Haute école d'ingénierie et d'architecture Fribourg
Hochschule für Technik und Architektur Freiburg

Filière Informatique et Systèmes de Communication
Orientation Informatique Logicielle

Cahier des charges
Travail de Bachelor
2025

Modèles numériques alternatifs pour WebAssembly par instrumentation de code

Luca Michel
luca.michel@edu.hefr.ch

Superviseur: Frédéric Bapst

Expert: Baptiste Wicht

Fribourg, 26 mai 2025
Version 0.2.0

Hes·so

Table des versions

Version	Date	Auteur(s)	Modifications
0.2.0	26/05/2025	LM	Révision des objectifs et de la planification
0.1.0	22/05/2025	LM	Version initiale

Table des matières

Chapitre 1
Contexte

WebAssembly (Wasm) est un langage bas niveau similaire à l'assembleur qui permet d'exécuter du code écrit depuis différents langages dans un navigateur web avec des performances similaires aux applications natives [1]. Cette technologie en plein essor est de plus en plus utilisée dans des contextes d'applications web grâce à sa performance et sa portabilité, mais il est également possible de faire fonctionner du code Wasm localement. Dans un contexte web, Wasm fonctionne de pair avec JavaScript (JS) qui permet de charger des modules Wasm au sein d'une application JS. Cela permet de donner accès à la flexibilité de JS et la performance de Wasm [1].

Le code Wasm fonctionne à la manière du code assembleur, et s'appuie sur une machine virtuelle à pile capable d'exécuter du code binaire. L'environnement de cette machine virtuelle est décrit comme efficace, *memory safe*, *sandboxed*,[1] ouvert et *debuggable* [2].

Cependant ce langage, comme beaucoup d'autres, est sensible aux anomalies arithmétiques usuelles. Ces dernières peuvent avoir des conséquences catastrophiques comme par exemple l'*Ariane flight V88* [3] ou la *VM escape via VGA device* [4]. Ces conséquences sont souvent dues à un manque de mesures de détection de ces anomalies.

COJAC [5] vise à détecter et à pallier les anomalies arithmétiques pouvant être rencontrées durant l'exécution d'un programme Java. Cet utilitaire est également capable d'altérer le comportement des opérations instrumentées de diverses manières [5] :

- Ajout de perturbations afin de tester la stabilité des opérations
- Substitution des types numériques standard par des types plus stables[2] via des wrappers
- Numerical sniffer - détection des anomalies arithmétiques
- Numerical profiler - analyse de l'efficacité des opérations arithmétiques telles qu'elles sont écrites et propositions d'altérations pour gagner en performance
- Autodiff - calcul de dérivée

COJAC a inspiré un projet de semestre réalisé par M. Julmy visant à implémenter une *proof of concept* (*POC*) démontrant la possibilité d'implémenter les fonctionnalités de COJAC pour du code Wasm dans un contexte Web.

Compte tenu de la présence grandissante de Wasm dans les nouvelles applications, il est intéressant de fournir un outil permettant de détecter et de pallier ces anomalies. Ce projet se concentre donc sur l'extension de la *POC* de M. Julmy.

[1] Contraint à l'espace mémoire qui lui est attribué, sans possibilité d'interagir avec le reste du système.
[2] BigInt, « Posit32 » Unum, BigDecimal, ...

Chapitre 2
Objectifs

La *POC* de M. Julmy permet d'instrumenter du code Wasm à la volée afin d'altérer le comportement des opérations instrumentées. Ces opérations sont substituées par des opérations de détection d'anomalies arithmétiques suivies des opérations initiales, ce qui permet de vérifier que les opérations se déroulent correctement sans altérer leur comportement.

Cette instrumentation est réalisée à l'aide de la librairie WAIL [6] et permet actuellement uniquement la détection des *integer overflow* et les résultats infinis sur les nombres à virgule flottante pour les opérations d'addition et de soustraction.

Afin de compléter cette fonctionnalité (appelée *numerical sniffer*), le support pour les opérations de multiplication et de division ainsi que la détection d'autres anomalies arithmétiques seront implémentés. Les anomalies additionnelles seront choisies parmi celles détectées par COJAC telles que :

- *cancellation*
- *absorption*
- *questionable comparisons*
- *underflow*

La logique actuelle, illustrée par la Fig. 1, exécute la détection des anomalies en Wasm et appelle conditionnellement une fonction de signalisation JS.

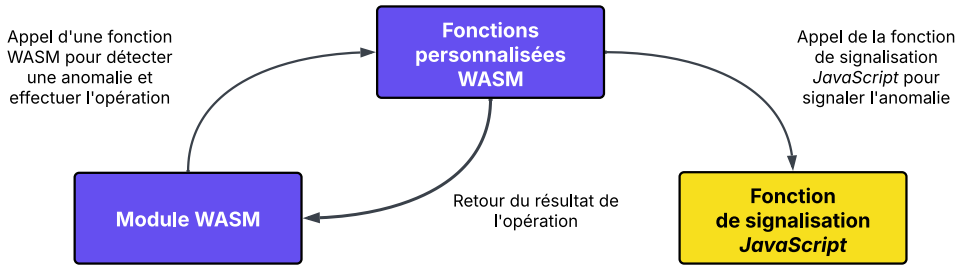


Fig. 1. – Déroulement de l'instrumentation et de la détection d'anomalies

Les « fonctions personnalisées WASM » décrites dans la Fig. 1 sont actuellement écrites en dur dans le code. Dans le but d'uniformiser le comportement du programme et d'augmenter sa maintenabilité ainsi que son extensibilité, ce projet définira une nouvelle architecture d'instrumentation. Cette architecture devra permettre de facilement substituer les opérations instrumentées par divers comportements prédéfinis. L'ajout de nouveaux comportements devra être le plus simple possible.

Les anomalies actuellement détectées sont signalées durant l'exécution du programme, à chaque détection et sans indication de la localisation exacte où l'anomalie a été rencontrée. Afin de faciliter la lecture des résultats de l'instrumentation et de procurer des résultats interprétables, la logique de signalisation sera réécrite.

Dans le but de potentiellement économiser de l'espace mémoire, COJAC propose une fonctionnalité *double-as-float* qui teste si des calculs entre *doubles* peuvent être faits avec

des *floats* sans dégradation notoire de la qualité des résultats. Cette fonctionnalité sera également implémentée durant ce travail de bachelor.

Enfin, COJAC propose une fonctionnalité permettant de *wrap* les nombres dans des structures de nombres plus riches afin de garantir une plus grande robustesse. À l'heure de la rédaction de ce cahier des charges il n'est pas certain que cette fonctionnalité soit réalisable, cependant une étude de l'approche nécessaire à son implémentation sera relatée dans le rapport de ce travail de bachelor. Si la fonctionnalité est réalisable alors une *POC* de cette dernière se trouvera dans le livrable final.

Les objectifs de ce projet sont donc les suivants :

- Conceptualiser et implémenter une nouvelle architecture de substitution des opérations capable de:
  - Détecter les opérations ciblées
  - Substituer les opérations détectées par des comportements prédéfinis :
    - Comportement « vide » ne faisant qu'effectuer l'opération initiale
    - *Numerical sniffer*
    - *Double-as-float*
    - *Wrappers* (si implémentés)
- Réaliser un *numerical sniffer* analogue à celui de COJAC avec les fonctionnalités suivantes :
  - Instrumentation des opérations suivantes sur les *int32*, *int64*, *float32*, *float64* :
    - Addition
    - Soustraction
    - Multiplication
    - Division
  - Détection de plusieurs sortes d'anomalies qui seront définies dans le rapport
  - Signalisation des anomalies détectées sous forme de synthèse indiquant idéalement la localisation de ces dernières
- Implémenter la fonctionnalité *double-as-float* de COJAC
- Validation du prototype une fois les objectifs précédents remplis
- Étudier et implémenter (si possible) la fonctionnalité *wrapper* de COJAC

Secondairement, une mesure de l'impact de l'instrumentation sur les performances sera également documentée.

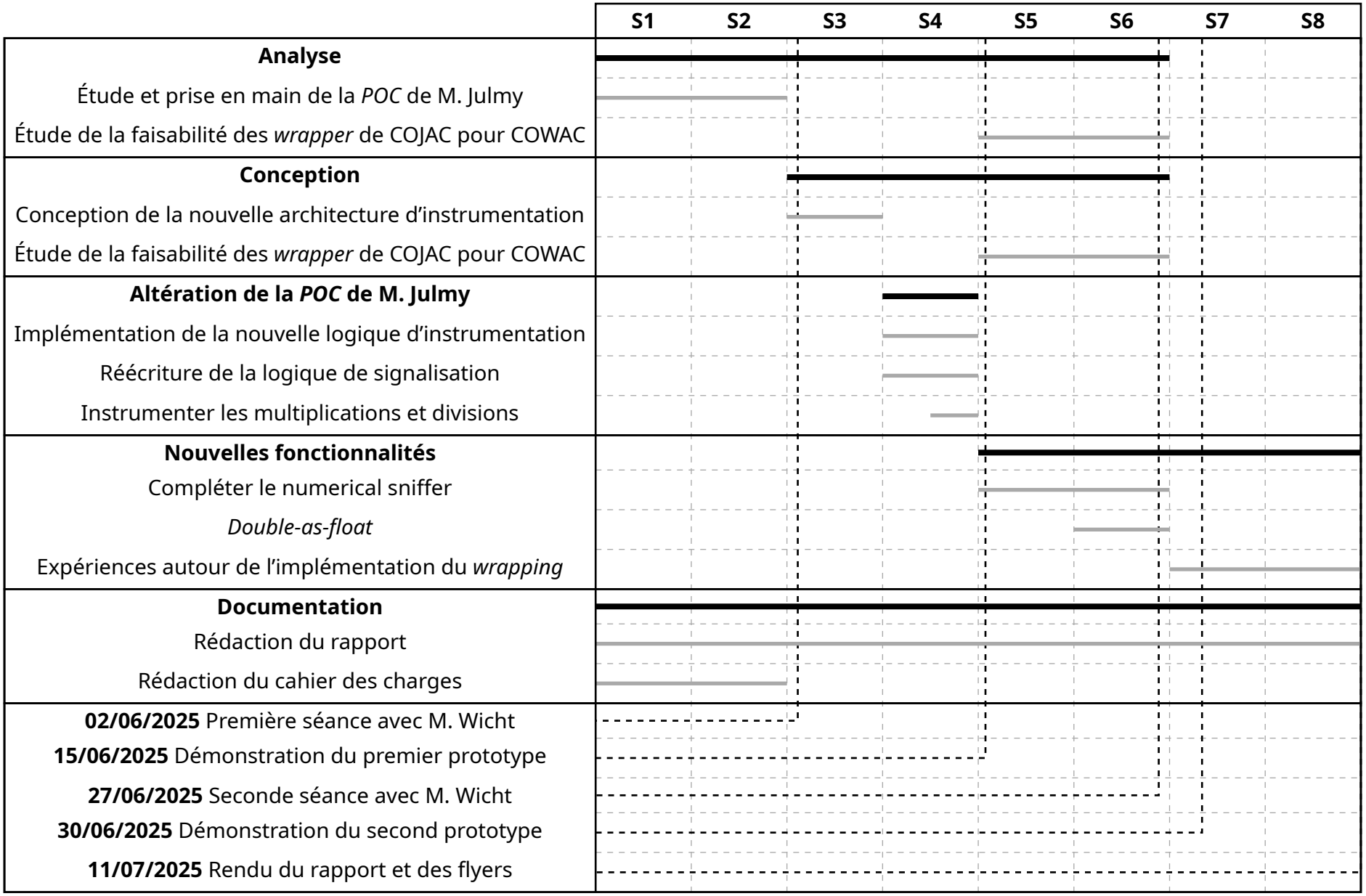
De plus un rapport documentant les processus d'implémentation, les notions abordées et l'état du projet à la fin du travail de bachelor sera rédigé.

Chapitre 3

Tâches

Chapitre 4

Planning


S1
S2
S3
S4
S5
S6
S7
S8
Analyse
Étude et prise en main de la POC de M. Julmy
Étude de la faisabilité des wrapper de COJAC pour COWAC
Conception
Conception de la nouvelle architecture d'instrumentation
Étude de la faisabilité des wrapper de COJAC pour COWAC
Altération de la POC de M. Julmy
Implémentation de la nouvelle logique d'instrumentation
Réécriture de la logique de signalisation
Instrumenter les multiplications et divisions
Nouvelles fonctionnalités
Compléter le numerical sniffer
Double-as-float
Expériences autour de l'implémentation du wrapping
Documentation
Rédaction du rapport
Rédaction du cahier des charges
02/06/2025 Première séance avec M. Wicht
15/06/2025 Démonstration du premier prototype
27/06/2025 Seconde séance avec M. Wicht
30/06/2025 Démonstration du second prototype
11/07/2025 Rendu du rapport et des flyers

Figures

Chapitre 5
Glossaire

absorption: Anomalie arithmétique survenant lorsque deux nombres à virgule flottante d'un ordre de grandeur excessivement différent sont additionnés ou soustraits sans effet. 2

cancellation: Anomalie arithmétique survenant lorsque deux nombres à virgule flottante dont la valeur absolue est presque similaire sont additionnés ou soustraits. Cette anomalie a pour effet de rendre les bits de poids faible plus signifiants sur le résultat alors que ceux-ci sont souvent plutôt du bruit. 2

COJAC: Utilitaire Java développé à l'HEIA-FR permettant de détecter et de pallier les anomalies arithmétiques pouvant être rencontrées à l'exécution d'un programme Java. 1, 1, 1, 2, 2, 3, 3, 3, 3, 6, 6

integer overflow: Anomalie arithmétique survenant lorsqu'un nombre entier sort du domaine permis par le type qui lui est attribué. 2

JS **– JavaScript**: Langage de scripting interprété utilisé notamment dans le contexte de applications web. 1, 1, 1, 2

POC – ***proof of concept***: Preuve de concept, démonstration de la faisabilité d'une idée par sa réalisation partielle et/ou à échelle réduite. 1, 1, 2, 3, 6, 6

questionable comparisons: Anomalie arithmétique survenant lorsque deux nombres à virgule flottante très proches sont comparés. 2

underflow: Anomalie arithmétique survenant lorsque le résultat de la division de deux nombres à virgule flottante est si petit qu'il est arrondi à zéro. 2

Ariane flight V88 *[3]*: Décolage de fusée ayant échoué à cause de plusieurs erreurs de conception du code, notamment un *overflow*. 1

VM escape via VGA device *[4]*: Vulnérabilité de VirtualBox, causée par un *overflow*, permettant à du code exécuté dans la machine virtuelle d'affecter la machine hôte. 1

WAIL: Librairie d'instrumentation de code Wasm. 2

Wasm **– WebAssembly**: Format d'instructions binaires basé sur une machine virtuelle à pile. 1, 1, 1, 1, 1, 1, 1, 1, 2, 2

Bibliographie

[1] 2025. Documentation de Mozilla concernant Wasm. Consulté à l'adresse https://developer.mozilla.org/fr/docs/WebAssembly

[2] 2025. Site officiel de Wasm. Consulté à l'adresse https://webassembly.org/

[3] 2025. Page wikipedia du vol Ariane V88. Consulté à l'adresse https://en.wikipedia.org/wiki/Ariane_flight_V88

[4] 2025. Rapport de sécurité concernant une vulnérabilité de Oracle VM VirtualBox permettant d'exécuter du code sur la machine d'hôte. Consulté à l'adresse https://en.wikipedia.org/wiki/Ariane_flight_V88

[5] 2025. Repository GitHub de COJAC. Consulté à l'adresse https://github.com/Cojac/Cojac?tab=readme-ov-file

[6] 2025. Repository GitHub de la librairie WAIL. Consulté à l'adresse https://github.com/Qwokka/WAIL