UNIVERSITÉ DE ROUEN

Compilateur LLVM

Langage jouet Kawa

Spécification Technique de Besoin 0.1.5

20 novembre 2014

Auteur(s): Kheireddine Berkane, Nasser Adjibi
Relecteur(s): Pierre-Luc BLOT

Version	Date	Changelog
0.1	04/11/2014	Version initiale.
0.1.5	18/11/2014	Définition des cas d'utilisations et des exigences.

Table des matières

1 Objet

LLVM est une infrastructure modulaire permettant la réalisation de chaînes de compilation et conçue pour l'optimisation. Elle met en oeuvre une représentation intermédiaire du code qui permet de découpler les langages de l'architecture. Nore objectif est de réaliser, à l'aide de l'infrastructure LLVM, un comilateur pour un langage jouet, que nous appellerons Kawa, ce dernier doit supporter :

- Les classes, les classes abstraites, les interfaces
- L'héritage
- Le polymorphisme
- Le système de types sera composé des types primitifs (int, float, etc.), des classes et des interfaces
- Les instructions de contrôle telles ques (if/else, for, while/do, switch, etc.).
- Les méthodes seront définies de manière identique à Java execpté que les paramètres pourront être préfixés du mot clé VALUE (transmission par valeur au lieu de référence)

1.1 Besoins opérationnels

1.2 Objectifs techniques

1.3 Contraintes et recommendations

1.4 Résultats attendus

2 Documents applicables et de référence

Différents documents de référence :

- Le site LLVM llvm.org.
- Le document de spécification du client Spec1.pdf

3 Terminologie et sigles utilisés

- LLVM : (Low Level Virtual Machine) est une infrastructure de compilateur conçue optimisation à la compilation.
- Kawa : langage jouet qui reprend quelques fonctionnalités de java.
- Clang : compilateur pour le langage c++, son interface de bas niveau se base sur des bibliothèques llvm pour la compilation. Il est utilisé par APPELE.
- GIT : logiciel qui permet de stocker un ensemble de fichiers en conservant la chronologie de toutes les modifications qui ont été effectuées dessus.
- ELF : (Executable and Linkable Format) est un format de fichier binaire utilisé pour l'enregistrement de code compilé (objets, exécutables, bibliothèques de fonctions).
- Ubuntu : Distribution linux sur base Debian.
- C++ : Langage de programmation orienté objet bas niveau.
- POO : Programmation orientée objet.
- Makefile : Fichier regroupant des instructions de compilation avec gestion de dépendances.

4 Exigences fonctionnelles

4.1 Présentation de la mission du produit logiciel

Id	Intitulé	Acteur(s)	Priorité
EF_1	Afficher l'aide	Utilisateur	Indispensable
EF_2	Compiler une application en mode monolithique	Utilisateur	Indispensable
EF_3	Compiler une application en mode partagé	Utilisateur	Secondaire
EF_4	Chosir un nom pour l'exécutable	Utilisateur	Important
EF_5	Compiler des bibliothèques	Utilisateur	Important
EF_6	Afficher la version du compilateur	Utilisateur	Important
EF_7	Définir des dépendances entre sources et classes	Utilisateur	Important
EF_8	Activer l'affichage en couleur	Utilisateur	Important

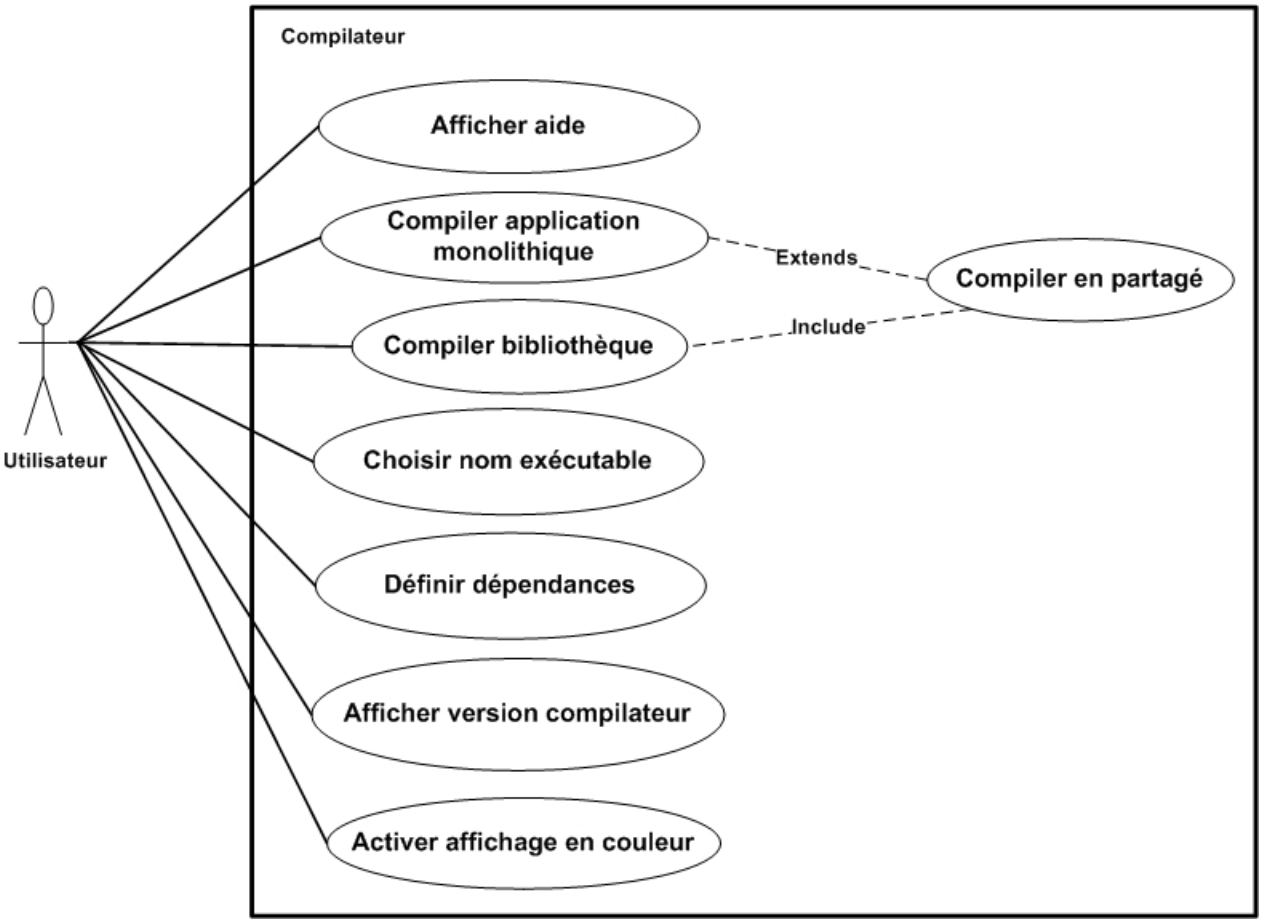

FIGURE 1 – **Cas d'utilisations du compilateur kawa.**

4.2 Cas d'utilisation EF_1

Nom	**Afficher l'aide**	
Acteurs concernés	Utilisateur du compilateur	
Description	le compilateur affiche la liste des options du compilateur sur la sortie standard à travers une ligne de commande.	
Préconditions		
Evénements déclenchants	Ligne de commande.	
Conditions d'arrêt	Action utilisateur	
Description du flot d'événements principal:		
Acteur(s)		**Système**
Flots d'exceptions:		

4.3 Cas d'utilisation EF_2

Nom	Compiler une application en mode monolithique	
Acteurs concernés	Utilisateur du compilateur	
Description	L'utilisateur introduit un ensemble de classes kawa afin de les pré-compiler et de générer le tous dans un seul exécutable.	
Préconditions	Ensemble de fichiers sources respectant la syntaxe du langage kawa.	
Evénements déclenchants	Ligne de commande.	
Conditions d'arrêt	Erreur (syntaxe, fichier introuvable, ..), ou succès de la compilation	
Description du flot d'événements principal:		
Acteur(s)		**Système**
Flots d'exceptions:	Abondant provoqué par l'utilisateur.	

4.4 Cas d'utilisation EF_3

Nom	Compiler une application en mode partagé	
Acteurs concernés	Utilisateur du compilateur	
Description	L'utilisateur introduit un ensemble de sources kawa ainsi des exécutables qui auraient été déjà compiler ailleurs,afin de compiler un programme kawa.	
Préconditions	Ensemble de fichiers source d'interfaces respectant la syntaxe du langage kawa.	
Evénements déclenchants	Ligne de commande.	
Conditions d'arrêt	Erreur (syntaxe, fichier introuvable, ..), ou succès de la compilation	
Description du flot d'événements principal:		
Acteur(s)		**Système**
Flots d'exceptions:	Abondant provoqué par l'utilisateur.	

4.5 Cas d'utilisation EF_4

Nom	Chosir un nom pour l'exécutable	
Acteurs concernés	Utilisateur du compilateur	
Description	L'utilisateur peut choisir le nom d'exécutable à produire via un ligne de commande.	
Préconditions		
Evénements déclenchants	Ligne de commande.	
Conditions d'arrêt	Action utilisateur.	
Description du flot d'événements principal:		
Acteur(s)		**Système**
Flots d'exceptions:		

4.6 Cas d'utilisation EF_5

Nom	**Compiler des bibliothèques**	
Acteurs concernés	Utilisateur du compilateur	
Description	l'utilisateur introduit l'ensemble des sources de la bibliothèque ce qui implique une compilation en mode partagé.Si le client ne précise pas le nom de l'exécutable, le nom est repris à partir du fichier contenant la méthode main.	
Préconditions		
Evénements déclenchants	Ligne de commande.	
Conditions d'arrêt	Erreur (syntaxe, fichier introuvable, ..), ou succès de la compilation.	
Description du flot d'événements principal:		
Acteur(s)		**Système**
Flots d'exceptions:	Abondant provoqué par l'utilisateur.	

4.7 Cas d'utilisation EF_6

Nom	**Afficher la version du compilateur**	
Acteurs concernés	Utilisateur du compilateur	
Description	L'utilisateur peut savoir la version du compilateur avec le quel compile ses sources et ses bibliothèques grâce à une ligne de commande.	
Préconditions		
Evénements déclenchants	Ligne de commande.	
Conditions d'arrêt	Action utilisateur.	
Description du flot d'événements principal:		
Acteur(s)		**Système**
Flots d'exceptions:		

4.8 Cas d'utilisation EF_7

Nom	**Définir des dépendances entre sources et classes**	
Acteurs concernés	Utilisateur du compilateur	
Description	L'utilisateur peut définir les dépendances utiles pour la compilation entre les sources et des fichiers déjà compilés	
Préconditions		
Evénements déclenchants	Ligne de commande.	
Conditions d'arrêt	Action utilisateur.	
Description du flot d'événements principal:		
Acteur(s)		**Système**
Flots d'exceptions:		

4.9 Cas d'utilisation EF_8

Nom	Activer l'affichage en couleur	
Acteurs concernés	Utilisateur du compilateur	
Description	L'utilisateur peut activer l'option de l'affichage en couleur, afin de décorer les messages renvoyés par le compilateur.	
Préconditions		
Evénements déclenchants	Ligne de commande.	
Conditions d'arrêt	Action utilisateur.	
Description du flot d'événements principal:		
Acteur(s)		**Système**
Flots d'exceptions:		

5 Exigences opérationnelles

Id	Intitulé	Priorité

6 Exigences opérationnelles d'interface

Id	Intitulé	Priorité

7 Exigences de qualité

Id	Intitulé	Priorité
EQ_1	les messages d'erreurs renvoyés par le compilateur doivent être explicites, les plus fines possibles et surtout par rapport à ce qu'il s'est passé.	Important

8 Exigences de réalisation

8.1 Intitulés

Id	Intitulé	Acteur(s)	Priorité
EXR_1	Nommage des fichiers de sortie	Utilisateur KAWAC	Secondaire
EXR_2	Reconnaissance de la grammaire KAWA	KAWAC	Indispensable
EXR_3	Compilation d'application en monolithique	KAWAC	Indispensable
EXR_4	Compilation d'application partagée	KAWAC	Secondaire
EXR_5	Compilation de bibliothèque en monolithique	KAWAC	Indispensable
EXR_6	Compilation de bibliothèque partagée	KAWAC	Secondaire
EXR_7	Gestion des mots clés	KAWAC	Indispensable
EXR_8	Fichier de sortie au format ELF	KAWAC	Indispensable
EXR_9	Garbage collector	KAWAC	Indispensable
EXR_10	Reconnaissance et compilation de classes	KAWAC	Indispensable
EXR_11	Reconnaissance et compilation de classes abstraites	KAWAC	Indispensable
EXR_12	Reconnaissance et compilation d'interfaces	KAWAC	Indispensable
EXR_13	Prise en charge du Polymorphisme	KAWAC	Indispensable
EXR_14	Prise en charge de l'héritage de classe	KAWAC	Indispensable
EXR_15	Prise en charge de l'implémentation d'interfaces	KAWAC	Indispensable
EXR_16	Prise en charge de l'héritage de classe	KAWAC	Indispensable
EXR_17	Prise en charge de l'implémentation d'interface	KAWAC	Indispensable
EXR_18	Mécanisme de constructeur	KAWAC & Utilisateur KAWAC	Indispensable
EXR_19	Mécanisme de finalisation	KAWAC & Utilisateur KAWAC	Secondaire
EXR_20	Reconnaissance et définition de méthode	KAWAC & Utilisateur KAWAC	Indispensable
EXR_21	Reconnaissance et définition d'attribut	KAWAC & Utilisateur KAWAC	Indispensable
EXR_22	Reconnaissance et définition des variables locales	KAWAC & Utilisateur KAWAC	Indispensable
EXR_23	Définition d'attributs statiques	Utilisateur KAWAC	Important
EXR_24	Définition d'attributs de constantes	Utilisateur KAWAC	Secondaire
EXR_25	Définition d'attributs ou de variables de type **value**	Utilisateur KAWAC	Important
EXR_26	Définition méthode value	Utilisateur KAWAC	Important
EXR_27	Définition méthode á référence	Utilisateur KAWAC	Important

EXR_28	Définition méthode sans référence	Utilisateur KAWAC	Important
EXR_29	Définition méthode finale	Utilisateur KAWAC	Important
EXR_30	Définition méthode statique	Utilisateur KAWAC	Important
EXR_31	Héritage de méthode	KAWAC	Indispensable
EXR_32	Héritage d'attribut	KAWAC	Indispensable
EXR_33	Redéfinition de méthode	Utilisateur KAWAC	Important
EXR_34	Surcharge des méthode	Utilisateur KAWAC	Important
EXR_35	Mutabilité	KAWAC	Indispensable
EXR_36	Concepte de visibilité	Utilisateur KAWAC	Indispensable
EXR_37	Portée de variable	KAWAC	Important
EXR_38	Paramétrage de méthode	KAWAC	Important
EXR_39	Mécanisme de contrôle	Utilisateur KAWAC	Important
EXR_40	Mécanisme de bouclage	Utilisateur KAWAC	Important
EXR_41	Gestion des exceptions	KAWAC & Utilisateur KAWAC	Important
EXR_42	Bloc de classe	Utilisateur KAWAC	Indispensable
EXR_43	Bloc d'instructions	Utilisateur KAWAC	Indispensable
EXR_44	Bloc de boucle	Utilisateur KAWAC	Important

8.2 Descriptions

Id	Intitulé
EXR_1	On pourra choisir le nom du fichier à l'issue de la compilation.
EXR_2	KAWAC est capable de dire si le code est valide pour la grammaire de KAWA, et émettre des erreurs pour signaler la ligne et un moyen de résolution.
EXR_3	KAWAC pourra compiler des fichiers et fournir un exécutable qui n'a pas besoin de bibliothèques externes pour fonctionner. L'ensemble du code donné en entrée devra fournir une méthode main, qui sera le point d'entrée de l'application.
EXR_4	KAWAC pourra compiler des fichiers et fournir un exécutable. L'exécutable s'il le faut dépendra de ressources externes pour pouvoir fonctionner. Le code en entrée devra fournir une méthode main, qui sera le point d'entrée de l'application.
EXR_5	KAWAC pourra compiler des fichiers et fournir une biblithèque qui dépendra d'aucune bibliothèque externe. Les fichier passés en entrée doivent ne pas contenir de méthode main.
EXR_6	KAWAC pourra compiler des fichiers et fournir une nouvelle bibliothèque. La bibliothèque s'il le faut dépendra de ressources externes pour pouvoir fonctionner. Le code en entrée devra ne pas contenir de methode main.
EXR_7	KAWAC gère une table de mots clé contenant les mots réservés par la spécification de la grammaire de KAWA.
EXR_8	Le compilateur utilisera le format ELF pour effectuer l'édition des liens.
EXR_9	Le programme exécutable intégrera un mécanisme permettant de gérer la mémoire au cours de l'exécution.
EXR_10	KAWAC est capable de reconnaître et compiler des classes écrites en KAWA.
EXR_11	KAWAC est capable de reconnaître et compiler des classes abstraites écrites en KAWA.
EXR_12	KAWAC est capable de reconnaître et compiler des interfaces écrites en KAWA.
EXR_13	KAWA permet d'utiliser des objets en utilsant des réferences de type différents, à la condition que le type statique soit un ancêtre du type dynamique. Seules les méthodes du type statique sont accéssibles.
EXR_14	KAWA autorise la dérivation de classe. KAWA ne gère pas l'héritage multiple de classe.
EXR_15	l'implémentation d'interface : KAWA autorise l'implémentation d'une ou plusieurs interfaces.
EXR_16	KAWA autorise la dérivation de classe. KAWA ne gère pas l'héritage multiple de classe.
EXR_17	KAWA autorise l'implémentation d'une ou plusieurs interfaces.
EXR_18	Chaque objet, pour être instancié, fournit une méthode qui permettra son intanciation. KAWAC fournira un constructeur si un n'est pas définit.
EXR_19	Chaque objet fournit une methode qui sera appelée lors de sa destruction par le garbage collector.
EXR_20	On peut déclarer un bloc d'instructions paramétrable s'exécutant s'il est appelé.
EXR_21	On peut allouer des espaces mémoires représentant les champs de chaque objet.
EXR_22	On peut définir des variables temporaires et de portées limitées. Les variables locales ne sont pas des attributs.
EXR_23	On peut définir des espaces mémoires associés aux classes. Les objets instanciant ou dérivant la classe où a été déclaré l'attribut et si la visibilité le permet, pointeront vers la même adresse pour cet attribut.

EXR_24	Un attribut constant ne peut être modifié après affectation. Il doit avoir été initialisé avant pour être utilisable par une autre opération que l'affectation.
EXR_25	En définissant un attribut avec le mot clé ŕvalue; on aura accès non pas a une réference vers un espace mémoire stockant les données, mais directement a un bloc de données.
EXR_26	La méthode renvoie un bloc de données représentant le resultat.
EXR_27	La méthode renvoie une référence vers un espace mémoire contenant les données de l'objet renvoyé.
EXR_28	La méthode ne retourne rien.
EXR_29	La méthode ne peut être surchargée ou redéfinie.
EXR_30	La méthode peut être accéssible à partir du nom de la classe.
EXR_31	Une classe dérivant une autre copira toute les méthodes déja définit dans l'arborescence de ses ancêtres. Mais ne pourra y acceder que si la visibilité le permet.
EXR_32	Une classe dérivant une autre copira tout les attributs déja definit dans l'arborescenece de ses ancêtres si la visibilité le permet.
EXR_33	Une méthode peut être definie avec le nom d'une autre existence, en renvoyant le meme type de valeur, ainsi que les memes parametres.
EXR_34	Une méthode peut être définit avec le nom d'une autre existence, en renvoyant le meme type de valeur, mais avec des paramètres différents.
EXR_35	Les valeurs stockées dans les attributs et les variables pourront être réaffectées.
EXR_36	A l'aide des mots clés, on peut définir la visibilité des attributs et methodes des classes sur plusieurs niveaux. Privée, pour empêcher l'accès en dehors de la classe. Protégée, pour empêcher l'accès en dehors de la classe, sauf pour les héritiés de la classe ou de l'interface. Publique pour autoriser l'accès à tous.
EXR_37	Une variable n'est effective qu'à l'interieur du bloc à dans lequel elle a été déclarée.
EXR_38	Une méthode accepte des références ou des bloc de données en paramètre et les remplace aux bons endroits dans son bloc d'execution.
EXR_39	KAWA admet des expressions conditionnelles.
EXR_40	KAWA admet des expressions de bouclage.
EXR_41	KAWAC gère une table de mots clé contenant les mots réservés par la spécification de la grammaire de KAWA.
EXR_42	La défintion du corps de l'objet se fera à l'intérieur d'un bloc délimité paramétré. Un bloc de classe ne peut en contenir un autre.
EXR_43	On peut définir une suite d'instructions délimité paramétré. Un bloc d'instruction peut contenir d'autres blocs.
EXR_44	On peut définir une suite d'instructions délimité paramétré. Les instructions contenues dans le bloc sont répétées tant qu'une certaines condtions sont remplies. Un bloc de bouclage peut contenir d'autres blocs.