MSLite: Tactiques d’interopérabilité et de testabilité proposées par le groupe

# Nature de l’exercice

* Le groupe est divisé en petites équipes (3-6 personnes)
* Chaque équipe se voit attribuer un des trois attributs de qualité (ces attributs sont importants dans le contexte de MSLite et n’ont pas encore été discutés dans ce contexte)
* Chaque équipe doit :
  1. Définir une exigences de qualité dans le document architectural de MSLite rattachée à l’attribut assigné;
  2. identifier et justifier le choix de deux (2) tactiques architecturales pertinentes pour l’exigence retenue dans le contexte de MSLite;
  3. identifier les portions de l’architecture où il faut intervenir pour appliquer les tactiques retenues.

## Interopérabilité

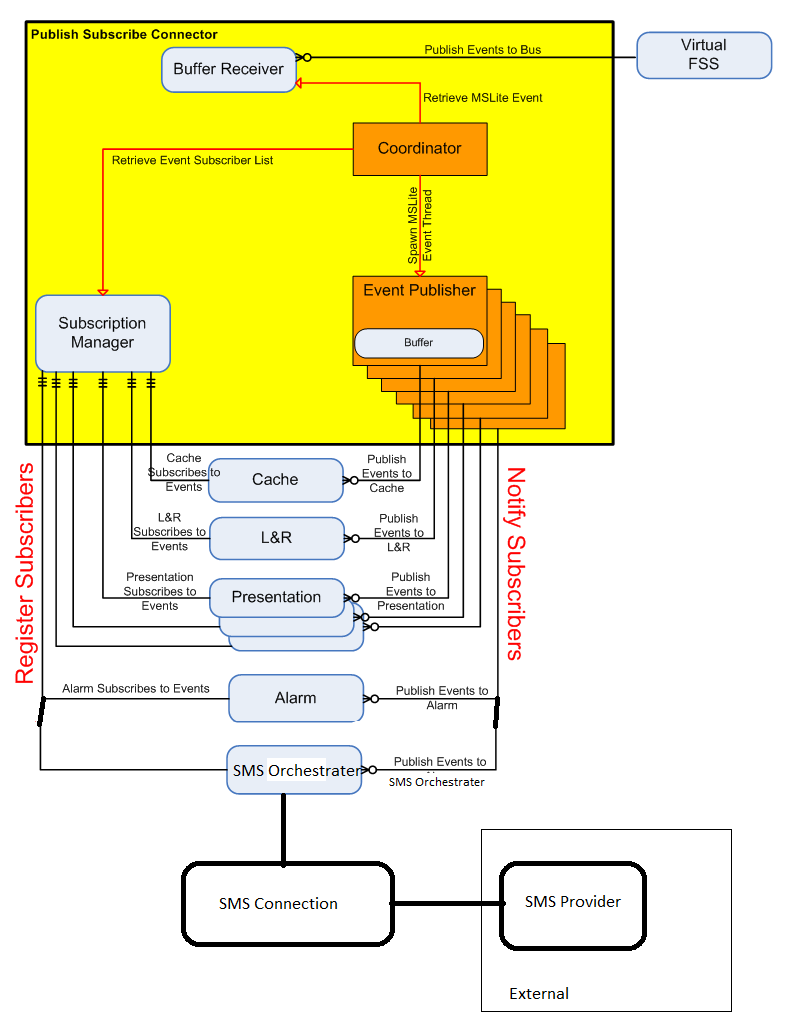
## Exigence

Dans notre scénario, le logiciel MSLite permettrait d’envoyer des messages texte d’avertissement aux utilisateurs. Ceux-ci auront la possibilité de choisir les avertissements qu’ils souhaiteraient recevoir en cochant ceux-ci dans une liste. Ce service sera offert en surplus aux utilisateurs et sera desservi par une compagnie spécialiste dans le domaine.

## Tactique

Pour arriver à notre solution, il sera nécessaire d’utiliser les deux tactiques « Tailored Interface » et « Orchestrate ». En premier lieu, il va être nécessaire de créer un nouveau module qui sera inscrit au Susbscription Manager et recevra les messages d’alerte. Ce nouveau module représente « Orchestrate » parce qu’il fait une suite d’opérations pour recueillir les informations nécessaires, en ajoutant le numéro de téléphone de l’utilisateur inscrit au service ainsi que de choisir les avertissements que le client veut recevoir. Ensuite, la tactique « Tailored Interface » est utilisée pour préparer ces informations pour qu’elles soient dans un format que la compagnie offrant le service de message SMS puisse utiliser.

## Référence dans le document d’architecture



# Testabilité

## Exigence

Afin d’augmenter la testabilité de MSLite, il faut établir deux tactiques de tests en lien avec les droits d’accès aux différentes notifications et alertes pour les utilisateurs.

## Tactiques

La tactique utilisée serait d’utiliser une source de données abstraite (Abstract Data Source) à la place du Virtual FSS en utilisant la même interface. De cette façon, il serait possible de charger les événements à tester et de connaître quels événements sont publiées au lieu de devoir gérer des données envoyées aléatoirement. En sachant quels événements seront publiés, il sera possible de tester les différents droits d’accès au niveau de la composante de présentation à l’utilisateur.

De plus, une interface spécialisée (specialized interface) permettant de modifier les données directement dans le Buffer Receiver pourrait être utilisée. Cette interface permet de différencier les méthodes de réception (remplissage) de données du Buffer Receiver, entre les données de test, et le fonctionnement normal du système (avec le Virtual FSS).

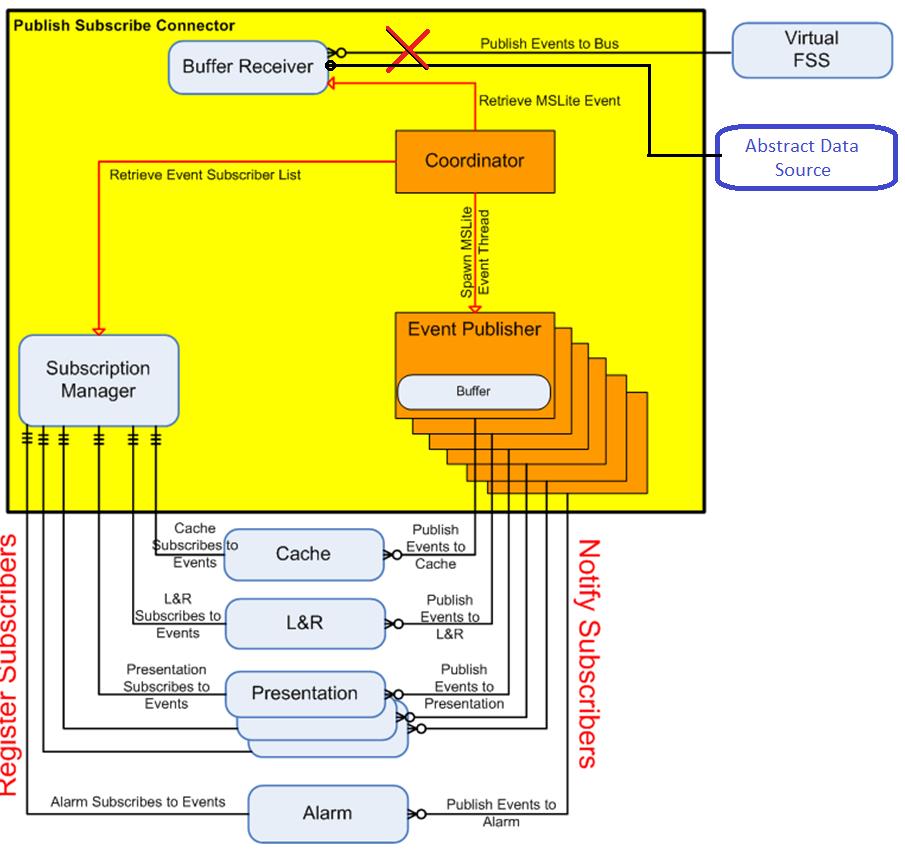
## Référence dans le document d’architecture

Au niveau du Pulish Subscribe Connector, il faudra modifier le Buffer Receiver afin qu’il puisse charger une liste d’événements précis et prédéfinis qui serviront aux tests des droits d’accès pour les différents types d’utilisateurs. À ce moment, il faut s’assurer que le Virtual FSS n’envoie pas de données au Buffer Receiver pour ne pas que la source de données de tests soit biaisée par des vrais change of values.



Hiver 2015 Page 2

LOG430 – Architecture logicielle Exercice sur les tactiques I



Hiver 2015 Page 3



# Testabilité

## Exigence

* On veut s’assurer que le Coordinateur recoit bien les évènements du Subscribe Manager et qu’il les communique bien avec l’ Event Publisher.

## Tactique

* **Specialize interfaces :** Ajouter un getter/setter au buffer receiver. Accéder aux nombres de fois que le Coordinateur envoie des Threads au Event Publisher et la comparer a la valeur du Subscriber.  Créer une liste d’évènement MSLite et les envoyer dans le Coordonnateur. Une fois qu’il a obtenu le nombre de processus à créer par l’Event Subscriber, conserver ce nombre et envoyer le résultat dans une version contrôlée de l’event publisher qui créera les processus à faire et renvera le résultat au Coordonnateur. Si le résultat diffère entre la valeur obtenue du Publisher et du Subscriber, le test à échoué.
* **Record/Playback**: sur le BufferReceiver : On ne veut pas toujours recréer de nouveaux évènements donc on va toujours réutiliser les même. Pour limiter l’utilisation de ressources et éviter de calculer différents ‘’states’’. Par la suite, on compare le contenu de la cache actuelle avec ce qui est attendu. De cette façon, on élargie la porté du test en validant la communication entre le Publisher et le Subscriber.

## Référence dans le document d’architecture

* Figure 8: Decomposition of the Publish Subscribe connector Table 4: Publish Subscribe subcomponents

# Testabilité

## Exigence

Nous souhaitons tester l’absence de cycles au niveau de l’« event publisher ». Lorsqu’un évènement est déclenché, il effectue généralement un appel à un autre évènement. Nous souhaitons donc nous assurer qu’aucune boucle infinie ne se produit.

## Tactique

La première tactique choisie est « Record/Playback ». Cette tactique nous permettra de garder une trace de l’exécution des processus. Cette trace nous permettra de reconstituer les évènements qui ont mené à l’erreur et d’ainsi la corriger. La fonctionnalité de « logger » sera implémentée entre l’ « event publisher » et le « subscription manager », puisque c’est à cet endroit que les processus sont appelés.

La seconde tactique choisie est « Localize state storage ». Cette tactique permet de stocker l’état interne d’un certain module sous une valeur. Dans notre cas, il serait intéressant de vérifier pour chaque évènement publié que le bon nombre de « listener » seront exécutés. En s’assurant que les évènements respectent le nombre d’exécutions individuel préétabli, on s’assure qu’il n’y a pas de boucle d’exécution indésirée.

## Référence dans le document d’architecture

MSLite Architecture documentation complete (p. 32)

# Testabilité

## Exigence

Améliorer la testabilité pour le module « Publisher-Subscriber » afin de s’assurer que le Publisher envoi l’information aux bons Subscribers.

## Tactique

Record/Playback : on génère un log lorsqu’un Subscriber contacté par un publisher. Le Subscriber écrira dans le journal de log les informations suivantes : « Je suis [nom subscriber], voici mes paramètres : [Paramètres] ».

Executable Assertions : on vérifie que les logs sont valides pour un event généré dans le cadre d’un test.

## Référence dans le document d’architecture

Dans la section *5.1.1. C&C View Packet 1: The MSLite System (Table 3: MSLite connector catalog)* :

Il faudrait ajouter le fait que chaque subscriber génère un log dans le journal de logs.

# Testabilité

## Exigence

Être en mesure de reproduire un problème de buffer overflow.

## Tactique

* Record/Playback
  + Cette tactique permet de savoir quelles séquences d’événements, ainsi que les états des composants du système, générant un buffer overflow dans un contexte réaliste près de la mise en production chez le client.
* Sandbox
  + Cette tactique permet d’isoler la composante Publish/Susbcribe Connector afin d’émuler les séquences d’évènements qui déclenchent les buffer overflow. Les composants émetteurs d’événements seraient remplacés par un composant bidon chargé d’envoyer la séquence d’événements problématique dans le but de déclencher le buffer overflow. Pour savoir comment faire les composants bidon et quelle séquence d’événements utiliser, on utilise les données obtenues dans la tactique Record/Playback.

## Portions de l’architecture impliquées

* Record/Playback
  + Buffer receiver (pour enregistrer les évènements)
  + Subscribtion manager (pour enregistrer l’état du système. I.E. Le nombre de subscribers)
* Sandbox
  + La partie Publish/Suscribe Connector serait mis dans un sandbox
  + Les composants VFSS, L&R, Alarm, Presentation et Cache serait remplacé par des composants bidon ayant pour but de générer l’état qui a déclenché le buffer overflow.

## Référence dans le document d’architecture

* Figure 8
* Tableau 4 > Buffer receiver
* Ajouter les spécifications des composants bidon