**TRAVAUX DIRIGES : n°3**

**Implémentation du type abstrait des Messages**

**MARQUEZ Lucas & ROBIN Florian**

**I – PROBLEMATIQUE :**

Le but de ce TD est de concevoir une implémentation complète pour le type abstrait des Messages en répondant au cahier des charges imposé par une personne compétente du secteur « contrôle de sureté de fonctionnement » au sein d’une centrale nucléaire. Faisant suite aux cahiers des charges la spécification CASL est également fournie, ce langage est universel et par conséquent adapté à ce type de travail. Cette implémentation va permettre au robot de diagnostique de transmettre un message à un des huit robots experts en maintenance pour qu’ils interviennent correctement sur un quelconque disfonctionnement.

**II – REALISATION :**

Pour chaque implémentation à réaliser il faut procéder en suivant 4 étapes définies de la manière suivante :

ETAPE 1 : Spécification du type abstrait.

Celle-ci est composée de 3 parties bien distinctes : l’en tête, la signature et la sémantique.

La signature correspond à l’interface de la spécification, c’est-à-dire la partie visible auquel toute personne a accès. Elle permet de déclarer le nom du module de spécification, les types utilisés, les types abstraits à définir ainsi que les opérations du type avec leur signature propre. Ensuite la sémantique est un ensemble de propriétés appelés axiomes qui permettent d’observer, modifier ou de créer l’état d’un objet. Pour finir l’en-tête permet de définir les librairies qui seront utilisés dans la spécification.

***Spécification en CASL des types messages :***



**EN-TETE**



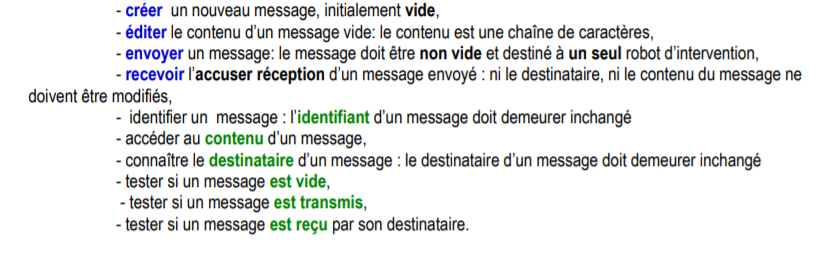
**SEMANTIQUE**

**SIGNATURE**



**SEMANTIQUE**

Afin d’être conforme au cycle de développement du type abstrait, il faut étudier le cahier des charges fourni pour bien respecter la demande. Nous devons donc respecter les fonctionnalités et conditions définies ci-dessous :



Il faut également détailler les mots clés de la spécification afin de ne pas commettre d’erreurs.

**Mots clés du CASL** :   
Library : Librairie(s)

From : Depuis

Get : Prendre (« from Basic/Numbers get Rat  » = Importer Rat (rationnel) depuis la librairie définie avant avec Library)

Spec … : Spécification du type abstrait de …

And : Et, permet de définir les types abstraits utilisés

Then : Puis/ensuite

Sort : Type abstrait défini par la spécification

Pred : Prédicat

Ops : (donner la signature des) Opérations (du type)

« : » :  
Forall … : Pour tout …

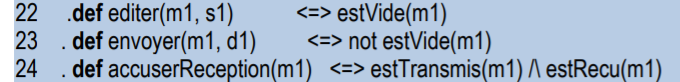
. def … : Axiomes définissant les opérations par leurs propriétés

… ⬄ … : … si et seulement si …

Not … : Non …

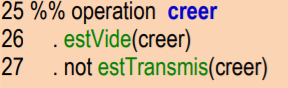
… /\ … : … et … (et logique)

Pour finir nous allons commenter tous les groupes d’axiomes de la spécification dans le but de les relier au cahier des charges.



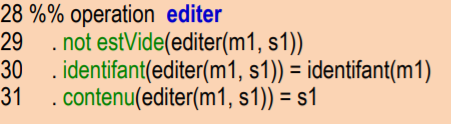
Ce groupe d’axiomes fait référence à 3 fonctions différentes, ici on nous impose des préconditions à exprimer afin d’utiliser ces fonctions.

* Ligne 22 : Un message ne peut être édité que si et seulement s’il est vide.
* Ligne 23 : Un message ne peut être envoyé que si et seulement s’il n’est pas un message vide.
* Ligne 24 : Le robot émetteur ne peut recevoir un accusé de réception d’un message que si et seulement si le message est transmis et ensuite reçu par le robot destinataire.



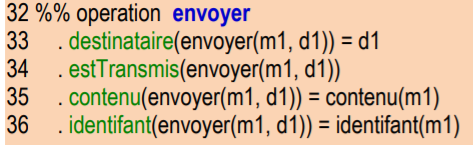
Ce groupe d’axiomes fait référence à la fonction créer un nouveau message, initialement vide.

* Ligne 26 : On initialise le message de façon à ce qu’il soit vide.
* Ligne 27 : Un message crée ne peut pas être transmis.



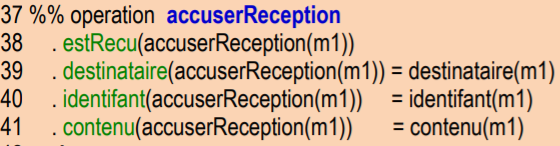
Ici le groupe d’axiomes fait référence à la fonction éditer du cahier des charges.

* Ligne 29 : Tout message ayant fait l’objet d’une édition n’est plus considéré comme un message vide.
* Ligne 30 : L’édition d’un message laisse invariant son identifiant.
* Ligne 31 : le contenu d’un message demeure toujours celui défini lors de l’édition du message. De plus cela doit forcément être une chaine de caractère.



Ce groupe d’axiomes fait référence à la fonction envoyer du cahier des charges.

* Ligne 33 : Le destinataire d’un message doit demeurer inchangé au moment de l’envoie du message et il doit être unique.
* Ligne 34 : Tout message envoyé est considéré comme transmis.
* Ligne 35 : Le contenu d’un message ne doit pas changer.
* Ligne 36 : Envoyer un message doit laisser l’identifiant inchangé.



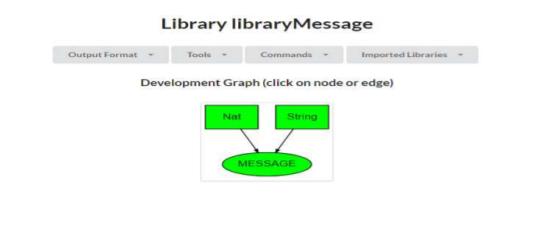
Pour finir la fonction Accuser réception possède ces axiomes.

* Ligne 38 : Pour recevoir un accusé de réception le message doit être reçu par le destinataire.
* Ligne 39 : le destinataire doit demeurer inchangé afin de recevoir l’accusé de réception.
* Ligne 40 : L’identifiant doit lui aussi demeurer inchangé.
* Ligne 41 : Le contenu d’un message ne doit pas être modifié.

ETAPE 2 : Validation de la spécification

Pour déterminer si la spécification est correcte il existe différents outils, nous utiliseront ici HETS (Heterogeneous ToolSet) qui est l’un des plus performants. Il a été développé par CoFi. L’analyseur HETS est disponible sur n’importe quel navigateur web à partir de DOLiator : ([*http://rest.hets.eu/*](http://rest.hets.eu/)*)*.

Ainsi valider une spécification c’est prouver formellement que cette dernière satisfait la consistance (pas d’axiomes exprimant des propriétés contradictoires) et la complétude (suffisamment d’axiomes pour savoir si une certaine propriété est vraie ou fausse). Vérifier une implémentation revient à prouver que cette dernière satisfait toutes les propriétés énoncées en spécification.



Voici le résultat de la spécification, on remarque que les types Nat et String sont définis, la spécification est donc correcte.

ETAPE 3 : Implémentation de la spécification

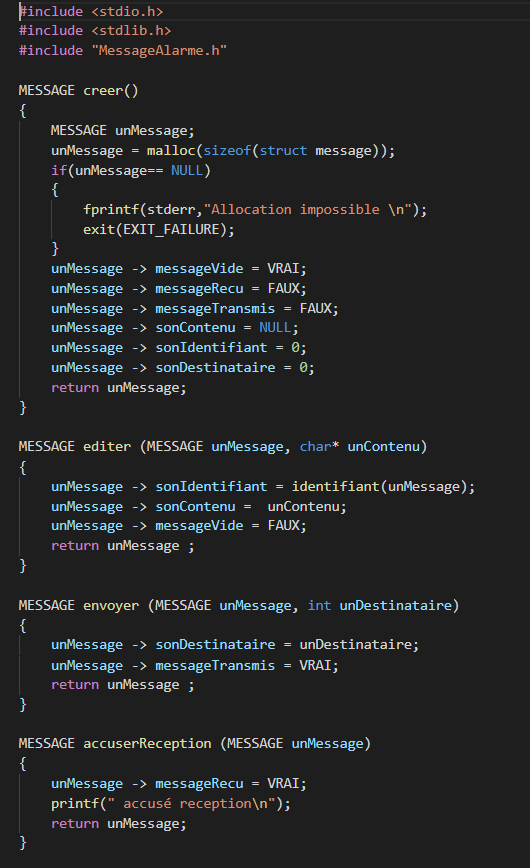
Nous avons donc commencé l’implémentation par le fichier d’interface, le header, dans ce fichier toutes les fonctions seront définies ainsi que le type abstrait Message.



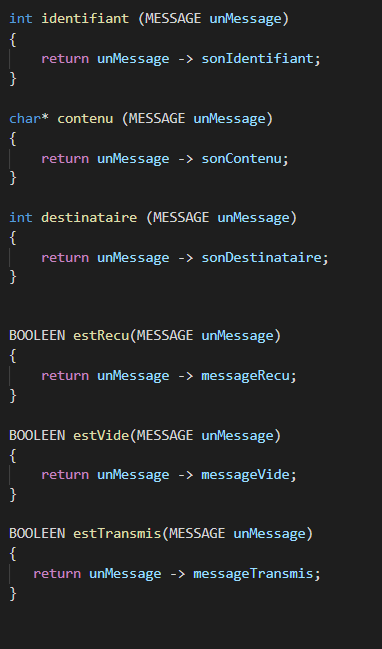
Définition de l’en-tête de toutes les fonctions

Définition du type message

Ensuite nous avons réalisé le fichier « messageAlarme.c » où toutes les opérations sur les fonctions y sont décrites. Ce fichier est non consultable et utilise le header car les en-têtes de fonctions sont déjà déclarés.



Sur l’image la fonction Créer () permet d’initialiser le message, c’est un constructeur. Ensuite les opérations sont décrites dans les fonctions qui suivent, ce sont les modificateurs.

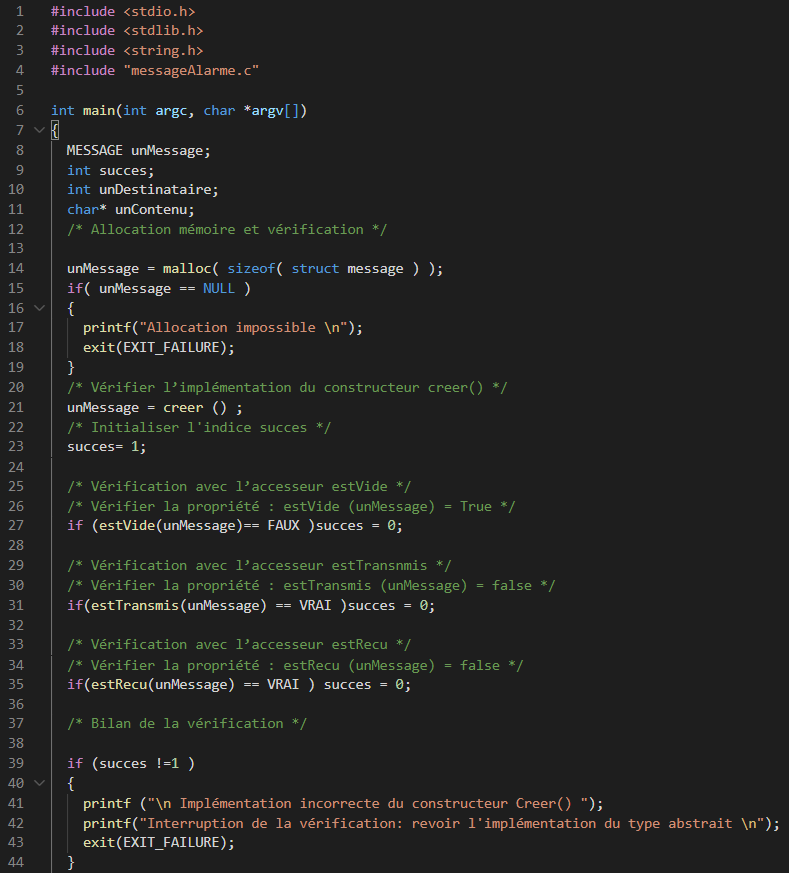


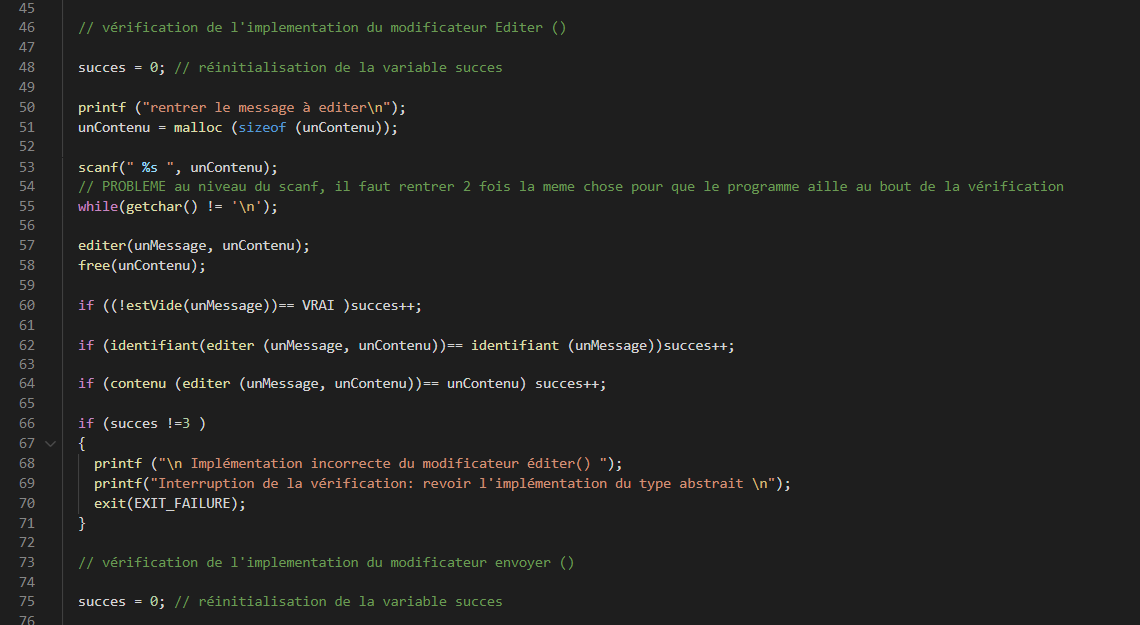
Sur cette autre image, tous les accesseurs sont définis, ils vont nous permettre d’observer l’état de l’objet message.

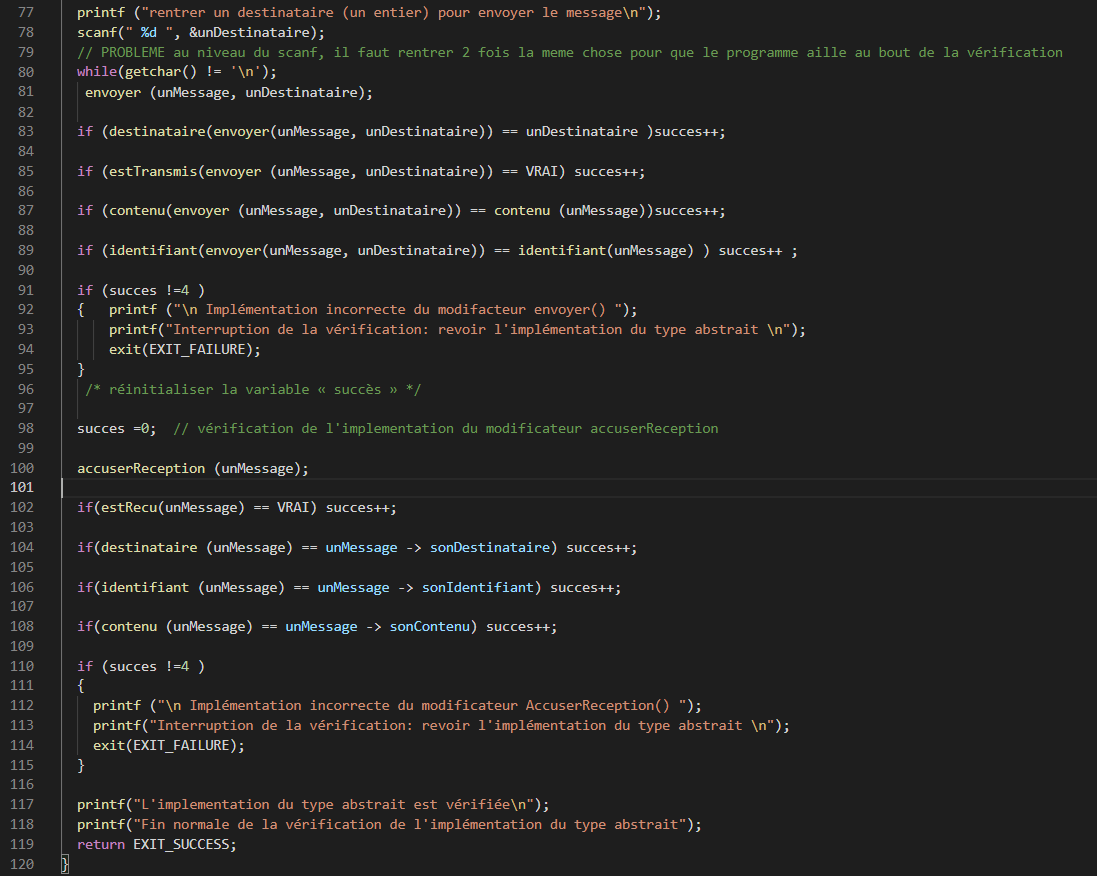
ETAPE 4 : Vérification de l’implémentation

Ici il suffit de vérifier que le constructeur Créer () et les modificateurs sont correctement implémentés. Tous les axiomes doivent être satisfaits pour que l’implémentation soit validée. Cette vérification se déroule dans un fichier nommé « PreuveMessage.c » en cas d’échec le programme s’arrête et renvoie une erreur.

**Petite précision :** au moment de l’exécution du programme de vérification, un bug subsiste sur les deux « scanf » il faut rentrer 2 fois le même message et 2 fois le même destinataire. Nous avons cherché mais l’erreur n’a pas réussi à être résolue.







**III – CONCLUSION :**

Sur le plan théorique, réaliser ce TD nous a permis de rentrer dans les détails du cycle de développement, nous pensions avoir mieux compris le rôle des 4 étapes. De plus la compréhension de la spécification est bien plus claire actuellement. On comprend mieux l’utilisation de la spécification qui s’avère très utile et indépendante de tout langage de programmation. La lecture d’une spécification est aussi un peu plus aisée maintenant même s’il reste tout de même du travail à fournir pour qu’elle soit optimale.

Sur le plan pratique, l’application des types abstraits permet de mettre en œuvre la spécification de type abstrait pour en faire un type concret. Elle présente plusieurs avantages, par exemple elle est indépendante de tout langage de programmation, la sémantique reste inchangée. Cette indépendance permet de palier facilement à un changement de langage ou à l’implémentation de nouvelles fonctions.