**Titre de la thèse :**
Modèles exécutables pour les simulateurs de drones marins.

| Encadrants | | |
|---|---|---|
| Jean-Philippe Babau | Equipe SHAKER, UBO | Directeur |
| Eric Cariou | Equipe P4S, UBO | Encadrant |
| Mickaël Kerboeuf | Equipe SHAKER, UBO | Encadrant |

**Mots clés :** DSL exécutable, code métier, drones marins, simulateurs

**Sujet de thèse :**

***Résumé***

La thèse a pour objectif de développer des environnements de production agiles de logiciels. L'application concerne la construction de simulateurs de drones marins. En terme de génie logiciel, il s'agit de pouvoir associer un modèle de comportement exécutable (comme une machine à états UML représentant un comportement d'un ascenseur) avec des opérations métier (faire monter l'ascenseur, fermer ses portes…) écrites dans un langage de programmation classique. L'intérêt de ces modèles exécutables définis par des langages dédiés (des DSL) est qu'ils permettent de valider le comportement du système par simulation dès la phase de conception. Ensuite, le modèle peut être intégré directement dans le système logiciel final sans modification et permet donc de supprimer une grande partie du développement du code qui se faisait à la main, ce qui est source d'introduction d'erreurs. Cette approche permettra de développer avantageusement des simulateurs logiciels de drones marins en étendant par exemple la plateforme CARES du Lab-STICC. L'idée est de définir le comportement de la simulation dans un modèle facilement modifiable et paramétrable, de pouvoir changer les opérations de simulation liées au modèle afin de trouver la meilleure configuration d'architecture d'un drone marin.

***Contexte et problème scientifique***

En Ingénierie Dirigée par les Modèles (IDM), un des objectifs est de produire des logiciels par la généralisation de l'utilisation et de la manipulation de modèles. Ces modèles peuvent être des modèles généralistes de type UML ou définis par des langages dédiés (*Domain Specific Language* ou DSL).

Les modèles exécutables sont des modèles qui intègrent la définition du comportement d'un système, c'est-à-dire concrètement quand et comment appeler des opérations métier [1]. Par exemple, une machine à états UML peut modéliser le comportement d'un ascenseur, les états et transitions étant associés à l'appel des fonctionnalités métier de l'ascenseur : monter ou descendre d'un étage, ouvrir ou fermer les portes de l'ascenseur… Ainsi la machine à état UML permet de faire effectivement bouger l'ascenseur. De même, une orchestration BPMN permettra de définir le parcours d'un client sur un site Web pour réserver des vacances en appelant tel ou tel service Web ou en faisant des requêtes sur des bases de données en fonction des choix du client. Le modèle exécutable devient alors un élément de programmation du système visé.

Les modèles exécutables sont intéressants d'un point de vue génie logiciel. Ils permettent de définir à un haut-niveau d'abstraction le comportement d'un système logiciel, ce qui le rend plus compréhensible. Ils peuvent être définis pendant la phase de spécification et réutilisés à l'identique à l'exécution dans le système final si on embarque leur moteur d'exécution. Ce moteur peut être utilisé pour simuler le modèle pendant sa spécification : si le modèle est validé dès cette phase, on sait alors que le comportement du système final sera correct. De même, il n'est plus utile de coder « à la main » le comportement du système puisqu'il est réifié dans le modèle exécutable, ce qui raccourcit les temps de développement et les sources d'erreurs.

Pour que le modèle exécutable permette de construire un système logiciel fonctionnel ou simulable, il faut associer des opérations métiers à ses éléments exécutables (états, activités, transitions…). Des spécifications récentes de l'OMG, comme fUML [2], permettent de modéliser l'équivalent de code abstrait exécutable via notamment une syntaxe textuelle. Ainsi, on peut obtenir un modèle exécutable complet, avec son comportement et son code métier défini en fUML. Néanmoins, nous considérons que cette approche est difficile à utiliser en pratique. Les ingénieurs devront apprendre un nouveau langage de programmation « abstrait » et quid de la capacité à écrire en fUML du code technique (par exemple une transaction JPA en Java ou le traitement d'un fichier JSON retourné par un service REST) ? Le problème de la capacité à intégrer du code existant (*legacy*) ou sur étagère dans des librairies se pose également. Pour toutes ces raisons, nous proposons une approche plus pragmatique où le comportement du système est défini dans un modèle exécutable et les opérations métier sont écrites dans un langage de programmation standard comme Java. Cela a été appliqué avec PauWare pour les machines à états UML [3, 4] et avec XmodelingStudio [5] qui est un plugin EMF pour tisser des opérations métier sur des modèles de DSL exécutables (appelés xDSL).

Si ces travaux ont montré la faisabilité de cette approche pragmatique de l'utilisation des xDSL, ils ont aussi montré des limites. Avec un langage de programmation classique, les méthodes s'appellent les unes les autres en se passant des paramètres : il y a un flot de contrôle sur lequel se tisse un flot de données. Avec un modèle exécutable, le flot de contrôle est intégralement défini par le modèle qui est totalement découplé du flot de données. Les opérations métier sont appelées par invocation dynamique par le moteur d'exécution du xDSL, sans contrôle de type ou de gestion des exceptions qui peuvent être levées. Si des solutions techniques ont été trouvées pour exprimer par exemple qu'une valeur retournée par une méthode sera le paramètre d'une autre, elles sont actuellement trop limitées d'un point de vue génie logiciel. Concrètement, rien n'est vérifiable à l'écriture du code métier ou de la définition du modèle, il faut exécuter le système pour détecter des erreurs.

***Contributions attendues en génie logiciel***

Le premier objectif de la thèse sera de travailler plus en détail sur le tissage des opérations métier sur des éléments exécutables d'un xDSL afin de pouvoir spécifier du typage précis lors de la définition et l'appel des méthodes ou de pouvoir gérer les exceptions levées par les appels de ces méthodes. Ce typage pourra se faire directement au niveau du modèle et on vérifiera qu'il existe du code compatible ou bien à l'inverse remonté à partir d'un code existant qui contiendra par exemple des annotations Java pour préciser certaines informations sur les opérations métier. Ce typage et ces annotations pourront également intégrer des contrats associés aux opérations métier pour faciliter et mieux maîtriser leurs liens avec les éléments exécutables. Ces fonctionnalités de typage et de gestion des erreurs lors de l'exécution du modèle seront implémentées sous la forme d'un plugin pour l'environnement EMF pour aider les ingénieurs logiciels à lier un modèle exécutable à ses opérations métier en leur offrant un environnement mixte de modélisation et de programmation.

Au niveau vérification, il faudra également pouvoir associer de manière générique des outils de model checking à un modèle exécutable (comme OBP2 [6] développé dans l'équipe P4S) pour valider le

comportement du système dès sa spécification. Le plugin EMF développé intégrera la possibilité de connexion avec des outils de vérification.

L'autre objectif de la thèse sera de faire de la rétro-ingénierie de code Java pour en déterminer des modèles ou des bouts de modèles exécutables par analyse de code. On pourra se baser là aussi sur les annotations placées dans le code. Un catalogue de patterns d'exécution pourra être déterminé après étude d'un ensemble de projets Java et permettra d'enrichir l'environnement de modélisation en proposant des patterns de modèles exécutables que l'on pourra intégrer dans la définition d'un xDSL. L'autre objectif de cette étude de codes Java est de déterminer quels sont les bons niveaux de granularité pour définir une opération métier. Il s'agit en effet de ne pas forcément considérer toute méthode de toute classe métier comme une opération liable à un élément exécutable d'un modèle mais d'avoir potentiellement des opérations de plus haut niveau.

***Application et contribution aux simulateurs de drones marins***

Comme cas d'étude et de validation, les résultats de la thèse seront appliqués à la construction de simulateurs de drones marins. Pour cela, ils seront notamment intégrés dans la plateforme CARES [7] développée par l'équipe SHAKER.

CARES est un environnement de spécification et de simulation d'architectures de drones. Plusieurs DSL y ont été définis pour spécifier des composants d'un drone marin ou configurer une simulation d'une architecture de composants. Ces composants sont reliés manuellement à des opérations métier, souvent livrées sur étagères, qui permettent de simuler un élément du drone ou l'environnement du drone (le vent ou le courant par exemple dans le contexte d'un drone marin). Les composants s'échangent également des flux de données qui sont typées. Un modèle exécutable obtenu permet de simuler une configuration d'architecture de composants mais il manque actuellement l'automatisation du lien modèle/code métier associé à une vérification de typage entre le modèle et les opérations métier. Les contributions de la thèse pourront donc directement s'appliquer à résoudre ces problèmes.

Ensuite, il est nécessaire de pouvoir lancer plusieurs simulations d'une même architecture avec différents paramètres ou différentes sémantiques d'exécution des composants pour trouver la meilleure configuration. Pour gérer les paramètres variants de simulation, ils pourront être rajouté dans une extension des DSL de Cares. Pour les variantes de sémantiques, un modèle exécutable de plus haut niveau (défini dans un nouveau xDSL) pourra réifier le comportement du moteur de simulation de CARES en permettant de combiner facilement les variantes de sémantiques d'exécution en changeant l'orchestration des opérations de simulation.

La thèse pourra également porter plus loin que la simulation de drones pour s'intéresser aux jumeaux numériques. Un jumeau numérique n'est pas qu'un simple simulateur mais est connecté au système réel. Par exemple, le jumeau numérique récupère des données générées au *runtime* par le drone physique et le jumeau numérique adapte son comportement en fonction de ces retours. En ce sens, on retrouvera des problématiques qui ont déjà été étudiées dans des travaux précédents sur l'adaptation des modèles exécutables [8] qui ici sont ceux du simulateur de CARES. Les solutions proposées dans cette thèse pourront donc être une première réflexion au problème de la définition d'un jumeau numérique lié à un drone marin physique.

**Publications récentes de l'équipe dans le domaine et références :**


1. Eric Cariou, Olivier Le Goaer, and Franck Barbier, *On the Executable Nature of Models*, 2nd International Workshop on Executable Modeling at MoDELS (EXE 2016), CEUR Workshop Proceedings, vol. 1760, 2016

2. OMG, *Semantics of a Foundational Subset for Executable UML Models (fUML) Specification*, version 1.4, 2018

3. Franck Barbier and Eric Cariou, *Executable Modeling for Reactive Programming*, Model-Driven Engineering and Software Development. MODELSWARD 2018, volume 991 of CCIS, Springer, 2019

4. Eric Cariou, Léa Brunschwig, Olivier Le Goaer, and Franck Barbier, *A software development process based on UML state machines*, The 4th Edition of the International Conference on Advanced Aspects of Software Engineering (ICAASE'20), IEEE, 2020

5. Eric Cariou, Olivier Le Goaer, Léa Brunschwig and Franck Barbier, *A generic solution for weaving business code into executable models*, 4th International Workshop on Executable Modeling at MoDELS (EXE 2018), CEUR Workshop Proceedings, vol. 2245, 2018

6. OBP2 : Modular Verification Meets xDSL Dynamics. http://www.obpcdl.org/

7. Loïc Salmon, Pierre-Yves Pillain, Goulven Guillou and Jean-Philippe Babau, *CARES, a framework for CPS simulation : application to autonomous underwater vehicle navigation function*, Forum on specification & Design Languages (FDL), IEEE, 2021

8. Franck Barbier, Eric Cariou, Olivier Le Goaer, and Samson Pierre, *Software Adaptation: Classification and a Case Study with State Chart XML*, IEEE Software, 32(5), Sept. Oct. 2015