

Implantation d'outils dans l'environnement Co²

Laboratoire I3S

encadreur : Pascal Rapicault

Luc Bourlier

14 septembre 2001

Table des matières

1	Introduction	2
2	Description du travail proposé	3
2.1	La notation SyncClass	4
	Exemple	5
2.2	Les outils	6
	Générateur de code	7
	Générateur de SyncClass	7
	Générateur de tests	7
	Finaliseur	8
	Outil de vérification dynamique	8
3	Description du travail réalisé	9
3.1	Structures de données	9
3.2	Générateur de code	11
3.3	Générateur de SyncClass	11
3.4	Finaliseur	12
3.5	Outil de vérification dynamique	12
3.6	Déroulement du stage	13
4	Travail à venir	14
5	Conclusion	15

Chapitre 1

Introduction

Dans le cadre de la formation du cycle ingénieur à l'École Supérieure en Sciences Informatiques (ESSI), j'ai effectué mon stage de fin d'étude au laboratoire I3S (UNSA / CNRS) de Sophia Antipolis, au sein de l'équipe RAINBOW.

Dans le cadre de sa thèse [Rap02], P. Rapicault a défini une notation graphique pour la description des composants, les SyncClass. Autour de cette notation, il a décrit un ensemble d'outils formant l'environnement Co², dont le but est d'aider les développeurs pour la spécification, le développement et l'utilisation de composants. L'objectif de mon stage était la conception et l'implantation d'une partie des outils de cet environnement.

Ce document présente la description du travail qui m'a été demandé ainsi que le travail que j'ai effectué, en précisant l'intérêt de ce projet dans le cadre de ma formation.

Je tiens à remercier les personnes du laboratoire que j'ai côtoyées durant mon stage, plus spécialement les membres de l'équipe RAINBOW et mon responsable de stage, Pascal Rapicault. J'ai spécialement apprécié l'aide qu'ils m'ont apportée ainsi que leur bonne humeur et l'ambiance de travail.

Chapitre 2

Description du travail proposé

Le but final du projet est de concevoir et développer les outils de l'environnement Co² [RBR01]. L'ensemble des outils de cet environnement sont représentés dans la figure 2.1.

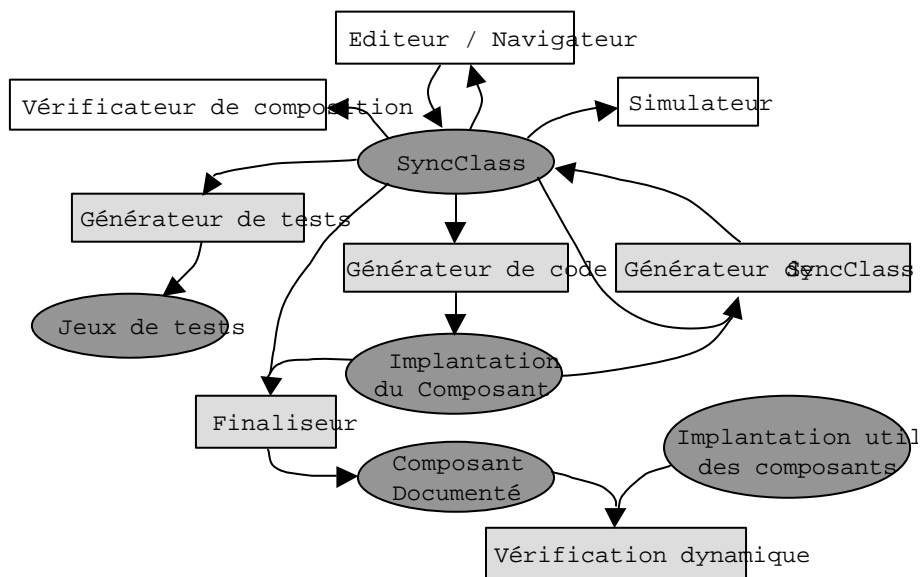


FIG. 2.1 – Organisation des outils de Co². Les ellipses grisées représentent les données manipulées par les outils (rectangles). Les rectangles grisés sont les outils sur lesquels portait le stage.

Ces outils ont pour but de permettre aux développeurs et aux utilisateurs de composant d'utiliser la notation SyncClass au cours du cycle de vie du composant.

La notation SyncClass associée à ces outils a été définie par P. Rapicault dans le cadre de sa thèse. La partie suivante présente de cette notation. Les outils qu'il m'a été demandé d'implanter sont ensuite décrits.

2.1 La notation SyncClass

La notation SyncClass est une notation graphique définie pour représenter le comportement des composants. On distingue deux points de vue pour représenter ce comportement, le point de vue client et le point de vue de composition.

Le point de vue client décrit le protocole d'un composant. Il représente les séquences d'appels de méthodes que le composant considère comme correctes, et la logique de chaque méthode du protocole par rapport aux autres composants. Cette logique, appelée action de la transition, correspond à la liste des appels de méthodes sur composants associés. Ils sont exprimés dans un langage permettant de représenter les structures de programmation (séquence, condition, boucle, exception).

Le point de vue de composition est une vue synthétique de l'utilisation des autres composants par le composant considéré. Il décrit les séquences d'appels de méthode qu'effectue un composant sur un autre composant avec lequel il est associé. De plus il indique les «callbacks» qu'effectuent le composant associé.

Les SyncClass servent à représenter ces différents points de vue.

Les SyncClass sont une extension des SyncCharts [And96]. Les SyncCharts sont eux même sont une représentation graphique du langage Esterel [Ber00]. Les SyncCharts sont utilisés pour représenter graphiquement les systèmes réactifs¹. Les SyncClass sont traduisibles en SyncCharts à l'aide de règles de transformation. Les SyncCharts, étant une représentation graphique du langage Esterel, sont eux-même traduisible dans ce langage. L'intérêt de cette suite de notation est la possibilité d'utiliser la famille d'outils d'Esterel avec les SyncClass.

¹Un Système réactif est un système qui interagit avec son environnement, au rythme imposé par celui-ci

Exemple

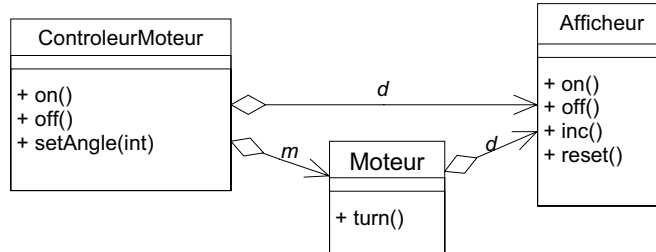


FIG. 2.2 – Diagramme de classe d’un contrôleur de moteur

Les figures 2.3 et 2.4 sont des exemples de la description d’un système à plusieurs composants dans la notation SyncClass. L’exemple choisi est un contrôleur de moteur à précision angulaire avec afficheur (figure 2.2). L’utilisateur indique au contrôleur l’angle dans lequel il veut placer l’axe du moteur, le contrôleur demande au moteur d’effectuer autant de mini-rotations que nécessaire pour que l’axe se place dans la bonne position, l’afficheur indique la position courante (l’angle par rapport à la position initiale). Lors de l’arrêt du système ou en cas de problème l’axe du moteur est placé à sa position initiale.

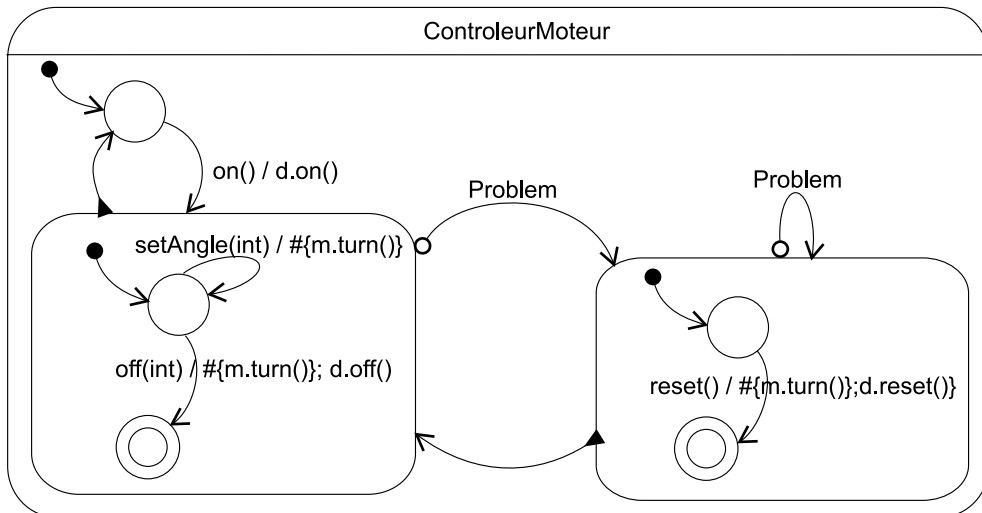
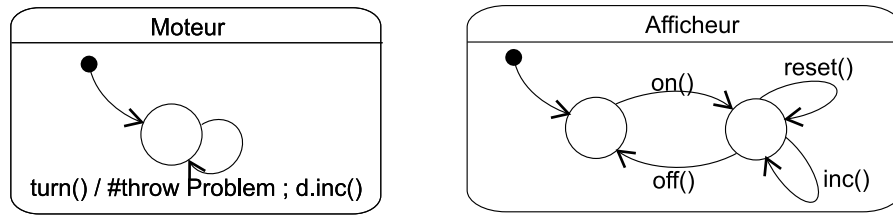


FIG. 2.3 – Point de vue client du contrôleur



(a) Point de vue client du moteur

(b) Point de vue client de l'afficheur

FIG. 2.4 – Points de vue client

Le SyncClass du point de vue client de l’Afficheur (figure 2.4(b)) représente le protocole qu’il suit. L’utilisation normale de ce composant commence par l’appel de la méthode *on()* pour initialiser l’afficheur, ensuite des appels à *inc()* pour incrémenter la valeur affichée ou à *reset()* pour remettre le compteur à zéro, pour arrêter l’afficheur, la méthode *off()* est appelée.

Sur le SyncClass du point de vue client du ContrôleurMoteur (figure 2.3), on voit aussi la description de l’action associée à chaque méthode du protocole. L’exécution de la méthode *off()* entraîne l’appel de la méthode *turn()* sur Moteur un nombre indéfini de fois (0 - n) puis l’appel de la méthode *off()* sur Afficheur.

Les SyncClass sont une notation qui permet de décrire précisément le comportement des composants. Cependant pour tirer pleinement bénéfice de cette notation, il est nécessaire de posséder des outils la gérant. Ainsi l’ensemble des outils couvre complètement le cycle de vie d’un composant, de l’analyse à la réutilisation.

2.2 Les outils

Dans le cadre de ce stage, mon objectif était la conception et le développement des outils ‘Générateur de code’, ‘Générateur de SyncClass’, ‘Générateur de test’, ‘Vérification dynamique’ et ‘Finaliseur’ (figure 2.1). L’implantation de ces outils était à effectuer en Java, pour un soucis de portabilité, de bibliothèques existantes et parce que ce langage est largement utilisé dans le domaine de la recherche. Cette section présente ces différents outils.

Générateur de code

Les SyncClass décrivent le comportement d'un composant. A partir des informations qu'ils contiennent, il est possible de générer un squelette du code du composant. L'outil a pour but de générer ce squelette du code du composant (la déclaration de chaque méthode du composant ainsi que le squelette des appels correspondant aux actions de chacune de ces méthodes) en fonction du SyncClass de son point de vue client.

Générateur de SyncClass

Cet outil doit permettre de compléter un SyncClass en étudiant le code de l'implantation du composant. A partir du SyncClass du point de vue client incomplet (qui contient au minimum son protocole mais pas les actions des transitions) et du code de l'implantation du composant, cet outil devra générer les actions des transitions en fonction de l'implantation de chaque méthode pour recréer le SyncClass complet.

Cet outil doit aussi permettre d'effectuer une vérification statique de l'implantation du composant. Cette vérification s'effectuera en comparant les actions des transitions contenues dans un SyncClass du point de vue client et celles générées par l'outil à partir de l'implantation. Cette vérification doit permettre de montrer, qu'au niveau des appels de méthodes sur les autres composants, l'implantation correspond aux informations contenues dans les SyncClass.

Ces deux outils sont utilisés en phase d'implantation.

Générateur de tests

Une fois l'implantation finie, il est important de tester le composant. Cet outil a pour but de générer des tests pour la phase test.

La partie vérification de l'outil précédent vérifie uniquement que l'implantation des méthodes correspond aux actions des transitions du SyncClass du point de vue client. Cet outil permet la vérification du protocole du composant. Ainsi, à partir d'un SyncClass il génère des tests mettant à l'épreuve le composant. Ces tests consistent à des séquences d'appels corrects suivant le protocole. Cet outil permet de vérifier le bon fonctionnement du composant lorsqu'il est utilisé correctement.

Finaliseur

La notation SyncClass a été définie pour que les développeurs et les utilisateurs de composant disposent d'informations précises. Jusqu'ici nous n'avons parlé que d'outil pour les développeurs de composant. Nous défendons l'approche qu'un composant est l'ensemble constitué de l'implantation de ce composant et du code le décrivant. Le finaliseur doit permettre au développeur d'intégrer les SyncClass dans la distribution du composant pour que ces SyncClass soit aussi accessibles pour les utilisateurs. Le finaliseur crée un composant "complet", contenant l'implantation et la documentation.

Outil de vérification dynamique

Lorsqu'un composant a été implanté, vérifié et finalisé, il est distribué aux utilisateurs. Les SyncClass qui documentent ce composant permettent aux utilisateurs de savoir comment le composant doit être utilisé. L'utilisateur du composant effectue son implantation en fonction de cette description.

Une fois l'implantation fini, cet outil permet à l'utilisateur de vérifier que l'application qu'il a implantée utilise correctement les composants. L'objectif de cet outil est d'effectuer de manière dynamique cette vérification, pendant l'exécution de l'application. L'outil compare l'utilisation qui est faite des composants avec les informations contenues dans leurs SyncCharts.

Chapitre 3

Description du travail réalisé

Cette partie décrit le travail réalisé pendant la durée du stage, la conception et le développement des outils décrits dans la section 2.2. Ce chapitre présente d'abord les structures de données utilisées par ces outils puis les spécificités de chaque outil.

3.1 Structures de données

Pour utiliser la notation SyncClass dans les différents outils, il était nécessaire de pouvoir la manipuler dans un programme. Pour cela, un ensemble de structures de données a été défini. Pour assurer la compatibilité avec les outils existants, des traducteurs permettant de passer entre les structures définies et vers d'autres structures de données (ou langages) ont été définis.

SyncCharts

Les SyncCharts étant une notation utilisée dans le domaine des systèmes réactifs, l'intention première était d'utiliser la structure existante et, comme les SyncClass sont une extension de cette notation, étendre cette structure pour les SyncClass. Malheureusement, cette structure n'était pas utilisable pour des programmes écrits en Java (structure en code Tcl/Tk). Une structure représentant les SyncCharts a donc été conçue et implantée en Java. L'éditeur existant permettait de générer le code Esterel correspondant à un SyncCharts, pour pouvoir utiliser les outils associés à ce langage. Ces outils

étant utiles pour la suite, et comme nous avons définis une nouvelle structure de données, il a été nécessaire d'implanter la traduction d'un SyncCharts représenté selon notre structure vers le code Esterel correspondant.

Pour effectuer cette partie, il m'a été nécessaire d'étudier les systèmes réactifs, domaine d'où provient la notation SyncCharts, ainsi que le langage Esterel.

SyncClass

Possédant une structure pour représenter les SyncCharts, il fallait posséder une structure permettant de représenter les SyncClass. Cette structure de données qui a été définie se base sur celle des SyncCharts avec en principale extension, une structure spécifique pour représenter les actions associées au transition. Cette structure spécifique pour les actions est nécessaire car la partie droite des transitions ne représente pas la même sémantique.

C'est cette différence de sémantique qui a rendu nécessaire l'implantation d'une traduction des SyncClass vers les SyncCharts. Cette traduction est utile pour utiliser la famille d'outil d'Esterel sur les SyncClass (SyncClass \rightarrow SyncCharts \rightarrow Esterel).

Transformations structure / XML

Bien que le but final soit la création d'un environnement complet de développement pour les composants basé sur les SyncClass, l'implantation de l'éditeur / navigateur, permettant une édition graphique des SyncClass n'est pas apparue comme une priorité (rapport temps de développement / intérêt de l'outil). Il était malgré tout nécessaire, au moins pour la mise au point des outils, de pouvoir créer et manipuler des SyncClass. Pour cela une structure de données XML [W3C00] a été défini ainsi que le XML Schema [W3C01] décrivant cette structure. Pour les mêmes raisons, une structure de données XML représentant les SyncCharts et le XML Schema correspondant ont été définis. Des outils permettant de passer des données XML aux structures de données des notations ont été implantés.

Les structures de données étant définies, la suite du chapitre présente les outils et leur implantations.

3.2 Générateur de code

Le générateur de code permet de créer un squelette du code du composant en fonction du SyncClass de son point de vue client. Pour cela, un parcourt du SyncClass est effectué de manière à récupérer la déclaration de chacune des méthodes du protocole, ainsi que les actions associées. À partir de cette liste, un squelette du code du composant est créé, en générant pour chaque méthode, sa déclaration et le code correspondant à son action. Pour le code correspondant aux actions, la génération s'effectue de la même manière qu'un compilateur génère le code exécutable, l'arbre correspondant à l'action est parcouru et les structures Java correspondantes sont générées au fur et à mesure.

Pour exemple, le code généré pour le composant du moteur (figure 2.4(a)) serait :

```
public class Moteur {
    public /*return_type*/ turn() {
        if (/*condition*/) {
            throw new Problem();
        }
        d.inc();
    }
}
```

3.3 Générateur de SyncClass

La partie complétion du générateur de SyncClass recrée les actions associées aux méthodes du protocole à partir de l'implantation du composant. L'objectif est donc de passer du code Java de chaque méthode à son équivalent sous forme d'action de transition. Pour effectuer cela, la partie avant d'un compilateur est utilisée : la partie qui parse le code et qui génère l'arbre syntaxique correspondant. En parcourant cet arbre, on récupère les appels de méthode effectués sur les autres composants, ainsi que les structures de contrôle utilisées (condition, boucle, ...) ce qui permet de générer les actions associées à chaque méthode. Une fois ces actions de transition générées, elles sont insérées à leur place dans SyncClass.

La seconde partie de cet outil est un vérificateur de l'implantation. La technique utilisée pour effectuer cette opération est à la base la même que pour

la complétion, les actions associés à chaque méthode sont extraites de l'implantation. Mais ensuite, ces actions ne sont pas insérées dans le SyncClass mais comparées à celle que le SyncClass contient déjà. Cette comparaison est permise par la structure de données décrivant les actions des transitions. Si les actions sont équivalentes, cela indique que l'implantation du composant correspond aux informations contenues dans ces SyncClass, au niveau de ces relations avec les autres composants.

Cet outil et le précédent permettent d'effectuer du "roundtrip engineering" pour conserver la correspondance entre l'implantation du composant et sa spécification.

Le développement de ces deux outils m'ont permis d'utiliser les techniques de compilation pour passer d'une structure de données au code cible et récupérer les appels de méthodes sur les autres composants.

3.4 Finaliseur

Cet outil ne possède pas de point technique particulier à développer. Par contre, son implantation m'a obligé à étudier quelque type de composant (classe simple, JavaBeans, EJB, ...) pour déterminer la manière la plus pratique et la plus simple d'intégrer les SyncClass à un composant. Pour une classe simple, par exemple, la solution retenue consiste à l'ajout des SyncClass dans les attributs utilisateur (user attributes), pour un EJB, dans les fichiers XML de description qui lui sont associés.

3.5 Outil de vérification dynamique

Cet outil a pour but de tester que le composant est utilisé correctement (le protocole est respecté). La technique utilisée est une vérification dynamique, c'est à dire que la bonne utilisation du composant est testée lors de l'exécution de l'application. Pour cela, on utilise les techniques de méta-programmation et d'instrumentation de code. Au chargement, les composants sont modifiés pour permettre la vérification. Ces modifications permettent que les appels de méthodes sur les composants soient attrapés et le système vérifie que ces appels de méthode sont possibles dans le contexte. Si cet appel est correct, la méthode demandée est exécutée, sinon une exception indiquant le problème est levée. De cette manière, on montre que dans les cas testés, l'utilisation du composant est correcte. Cette vérification peut aussi

être utile pour la mise au point de l'application, elle permet de détecter à quel moment le protocole n'est pas respecté.

3.6 Déroulement du stage

Le stage a débuté par le développement de l'outil de vérification dynamique puis par l'étude de différents types de composants pour la conception du finaliseur. Le développement des structures de données n'a pu être effectué dès le début du stage car nous attendions des informations de la part de l'équipe développant les SyncCharts. Après avoir obtenu ces informations, il nous est apparu nécessaire de définir des structures de données utilisable par des programmes Java (ce qui n'était pas le cas des structures existantes). Cela effectué, j'ai implanté les traductions entre les différents type de données. A partir de ce moment, il a été possible de créer les outils générateur de code et générateur de SyncClass. Le stage s'est terminé par la modification de la vérification dynamique pour prendre en compte les structures de données créées et l'implantation du finaliseur.

Au niveau de la gestion du projet, des réunions étaient effectuées tous les quinze jours. Au cours de ces réunions, nous faisons un point sur ce qui avait été développé puis mettions à jour le calendrier pour les quinze jours suivant. De plus, nous faisons un point quotidien sur ce l'état d'avancement du projet.

Chapitre 4

Travail à venir

L'objectif final de ce projet est de créer l'environnement Co2. Dans ce but, il est prévu de développer une application intégrant cet ensemble d'outils, qui sera elle-même intégrée dans l'interface de développement Eclipse¹.

Pour cela les autres outils devront être développés pour créer un produit consistant.

Comme vous l'avez vu, il existe une différence entre le travail proposé et le travail effectivement réalisé, l'outil générateur de test n'a pas été implanté. Cela est en partie dû au fait qu'il a fallu effectuer la conception et l'implantation de la structure représentant les SyncCharts, ainsi que la traduction SyncCharts vers Esterel. Ce développement n'était pas prévu au début du stage et a utilisé le temps qui aurait permis d'effectuer l'implantation du générateur de test.

Je vais continuer de travailler sur ce projet et effectuer le développement des outils générateur de tests et vérificateur de composition. Ce dernier outil a pour but de tester si un ensemble de composants peuvent être utilisés ensemble, sont compatibles entre eux. Cette vérification doit s'effectuer à partir des SyncClass de chaque composant.

Les outils Éditeur / Navigateur et Simulateur font l'objet d'une proposition de sujet pour un projet pour des étudiants en ESSI 3.

¹www.eclipse.org

Chapitre 5

Conclusion

Ce stage à été positif sur plusieurs points dans le cadre de ma formation. Tout d'abord, le projet sur lequel il reposait m'a permis d'étudier et de mettre en oeuvre des techniques issues de différents domaines : compilation, méta-programmation, instrumentation de code, systèmes réactifs. Le poste que j'occupais était aussi nouveau pour moi, j'étais l'équivalent d'un ingénieur de recherche. Le travail n'est pas le même que dans l'industrie, les spécifications sont plus changeantes, en fonction des résultats obtenus, et il est nécessaire de s'intéresser aux travaux des autres équipes pour voir si il s'en rapproche.

Dans le cadre de ce projet, j'ai aussi collaboré à la rédaction d'un article sur l'environnement Co² pour la conférence LMO. Cela est une expérience supplémentaire utile à ma formation, où l'aspect recherche n'est pas très développé

Pour conclure, ce stage m'a permis d'effectuer le travail complet sur un projet : la conception, le développement et la documentation, dans un cadre de recherche.

Bibliographie

- [And96] Charles André. Representation and analysis of reactive behaviors : a synchronous approach. In *CESA*, pages 19–29, july 1996.
- [Ber00] Gérard Berry. The esterel language primer, version v5_91, june 2000.
- [Rap02] Pascal Rapicault. *Modèles et techniques pour spécifier, développer et utiliser un framework : une approche par méta-modélisation*. PhD thesis, I3S, Université de Nice, 2002. En cours de rédaction.
- [RBR01] Pascal Rapicault, Luc Bourlier, and Jean-Paul Rigault. Syncclass et Co², notation et environnement pour la spécification, le développement et l'utilisation de composant. 2001. Soumis.
- [W3C00] Extensible markup language (xml) 1.0 (second edition). W3C, october 2000. <http://www.w3.org/TR/2000/REC-xml-20001006>.
- [W3C01] Xml schema part 0 : Primer. W3C, may 2001. <http://www.w3.org/TR/2001/REC-xmlschema-0-20010502/>.

Résumé

Dans le cadre de sa thèse, Pascal Rapiçault a défini une notation décrivant le comportement des composants ainsi que les outils d'un environnement de programmation basé sur cette notation, l'environnement Co². Cette notation et cet environnement ont pour but d'aider les développeurs pour la spécification, le développement et l'utilisation de composant. C'est dans ce cadre que j'ai effectué mon stage de fin d'étude, dont l'objectif était l'implantation d'une partie des outils de l'environnement Co². Ce document décrit le travail qui m'a été demandé de réaliser, la réalisation de ce travail et ce qu'il a apporté à ma formation.