactions need not bring about the destruction of mankind any more than the discovery of matches.»

Albert Einestein

Contents

Résumé en français

(French Summary)

L'Ingènierie dirigèe par les modèles (IDM) est une approche de spècifier, construire, valider et maintenir des systèmes logiciels complexes en utilisant des objets de première classe appelè *modèles*. IDM a èmergè à partir d'un certain nombre de domaines dans le dèveloppement de logiciels tels que l'analyse orientèe objet et langages de conception, des mèthodologies orientèes objet [7] [33] [63], et Computer-Aided Software Engineering (CASE) efforts dans les annèes 80 et 90 afin d'automatiser plusieurs ètapes dans gènie logiciel [55] [8].

Les modèles sont *graphs d'objets interconnectès* dans un *domaine de modèlisation*. Un domaine de modèlisation dèfinit une *ensemble des modèles* où chaque modèle est construit en utilisant un ensemble commun de concepts et des relations. Par exemple, dans cette thèse, nous considèrons la spècification de deux domaines de modèlisation : (a) les *mètamodèles* qui spècifient un ensemble de modèles dans un langage de modèlisation (b) diagrammes des features qui spècifient un ensemble des produits qui donne lieu à un ligne de produits logiciel (LPL). Très souvent, la crèation des *modèles efficaces* dans un domaine de modèlisation exigent la satisfaction de contraintes à partir de sources hètèrogènes. Par exemple, la crèation d'un modèle de workflow en utilisant le diagramme d'activitès bien connues Unified Modelling Language (UML) exige que le modèle de satisfaire UML règles de forme, la logique mètier, les contraintes èconomiques, la qualitè de contraintes de service, et les restrictions de sècuritè. Les modèlisateurs

avec l'expèrience progressivement crèer les modèles en vigueur par tacite veiller à ce que les modèles sont *correcte par construction* et satisfaire les contraintes provenant de sources hètèrogènes. Malgrè tout, ce processus est extrêmement pènible et parfois impossible, s'il ya un besoin de crèer des milliers de modèles. **Peut-on automatiser la crèation de modèles efficaces compte tenu de l'hètèrogènèitè des sources de la connaissance ?** C'est la question qui nous intrigue et le sujet de cette thèse.

L'introduction est organisè comme la suit. La notion de dècouverte modèle efficace se situe dans le contexte global de dècouvrir des structures efficaces dans les sciences de l'ingènierie. Nous dècrivons brièvement ce contexte global dans la section 0.1. Cette thèse aborde le problème de l'automatisation de la dècouverte dans le contexte contemporain et spècifiques de la IDM que nous dècrivons dans la section 0.2. Un certain nombre de scènarios dans IDM nècessite la gènèration de modèles efficaces. Notre motivation vient de ces scènarios que nous dècrivons dans la section 0.3. Dans la section 0.4, nous prèsentons le fichier *contexte du problème gènèral* et ses dèfis. Nous prèsentons notre thèse et de dècrire notre mèthode de modèle efficace de dècouverte automatique et de produits dans la section 0.5. Nous faisons appel à des contributions de notre thèse dans la section 0.6. Enfin, nous prèsentons l'organisation de la thèse dans la section 0.7.

0.1 Discovery of Effective Structures in Science and Engineering

Les découvertes scientifiques se termine souvent en représentant la structure dans la nature comme un *réseaux d'entités* ou *graph des objets*. Par exemple,

- **Les réseaux trophiques** sont des représentations des relations prédateur-proie entre les espèces dans un écosystème ou d'habitat. Un exemple courant est le *réseau alimentaire du sol* illustré à la figure 1. La chaîne alimentaire du sol est souvent trouvé dans un jardin bio-compost.
- **Réseaux réaction biochimique** ou des voies métaboliques représentent essentiel des

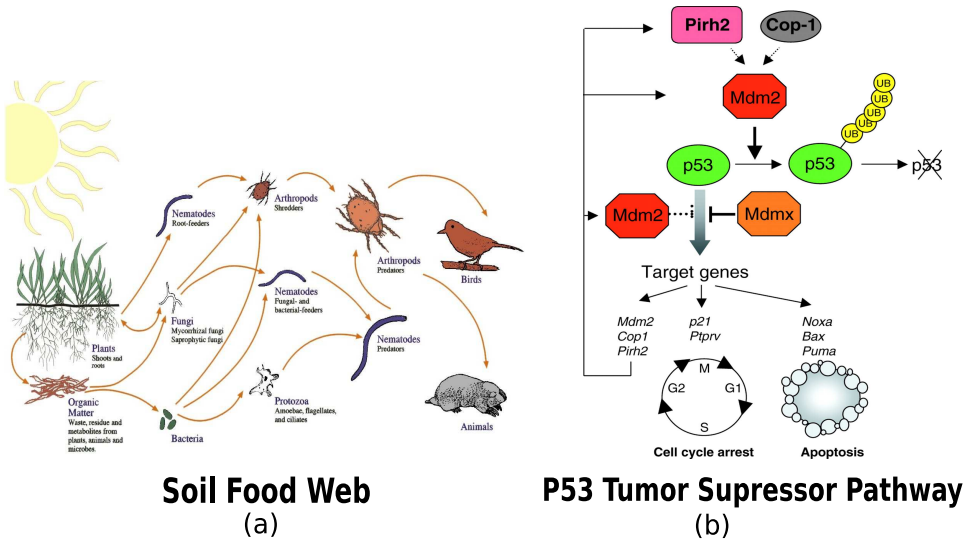

FIG. 1 – Des structures efficaces en découverte scienctific : (a) Web alimentaire du sol (b) Voie de suppression tumoral

échanges moléculaires dans les êtres vivants. La largement étudié *voie suppresseur de tumeur* de la figure 1 (b) illustre le rôle crucial de la protéine p53 dans la mort cellulaire. La mort cellulaire est important pour réguler la croissance cancéreuse.

De connaissances à partir d'expériences, l'analyse des données et mentale de plomb tacite à la découverte de tel *structures efficaces* dans la nature. L'existence de structures efficaces n'est pas limité à la virtuosité de la nature. Nous, les humains sont doués de la capacité de représenter et de créer des structures efficaces tels que les bâtiments, les ponts, les robots et logiciels complexes.

Résultats de *la conception en ingénierie* souvent en représentant efficace structures artificielles comme des graphes d'objets. Par exemple,

– **Diagrammes de circuits électroniques** représentent un réseau de composants électriques qui permettent d'atteindre un but donné. Le *récepteur FM Circuit* de la figure 2, par exemple, est utilisé dans des millions d'appareils radio.
– **Design patterns** représentent en général des solutions réutilisables aux problèmes fréquemment rencontrés dans la conception de logiciels. Ils sont souvent représentés comme

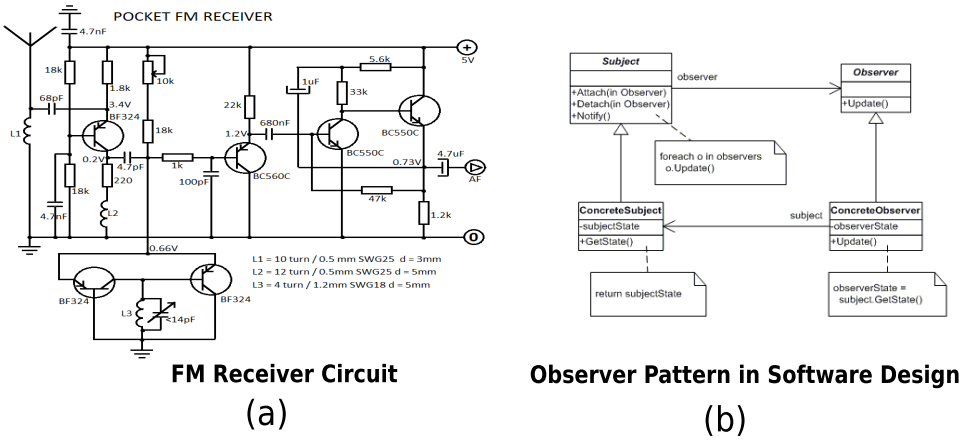

FIG. 2 – Des structures efficaces en ingénierie : (a) Circuit radio FM(b) Design pattern observateur

*les diagrammes de classes.*The de logiciels orientés objet emph (motif) observateur dans Figure 2 (b) est un modèle commun dans les logiciels distribués nécessitant la gestion des événements. Le célèbre logiciel d'édition photo Adobe Photoshop est un exemple.

Très semblable à la découverte de la science, de la conception en ingénierie est guidée par la connaissance d'un certain nombre de sources couplées avec la créativité d'un ingénieur. Puis ce processus de découverte scientifique ou de conception en ingénierie en utilisant diverses sources de connaissances sont *automatisé* ? Cette question a fait l'objet d'études depuis plusieurs décennies avec l'avènement de l'informatique moderne.

Programmes informatiques ont été utilisés pour découvrir la structure dans la nature. Par exemple, inspiré par une logique de Karl Popper, de la découverte scientifique [61], Pat Langley, Herbert Simon, G. Bradshaw, et Zytkow mis au point plusieurs programmes informatiques tels que Bacon, Glauber, Dalton, and Stahl décrit dans leur livre [47]. Ces programmes ont été guidés par des heuristiques pour réussir à re-découvrir des lois historiques de la chimie.

Approches de calcul de l'évolution ont été développer pour automatiser la conception dans l'ingénierie tels que la production de circuits électroniques [43]. Les programmes d'ordinateur la mise en œuvre d'un concours approche évolutive pour le "Humies Award" conféré chaque année

à la GECCO conference. L'attribution de $10,000 est accordée à l'approche la plus compétitive à l'égard de l'homme. Dans la communauté du génie logiciel, des conférences récentes, telles que l'Automated Software Engineering(ASE) conférence fournir des sites de compétition pour la présentation des approches pour générer des structures de logiciels.

Dans cette thèse, nous abordons la question de la découverte automatique dans le contexte contemporain du l'ingènierie dirigèe par les modèles de systèmes logiciels complexes.

0.2 Contexte : L'Ingènierie dirigèe par les mod èles

IDM [53] vise à graisser les roues de la création de logiciels complexes en utilisant des objets de première classe appelé *modèles*. La philosophie IDM utilise des modèles pour représenter des objets importants dans un système comme les exigences, les dessins de haut niveau, des structures de données, les vues, les interfaces, les transformations de modèles, des scénarios de test, et des objets de mise en $\sqrt{}$èuvre au niveau telles que le code source.Un modèle est construit dans un *domaine de la modélisation* qui capture un ensemble de concepts communs et les relations. La construction d'un modèle dans un domaine de modélisation peutêtre encore réduite en utilisant des règles de bonne formation et des contraintes de diverses sources.

La notion générale d'un domaine de modélisation peut être *spécialisées* à de nombreux égards. Une description précise de concepts et de relations qui définit un ensemble de modèles est un domaine de modélisation. Par exemple, un *métamodèle* spécifie le domaine de la modélisation d'un langage de modélisation. Le célèbre Unified Modelling Language (UML) langage de modélisation [57] a son propre méta-modèle qui spécifie l'ensemble des modèles UML. Un autre exemple, d'un domaine de modélisation est un *feature diagram* ou *feature model* qui spécifie un ensemble de produits dans un ligne des produits logiciels (LPL). Modèles dans un domaine de modélisation peutêtre chargé stocké, manipulé et transformé à d'autres modèles de mise en $\sqrt{}$èuvre des artefacts / pour résoudre les problèmes logiciels.

IDM fournit un certain nombre de processus logiciels et des technologies pour permettre la

création de la modélisation des domaines et la transformation de ses modèles. Historiquement, le Model-Driven Architecture (MDA) est la marque commercialisés par l'Object Management Group (OMG), présente une approche pilotée par les modèles de développement du système. L'approche MDA commence le développement d'un domaine de modélisation pour les modèles indépendants de la plateforme (PIM), qui sont progressivement transformés ou raffinés dans des modèles de plateforme de niveau inférieur spécifiques (PSM) dans un autre domaine de la modélisation. Le PSM sont réifiés dans des artefacts tels que la code executable. Cette construction automatique des systèmes à partir de modèles de haut niveau permet à l'expertise en génie logiciel àêtre capturé comme *transformations de modèles réutilisables* appliquées de manière plus fiable et efficace. Actuellement, le cadre largement accepté pour la spécification des domaines de modélisation est la Eclipse Modeling Framework (EMF) [27]. Par exemple, métamodèles sont créés dans le EMF *Ecore* format de préciser le domaine d'un langage de modélisation. Langages de transformation des modèles [74] telles que le Kermeta impératif [40] [52], fondé sur des règles ATL [36] [35] [1], baser sur grammaire de graphes AToM^3 [29], Viatra [77] transformer les modèles. langages de transformation de modèle sont tenus de se conformer à standard Query-View-Transformation (QVT) [35]. Différents types de transformations de modèles peuventêtre créés en utilisant ces langues, comme classés dans [20]. Transformations de modèles peuvent transformer des modèles dans le même domaine de modélisation (transformations endogènes), entre les différents domaines de modélisation (transformations exogènes) et même réaliser la vision classique de la génération du code exécutable à partir d'un modèle de haut niveau.

Notre objectif dans cette thèse est la découverte automatique ou assistée, de découverte de modèles dans un domaine de modélisation.

0.3 Motivation : Pourquoi le besoin de découverte automatique modèle ?

Notre motivation pour la découverte automatique dans le contexte général de tiges d'ingénierie dirigée par les modèles existants de la découverte de calcul efforts dans des domaines hétérogènes. Ces domaines vont des systèmes [12] [62], au génie des systèmes physiques [49] [68] [26], [43]. Nous voyons la découverte automatique de modèles efficaces dans un domaine de la modélisation en tant que cadre général subsumant les approches existantes à la découverte de structure efficace dans des domaines hétérogènes de la science et l'ingénierie. IDM des systèmes à logiciel ne fait pas exception. Dans cette thèse, nous étudions trois scénarios IDM comme décrit ci-dessous :

0.3.1 Scénario 1 : La génération de tests pour les transformations de modèles

Transformations de modèles sont des artefacts logiciels de base en IDM. Un modèle simple de transformation MT prend les modèles d'entrée conforme à un métamodèle MM_I d'entrée et de sortie produit des modèles conformes au méta-modèle de sortie MM_O comme le montre la figure Figure 3. Pas tous les modèles spécifiés par le méta-modèle d'entrée peutêtre executer par la transformation du modèle. Par conséquent, nous composons des post-conditions $post(MT)$. Les transformations du modèle lui-même est construit en utilisant des connaissances à partir d'un ensemble d'exigences $MT_{Requirements}$.

Test d'une transformation de modèles nécessite modèle d'entrée qui permet de détecter des bogues dans la transformation MT. Création manuelle des modèles de test est fastidieuse car il doitêtre un graphe d'objets qui doiventêtre conformes aux MM_I, $pre(MT)$, et d'utiliser les connaissances de $MT_{Requirements}$. Création manuelle devient impossible lorsque nous avons besoin de créer des milliers de modèles de ces essais qui codent pour objectifs de test différentes. Par conséquent, il est clairement nécessaire de automatiser la génération de modèles d'essai qui satisfont les connaissances provenant de diverses sources telles que MM_I, $pre(MT)$, et

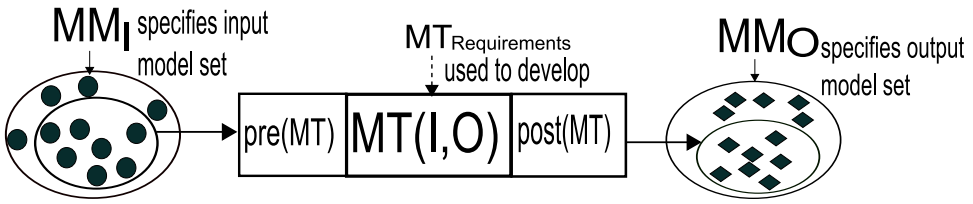

FIG. 3 – Une transformation

$MT_{Requirements}$. La génération automatique de modèles d'entrée exalte au niveau de la découverte automatique si nous validons qu'ils peuvent détecter les bugs dans une transformation. On peut qualifier l'efficacité des modèles d'essai par des techniques telles que *l'analyse de mutation* pour les transformations modèle [51]. Basé sur une description de ce scénario, nous demandons, *Comment pouvons-nous générer des modèles de tester et de qualifier leur efficacité pour la détection des bugs* ?

0.3.2 Scénario 2 : Achèvement d'un modèle partiel dans un éditeur de modèles

Les modélisateurs ont souvent recours à des éditeurs de modèles à construire des modèles progressivement. Par exemple, le éditeur TopCaseD [24] peutêtre utilisé à construire des modèles UML comme dans la figure 4. Le modèle présenté est une machine d'état incomplète en UML. Par exemple, le modèle ne contient pas un état initial qui est une règle de bonne formation. Il ya une infinité des moyens possible pour compléter le modèle tel qu'il devient une valable modèle UML de la machine d'état et répond à toutes les règles de bonne formation. Ce qui est probablement plus intéressant est la modèle plus proche qui est compatible à UML et qui contient tous les éléments du modèle partiel. Il peut y avoir un certain nombre de possibilités de mener à bien les modèles partiellement spécifié. On peut rapporter l'achèvement mode automatique au problème de complétion de code automatique dans les environnements de programmation [4]. Ce scénario soulève la question suivante : *Comment pouvons-nous des modèles de découverte automatique complète ou recommandations pour compléter les modèles partielle ?*

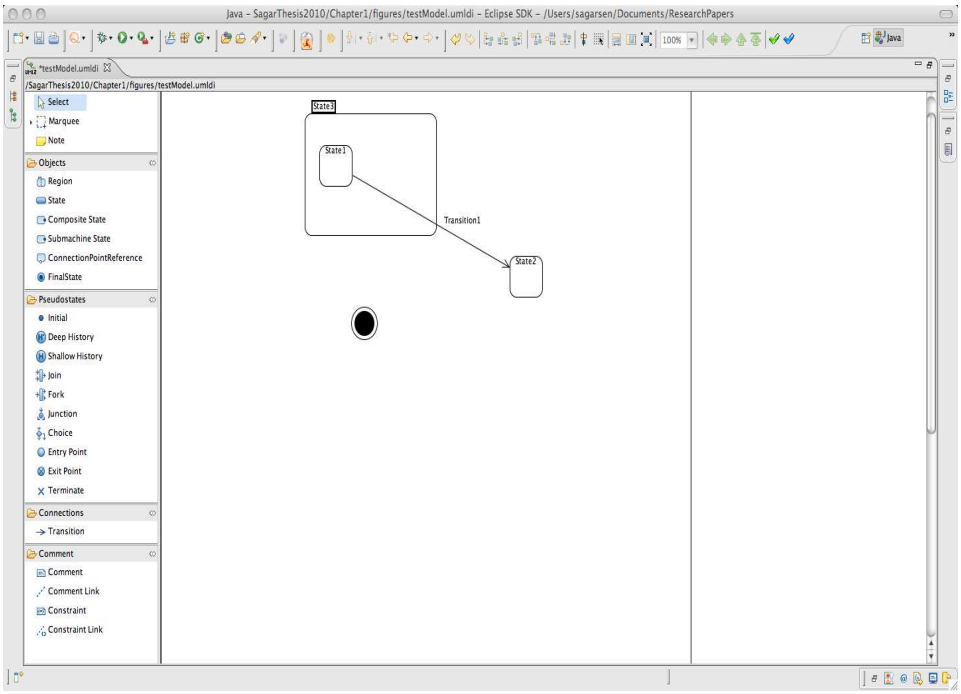

FIG. 4 – Modèle partiel dans l'éditeur de modèles UML : TopCaseD

0.3.3 Scénario 3 : La génération de produits dans une ligne des produits logiciels

Un ligne des produits logiciel (LPL) se réfère à un ensemble de produits partageant ensemble commune de caractéristiques/features qui répondent aux besoins spécifiques d'une mission particulière [15]. Un *Feature Diagram* (FD) ou un *feature model* précise un domaine de la modélisation d'un LPL. Feature diagrams introduite par Kang et al. [37] [38] compacte représentent tous les produits d'un LPL en termes de caractéristiques qui peuventêtre composés. Un FD se compose de *k* features $f_1, f_2, ..., f_k$ et les contraintes de dépendance entre les features. Par exemple, la sélection de certaines features dans un produit peut obligatoirement imposer la sélection d'autres features. En outre, certaines des features peutêtre associé à un actif de logiciels tels que service web. Considérons le FD pour un système de gestion des crises accident de voiture dans la figure 5. Le FD contient 47 éléments dont 25 d'entre eux sont optionnels. Certaines des features sont associées à des services ou des actifs logiciels. Le FD décrit 33, 554, 432 configurations différentes de features. Puis toutes les configurations se composer en des produits valide ? Pour répondre à cette question, il faut créer tous les produits ou un sous-ensemble représentatif de tous les produits. Par exemple, ce sont l'ensemble des produits qui répondent à l'interaction entre les paires de features. La création de ces produits nous aidera à dévoiler des produits non valide. Manuellement créer des produits qui satisfont toutes les contraintes FD est très fastidieux. Par conséquent, nous demandons, *Comment peut-on automatiser la génération de produits dans une ligne des produits logiciels pour différents objectifs ?*

0.4 Contexte du problème et défis

Nous sommes motivés par la nécessité pour la génération automatique de modèles efficaces dans un domaine de modélisation.Le contexte du problème de découverte automatique de modèle est illustré dans la figure6. Le contexte identifes les apports suivants :

– **Spécification d'un domaine de modélisation :** Le domaine de la modélisation précise d'un ensemble de modèles *M*. Exemples de spécifications pour les domaines de modéli-

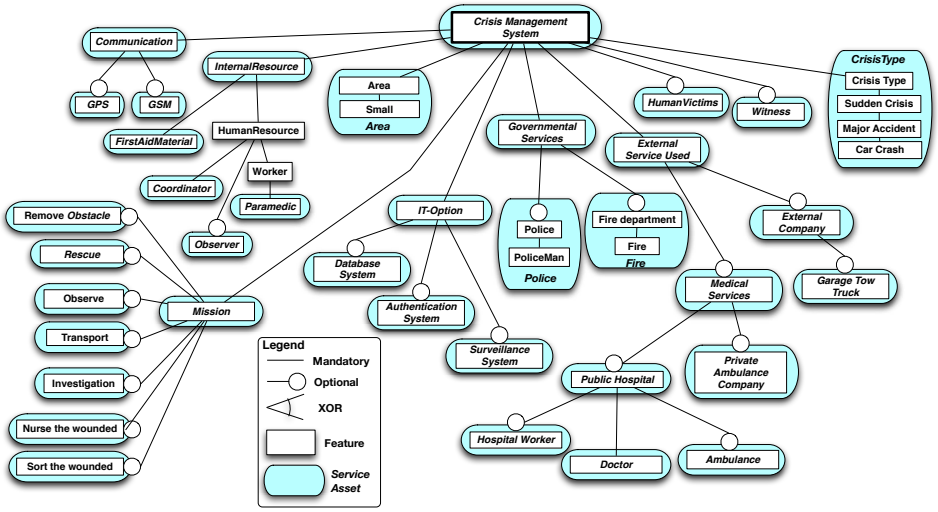

FIG. 5 – Un feature diagram pour le système de gestion de crise pour accident des voitures

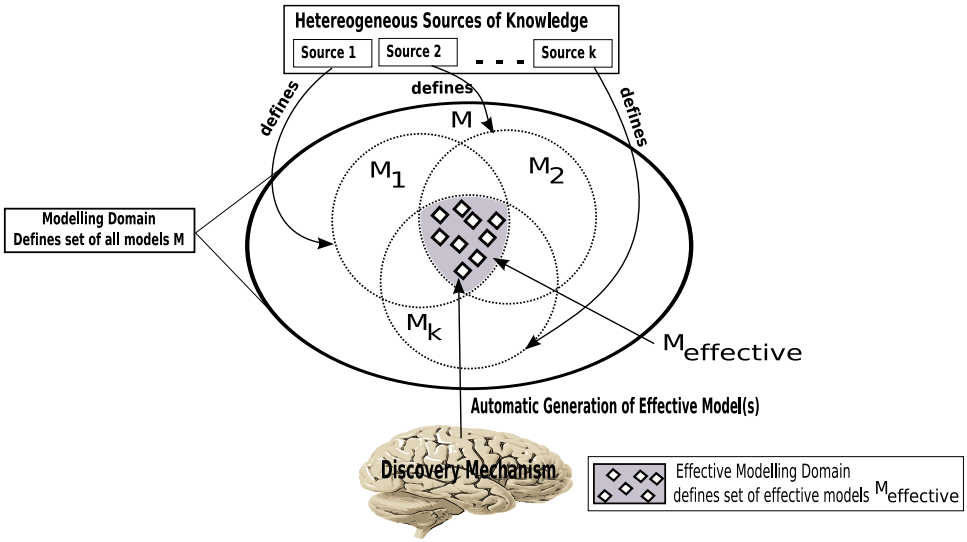

FIG. 6 – Contexte du problème pour la découverte automatique modèle

sation sont métamodèle pour la langage de modélisation et feature diagrams pour LPLs.

– **Sources hétérogènes de connaissances :** Connaissances provenant de sources hétérogènes

$Source_1, Source_2, ...Source_k$ éventuellement dans différents langages de modélisation précise des sous-ensembles du domaine de modélisation $M_1, M_2, .., M_k$. L'intersection de ces sous-ensembles

$M_1, M_2, ..., M_k$ est le domaine de modélisation efficaces représenté par un ensemble de modèles efficaces $M_{effective}$. Nous pouvons voir les sources hétérogènes de la connaissance comme un ensemble de contraintes dans les différentes langues qui limitent l'ensemble des modèles M à un sous-ensemble $M_{effective}$.

Compte tenu de ces apports, nous demandons : Quel est le mécanisme de découverte automatique qui peut créer des modèles dans l'ensemble $M_{effective}$? Telle est la question globale qui nous intrigue.

Cette question donne lieu à un certain nombre de défis ayant trait à la découverte modèle automatique. Nous décrivons les défis les plus importants ci-dessous :

Défi 1. Mécanisme de découverte : Generative vs. satisfaction de contraintes ? Notre recherche a commencé avec l'exploration des mécanismes existants pour automatiser la génération / découverte de modèles dans un domaine de modélisation. Nous classons les approches existantes que soit *générative* ou ceux basés sur les *satisfaction de contraintes*. La question était de savoir lequel est le plus prometteur ?

Une tentative d'approche générative progressivement créer des modèles dans un domaine de la modélisation par instanciation de l'objet. Par exemple, dans [10], les auteurs présentent un algorithme impératif et un outil pour générer des modèles qui sont conformes seulement aux spécification Ecore d'un méta-modèle. L'approche ne garantit pas la satisfaction de contraintes à partir de sources hétérogènes de connaissances telles que les règles de bonne formation. De même, dans Ehrig et al. [22], les auteurs proposent une approche basée sur les grammaires de graphs pour générer des modèles conformes à un diagramme de classes (comme un modèle

Ecore). Ces modèles ne sont pas conformes à toute contraintes OCL sur le méta-modèle.

Les approches fondées sur la satisfaction de contraintes tentative de transformer un domaine de la modélisation à un ensemble de variables et de contraintes. L'ensemble des contraintes est résolu en utilisant un solveur de contraintes [45]. Une ou plusieurs solutions de bas niveau sont transformés comme des modèles du domaine de modélisation. Cette approche a été utilisée dans des contextes spécifiques à un domaine comme les tests de logiciels. Le système Korat (Chandra et al.) [9] est capable de générer des structures de données implementer dans le cadre de Java Collections Framework qui satisfont des prédicats. De même, Sarfraz Khurshid dans son thèse doctoral [41] présente l'outil TestEra tool pour générer des structures de données Java telles que les listes chaînées, arbres cartes, jeux de hachage, les tableaux tas, et les arbres binaires pour les tests. Les deux approches sont limitées à des structures de données standard et non pas à la notion plus générique de modèles. L'approche la plus intrigante est l'outil UML2Alloy [46]. L'outil tente de transformer les diagramme de classe UML , qui largement ressemblent métamodèles, à le langage de spécification formelle ALLOY [32]. On peut alors utiliser ALLOY pour analyser modèles UML en générant des exemples et des contre-exemples. Bien que l'outil n'est pas directement lié à la découverte du modèle, il vise à transformer les éléments de un diagramme de classe à un problème de satisfaction de contraintes dans ALLOY. Toutefois, UML2Alloy ne transforme pas les éléments complexes d'un métamodèle telles que l'héritage multiple et les confinements multiples. UML2Alloy ne parvient pas à solliciter l'utilisation deALLOY lorsque la taille de le modèle UML est grand qui rendre l'approche non scalable.

Les approches génératives créer des modèles progressivement et ne peuvent pas satisfaire les contraintes simultanément. Par conséquent, un certain nombre de modèles as peut-être besoin d'être rejetés parce qu'ils ne peuvent pas satisfaire des contraintes. Par conséquent, les approches de satisfaction de contraintes semblent plus prometteuses.

Défi 2. Transformer la spécification d'un domaine de la modélisation à un problème de satisfaction de contraintes La spécification d'un domaine de modélisation contient un ensemble de concepts et de relations entre eux. Ces relations pourraient coder des contraintes complexes

qui ne sont pas facilement transformé en un problème de satisfaction de contraintes. En outre, un grand nombre de concepts et de relations peut conduire à un problème de satisfaction de contraintes très grand qui devient calculs inextricables.

Par exemple, la transformation d'un cahier des méta-modèle à un problème de satisfaction de contraintes requiert un modèle de contraintes pour des constructions telles que :

- Héritage multiple
- Plusieurs conteneurs pour une classe
- Propriétés opposées
- Propriétés d'identité
- Propriétés composite

La grande taille d'un métamodèle tels que le UML avec environ 246 classes ne peuvent pasêtre directement transformé en un problème de satisfaction de contraintes traitable.

Défi 3. Transformer les connaissances provenant de sources hétérogènes à des contraintes Connaissances provenant de sources hétérogènes sont spécifiées dans les différents langages de modélisation. Toutefois, pour la satisfaction de contraintes, ils doivent tousêtre transformés à des contraintes dans une langue commune. Par exemple, la tâche de génération des modèles de tests pour une transformation de modèles doivent satisfaire les contraintes spécifiées dans un langage de contraintes textuelles, telles que Object Constraint Language,objectifs de test, et la pre-condition de la transformation de modèles exprimés dans la langue de la transformation.

Défi 4. Génération de modèles doiventêtre dans des limites maniable et finis La découverte de modèles dans un domaine de modélisation nécessite la génération de modèles de taille finie. Quels sont les heuristiques pour déterminer la taille appropriée d'un modèle qui est suffisant pour satisfaire à la connaissance à partir de sources hétérogènes de la connaissance.

Défi 5. Détection des sources incohérentes de la connaissance Connaissances provenant de diverses sources peuventêtre incompatibles avec le spécification du domaine de modélisation. Comment pouvons-nous détecter de telles sources de connaissances incompatibles et les éliminer ?

Défi 6. Validation de l'efficacité des modèles Il est nécessaire de procéder *expériences rigoureuses*qui qualifient les modèles générés par satisfaction de contraintes. La qualification garantie si les modèles sont efficaces ou utiles pour des objectifs donnés. Ces expériences doivent considérer l'effet de divers facteurs qui influent sur la qualité des modèles générés. Par exemple, on peut se demander quelle est l'influence de la génération de modèles multiples en utilisant la même solveur des contraintes sur leur efficacité en tant que modèles de test ? Ne différents paramètres de la contrainte de résoudre une incidence considérable sur la qualité des solutions ?

0.5 Thesis

Dans cette thèse, nous proposons qu'il est possible de découvrir automatiquement des modèles efficaces dans un domaine de modélisation. Catégoriquement, nous abordons le problème de la découverte de modèle efficace dans deux domaines de modélisation : (a) Métamodèles (b) Feature Diagrams.

Un métamodèle est une spécification très générale du domaine un langage de modélisation. Un métamodèle peutêtre utilisé pour spécifier le domaine d'une langue spécifique au domaine de la modélisation. Cependant, les systèmes logiciels existants et des composants ne peuvent pas toujoursêtre modélisées ou transformées dans un langage de modélisation à partir de zéro. Idéalement, des composants fiables doiventêtre réutilisés dans leur forme patrimonial pour la combinaison avec d'autres composants pour construire un système logiciel. Si nous voyons ces composants patrimonial comme des features puis les combinaisons possibles de features est bien créé avec la langage feature diagram. Les éléments granuleux associée aux features peuventêtre combinées dans des configurations différentes qui font partie du domaine de modélisation d'un feature diagram. Cette distinction entre les modèles pure dans un domaine d 'un langage de modélisation et de la configuration des composants patrimonial dans une ligne de produits logiciels réaliser la construction dirigée par les modèles à différents niveaux. Par conséquent, nous considérons à la fois les spécifications des domaines de modélisation dans cette thèse.

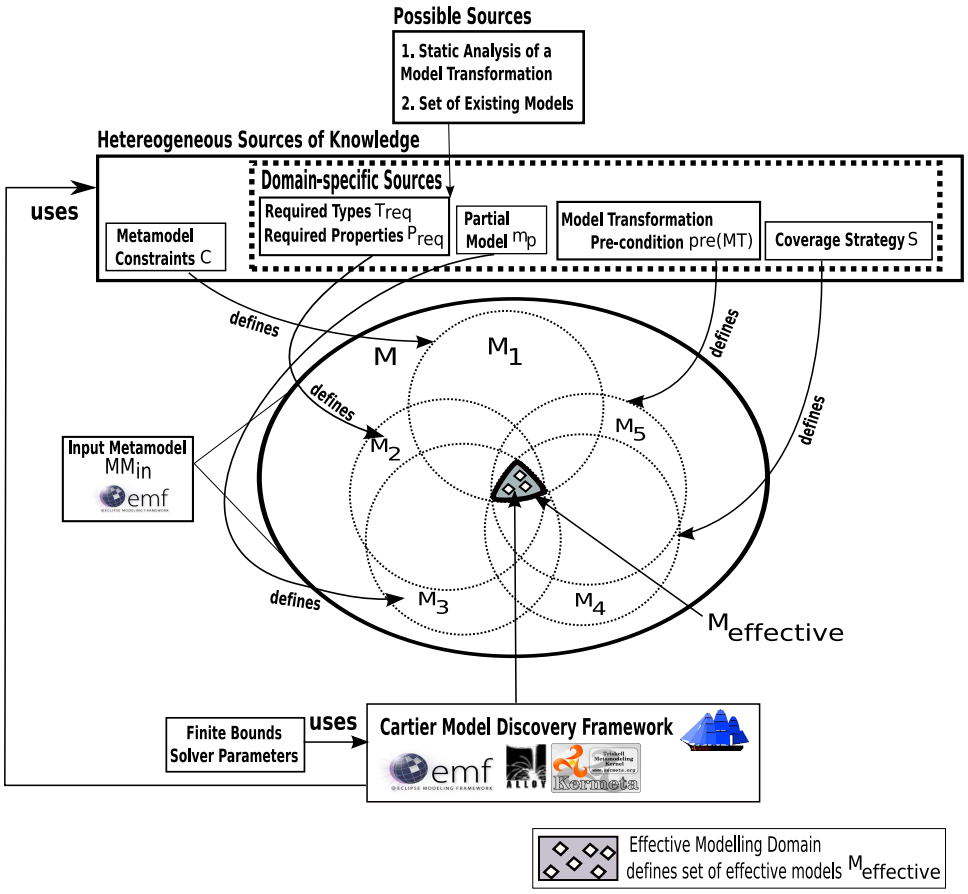

FIG. 7 – Un cadre pour la découverte automatique modèle efficace

Par conséquent, nous proposons deux cadres pour la découverte de modèle qui se spécialisent le cadre général de la figure 6 :

1. Le cadre pour la découverte automatique de modèle efficace dans le domaine de la modélisation spécifiée par un métamodèle. Ce cadre est incorporé dans l'outil CARTIER.
2. Le cadre pour la découverte automatique des produits efficaces dans le domaine de la modélisation spécifiée par un feature diagram. Ce cadre est incorporé dans l'outil AVISHKAR.

0.5.1 A Framework for Automatic Effective Model Discovery

La figure 7 présente la vue d'ensemble du cadre pour la découverte automatique de modèle efficace. Le cadre est incorporé dans l'outil CARTIER. Le nom CARTIER vient de la célèbre découvreur français de Saint-Malo qui a découvert en terres du Québec au Canada. L'entrée principale du cadre est la spécification du domaine de la modélisation donnée par un *métamodèle d'entrée*. Le *input metamodel* MM_{in} spécifie un ensemble de modèles M. Le métamodèle d'entrée se compose d'un ensemble de types (classe avec des propriétés, de enumerations, primitive) pour instancier des modèles d'un langage de modélisation. Concrètement, le métamodèle d'entrée est stockée comme une instance de la métamodèle ECORE qui fait partie de la norme de l'industrie Eclipse Modeling Framework (EMF) [27]. Les modèles eux-mêmes sont stockées sous forme XMI [2] fichiers représentant des instances de la métamodèle Ecore.

Sources hétérogènes de connaissances limiter le domaine de la modélisation spécifiée par un métamodèle :

- **Types** T_{req} **et propriétés** P_{req} **requises** dans le métamodèle d'entrée. L'ensemble des types et propriétés requises aide à extraire un sous-ensemble du métamodèle d'entrée appelé *métamodèle efficace*. Le métamodèle efficaces précise le sous-ensemble de modèles $M_1 \subset M$.Il peut y avoir plusieurs sources possibles pour l'ensemble des types et propriétés requises :
  - L'analyse statique d'une transformation modèle donne un ensemble de types et de propriétés dans le métamodèle d'entrée effectivement manipulé par la transformation.
  - Un ensemble de modèles conformes au métamodèle d'entrée est une autre source de types et propriétés requises. Visiter les modèles dans l'ensemble nous donne un ensemble de types et propriétés utilisées dans la métamodèle. Un exemple typique du monde réel de ce qui pourraitêtre dans une salle de classe pour la conception orientée objet en utilisant UML. Le professeur peut faire remarquer aux élèves les types et propriétés requises, utilisé à créer UML, en visitant tous les objets d'un ensemble de modèles automatiquement.
- **Contraintes sur métamodèle** C sont exprimés sur un métamodèle d'entrée en utilisant

un langage de contraintes textuelles, telles queObject Constraint Language (OCL) [56]. Ces contraintes encoder des restrictions qui ne peutêtre spécifié en utilisant une modèle Ecore. Nous illustrons ce dans l'ensemble $M_2 \subset M$.

– **Sources spécifiques à un domaine de connaissances** peut également aider à définir le domaine de la modélisation efficace. Nous présentons quelques-unes ci-dessous :
  – **Modèle partiel** m_p est un modèle partiellement spécifié en utilisant les métamodèle d'entrée. Par exemple, un éditeur de modèle graphique permet à un utilisateur de créer des modèles dans un langage de modélisation telles que les machines d'état UML. Un modèle incomplet dans l'éditeur est un modèle partiel dans langage machine d'etat de UML. Le modèle partielle peut ne pas respecter toutes les contraintes du métamodèle UML. Par conséquent, un modèle partiel est souvent exprimée comme une instance d'un *version relaxée du métamodèle d'entrée*. Le modèle partiel définit le sous-ensemble $M_3 \subset M$.
  – **Stratégie de couverture** S aider à définir et générer des *fragments de modèles* [25] qui couvrent un large éventail d'aspects structurels dans le métamodèle d'entrée. Par exemple, la stratégie d'une partition de domaine d'entrée permet de générer un ensemble de fragments modèle MF qui couvrent les partitions sur tous les types et les propriétés du métamodèle d'entrée. Ces fragments de modèles aider à définir un domaine de modélisation efficaces pour *coverage-based testing* d'une transformation de modèles. Tous les modèles de teste qui répondent à une stratégie de couverture contiennent le modèle de fragments générés par la stratégie. Fragments de modèles sont exprimées dans un langage de modélisation qui permet de préciser des ranges sur les propriétés d'un métamodèle d'entrée. Une stratégie de couverture définit le sous-ensemble $M_4 \subset M$.
  – **Pré-condition d'une transformation** $pre(MT)$ est un ensemble d'invariants sur le métamodèle qui est spécifique à une transformation de modèles MT. Une transformation de modèles souvent ne peuvent pasêtre conçu pour transformer tous les modèles spé-

cifiés par son métamodèle d'entrée. Par exemple, la transformation des modèles de diagramme de classe à des modèles entité relation diagramme [5] exiger que toutes les classes dans le modèle d'entrée ont au moins un attribut principal. Le OCL [56] est souvent utilisé pour exprimer des pré-conditions. Un pré-condion définit le sous-ensemble $M_5 \subset M$.

L'intersection de toutes les sources de connaissances définit le *domaine de la modélisation efficace*. Le domaine de la modélisation efficace est l'ensemble des modèles définis par $M_{effective} \leftarrow M \cap M_1 \cap M_2 \cap M_3 \cap M_4 \cap M_5$.

La méthodologie pour la découverte de modèle utilise les sources de connaissances présentées ci-dessus pour générer automatiquement des modèles efficaces dans le domaine de la modélisation. Nous mobiliser les étapes ci-dessous :

Étape 1. Identification métamodèle efficace : Nous élaguons les métamodèle d'entrée MM_{in} pour obtenir le métamodèle efficace $MM_{effective}$ utilisant un algorithme de l'élagage métamodèle[73]. Le métamodèle efficaces contient l'ensemble des types T_{req} et propriétés requises P_{req} fournies en entrée et toutes ses dépendances obligatoires calculé par l'algorithme de l'élagage métamodèle. Tous les types de biens inutiles et sont éliminés. $MM_{effective}$ est un super type de MM_{in} d'un point de vue théorique de typage et un sous-ensemble de MM_{in} d'un point de vue théorie des ensembles. La taille de la métamodèle efficace $MM_{effective}$ est souvent beaucoup plus petits que la taille de la métamodèle d'entrée MM_{in}.

Étape 2. Transformation de la spécification de domaine efficace à ALLOY : La spécification de domaine efficace la modélisation est définie par un certain nombre d'artefacts. Il est d'abord définie par le métamodèle efficace $MM_{effective}$ et contrainte par la connaissance d'une ou plusieurs sources : (b) Contraintes sur métamodèle C (b) Modèle partiel m_p (c) Modèle fragments MF de la stratégie de couverture S, and (d) Pré-condition $pre(MT)$ d'une transformation de modèles MT. Nous transformons ces artefacts ont exprimé dans des langues différentes, éventuellement à un **constraint satisfaction problem** (CSP) dans la langue pour la spécification formelle ALLOY [31] [32]. Le formalisme théorique pour exprimer le CSP est **premier ordre**

logique relationnelle.

Étape 3. Génération de modèles dans de domaine de modélisation efficace : Nous résolvons le CSP enALLOY pour générer des modèles efficaces dans le domaine de la modélisation. CARTIER atteint cet objectif en invoquant KodKod [23] en ALLOY de transformer le CSP à Boolean Conjunctive Normal Form (CNF) . Nous invoquer une satisfiabilité (SAT) solveur comme MiniSAT [54], ZChaff [78] pour résoudre le Boolean CNF. Enfin, nous transformons des solutions à faible niveau de la CNF à des modèles conformes au méta-modèle d'entrée MM_{in}.

La génération de modèles dans un domaine de modélisation est souvent orienté vers un objectif. Nous devons nous assurer que l'objectif est atteint de manière cohérente tenant compte de tous les facteurs déterminants. Une question typique peut-être quel est l'effet d'un solveur SAT sur la qualité de la solution ? Pour répondre à cette question nous avons besoin de réaliser des expériences qui génèrent plusieurs solutions pour le problème de satisfaction de contraintes même. Il existe de nombreux autres facteurs pour lesquels nous effectuons des expériences rigoureuses pour valider l'efficacité de découverte. Dans cette thèse, nous réalisons des expériences dans les domaines d'application suivants :

1. Génération de modèles de teste pour les transformation de modèles
2. Achèvement du modèle partiel dans les éditeurs de modèle de domaine spécifique

0.5.2 Un cadre pour la découverte automatique de produits efficaces

La figure 8 présente la vision globale du cadre de la découverte de produits efficaces. Le cadre est incorporé dans l'outil AVISHKAR. AVISHKAR en hindi signifie *invention* qui signifie le caractère de l'outil pour découvrir les produits dans un LPL. L'entrée principale du cadre est la spécification du domaine de la modélisation donnée par un *feature diagram* or *feature model*. Le *feature diagram* FD spécifie un ensemble de produits P. *Feature Diagrams* (FD) introduit par Kang et al. [38] représentent tous les produits (ou configurations) d'un LPL en termes de features qui peutêtre composé. Feature diagrams ont été formalisés pour effectuer des analyses LPL[69]. Dans [69], Schobbens et al. proposer une définition générique formelle de FD qui subsume les

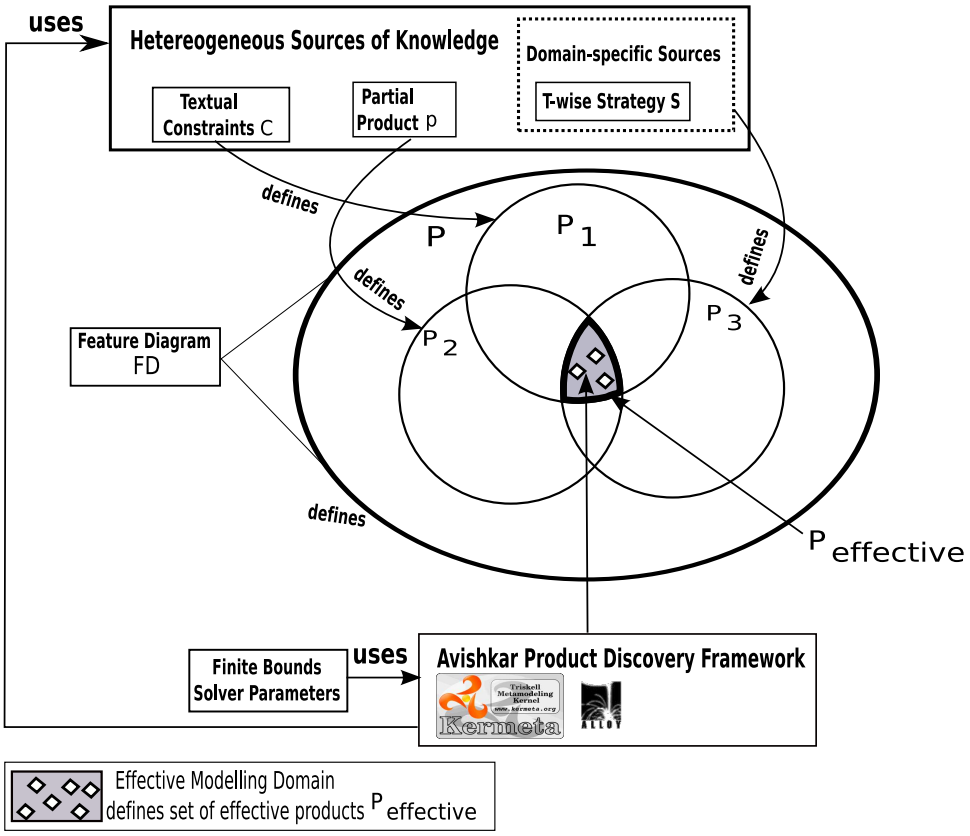

FIG. 8 – Un cadre pour la découverte automatique de produits efficaces

nombreux dialectes FD existants. Nous définissons une FD comme suit :

- Un FD se compose de *k* features $f_1, f_2, ..., f_k$
- Un feature f_i peutêtre associé à un actif logiciel.
- Features sont organisés dans une relation parent-enfant dans un arbre T. Un feature avec aucun autre enfant est appelé une feuille.
- Une relation parent-enfant entre les features f_p and f_c sont classés comme suit :
  - *Mandatory* - enfant feature f_c est requis si f_p est sélectionné.
  - *Optional* - enfant feature f_c peutêtre sélectionnée si f_p est sélectionné.
  - *OR* - au moins une des enfant-features $f_{c1}, f_{c2}, .., f_{c3}$ de f_p doitêtre sélectionné.
  - *Alternative (XOR)* - l'une des child-features $f_{c1}, f_{c2}, .., f_{ck}$ de f_p doitêtre sélectionné.
- Relations à travers d'arbre entre deux features f_i et f_j dans l'arbre T sont classés comme suit :
  - f_i requires f_j - La sélection de f_i dans un produit implique la sélection de f_j.
  - f_i excludes f_j - f_i et f_j ne peut pas faire partie du même produit et sont *mutuellement exclusives.*

Utilisation de la FD nous créons des produits / configurations de features. Nous pouvons composer des actifs logiciels. associés à ces features pour obtenir le produit final.

Sources hétérogènes de connaissances restreindre le domaine de la modélisation spécifiée par un

- **Contraintes textuels** C sont exprimés sur un ensemble de features. Les contraintes sont exprimées textuellement quand ils ne peuvent pasêtre directement encodée dans le FD. Ces contraintes préciser le sous-ensemble $P_1 \subset P$
- **Produit partiel***p* iest un ensemble de features choisies dans le produit. L'ensemble des features peuvent nécessiter la sélection d'autres features pour obtenir un produit complet. Le produit partielle précise le sous-ensemble $P_2 \subset P$
- **Stratégie T-wise** S est une stratégie de génération de produit pour détecter des défauts dans les lignes de produits logiciels [44] [60]. Le grand nombre de produits visés par

un diagramme feature diagram peutêtre échantillonnées en utilisant une stratégie tels que $T-wise$. L'objectif est de générer un nombre minimal de produits qui répondent à tous les interactions $T-wise$ entre les features. Par exemple, dans un FD avec 25 optionnels (voir la figure 5) spécifie au moins 2^{25} produits. Une stratégie $2-wise$ où $T=2$ permettra à la génération de seulement $4\times{}_{25}C_2=300$ de produits qui couvrent toutes les interactions par paires entre les features. La stratégie $T-wise$ pour une valeur particulière de T spécifie le sous-ensemble $P_3\subset P$.

L'intersection de toutes les sources de connaissances définit le *domaine de la modélisation efficace*. Le domaine de la modélisation efficace est l'ensemble des produits définis par $P_{effective}\leftarrow P\cap P_1\cap P_2\cap P_3$.

La méthodologie de découverte produit utilise les sources de connaissances présentées ci-dessus pour générer automatiquement des produits dans le domaine de la modélisation efficace FD. Nous mobiliser les étapes ci-dessous :

Étape 1. Transformation du feature diagram à ALLOY : Nous transformons un feature diagram to problème de satisfaction de contraintes dans la langage formelle ALLOY [32] [31].

Étape facultative. Transformation des produits partiels à ALLOY et leur achèvement : Nous pouvons transformer un produit partiel p à ALLOY. Il génère un prédicat ALLOY qui représente des informations partielles sur les features sélectionnées dans le produit partiel. On peut alors résoudre le modèle ALLOY pour générer un ou plusieurs produits complète.

Étape 2. Génération de tuples $T-wise$ et la détection de tuples valide à l'aide d'ALLOY : Dans cette thèse nous nous concentrons sur la création de produits qui répondent à $T-wise$ interaction entre les features. Nous avons d'abord générer les prédicats ALLOY représente tuples $T-wise$ et détecte ceux qui ne sont pas compatibles avec les contraintes dans la FD.

Étape 3. Generation scalable des produits Nous proposons *divide-and-compose* stratégies pour générer un ensemble de produits qui couvrent toutes les lignes entre les features qui couvrent interaction $T-wise$. L'approche divise le problème de satisfaction pour tous les tuples à la résolution de sous-ensembles de tuples. Nous résolvons multiples modèles ALLOY avec ces

sous-ensembles d'obtenir des ensembles de produits. Les assortiments de produits sont fusionnés en un ensemble final des produits.

Est découvert les produits en utilisant le cadre capable à constamment atteindre leurs objectifs ? Par exemple on peut se demander quel est l'effet du stratégies divide-and-compose sur la redondance des produits générés ? Pour répondre à cette question nous avons besoin de générer des produits compte tenu de tous autres facteurs déterminants. Dans cette thèse, nous validons notre cadre de l'aide d'expériences rigoureuses dans les domaines d'application suivants :

1. Génération de produits de teste qui satisfont aux critères de interaction t-wise.
2. En cours / travaux futurs, nous montrons que notre cadre peut effectivement échantilloner l'espace Qualité de Service (QoS) d'un service web dynamique. La variabilité de la service web dynamique est modélisée avec un FD.

0.6 Contributions

Les cadres pour le découverte de modèle et les produits ont conduit à des contributions scientifiques dans cette thèse. Nous expliquons ces contributions dans les sous-sections suivantes. Certaines des contributions sont extraites et pin-fait de la méthode déjà décrite dans la section 0.5. Nous citons les publications pertinentes des conférences par des pairs et des revues.

0.6.1 Contributions à la découverte automatique modèle efficace

Contribution 1.1 Nous présentons un cadre global pour la génération de modèles efficaces de taille finie dans tout langage de modélisation et restreinte par des sources hétérogènes de la connaissance. Le cadre est incorporé dans l'outil CARTIER. Nous utilisons le langage de spécification formelle ALLOY pour sa capacité représentent des contraintes sur les graphes d'objets et donc de représenter la métamodèle comme un problème de satisfaction de contraintes. Cette contribution résume la réponse à tous les défis présentés dans la section 0.4 pour un domaine

de modélisation spécifiée par un métamodèle. L'outil CARTIER, vu ses origines dans nos documents [66], [70].

Contribution 1.2. Le cadre transforme toutes les éléments d'un méta-modèle à ALLOY pour la satisfaction de contraintes. Il traite également de méta-modèle avec l'héritage multiple en l'aplatissant à l'héritage unique en ALLOY. En outre, le cadre présente la transformation à ALLOY factsde contraintes imposées par multiple containers, opposite properties, identify properties, et composite properties. Cette contribution adresses défi 2 de la section 0.4. La transformation à ALLOY a été brièvement décrit dans deux de nos contributions [70] and [72].

Contribution 1.3. Le cadre est construit en utilisant Kermeta, la langage pour simultanément traiter des modèles venant de langage différentes. Chaque source de connaissance est exprimée comme un modèle dans un langage de modélisation. Par exemple, des fragments de modèle sont exprimées en tant que modèles d'une langage fragment modèle. Kermeta pouvez charger, enregistrer et manipuler des modèles conformes aux métamodèles différents en même temps. Par conséquent, CARTIER,écrit en Kermeta, transforme la connaissance des différents modèles à des faits dans la langage cible ALLOY. Cette contribution adresses défi 3 de la section 0.4 et est publié dans nos papiers [70] [50].

Contribution 1.4. Dans le cadre que nous présentons un algorithme pour tailler un métamodèle[73] qui utilise un ensemble de types et propriétés requises pour générer un métamodèle à compter de méta-modèle d'entrée de grandes. Le métamodèle efficace est souvent très petite et peut facilementêtre transformé en ALLOY comme un problème de satisfaction de contraintes traitable. Cette contribution porte sur une partie du défi 2 de la section 0.4 et présenté dans le papier [73].

Contribution 1.5. Le cadre comprend des installations d'assigner des limites finies pour le nombre d'objets pour chaque type dans le modèle. Il transforme aussi les solutions du solveur SAT en ALLOY appelé ALLOY à modèles du haut niveau conforme à un métamodèle. La génération de modèles conformes à des sources hétérogènes de la connaissance permet de déterminer les incohérences entre eux le cas échéant. Une sélection de sources incohérentes de la connaissance est faite et soit modifiée ou supprimée à partir de la spécification du domaine de

modélisation efficaces. Cette contribution porte sur les défis 4 et 5 de la section 0.4 et est publiée dans [70] et [72].

Contribution 1.6. Nous validons les modèles générés pour leur efficacité en utilisant le cadre en effectuant les expériences suivantes :

- **Génération des modèle de test pour des transformations de modèles :** Nous générons des milliers de modèles pour une transformation représentant. Nous utilisons l'analyse mutation [51] pour démontrer que les modèles d'testes obtenues en utilisant *stratégie de partitionnement* peut détecter 93% des bugs par rapport à la génération arbitraire de 70%. Nous montrons que la stratégie de partitionnement n'est pas affectée par divers biais tels que la dépendance sur les ALLOY solveur. L'étude expérimentale est publié dans [71] et la version journal en revue [64].
- **La complétion du modèle partiel dans les éditeurs de modèle de domaine spécifique :** Nous utilisons notre cadre de produire des recommandations pour compléter les modèles partiels dans l'éditeur de modèle AToM^3 [29]. Nous montrons que notre cadre peut compléter automatiquement modèles partiels dans un éditeur de modèle. Les expériences montrent que cela peutêtre fait pour les petits exemples dans des délais raisonnables. Ces travaux sont publiés dans [67], [72].

Cette contribution adresses défi 6 de la section 0.4.

0.6.2 Contributions à la découverte automatique produits efficace

Contribution 2.1. Nous présentons un cadre global pour la production de produits efficaces dans une ligne de produits logiciels spécifié par un feature diagram. Le cadre est incorporé dans l'outil AVISHKAR. Le cadre contient la transformation d'un feature diagram à un problème de satisfaction de contraintes en ALLOY. Le cadre invoque un solveur sur la modèle ALLOY pour générer automatiquement des produits conformes au feature diagram. Cette contribution résume la réponse à tous les défis dans la section Section 0.4 pour un domaine de modélisation spécifiée par un feature diagram.

Contribution 2.2. Étant donné un ensemble des features sélectionner (disponible / non disponible) le cadre utilise ALLOY à détecter si un produit peutêtre créé de telle sorte que ces sélections des features sont remplies. Une contrainte par exemple dit que les features f_1 existe dans le produit, tandis que f_2 ne devrait pas exister. Si f_2 est un élément obligatoire alors AVISHKAR utilise ALLOY à détecter que la contrainte n'est pas valide. Cette contribution adresses défi 5 de la section 0.4.

Contribution 2.3. Scalable génération de produits de test à partir d'un feature diagram Feature diagrams ont été transformées à des problèmes de satisfaction de contraintes pour tester une LPL. Par exemple, Cohen et. al.ont appliqué les tests d'interaction combinatoire à sélectionner systématiquement les configurations [19] partir d'un feature diagram. Ils considèrent les différents algorithmes afin de calculer les configurations qui répondent à des critères pair-wise et t-wise [18]. Les contraintes imposées en raison de relations entres les features dans un feature model sont résolus par invoquant les SAT solveurs tels que ZChaff [78].Toutefois, leur approche n'est pas très extensible si l'on considère les feature diagrams grande taille. Notre cadre contient des stratégies *divide-and-compose* visant à scinder le problème de la génération de produits de test satisfaisant $T-wise$ en sous-problèmes. L'outil AVISHKAR résout les sous-problèmes et fusionne les résultats dans un petit ensemble de produits qui contiennent tous les tuples valides requis par le critères $T-wise$. Ce mécanisme rend notre méthodologie à adopter une approche évolutive pour générer des produits dans une gamme de produits logiciels. Cette contribution adresses défi 4 de la section 0.4.

Contribution 2.4. Validation de l'efficacité des produits de test : Il est nécessaire de réaliser des expériences qui sont admissibles les produits générés à l'aide de notre cadre. Nous effectuons des expériences pour générer des produits d'un feature diagram AspectOPTIMA. Nous montrons que *redondance* est introduit dans les produits en raison de stratégies de divide-and-compose. Dans les travaux en cours que nous effectuons des expériences pour générer des configurations différentes d'une orchestration dynamique de services Web. Nous démontrons que la qualité de service d'un web-service varie en fonction de différentes configurations de la web-

service. Ces expériences d'analyse nous aident à définir une méthodologie efficace d'ensemble robuste d'accords contractuels pour le service Web dynamiques.

Les contributions ci-dessus sont publiés dans [60]. Deux papiers [3] [39] ont été soumis à appliquer l'outil de découverte des produits AVISHKAR à l'analyse des variables de la QoS dans une orchestration de services web.

0.7 Organisation de la thèse

La thèse comprend 6 chapitres, y compris l'introduction. Les 5 chapitres suivants sont organisés comme suit :

- Chapitre 2,nous introduisons le contexte de MDE et l'état de l'art dans la découverte automatique de modèle efficace dans un domaine de modélisation.
- Chapitre 3, nous présentons la découverte automatique de modèle efficace dans le domaine spécifié par un métamodèle.
- Chapitre 4, présente la validation empirique du cadre pour la découverte de modèle. En particulier, nous nous concentrons sur deux domaines d'application pour la validation : (a) la génération de test du modèle de transformation de modèles (b) la réalisation partielle du modèle dans l'éditeur de modèle AToM^3
- Chapitre 5, nous décrivons le cadre de la découverte de test automatique du produit dans une ligne de produits logiciels. Nous valider empiriquement le cadre de la redondance dans les produits générés.
- Chapitre 6, nous résumons notre travail et les perspectives actuelles pour la recherche future. Nous décrivons brièvement les travaux en cours sur l'analyse de la qualité de service variable dans un service de web dynamique.

Chapter 1

Introduction

Model-driven engineering (MDE) is an approach to specify, construct, validate and maintain complex software systems using first class artifacts called *models*. MDE has emerged from a number of areas in software development such as object-oriented analysis and design languages, object-oriented methodologies [7] [33] [63], and Computer-Aided Software Engineering (CASE) endeavours in the 80s and 90s to automate several steps in software engineering [55] [8].

Models are *graphs of inter-connected objects* in a *modelling domain*. A modelling domain defines a *set of models* where each model is constructed using a common set of concepts and relationships. For instance, in this thesis we consider the specification of two modelling domains: (a) *metamodels* that specify a set of models in a modelling language (b) *feature diagrams* or *feature models* that specify a set of product models or simply products in a Software Product Line (SPL). Very often the creation of useful or *effective models* in a modelling domain require the satisfaction of constraints from heterogeneous sources. For instance, creating a workflow model for a business process using the well-known Unified Modelling Language (UML) activity diagram requires the model to satisfy UML well-formedness rules, business logic, economic constraints, quality of service constraints, and security restrictions. Human modellers with experience incrementally create such effective models by tacitly ensuring that the models are *correct*

by construction and satisfy constraints from heterogenous sources. Still and all, this process is extremely tedious and sometimes impossible if there is a need to create thousands of models. **Can we automate the creation of effective models given the heterogenous sources of knowledge?** This is the question that intrigues us and the subject of this thesis.

The introduction is organized as follows. The notion of effective model discovery situates itself in the *global context* of discovering effective structures in science of engineering. We briefly describe this global context in Section 1.1. This thesis addresses the problem of automating discovery in the contemporary and specific context of MDE which we describe in Section 1.2. A number of scenarios in MDE necessitate generation of effective models. Our motivation stems from these scenarios that we describe in Section 1.3. In Section 1.4, we present the general *problem context* and its challenges. We present our thesis and describe our methodology for automatic effective model and product discovery in Section 1.5. We enlist the contributions of our thesis in Section 1.6. Finally, we present the organization of the thesis in Section 1.7.

1.1 Discovery of Effective Structures in Science and Engineering

Scientific discovery often culminates into representing structure in nature as *networks of entities* or *graphs of objects*. For instance,

- **Food webs** are representations of the predator-prey relationships between species within an ecosystem or habitat. A common example is the *soil food web* shown in Figure 1.1. The soil food web is often found in a garden bio-compost.

- **Biochemical reaction networks** or metabolic pathways represent vital molecular exchanges in living beings. The widely studied *tumor suppressor pathway* shown in Figure 1.1 (b) illustrates the crucial role of protein p53 in cell death. Cell death is important in order to regulate cancerous growth.

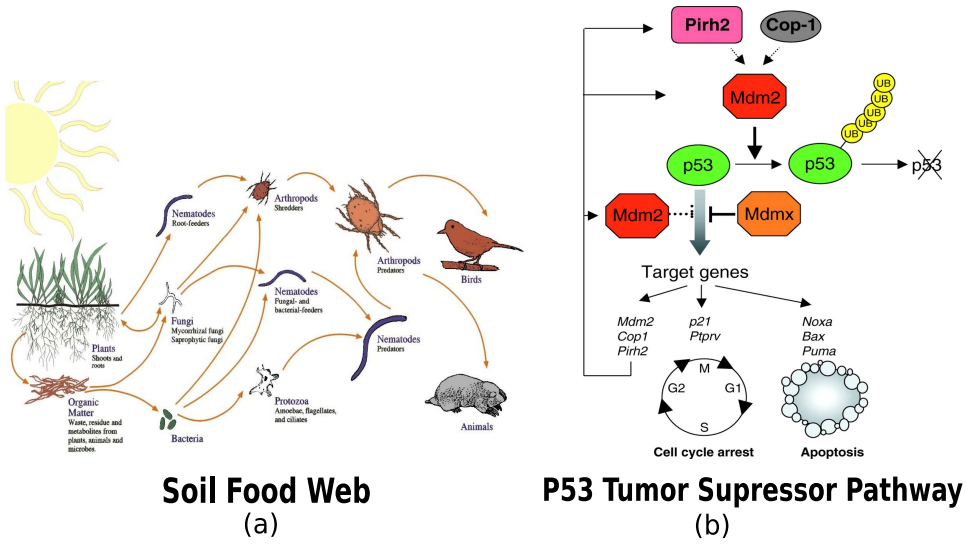

Figure 1.1: Effective Structures in Scienctific Discovery: (a) Soil Food Web (b) Tumor Suppression Pathway

Knowledge from experiments, data analysis and mental tacit lead to the discovery of such *effective structures* in nature. The existence of effective structures is not limited to the virtuosity of nature. We humans are endowed with the ability to represent and create effective structures such as buildings, bridges, robots, and complex software.

Design in engineering often results into representing effective man-made structures as graphs of objects. For instance,

- **Electronic circuits diagrams** represent a network of electrical components that achieve a given purpose. The *FM Receiver Circuit* shown in Figure 1.2, for instance, is used in millions of radio devices.

- **Software Design Patterns** represent general reusable solutions to commonly occurring problems in software design. They are often represented as *class diagrams* in object-oriented software engineering.The *observer pattern* in Figure 1.2 (b) is a common pattern in software requiring distributed event handling. The well-known photo editing program Adobe Photoshop is one such software product.

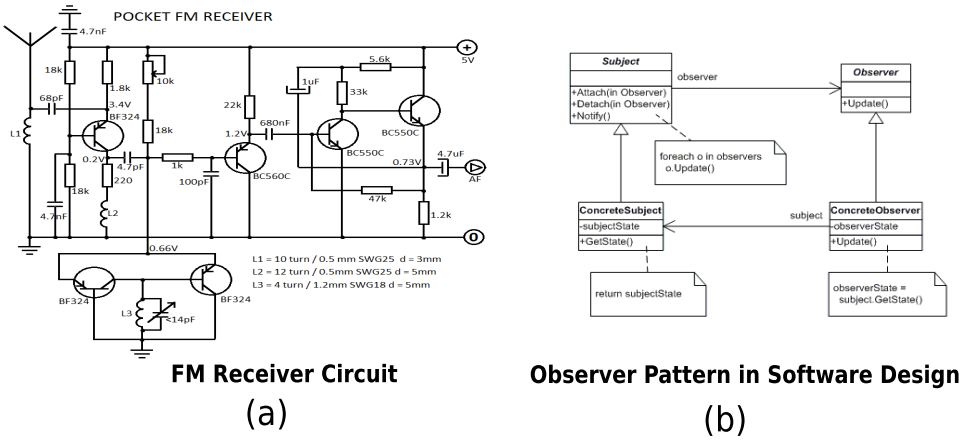

Figure 1.2: Effective Structures in Engineering: (a) FM Radio Circuit (b) Observer Design Pattern

Very much like discovery in science, design in engineering is guided by knowledge from a number of sources coupled with the creativity of an engineer. Can this process of scientific discovery or design in engineering using various sources of knowledge be *automated*? This question has been a subject of study for several decades with the advent of the modern computer.

Computer programs have been used to discover structure in nature. For instance, inspired by Karl Popper's logic of scientific discovery [61], Pat Langley, Herbert Simon, G. Bradshaw, and Zytkow developed several computer programs such as Bacon, Glauber, Dalton, and Stahl described in their book [47]. These programs were guided by heuristics to successfully re-discover historical laws in chemistry.

Evolutionary computing approaches have been develop to automate design in engineering such as generation of electronic circuits [43]. Computer programs implementing an evolutionary approach contest for the "Humies Award" conferred each year at the GECCO conference. The award of $10,000 is given to the approach with most human-competitive results. In the software engineering community, recent conferences such as the Automated Software Engineering (ASE) conference provide competitive venues for presenting approaches to generating software structures.

In this thesis, we address the question of automatic discovery in the contemporary context of Model-driven Engineering of complex software systems.

1.2 Context: Model Driven Engineering

MDE [53] aims to grease the wheels of complex software creation using first class artifacts called *models*. The MDE philosophy is using models to represent important artifacts in a system such as requirements, high-level designs, data structures, views, interfaces, model transformations, test cases, and implementation-level artifacts such as source code. A model is constructed in a *modelling domain* that captures a set of common concepts and relationships. The construction of a model in a modelling domain may be further constrained using well-formedness rules and constraints from various sources.

The general notion of a modelling domain can be *specialized* in many ways. A precise specification of concepts and relationships that defines a set of models is a modelling domain. For instance, a *metamodel* specifies the modelling domain of a modelling language. The well-known Unified Modelling Language (UML) modelling language [57] has its own metamodel that specifies the set of all UML models. Another, example of a modelling domain is a *feature diagram* or *feature model* that specifies a set of products in a Software Product Line(SPL). Models in a modelling domain can be loaded/stored, manipulated, and transformed to other models/implementation artifacts to solve software problems.

MDE provides a number of software processes and technologies to allow creation of modelling domains and the transformation of its models. Historically, the Model-driven Architecture (MDA) trademark marketed by the Object Management Group (OMG), presents a model-driven approach to system development. The MDA approach begins development of a modelling domain for platform-independent models (PIMs), which are incrementally transformed or refined into lower-level platform specific models (PSMs) in another modelling domain. The PSMs are reified into implementation artifacts such as implementation code. This automatic construc-

tion of systems from high-level models allows software engineering expertise to be captured as reusable *model transformations* applied more reliably and efficiently. Currently, the widely accepted framework for specifying modelling domains is the Eclipse Modeling Framework (EMF) [27]. For instance, metamodels are created in the EMF *Ecore* format to specify the domain of a modelling language. Model transformation [74] languages such as the imperative Kermeta [40] [52], rule-based ATL [36] [35] [1], graph grammar based AToM^3 [29], Viatra [77] transform models. Model transformation languages are expected to conform to the Query-View-Transformation (QVT) standard [35]. Different types of model transformations can be created using these languages as classified in [20]. Model transformations may transform models within the same modelling domain (endogenous transformations), between different modelling domains (exogenous transformations) and even realize the classical view of generating executable code from a high-level model.

Our focus in this thesis is the automatic discovery or computer-assited discovery of models in a modelling domain.

1.3 Motivation: Why the Need for Automatic Model Discovery?

Our motivation for automatic discovery in the general context of model-driven engineering stems from existing computational discovery endeavors in heterogenous domains. These domains range from systems biology [12] [62], to engineered physical systems [49] [68] [26], [43]. We see automatic discovery of effective models in a modelling domain as general framework subsuming existing approaches to effective structural discovery in heterogeneous areas of science and engineering. MDE of software systems is no exception. In this thesis, we investigate three scenarios in MDE as described below:

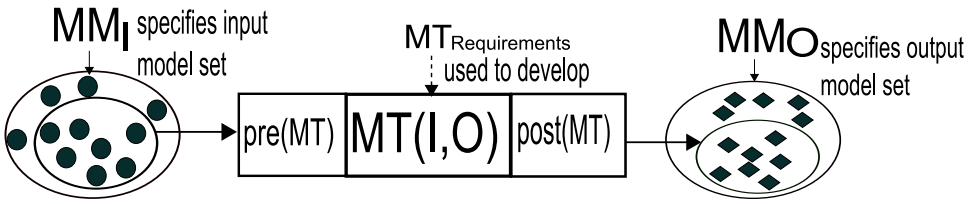

Figure 1.3: A Model Transformation

1.3.1 Scenario 1: Test Generation for Model Transformations

Model transformations are core software artifacts in MDE. A simple model transformation MT takes input models conforming to an input metamodel MM_I and produces output models conforming the output metamodel MM_O as shown in Figure 1.3. Not all models specified by the input metamodel can be processed by the model transformation. Therefore, we compose pre-conditions $pre(MT)$ that restrict some models from being processed by the model transformation. The output models must also satisfy a set of constraints called the post-condition $post(MT)$. The model transformations itself is built using knowledge from a set of requirements $MT_{Requirements}$.

Testing a model transformation requires input model that can detect bugs in the transformation MT. Manually creating such test models is tedious since it must be a graph of objects that must conform to MM_I, $pre(MT)$, and use information from $MT_{Requirements}$. Manual creation becomes impossible when we need to create thousands of such test models that encode different test objectives. Therefore, there is a clear need to automate the generation of test models that satisfy knowledge from various sources such as MM_I, $pre(MT)$, and use information from $MT_{Requirements}$. The automatic generation of input models exalts to the level of automatic discovery of test models if we validate that they can indeed detect bugs in a transformation. We can qualify the effectiveness of test models via techniques such as *mutation analysis* for model transformations [51]. Based on a description of this scenario, we ask *how do we generate test models and qualify their bug detecting effectiveness*?

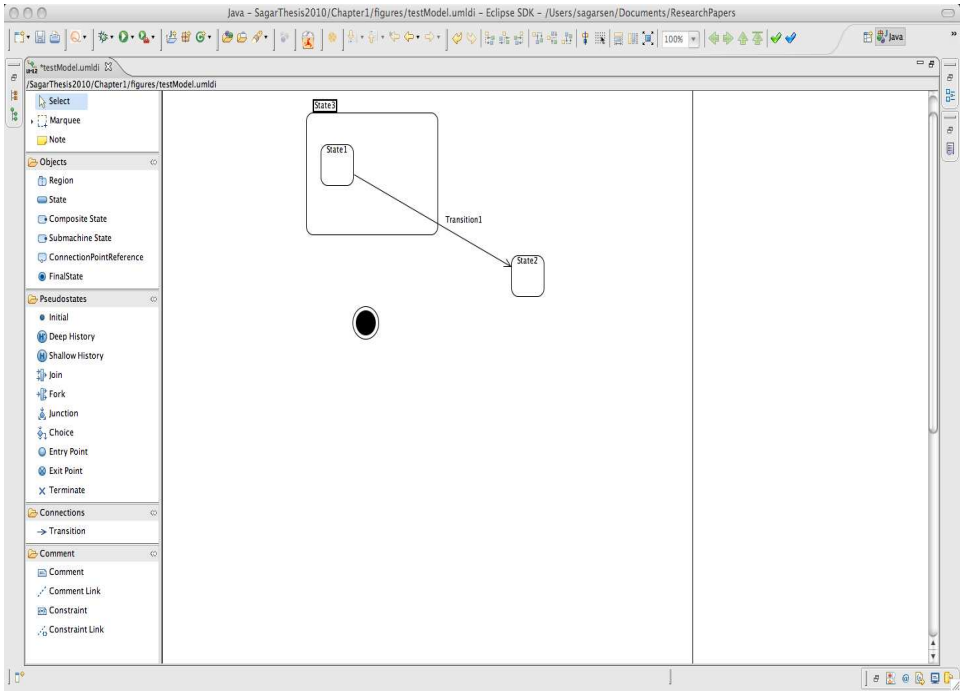

Figure 1.4: Partial Model in the TopCaseD UML Model Editor

1.3.2 Scenario 2: Partial Model Completion in a Model Editor

Modellers often use model editors to incrementally build models. For instance, the TopCaseD editor [24] can be used build UML models as shown in Figure 1.4. The model shown is an incomplete UML state machine. For instance, the model does not have an initial state which is a well-formedness rule. There are infinite possible ways to complete the model such that it becomes a valid UML state machine model and satisfies all the well-formedness rules of a state machine. What is probably more interesting is the nearest consistent UML state machine that contains all elements of the partial model. There may be a number of possibilities to complete such partially specified models. We can relate automatic mode completion to the automatic code completion problem in programming environments [4]. This scenario raises the question: *How do we automatically discovery complete models or recommendations to complete partial models?*

1.3.3 Scenario 3: Generation of Products in a Software Product Line

A Software Product Line (SPL) references to a set of products sharing a common, managed set of features that satisfy the specific needs of a particular mission [15]. A *Feature Diagram* (FD) or a *feature model* specifies of a modelling domain for a SPL. Feature diagrams introduced by Kang et al. [37] [38] compactly represent all the products of an SPL in terms of features which can be composed. A FD consists of k features $f_1, f_2, ..., f_k$ and dependency constraints between features. For instance, selection of some features in a product may compulsorily link the selection of other features. Further, some of the features may be associated with a software asset such as web service. Consider the FD for a car crash crisis management system in Figure 1.5. The FD contains 47 features where 25 of them are optional. Some of the features are associated with services or software assets. The FD describes 33, 554, 432 different configurations of features. Can software assets in all configurations be composed into a valid product? Answering this requires creating either all products or a representative subset of all products. For instance, what are the set of all products that satisfy pairwise interaction between features? Creating these products will help us reveal invalid products. Manually creating products that satisfy all FD constraints is very tedious. Therefore, we ask, *how do we automate product generation in a software product line for various objectives?*

1.4 Problem Context and Challenges

We are motivated by the need for automatic generation of effective models in a modelling domain. The problem context for automatic model discovery is shown in Figure 1.6. The context identifes the following inputs:

- **Specification of a Modelling Domain:** The modelling domain specifies a set of models M. Examples of modelling domain specifications are metamodel for modelling languages and feature diagrams for SPLs.

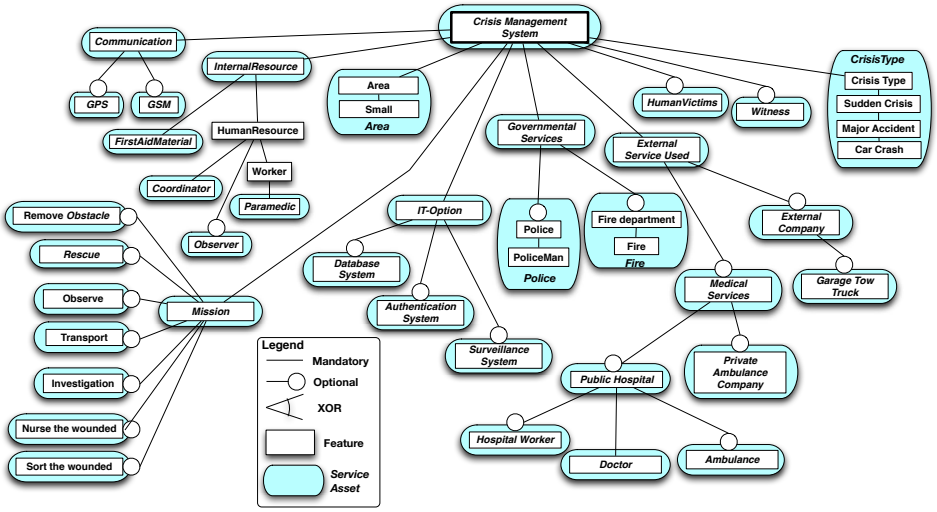

Figure 1.5: A Feature Diagram for Car Crash Crisis Management System

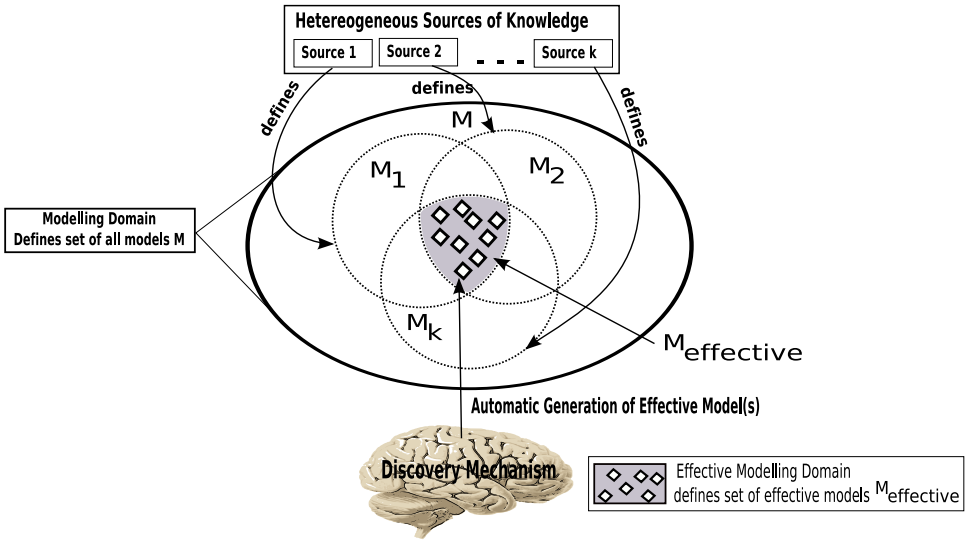

Figure 1.6: Problem Context for Automatic Model Discovery

- **Heterogenous Sources of Knowledge:** Heterogeneous sources of knowledge $Source_1, Source_2, ...Source_k$ possibly in different modelling languages specify subsets of the modelling domain $M_1, M_2, .., M_k$. The intersection of these subsets $M_1, M_2, ..., M_k$ is the effective modelling domain represented by a set of effective models $M_{effective}$. We can see the heterogeneous sources of knowledge as a set of constraints in different languages that limit the set of models M to a subset $M_{effective}$.

Given these inputs we ask: What is the automatic discovery mechanism that can create models in the set $M_{effective}$? This is the global question that intrigues us.

This question gives rise to a number of challenges pertaining to automatic model discovery. We describe the most important challenges below:

Challenge 1 Discovery mechanism: Generative vs. Constraint Satisfaction? Our research began with the exploration of existing mechanisms to automate the generation/discovery of models in a modelling domain. We classify existing approaches as either *generative* or those based on *constraint satisfaction*. The question was which approach is promising?

A generative approach attempts to incrementally create models in a modelling domain by object instantiation. For instance, in [10], the authors present an imperative algorithm and a tool to generate models that conform only to the Ecore specification of a metamodel. The approach does not ensure the satisfaction of constraints from heterogeneous sources of knowledge such as well-formedness rules. Similarly, in Ehrig et al. [22], the authors propose a graph grammar based approach to generate models that conform to a class diagram (or Ecore model). These models do not conform to any OCL constraints on the meta-model.

Constraint satisfaction based approaches attempts to transform a modelling domain to a set of variables and constraints on them. The set of constraints is solved using a constraint solver [45]. One or more low-level solutions are transformed as models of the modelling domain. This approach has been used in domain-specific settings such as software testing. The Korat (Chandra et al.) [9] system is able to generate data structures implemented in the Java Collections Framework that satisfy predicates. Similarly, Sarfraz Khurshid in his Ph.D. thesis [41] presents

the TestEra tool for generating Java data structures such as linked lists, tree maps, hash sets, heap arrays, and binary trees for testing. Both approaches are limited to standard data structures and not to the more generic notion of models. The most intriguing approach was the tool UML2Alloy [46]. The tool attempts to transform UML class diagram models, that largely resemble metamodels, to the formal specification language ALLOY [32]. One may then use ALLOY to analyze UML models by generating examples and counterexamples. Although the tool is not directly related to model discovery it aims to transform class diagram constructs to a constraint satisfaction problem in ALLOY. However, UML2Alloy does not transform complex metamodel constructs such as multiple inheritance and multiple containments. UML2Alloy fails to solicit the use of ALLOY when the size of the UML model is large making the approach unscalable.

Generative approaches create models incrementally and cannot satisfy constraints simultaneously. Therefore, a number of models may need to be rejected as they may not satisfy constraints. Therefore, constraint satisfaction based approaches seem more promising.

Challenge 2. Transforming the specification of a modelling domain to a constraint satisfaction problem The specification of a modelling domain contains a set of concepts and relationships between them. These relationships might encode complex constraints that are not easily transformed to a constraint satisfaction problem. Further, a large number of concepts and relationships may lead to a very large constraint satisfaction problem that becomes computationally intractable.

For instance, the transformation of a metamodel specification to a constraint satisfaction problem requires a constraints model for constructs such as:

- Multiple Inheritance
- Multiple containers for a class
- Opposite properties
- Identity properties

- Composite properties

The large size of a metamodel such as the UML with about 246 classes cannot be directly transformed to a tractable constraint satisfaction problem.

Challenge 3. Transforming heterogeneous sources of knowledge to constraints Heterogeneous sources of knowledge are specified in different modelling languages. However, for constraint satisfaction they all need to be transformed to constraints in a common language. For instance, the task of generation test models for a model transformation must satisfy constraints specified in a textual constraint language such as Object Constraint Language, test objectives, and the pre-condition of the model transformation expressed in the language of the transformation.

Challenge 4. Generation of models must be within tractable and finite bounds The discovery of models in a modelling domain requires generation of models of finite size. What are the heuristics to determine the appropriate size of a model that is sufficient to satisfy knowledge from heterogenous sources of knowledge.

Challenge 5. Detection of Inconsistent Sources of Knowledge Knowledge from various sources may be inconsistent with respect to the modelling domain specification. How can we detect such inconsistent sources of knowledge and eliminate them?

Challenge 6. Validating the Effectiveness of Models There is a need to conduct *rigorous experiments* that qualify models generated by constraint satisfaction. The qualification guarantees whether models are effective or useful for given objectives. These experiments must consider the effect of various influencing factors on the quality of the generated models. For instance, one may ask what is the influence of generating multiple models using a particular constraint solver on their effectiveness as test models? Do different parameters to the constraint solvers drastically affect the quality of the solutions?

1.5 Thesis

In this thesis we propose that it is possible to automatically discovery effective models in a modelling domain. Categorically, we address the problem of effective model discovery in two modelling domains: (a) Metamodels (b) Feature Diagrams. A metamodel is a very general specification of a modelling language's domain. A metamodel can be used to specify the domain of any domain-specific modelling language. However, legacy software systems and components cannot always be modelled or remodeled in a modelling language from scratch. Ideally, time tested components must be reused in their legacy form for combination with other components to build a software system. If we see these legacy components as features then the possible combinations of features is best modelled using the feature diagram language giving rise to a Software Product Line. The coarse-grained components associated with features may be combined in different configurations which are part of the feature diagram modelling domain. This distinction between pure models in a the domain of a modelling language and configurations of coarse-grained legacy components in a product line realize model-driven software construction at different levels .Therefore, we consider both specifications of modelling domains in this thesis.

Consequently, we propose two frameworks for model discovery specializing the general framework shown in Figure 1.6:

1. The framework for automatic effective model discovery in the modelling domain specified by a metamodel. This framework is embodied in the tool CARTIER.

2. The framework for automatic effective product discovery in the modelling domain specified by a feature diagram. This framework is embodied in the tool AVISHKAR.

1.5.1 A Framework for Automatic Effective Model Discovery

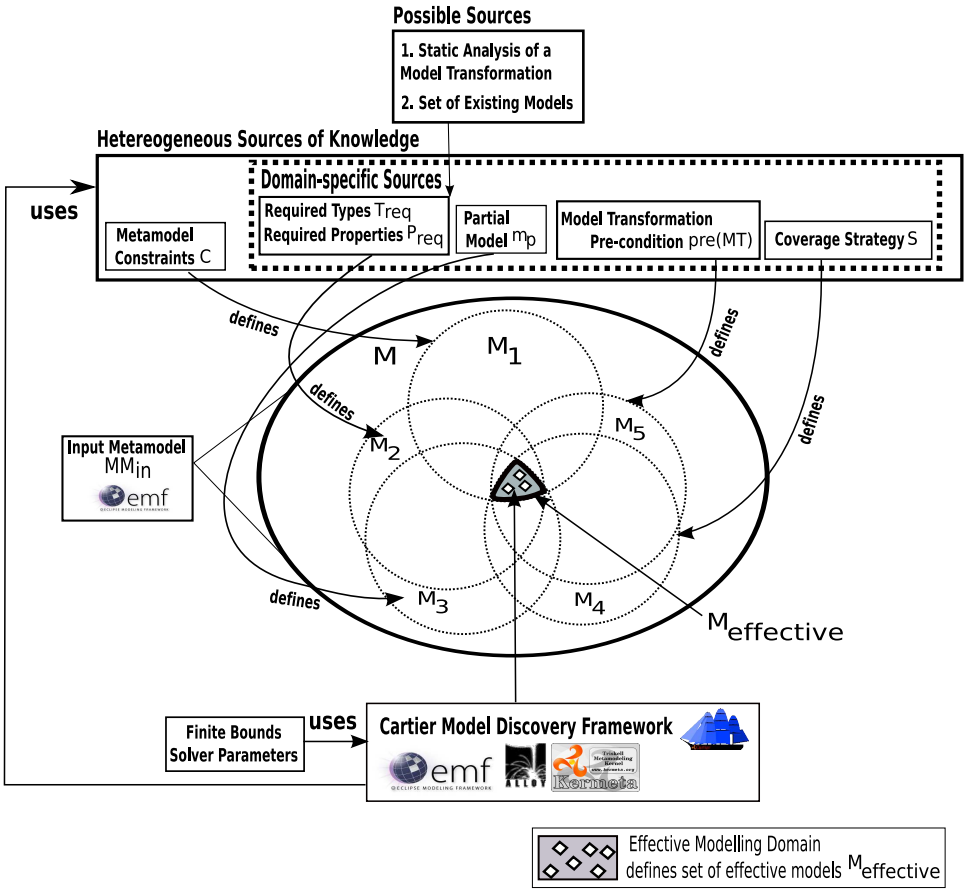

Figure 1.7: A Framework for Automatic Effective Model Discovery

The Figure 1.7 presents the overall view of the framework for automatic effective model discovery. The framework is embodied in the tool CARTIER. The name CARTIER comes from the famous French discoverer from St. Malo who discovered Canadian in-lands in Quebec. The primary input to the framework is the specification of the modelling domain given by an *input metamodel*. The *input metamodel* MM_{in} specifies a set of models M. The input metamodel consists of a set of types (class with properties, enumeration, primitive) to instantiate models of a modelling language. Concretely, the input metamodel is stored as an instance of the ECORE metamodel which is part of the industry standard Eclipse Modeling Framework (EMF) [27]. The models themselves are stored as XMI [2] files representing instances of the Ecore metamodel.

Heterogenous sources of knowledge constrain the modelling domain specified by a metamodel:

- **Required types** T_{req} **and properties** P_{req} in the input metamodel. The set of required types and properties helps extract a subset of the input metamodel called the *effective metamodel*. The effective metamodel specifies the subset of models $M_1 \subset M$. There can be many possible sources for the set of required types and properties:
  - Static analysis of a model transformation gives a set of types and properties in the input metamodel actually manipulated by the transformation.
  - A set of models conforming to the input metamodel is another source of required types and properties. Visiting the models in the set gives us a set of types and properties used in the metamodel. A typical real-world example of this could be in a classroom setting for object-oriented design using UML. The professor can point out to students the required types and properties he used to create UML by visiting every object of a set of models automatically.
- **Metamodel Constraints** C are expressed on an input metamodel using a textual constraint language such as Object Constraint Language (OCL) [56]. These constraints encode restrictions that cannot be specified using a diagrammatic Ecore model. We illustrate this as

the set $M_2 \subset M$.

- **Domain-specific sources of knowledge** may also help define the effective modelling domain. We present some of them below:

  – **Partial Model** m_p is a partially specified model using the input metamodel. For instance, a graphical model editor allows an user to create models in a modelling language such as UML state machines. An incomplete model in the editor is a partial model in the UML state machine language. The partial model may not respect all metamodel constraints of UML. Therefore, a partial model is often expressed as an instance of a *relaxed version of the input metamodel*. The partial model defines the subset $M_3 \subset M$.

  – **Coverage Strategy** S help define and generate *model fragments* [25] that cover a wide range of structural aspects in the input metamodel. For instance, an input domain partition based strategy helps generate a set of model fragments MF that cover partitions on all types and properties of the input metamodel. These model fragments help define an effective modelling domain for *coverage-based testing* of a model transformation. All test models that satisfy a coverage strategy contain the model fragments generated from the strategy. Model fragments are expressed in a modelling language that permits specification of ranges on properties of an input metamodel. A coverage strategy defines the subset $M_4 \subset M$.

  – **Transformation Pre-condition** $pre(MT)$ is a set of invariants on the metamodel that is specific to a model transformation MT. A model transformation often may not be designed to transform all models specified by its input metamodel. For instance, the transformation from class diagram models to entity relationship diagram models [5] require that all classes in the input class diagram have at least one primary attribute. The OCL [56] is often used to express pre-conditions. A pre-condion defines the subset $M_5 \subset M$.

The intersection of all the sources of knowledge defines the *effective modelling domain*. The effective modelling domain is the set of models defined by $M_{effective} \leftarrow M \cap M_1 \cap M_2 \cap M_3 \cap M_4 \cap M_5$.

The methodology for model discovery uses the sources of knowledge presented above to automatically generate models in the effective modelling domain. We enlist the steps below:

Step 1. Effective Metamodel Identification : We prune the input metamodel MM_{in} to obtain the effective metamodel $MM_{effective}$ using a metamodel pruning algorithm [73]. The effective metamodel contains the set of required types T_{req} and properties P_{req} provided as input and all its obligatory dependencies computed by the metamodel pruning algorithm. All unnecessary types and properties are removed. $MM_{effective}$ is super type of MM_{in} from a type theoretic point of view and a subset of MM_{in} from a set-theoretic point of view. The size of the effective metamodel $MM_{effective}$ is often considerably smaller than the size of the input metamodel MM_{in}.

Step 2. Transformation of Effective Modelling Domain Specification to ALLOY : The effective modelling domain specification is defined by a number of artifacts. It is initially defined by the effective metamodel $MM_{effective}$ and constrained by knowledge from one or more sources: (b) Metamodel constraints C (b) Partial model m_p (c) Model fragments MF from coverage strategy S, and (d) Pre-condition $pre(MT)$ of a model transformation MT. We transform these artifacts expressed in possibly different languages to a **constraint satisfaction problem** (CSP) in the unique formal specification language ALLOY [31] [32]. The theoretical formalism for expressing the CSP is **first-order relational logic**.

Step 3. Generation of Models in Effective Modelling Domain : We solve the CSP in ALLOY to generate models in the effective modelling domain. CARTIER achieves this by invoking Kod-Kod [23] in ALLOY to transform the CSP as relational model to Boolean Conjunctive Normal Form (CNF) . We invoke a satisfiability (SAT) solver such as MiniSAT [54], ZChaff [78] to solve the Boolean CNF. Finally, we transform low-level solutions of the CNF to models conforming to the input metamodel MM_{in}.

The generation of models in a modelling domain is often directed towards an objective. We

need to ensure that the objective is consistently achieved considering all influencing factors. A typical question maybe what is the effect of a SAT solver on the quality of the solution? To answer this question we need to perform experiments that generate several solutions for the same constraint satisfaction problem. There are many other influencing factors for which we conduct rigorous experiments to validate discovery effectiveness. In this thesis, we perform experiments in the following application domains:

1. Test model generation for model transformation testing
2. Partial model completion in domain-specific model editors

1.5.2 A Framework for Automatic Effective Product Discovery

The Figure 1.8 presents the overall view of the effective product discovery framework. The framework is embodied in the tool AVISHKAR. AVISHKAR in Hindi means *Invention* which signifies the character of the tool to discover products in a SPL. The primary input to the framework is the specification of the modelling domain given by a *feature diagram* or *feature model*. The *feature diagram* FD specifies a set of products P. *Feature Diagrams* (FD) introduced by Kang et al. [38] compactly represent all the products (or configurations) of a SPL in terms of features which can be composed. Feature diagrams have been formalized to perform SPL analysis [69]. In [69], Schobbens et al. propose an generic formal definition of FD which subsumes many existing FD dialects. We define a FD as follows:

- A FD consists of k features $f_1, f_2, ..., f_k$
- A feature f_i may be associated with a software asset.
- Features are organized in a parent-child relationship in a tree T. A feature with no further children is called a leaf.
- A parent-child relationship between features f_p and f_c are categorized as follows:

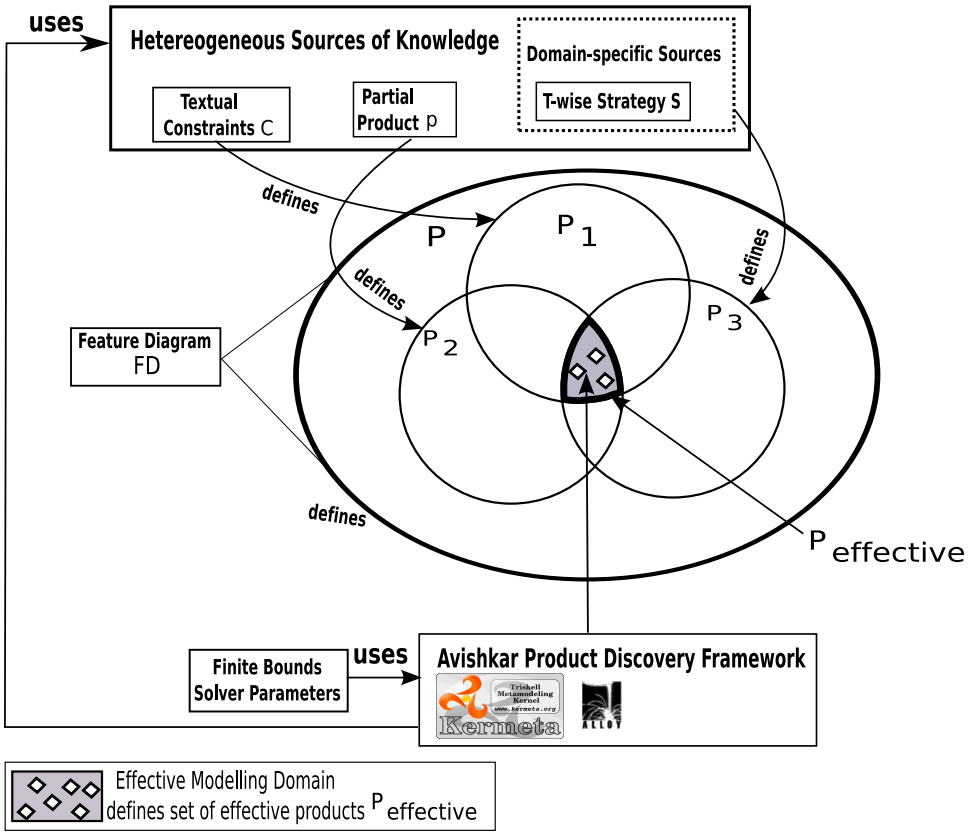

Figure 1.8: A Framework for Automatic Product Discovery

  - *Mandatory* - child feature f_c is required if f_p is selected.
  - *Optional* - child feature f_c *may* be selected if f_p is selected.
  - *OR* - at least one of the child-features $f_{c1}, f_{c2}, .., f_{c3}$ of f_p must be selected.
  - *Alternative (XOR)* - one of the child-features $f_{c1}, f_{c2}, .., f_{ck}$ of f_p must be selected.

- Cross tree relationships between two features f_i and f_j in the tree T are categorized as follows:
  - f_i requires f_j - The selection of f_i in a product implies the selection of f_j.
  - f_i excludes f_j - f_i and f_j cannot be part of the same product and are *mutually exclusive*.

Using the FD we create products/configurations of features. We can compose software assets. associated with these features to derive the final product.

Heterogenous sources of knowledge constrain the modelling domain specified by a feature diagram:

- **Textual Constraints** C are expressed on a set of features. Constraints are expressed textually when they cannot be directly encoded in the FD. These constraints specify the subset $P_1 \subset P$
- **Partial Product** p is a set of features chosen in product. The set of features may require the selection of other features to derive a complete product. The partial product specifies the subset $P_2 \subset P$
- **T-wise Strategy** S is a product generation strategy to detect faults in software product lines [44] [60]. The large number of products specified by a feature diagram can be sampled using a strategy such as $T-wise$. The objective is to generate a minimum number of products that satisfy all $T-wise$ interactions between features. For instance, in a FD with 25 optional features (see Figure 1.5) specifies at least 2^{25} products. A $2-wise$ strategy

where $T = 2$ will lead to generation of only $4 \times {}_{25}C_2 = 300$ products that cover all pairwise interactions between features. The $T-wise$ strategy for a particular value of T specifies the subset $P_3 \subset P$.

The intersection of all the sources of knowledge defines the *effective modelling domain*. The effective modelling domain is the set of products defined by $P_{effective} \leftarrow P \cap P_1 \cap P_2 \cap P_3$.

The product discovery methodology uses the sources of knowledge presented above to automatically generate products in the effective modelling domain of a FD. We enlist the steps below:

Step 1. Transformation of Feature Diagram to ALLOY : We transform a feature diagram to constraint satisfaction problem in the formal specification language ALLOY [32] [31].

Optional Step. Transformation of Partial Product to ALLOY and their Completion : We can transform a partial product p to ALLOY. It generates an ALLOY predicate that represents the partial information about selected features in the partial product. It can then solve the ALLOY model to generate one or more complete products.

Step 2. Generation of $T-wise$ Tuples and Detection of Valid Tuples using ALLOY: In this thesis we focus on generating products that satisfy $T-wise$ interaction between features. We first generate ALLOY predicate represents $T-wise$ tuples and detects those that are not consistent with the constraints in the FD.

Step 3. Scalable Generation of Products We propose *divide-and-compose* strategies to generate a set of products that cover all valid tuples that cover $T-wise$ interactions between features. The approach splits the satisfaction problem for all tuples to solving subsets of tuples. We solve multiple ALLOY models with these subsets to obtain sets of products. The sets of products are merged into a final set of products.

Do products discovered using the framework consistently attain their objectives? For instance we may ask what is the effect of divide-and-compose strategies on the redundancy of products generated? To answer this question we need to generate products considering all important influencing factors. In this thesis, we validate our framework using rigourous experiments

in the following application domains:

1. Test product generation that satisfy the t-wise interaction criteria

2. In ongoing/future work, we show that our framework can effectively sample the space of Quality of Service (QoS) of a dynamic web service who's variability is modelled as a FD.

1.6 Contributions

Both the frameworks for model and product discovery have led to the scientific contributions in this thesis. We explain these contributions in the following sub-sections. Some of the contributions are extracted and pin-pointed from the methodology already described in Section 1.5. We cite the relevant publications in peer-reviewed conferences and journals.

1.6.1 Contributions in Automatic Effective Model Discovery

Contribution 1.1 We present a comprehensive framework for generation of finite-sized effective models in any modelling language and constrained by heterogeneous sources of knowledge. The framework is embodied in the tool CARTIER. We use the formal specification language ALLOY for its ability represent constraints on graphs of objects and consequently to represent he entire metamodel as a constraint satisfaction problem. This contribution summarizes the answer to all challenges presented in Section 1.4 for a modelling domain specified by a metamodel. The tool CARTIER, saw its origins in our papers [66], [70].

Contribution 1.2. The framework transforms all metamodel constructs to ALLOY for constraint satisfaction. It also deals with metamodel with multiple inheritance by flattening it to single inheritance in ALLOY. Further, the framework presents transformation to ALLOY facts from constraints imposed by multiple containers, opposite properties, identify properties, and composite properties. This contribution addresses challenge 2 of Section 1.4. The transformation to ALLOY has been briefly described in two of our contributions [70] and [72].

Contribution 1.3. The framework is built using Kermeta modelling and model transformation language to simultaneously process models of knowledge in different languages. Each source of knowledge is expressed as a model in a modelling language. For instance, model fragments are expressed as models of a model fragment language. Kermeta can load, save, and manipulate models conforming to different metamodels at the same time. Therefore, CARTIER, written in Kermeta, transforms knowledge from various models to facts in the target language ALLOY. This contribution addresses challenge 3 of Section 1.4 and is published in our papers [70] [50].

Contribution 1.4. In the framework we present a metamodel pruning algorithm [73] that uses a set of required types and properties to generate an effective metamodel from large input metamodel. The effective metamodel is often very small and can be easily transformed to ALLOY as a tractable constraint satisfaction problem. This contribution addresses part of challenge 2 of Section 1.4 and presented in the paper [73].

Contribution 1.5. The framework contains facilities to assign finite bounds to the number of objects for each type in the model. It also transforms the solutions from the SAT solver in ALLOY called ALLOY instances back to high-level model conforming to a metamodel. The generation of models conforming to heterogeneous sources of knowledge helps determine inconsistencies between them if any. A selection of inconsistent sources of knowledge is made and either modified or eliminated from the specification of the effective modelling domain. This contribution addresses challenges 4 and 5 of Section 1.4 and is published in articles [70] and [72].

Contribution 1.6. We validate models generated for their effectiveness using the framework by performing the following experiments:

- **Test model generation for model transformation testing :** We generate thousands of models for a representative transformation. We use mutation analysis [51] to demonstrate that test models generated using *partitioning strategy* can detect 93% of the bugs compared to arbitrary generation 70%. We show that the partitioning strategy is not affected by various biases such as dependence on the ALLOY solver. The experimental study is published in [71] and journal version of the paper [64] has been submitted.

- **Partial model completion in domain-specific model editors:** We use our framework to generate recommendations to complete partial models in the model editor AToM^3 [29]. We illustrate that our framework can automatically complete partial models in a model editor. The experiments show that this can be done for small examples within reasonable time limits. This work is published in [67], [72].

This contribution addresses challenge 6 of Section 1.4.

1.6.2 Contributions in Automatic Effective Product Discovery

Contribution 2.1. We present a comprehensive framework for generation of effective products in a Software Product Line specified by a feature diagram. The framework is embodied in the tool AVISHKAR. The framework contains the transformation of a feature diagram to a constraint satisfaction problem in ALLOY. The framework invokes a solver on the ALLOY model to automatically generate products conforming to the feature diagram. This contribution summarizes the answer to all challenges in Section 1.4 for a modelling domain specified by a feature diagram.

Contribution 2.2. Given a set of feature selections (available/not available) the framework uses ALLOY to detect if a product can be created such that these feature selections are satisfied. A constraint for instance states that features f_1 exists in the product, while f_2 should not exist. If f_2 is a mandatory feature then AVISHKAR uses ALLOY to detect that the constraint is invalid. This contribution addresses challenge 5 of Section 1.4.

Contribution 2.3. Scalable generation of test products from a feature diagram Feature diagrams have been transformed to constraint satisfaction problems for testing a software product line. For instance, Cohen et. al. have applied combinatorial interaction testing to systematically select configurations/products [19] from a feature diagram. They consider various algorithms in order to compute configurations that satisfy pair-wise and t-wise criteria [18]. The constraints imposed due to feature relationships in a feature model are solved by calling SAT solvers such as ZChaff [78]. However, their approach is not very scalable when we consider large feature diagrams. Our framework contains *divide-and-compose* strategies to split the problem of test

product generation satisfying $T-wise$ into sub-problems. The tool AVISHKAR solves the sub-problems and merges the results into a small set of products that contain all valid tuples required by the $T-wise$ criteria. This mechanism renders our methodology to be a scalable approach to generate products in a software product line. This contribution addresses challenge 4 of Section 1.4.

Contribution 2.4. Validation of Effectiveness of Test Products: There is a need to perform experiments that qualify the products generated using our framework. We perform experiments to generate products for a transaction processing feature diagram AspectOPTIMA. We show that *redundancy* in $T-wise$ tuples is introduced in the products due to divide-and-compose strategies. In on-going work we perform experiments to generate different configurations of a dynamic web-service orchestration. We demonstrate that the QoS of a web-service varies with different configurations of the web-service. These variable QoS analysis experiments help us define an effective methodology to set robust contractual agreements for dynamic web service.

The above contributions are published in [60]. Two papers [3] [39] have been submitted to apply the product discovery tool AVISHKAR to analysis of varying QoS in a web service orchestration.

1.7 Thesis Organization

The thesis contains 6 chapters including the introduction. The next 5 chapters are organized as follows:

- Chapter 2, we introduce the context of MDE and the state of the art in automatic effective model discovery in a modelling domain.
- Chapter 3, we present automatic effective model discovery in the domain specified by a metamodel.
- Chapter 4, presents empirical validation of the framework for model discovery. In partic-

ular, we focus on two application domains for validation: (a) test model generation for a model transformation (b) partial model completion in the model editor AToM^3

- Chapter 5, we describe the framework for automatic test product discovery in a software product line. We empirically validate the framework for the redundancy in the generated products.

- Chapter 6, we summarize our work and present perspectives for future research. We briefly describe ongoing work on analysis of variable QoS in a dynamic web service.

Chapter 2

Context and State of the Art

Chapter 3

Automatic Effective Model Discovery

Chapter 4

Experiments in Effective Model Discovery

Chapter 5

Automatic Effective Product Discovery

Chapter 6

Conclusion and Perspectives

Model-driven engineering is leveraging the use of models in all several aspects of software development. Research into the theories, techniques, and tools for the various parts that make up a model driven system -models and transformations- is active and is seeing uptake in industrial contexts. However, as MDE is advancing it is facing challenges that characterize software engineering such as managing scalability, reliability and of particular interest in this thesis *automatic discovery of effective models to facilitate test-based validation and model construction.*

In order to address the challenges in *automatic model discovery*, we must develop mechanisms to explore and discover models in a modelling domain. Further, the models must conform to constraints heterogenous sources of knowledge such as metamodel constraints, search strategies, and partial models. How can we discover models in a modelling domain?

We address this question in the thesis by presenting a generic methodology that transforms a modelling domain and heterogeneous sources of knowledge to a constraint satisfaction problem in the formal specification language ALLOY. We solve the constraint satisfaction problem to discover models of interest. We specialize the generic methodology to first consider discovery in a modelling domain specified by a metamodel and constrained by heterogeneous sources of knowledge. This approach is concretely embodied in the tool CARTIER. We validate our approach and CARTIER by performing experiments in test model generation and partial model

completion. Second, we specialize our generic methodology for discovery in a modelling domain specified by a feature diagram of a Software Product Line. An SPL allows modelling variability in software systems using legacy software assets. This proves to be better than modelling everything from scratch in a modelling language specified by a metamodel. The methodology is embodied in the tool AVISHKAR. We validate AVISHKAR using experiments to generate test products for a transaction processing SPL AspectOPTIMA. Using both methodologies and tools CARTIER and AVISHKAR we demonstrate the feasibility of automatic model discovery in different modelling domains.

The rest of the chapter is organized as follows. In Section 6.1, we present a summary of the different chapters in this thesis. In Section 6.2, we present ongoing work on the use of AVISHKAR to analyze variation in QoS of web service orchestrations. Finally, in Section 6.3, we present perspectives for future work.

6.1 Summary and Conclusion

Chapter 2, presents the general context of MDE and the creation of modelling domains in MDE. In particular, we discuss (a) The modelling domain specified by a metamodel and its constraints in OCL and (b) The modelling domain specified by a feature diagram. The modelling domain specification are transformed to constraint satisfaction problem (CSP) in the formal specification language ALLOY which we describe in this chapter. The model transformation language to perform the transformation from modelling domain to ALLOY is Kermeta. We describe Kermeta, aspect-weaving in Kermeta, and model typing in Kermeta in this chapter. The chapter presents the state of the art in automatic model discovery with emphasis on test model generation and partial model completion. It also presents the state of the art in automatic product discovery.

In Chapter 3, we present a framework for automatic model discovery in the modelling domain specified by an input metamodel. The framework is embodied in the tool CARTIER. First, we present a metamodel pruning algorithm to extract an effective metamodel from the input

metamodel. The effective metamodel is a supertype of the input metamodel from a type-theoretic point of view and a subset of the input metamodel from a set-theoretic point of view. Second, we present a transformation of any metamodel or the effective metamodel to a CSP in ALLOY. The transformation takes into account all non-trivial artifacts in a metamodel such as multiple inheritance, multiplicity, containers, composite properties, opposite properties, and identity properties. A discussion on the validity and complexity of the transformation is presented. Third, we discuss how heterogeneous sources of knowledge such as OCL constraints may be transformed to ALLOY. Finally, we demonstrate the generation of models for the large case study of the UML metamodel.

In Chapter 4, we present experiments to validate automatic model discovery presented in Chapter 3. We present experiments in test model generation and partial model completion in a model editor. First we consider test model generation where we use input domain partitioning strategies to generate test models using CARTIER. These models detect 93% of the bugs in a representative model transformation compared to only 70% for unguided generation. The representative transformation from UML class diagrams to RDBMS models exercises most model transformation operators. The input metamodel UML contains almost all complex metamodel constructs and is a widely used industrial metamodel. In the second experiment we perform partial model completion in a model editor. Given a partially specified model in a model editor we use CARTIER to generate recommendations to complete partial model. We present an algorithm to transform a partial model to an ALLOY predicate. We solve the predicate to generate one or more model completions for models in the Hierarchical Finite State Machine modelling language. We present the different times taken for completion of partial models of various size.

Chapter 5, we present a framework for automatic product discovery in the modelling domain specified by the feature diagram (FD) of a Software Product Line (SPL). The framework is embodied in the tool AVISHKAR. We first transform a FD to a CSP in ALLOY. We solve the resulting ALLOY model to generate products. The focus of this chapter is to discover test products that satisfy the T-wise coverage criteria between features in the FD. Generation of test

products for large FDs using ALLOY is not tractable. We scale the use of ALLOY using divide-and-compose strategies that can generate a close to minimal set of test products that satisfy T-wise coverage. A side-effect of using divide-and-compose strategies is the introduction of redundancies of pairs in products. We presents metrics to measure these redundancies. Using pairwise coverage we show that AVISHKAR generates test products with acceptable redundancy for a transaction processing FD AspectOPTIMA.

6.2 Ongoing Work: Variability Modeling and QoS Analysis of Web Services Orchestrations

In ongoing work we model the variability in a composite web services orchestration using FDs. We apply AVISHKAR to generate different possible orchestrations of a composite web service. We analyze the consequent variation in Quality of Service (QoS) of these orchestrations using probabilistic models of QoS. This work is described below.

Inherent choice in an ever-growing world of services is making *orchestration variability* a significant aspect of a composite web service. The different ways of orchestrating atomic services can be seen as either multiple variants of a composite service created offline or an online composite service that reconfigures dynamically. In either case, we expect to observe variation in Quality of Service (QoS) across different orchestrations. This variation in QoS must not only take into account service variability but also the uncertainty/probabilistic nature of QoS itself.

It is important to consider orchestration variability and its implications on composite service behavior. For instance, not considering variability leads to misrepresentation of contractual agreements on QoS [75]. Contractual agreements such as service level agreements (SLAs) [58] is the industry standard to ensure QoS compliance between service providers and customers. Usual deviations from SLAs are a result of non-incorporation of QoS variability and in particular QoS outliers in its specification. Therefore, we need systematic analysis of variability in order

to improve robustness of contractual SLAs.

Modeling variability in web service orchestrations and analyzing the consequent variation in QoS is the principal subject of this work. We present a methodology to model orchestration variability using *feature diagrams* (FDs). Feature diagrams [37] provide a graphical constraints-based framework to specify a product-line of orchestrations. Each orchestration in the product-line is represented as an authorized configuration of invoked/rejected atomic services. In most cases the FD specifies a very large set of configurations making exhaustive sampling infeasible. Instead, we sample the set of all possible configurations by systematically analyzing configurations covering all valid pairwise service interactions [21]. Finally, we use probabilistic models of QoS [65] to analyze variants of orchestrations derived from all valid configurations.

We use our methodology to investigate merits of systematically sampling the set of all configurations of web service orchestrations. Random sampling of configurations, generally employed, is both ineffective and expensive because it cannot be systematic and requires computing QoS values for a large number of configurations. Moreover, random sampling is not easy when FD constraints like mutual exclusion/requirement need to be satisfied. This work focuses on the adaptation of combinatorial interaction testing (CIT) [16] to select a sample of configurations that covers all pairwise interactions of services while satisfying all FD constraints. We use the recently proposed scalable approach in [60] for generating these configurations. CIT is based on the observation that most of the faults are triggered by interactions between a small number of variables [44]. For example, consider the output quality of printing web pages depending on a hypothetical combination of parameters represented in Table 6.1.

Parameters	**Options**
Operating System	Windows, Linux, Macintosh
Browser	IE, Firefox, Chrome, Opera
Printer Model	HP, Canon, Xerox, Epson
Printer Type	Ink-Jet, Laser
Orientation	Portrait, Landscape
Size	A3, A4, A5, A6
Color	B/W, Multicolor

Table 6.1: Examples of printing parameters requiring comparison.

An exhaustive generation of combinations of these parameter options would entail 1536

cases with many redundancies. Pairwise coverage of optional combinations would require just 17 tests, resulting in a reduction of close to 99%. The number of exhaustive tests will increase exponentially with addition of more parameters/options requiring an employment of efficient sampling strategies.

Pairwise coverage test generation has been used to detect faults in software systems in prior work [21], [16]. However, the application of these coverage-based techniques to sample configurations in service orchestrations is yet to be examined. This work performs such an examination through a series of experiments that aim at investigating several facets of the question: is pairwise service interaction sampling of orchestration configurations effective for overall QoS analysis and the consequent definition of a global SLA?

Our experiments are based on a *crisis management system* (CMS) case study described comprehensively in [42]. This work reports on the following questions:

- Is it possible to automatically sample the orchestration configurations space to select configurations that cover all pairwise service interactions?
- What global QoS metrics can we infer from a pairwise sample?
- How stable is the SLA computed from a pairwise sample? This question is related to the fact that the automatic generation of pairwise configurations is not deterministic and thus the global contract might vary depending on the generated *sample*.
- Is pairwise sampling more effective and efficient compared to exhaustive sampling of the configuration space?

From our experimentation, we have seen that analysis of a family of configurations (and their corresponding QoS values) can be accurately represented by a small set of configurations satisfying pairwise interactions. Consistency of various generated pairwise solutions are also demonstrated through simulations. This comprehensive analysis of variability helps the orchestrator understand the global QoS extremities of the composite service before negotiating a SLA

agreement. Deterioration in service quality or non-compliance of SLA standards during online deployment of the service is thus prevented. Improvements in the orchestration model to eliminate some deviant configurations (causing excessive deterioration of end-to-end QoS) or grouping a family of configurations with similar QoS behavior are other extensions of this technique.

Accurate offline analysis of a composite web service before its deployment is essential to ensure non-repudiation of a SLA contract. This is necessary to maintain optimal QoS behavior of mission-critical services such as crisis management. In order to do this, the service provider must keep in mind the probabilistic aspect of QoS parameters and the variable configurations in a composite service. In this work, we study an analysis framework to test the QoS of an orchestration before deployment. Further, the notion of systematic pairwise sampling procedure has also been demonstrated, which provides a more efficient sampling of the configuration space than exhaustive trails while still maintaining sufficient coverage. Larger FD and orchestration models can be analyzed using the divide-and-compose approaches [60] to handle this scalability issue. This should provide a simple, systematic and stochastically correct methodology for pre-deployment QoS analysis of a composite service.

While this work concentrates on a particular composition of fixed atomic services, a future area of interest would be optimal compositions. The use of configurations and scenarios modeled by a FD leads to a family of composite services. These, in turn, may be used to generate many versions of the orchestrations. Further implementation of these techniques to study larger composite orchestrations is useful for both obtaining realistic QoS bounds and product generation of families of services.

6.2.1 Related Work

The combinatorial testing framework described by Cohen et al. [16] has been applied extensively to efficient testing for fault detection. In the work of Cohen et al. [17], this technique is extended to software product lines with highly configurable systems. Modeling variability in SPLs using

feature models is the work of Jaring and Boschet [34] where they show that the robustness of a SPL architecture is related to the type of variability. To ensure that constraints in the FD are incorporated in the efficient sampling of t-wise tests, the solver proposed by Perrouin et al. [60] is used. In [48] Larsen et al. define modal I/O automata, an extension of interface automata with modality. These allow models of varying configurations to be developed from a single produce line while disallowing trivial implementations. Such a notion when extended to a composite service can provide interesting configurations and versions of composite products as described in [48].

Pre-deployment testing of SLAs has been studied by Di Penta et al. [59], where they make use of genetic algorithms to generate test data causing SLA violations. Analysis of white and black box approaches are provided in the paper. In [11], Bruno et al. make use of regression testing to ensure that an evolving service maintains the functional and QoS assumptions. The service consistency verification due to evolution is done by executing test suites contained in a XML encoded facet attached to the service.

The use of probabilistic QoS and soft contracts was introduced by Rosario et. al [65] and Bistarelli et al. [6]. Instead of using fixed hard bound values for parameters such as response time, the authors proposed a soft contract monitoring approach to model the QoS measurement. The composite service QoS was modeled using probabilistic processes by Hwang et al. [30] where the authors combine orchestration constructs to derive global probability distributions.

In our work, we extend these two notions to analyze the QoS of a composite orchestration under various configurations. The hard contract notions of end-to-end QoS are replaced by the probability quantile based approach. This provides the service provider the technique for estimating composite service QoS distributions and estimating the global soft contract SLA. Though formal analysis of end-to-end QoS has been studied in Cardoso et al. [14], there are no practical testing tools available for the service provider. The pairwise testing procedure has been shown to outperform other testing techniques in [16]. We extend this testing tool to develop a generic testing methodology to query end-to-end QoS of a web service. The efficacy of this

scheme is provided though experimental verification.

Related empirical studies of optimal QoS compositions make use of genetic programming in Canfora et al. [13] and linear programming in Zeng et al. [79]. These are dynamic techniques to choose the best possible atomic services and configurations keeping QoS in mind. This differs from our work due to the assumption that the atomic services and their composition have already been defined. The goal is to analyze the variable configurations that may result due to invocation or non-invocation of particular web services. This is of need when atomic SLAs and their interactions in an orchestration have already been established. Such efficient, systematic and stochastically correct analysis provides an accurate estimate of the global QoS distributions of composite services.

6.3 Perspectives

The ideas presented in this thesis represents a first step towards automating discovery of models in a modelling domain. The work evokes a number of future avenues of research.

6.3.1 A Family of Metamodel Pruning Algorithms

In Chapter 3, we present the metamodel pruning algorithm to extract an effective metamodel from an input metamodel. We show that the effective metamodel is a supertype of the input metamodel from a type-theoretic point of view. It is also a subset of the input metamodel from a set-theoretic point of view. The supertype property of the effective metamodel makes it *backward compatible* with the input metamodel. By backward compatibility we mean all model transformations or operations for the effective metamodel are valid for the input metamodel. Similarly, all models of the effective metamodel are also valid instances of the input metamodel. This property has practical implications to the usage of large industry standard metamodels such as the UML. Experts may extract a small and relevant subset of the UML to create models or transformations while preserving type conformance with UML itself. Therefore, the type confor-

mance property between an effective metamodel and the large input metamodel leverages several applications of the metamodel pruning algorithm. In future work, we would like to investigate the possibility of creating a family of metamodel pruning algorithms.

The notion of a family metamodel pruning algorithms is based on the possibility of developing combinations of atomic pruning operators that satisfy type conformance. An atomic pruning operator has an input metamodel and gives an effective metamodel as output. The effective metamodel shows type conformance with the input metamodel. A given sequence of pruning operators on an input metamodel should give an effective metamodel as output such that it shows type conformance with the original input metamodel. This is due to a transitivity property of pruning operators in a sequence. What are the different possible sequences of pruning operators? Which pruning operators are commutative? Which pruning operators in sequence show transitivity? These are some of the questions that need explorations.

6.3.2 Transforming OCL Subset to ALLOY

In Chapter 3, we present a complete transformation of a metamodel to ALLOY implemented in the CARTIER framework. However, not all constraints may be expressed in the metamodel. A textual constraint language such as the Object Constraint Language (OCL) is the industrial standard to expressed additional metamodel constraints. OCL is a side-effect language that queries a model of a modelling language and check structural properties on the model. There are several similarities between OCL and ALLOY in the way constraints are expressed. In future work, we would like to focus on transforming a subset of OCL to ALLOY facts or predicates. ALLOY also has some features not yet exploited in OCL which may help concurrently improve OCL itself. In [76], the authors presents some shortcomings of OCL with respect to ALLOY.

6.3.3 Product Discovery Strategies based on Feature Diagram structure

In Chapter 5, we present the AVISHKAR framework to generate products that satisfy all T-wise feature interactions in a FD. We believe that the quality of the test products and the number

of effective test products may be improved if we consider the structural semantics of the FD in developing new strategies. New strategies will essentially comprise of analyzing the tree structure of the FD to obtain knowledge to generate test products. The idea is to generate test products using knowledge that explore the FD's product space while respecting FD constraints. This is in contrast, to T-wise generation where a lot of feature interactions are generated that do not satisfy the FD constraints. Only a subset of the T-wise interactions are valid and are used to generate test products.

6.3.4 Scaling Constraint Solving using ALLOY

In most of the thesis we have used ALLOY to generate models or products. Generation using ALLOY is based on the hypothesis that small models are often effective. We demonstrate this using experiments in test model generation. However, for product generation we make advances in scaling ALLOY to generate products for a large FD. The idea is based on dividing the constraint satisfaction problem and composing the results into a final set of products. This approximate approach can handle large FDs but introduces some tuple redundancy in the generated products. What are other ways to scale the size and number of models that can be generated using ALLOY ? This is a question that intrigues us. We would like to research this question in two axes: (a) Develop divide and compose strategies to first create small models and then weave them together into larger models (b) Leverage SAT solving using parallel SAT solvers such as ManySAT [28] in order to generate instances from a large and highly-constrained ALLOY model.

Appendix

List of Figures

List of Tables

Listings

Bibliography

[1] The atl homepage, http://www.sciences.univ-nantes.fr/lina/atl/contrib/bezivin.

[2] Xmi meta-data interchange format. http://www.omg.org/technology/documents/formal/xmi.htm.

[3] Kattepur Ajay, Sen S., and Baudry B. Pairwise interactions to effectively sample qos in dynamic web services. 2010.

[4] Andrew J. Ko, Htet Htet Aung, and Brad A. Myers. Design requirements for more flexible structured editors from a study of programmer's text editing. In *CHI 2005*, 2005.

[5] Jean Bezivin, Bernhard Rumpe, Andy Schurr, and Laurence Tratt. Model transformations in practice workshop, october 3rd 2005, part of models 2005. In *Proceedings of MoDELS*, 2005.

[6] S. Bistarelli and F. S. Santini. Soft constraints for quality aspects in service oriented architectures. In *Fourth European Young Researchers Workshop on Service Oriented Computing*, Italy, 2009.

[7] Grady Booch. *Object-oriented analysis and design with applications*. Benjamin-Cummings Publishing Co., Inc., Redmond City, CA, USA, 1994.

[8] Gerard Boudier, Ferdinando Gallo, Regis Minot, and Ian Thomas. An overview of pcte and pcte+. volume 24, pages 248–257, New York, NY, USA, 1989. ACM.

[9] Chandrasekhar Boyapati, Sarfraz Khurshid, and Darko Marinov. Korat: automated testing based on java predicates. In *Proceedings of the 2002 ACM SIGSOFT international symposium on Software testing and analysis*, 2002.

[10] E. Brottier, F. Fleurey, J. Steel, B. Baudry, and Y. Le Traon. Metamodel-based test generation for model transformations: an algorithm and a tool. In *Proceedings of ISSRE'06*, Raleigh, NC, USA, 2006.

[11] M. Bruno, G. Canfora, M. Di Penta, G. Esposito, and V. Mazza. Using test cases as contract to ensure service compliance across releases. In *Proc. of the 3rd Intl. Conf. in Service-Oriented Computing*, pages 87–100, Amsterdam, The Netherlands, 2005.

[12] Eugene C Butcher, Ellen L Berg, and Eric J.Kunel. Systems biology in drug discovery. *Nature Biotechnology*, pages 1253–1259, 2004.

[13] G. Canfora, M. Di Penta, R. Esposito, and M. L. Villani. An approach for qos-aware service composition based on genetic algorithms. In *Conf. on Genetic and evolutionary computation*, pages 1069–1075, USA, 2005.

[14] J. Cardoso, J. Miller, A. Sheth, and J. Arnold. Modeling quality of service for workflows and web service processes, lsdis lab technical report pp 1-44. Technical report, University of Georgia, 2002.

[15] Paul Clements and Linda Northrop. *Software Product Lines: Practices and Patterns*. Addison Wesley, Reading, MA, USA,, 2001.

[16] David M. Cohen, Ieee Computer Society, Siddhartha R. Dalal, Michael L. Fredman, and Gardner C. Patton. The aetg system: an approach to testing based on combinatorial design. *IEEE Transactions on Software Engineering*, 23:437–444, 1997.

[17] M. B. Cohen, M. B. Dwyer, and J. Shi. Constructing interaction test suites for highly-configurable systems in the presence of constraints: A greedy approach. *IEEE Trans. on Software Engineering*, 34(5):633–650, 2008.

[18] M.B. Cohen, M.B. Dwyer, and J. Shi. Interaction testing of highly-configurable systems in the presence of constraints. In *ISSTA*, pages 129–139, 2007.

[19] Myra B. Cohen, Matthew B. Dwyer, and Jiangfan Shi. Coverage and adequacy in software product line testing. In *ROSATEA '06: Proceedings of the ISSTA 2006 workshop on Role of software architecture for testing and analysis*, pages 53–63, New York, NY, USA, 2006. ACM.

[20] Krzysztof Czarnecki and Simon Helsen. Classification of model transformation approaches, 2003.

[21] J. Czerwonka. Pairwise testing in the real world. In *24th Pacific Northwest Software Quality Conference*, 2006.

[22] K. Ehrig, J.M. Kuster, G. Taentzer, and J. Winkelmann. Generating instance models from meta models. In *FMOODS'06 (Formal Methods for Open Object-Based Distributed Systems)*, pages 156 – 170., Bologna, Italy, June 2006.

[23] Emina Torlak and Daniel Jackson. Kodkod: A Relational Model Finder. In *Tools and Algorithms for Construction and Analysis of Systems*, Braga,Portugal, March 2007.

[24] Patrick Farail, Pierre Gaufillet, Agusti Canals, Christophe Le Camus, David Sciamma, Pierre Michel, Xavier Crégut, and Marc Pantel. The TOPCASED project: a toolkit in open source for critical aeronautic systems design. In *Embedded Real Time Software (ERTS)*, Toulouse, FebruaryMay 2006.

[25] Franck Fleurey, Benoit Baudry, Pierre-Alan Muller, and Yves Le Traon. Towards dependable model transformations: Qualifying input test data. *Software and Systems Modelling (Accepted)*, 2007.

[26] C. A. Floudas, A. R. Ciric, and I. E. Grossmann. Automatic synthesis of optimum heat exchanger network configurations. *AIChE Journal*, 32(2):276–290, 1986.

[27] Budinsky Frank. *Eclipse Modeling Framework*, volume 1 of *The Eclipse Series*. Addison-Wesley, 2004.

[28] Youssef Hamadi, Said Jabbour, and Lakhdar Sais. Manysat: a new parallel sat solver. *Journal of Satisfiability Boolean Modeling and Computation, In special issue on Parallel SAT solving*, 6:245–262, 2009.

[29] Hans Vangheluwe and Juan de Lara. Domain-Specific Modelling with AToM3. In *The 4th OOPSLA Workshop on Domain-Specific Modeling*, Vancouver, Canada, October 2004.

[30] S. Y. Hwang, H. Wang, J. Tang, and J. Srivastava. A probabilistic approach to modeling and estimating the qos of web-services-based workflows. *Elsevier Information Sciences*, 177:5484–5503, 2007.

[31] Daniel Jackson. *Software Abstractions: Logic, Language, and Analysis*. The MIT Press, April 2006.

[32] Daniel Jackson. http://alloy.mit.edu. 2008.

[33] Ivar Jacobson. *Object-oriented software engineering*. ACM Press, 1991.

[34] M. Jaring and J. Bosch. Representing variability in software product lines: A case study. In *Proc. of the Second Intl. Conf. on Software Product Lines*, pages 15–36, London, UK, 2002.

[35] F Jouault and I Kurtev. On the Architectural Alignment of ATL and QVT. In *Proceedings of ACM Symposium on Applied Computing (SAC 06)*, Dijon, FRA, April 2006.

[36] Frédéric Jouault, Freddy Allilaire, Jean Bézivin, and Ivan Kurtev. Atl: A model transformation tool. *Science of Computer Programming*, 72(1-2):31–39, June 2008.

[37] K. Kang, S. Cohen, J. Hess, W. Novak, and S. Peterson. Feature-Oriented Domain Analysis (FODA) Feasibility Study. Technical Report CMU/SEI-90-TR-21, Software Engineering Institute, November 1990.

[38] Kyo C. Kang, Sajoong Kim, Jaejoon Lee, Kijoo Kim, Euiseob Shin, and Moonhang Huh. FORM: A Feature-Oriented Reuse Method with Domain-Specific Reference Architectures. *Ann. Softw. Eng.*, 5:143–168, 1998.

[39] Baudry B. Beneveniste A. Jard C. Kattepur Ajay, Sen S. Variability modeling and qos analysis of web services orchestrations. 2010.

[40] Kermeta. http://www.kermeta.org/.

[41] Sarfraz Khurshid. *Generating Structurally Complex Tests from Declarative Constraints*. PhD thesis, MIT, 2003.

[42] J. Kienzle, N. Guelfi, and S. Mustafiz. Crisis management systems: A case study for aspect-oriented modeling, mcgill univ.

[43] John R. Koza, Forrest H. Bennett, III, David Andre, and Martin A. Keane. Automated wywiwyg design of both the topology and component values of electrical circuits using genetic programming. In *GECCO '96: Proceedings of the First Annual Conference on Genetic Programming*, pages 123–131, Cambridge, MA, USA, 1996. MIT Press.

[44] D. Richard Kuhn, Dolores R. Wallace, and Albert M. Gallo. Software fault interactions and implications for software testing. *IEEE Trans. Softw. Eng.*, 30(6):418–421, 2004.

[45] Vipin Kumar. Algorithms for constraint satisfaction problems: A survey. *AI Magazine*, pages 32–44, 1992.

[46] Kyriakos Anastasakis, Behzad Bordbar, Geri Georg, and Indrakshi Ray. UML2Alloy: A Challenging Model Transformation. In *MoDELS*, pages 436–450, 2007.

[47] Pat Langley, Herbert A. Simon, G. Bradshaw, and y J. Zytkow. *Scientific Discovery, Computational Explorations of the Creative Mind*. MIT Press, Cambridge, Massachusetts, 1987.

[48] K. G. Larsen, U. Nyman, and A. Wasowski. Modal i/o automata for interface and product line theories. In *Joint European Conf. on Theory and Practices of Software*, pages 64–79, Braga, Portugal, 2007.

[49] Hod Lipson and Jordan B. Pollack. Automatic design and manufacture of robotic lifeforms. *Nature*, 406(6799):974–978, 2000.

[50] Faucher C. Barais O. Jezequel J.M. Moha N., Sen S. Evaluation of kermeta for solving graph-based problems. *,International Journal on Software Tools for Technology Transfer (STTT), 10 p. (In Press)*, 2010.

[51] J.-M. Mottu, B. Baudry, and Y. Le Traon. Mutation analysis testing for model transformations. In *Proceedings of ECMDA'06*, Bilbao, Spain, July 2006.

[52] Pierre-Alain Muller, Franck Fleurey, and Jean-Marc Jézéquel. Weaving executability into object-oriented meta-languages. In *Proc. of MODELS/UML'2005*, LNCS, Jamaica, 2005. Springer.

[53] Jean Bezivin Nicolas, Nicolas Farcet, Jean marc J?z?quel, Beno?t Langlois, and Damien Pollet. Reflective model driven engineering. In *The 6th International Conference on the Unified Modeling Language, Modeling Languages and Applications (UML 2003)*, pages 175–189. Springer, 2003.

[54] Niklas Een and Niklas S?rensson. MiniSat a SAT solver with conflict-clause minimization. In *SAT*, 2005.

[55] Anders Olsen, Ove Faergemand, Birger Moller-Pedersen, Rick Reed, and J.R.W. Smith. *Systems Engineering with SDL-92*. North Holland, 1995.

[56] OMG. UML 2.0 OCL 2.0 specification. Technical Report ptc/05-06-06, Object Management Group, June 2005.

[57] OMG. The uml 2.1.2 infrastructure specification. Technical Report formal/2007-11-04, OMG, April 2007. OMG Available Specification.

[58] A. Paschke and M. Bichler. Knowledge representation concepts for automated sla management. *Journal of Decision Support Systems*, 46:187–205, 2008.

[59] M. Di Penta, G. Canfora, and G. Esposito. Search-based testing of service level agreements. In *Proc. of the 9th Conf. on Genetic and evolutionary computation*, pages 1090–1097, London, England, 2007.

[60] Gilles Perrouin, Sagar Sen, Jacques Klein, Benoit Baudry, and Yves le Traon. Automatic and scalable t-wise test case generation strategies for software product lines. In *International Conference on Software Testing*, 2010.

[61] Karl Popper. *The Logic of Scientific Discovery*. Hutchinson & Co., 1959.

[62] Isidore Rigoutsos, Aris Floratos, Laxmi Parida, Yuan Gao, and Daniel Platt. The emergence of pattern discovery techniques in computational biology. *Metabolic Engineering*, 2(3):159 – 177, 2000.

[63] James Rumbaugh, Michael Blaha, William Premerlani, Frederick Eddy, and William Lorensen. *Object-oriented modeling and design*. Prentice-Hall, 1991.

[64] Sen S., Mottu J.M., and Baudry B. Automatic model generation for transformation testing. *SoSyM special issue*, 2010.

[65] S. Haar S. Rosario, A. Benveniste and C. Jard. Probabilistic qos and soft contracts for transaction-based web services orchestrations. *IEEE Trans. on Services Computing*, 1(4):187 – 200, 2008.

[66] Sagar Sen, Benoit Baudry, and Doina Precup. Partial Model Completion in Model Driven Engineering using Constraint Logic Programming. In *International Conference on the Applications of Declarative Programming*, 2007.

[67] Sagar Sen, Benoit Baudry, and Hans Vangheluwe. Domain-specific Model Editors with Model Completion. In *Multi-paradigm modelling workshop associated with MoDeLs 2007*, Nashville, TN, USA, October 2007.

[68] Sandeep Neema, Janos Szitpanovits, and Gabor Karsai. Constraint-Based Design Space Exploration and Model Synthesis. In *Proceedings of EMSOFT 2003, Lecture Notes in Computer Science*, number 2855, pages 290–305, 2003.

[69] P.Y. Schobbens, P. Heymans, J.C. Trigaux, and Y. Bontemps. Generic semantics of feature diagrams. *Computer Networks*, 51(2):456–479, 2007.

[70] Sagar Sen, Benoit Baudry, and Jean-Marie Mottu. On combining multi-formalism knowledge to select test models for model transformation testing. In *IEEE International Conference on Software Testing*, Lillehammer, Norway, April 2008.

[71] Sagar Sen, Benoit Baudry, and Jean-Marie Mottu. Automatic model generation strategies for model transformation testing. In *ICMT*, pages 148–164, 2009.

[72] Sagar Sen, Benoit Baudry, and Hans Vangheluwe. Towards domain-specific model editors with automatic model completion. *Simulation*, 86(2):109–126, 2010.

[73] Sagar Sen, Naouel Moha, Benoit Baudry, and Jean-Marc Jezequel. Meta-model pruning. In *Model Driven Engineering Languages and Systems, 12th International Conference (MODELS)*, Denver, CO, USA, October 4-9 2009.

[74] Shane Sendall and Wojtek Kozaczynski. Model transformation: The heart and soul of model-driven software development. *IEEE Softw.*, 20(5):42–45, 2003.

[75] Vladimir Tosic and Bernard Pagurek. On comprehensive contractual descriptions of web services. In *EEE '05: Proceedings of the 2005 IEEE International Conference on e-Technology, e-Commerce and e-Service (EEE'05) on e-Technology, e-Commerce and e-Service*, pages 444–449, Washington, DC, USA, 2005. IEEE Computer Society.

[76] Mandana Vaziri and Daniel Jackson. Some shortcomings of ocl, the object constraint language of uml. In *TOOLS '00: Proceedings of the Technology of Object-Oriented Languages and Systems (TOOLS 34'00)*, page 555, Washington, DC, USA, 2000. IEEE Computer Society.

[77] Viatra2. Department of measurement and information systems, budapest university of technology and economics. http://www.eclipse.org/gmt/VIATRA2/.

[78] Y. S. Mahajan, Z. Fu, and S. Malik. ZChaff2004: An Efficient SAT Solver. In *Lecture Notes in Computer Science SAT 2004 Special Volume LNCS 3542.*, pages 360–375, 2004.

[79] L. Zeng, B. Benatallah, A. H. Ngu, M. Dumas, J. Kalagnanam, and H. Chang. Qos-aware middleware for web services composition. *IEEE Trans. on Software Engineering*, 30(5):311–327, 2004.