Analyse – V1

|  |  |
| --- | --- |
| projet | NAO BOX |
| domaine d’ACTION | Pilotage du robot NAO lors de sa présentation sur les salons |
| DESCRIPTION | Mettre à disposition des personnes qui présentent NAO sur les salons un boitier de contrôle de ses actions. Le robot sera commandé de manière intuitive via un seul périphérique afin d’alléger le package nécessaire à son fonctionnement |
| maitrise d’ouvrage | Bernard BUFFIERE |
| chef de projet | Elie METAHRI |
| etudiants | Elie METAHRI, Bastien VAUTIER, Jérémie LIECHTI |



Sommaire

[I. **Généralités** 3](#_Toc433463570)

[1. Analyse du besoin 3](#_Toc433463571)

[2. Contraintes de développement 3](#_Toc433463572)

[3. Caractéristiques de l’application 3](#_Toc433463573)

[4. Contraintes de l’application 4](#_Toc433463574)

[II. **Représentation UML** 4](#_Toc433463575)

[1. Diagramme de Cas d’Utilisation Général 4](#_Toc433463576)

[2. Diagramme de Classes Général 6](#_Toc433463577)

[3. Diagrammes de Séquence 8](#_Toc433463578)

[a) Administrer 8](#_Toc433463579)

[b) Ajouter fonctionnalités 9](#_Toc433463580)

[c) Consulter Etat Nao 9](#_Toc433463581)

[d) Démarrer Raspberry 10](#_Toc433463582)

[e) Effectuer une tâche 10](#_Toc433463583)

[f) Gérer Nao 11](#_Toc433463584)

[g) Gérer Périphériques 12](#_Toc433463585)

[h) Se Connecter 13](#_Toc433463586)

[i) Vérifier Connexion Raspberry – Nao 14](#_Toc433463587)

[4. Diagramme de déploiement 15](#_Toc433463588)

# Généralités

## Analyse du besoin

L’application doit être une application client / serveur. Cette dernière doit permettre à un utilisateur lambda de pouvoir donner des ordres à Nao afin que ce dernier les exécute. Pour cela, nous allons utiliser plusieurs modules dans le but que la communication entre l’utilisateur et Nao soit simple et fonctionnelle. Pour cela, nous allons utiliser un Raspberry Pi 2 qui va jouer le rôle de serveur pour héberger notre application Web ainsi que de point d’accès Wifi. L’application sera alors décomposée en deux parties :

* Une partie utilisateur
* Une partie administrateur

La partie utilisateur va alors permettre de gérer Nao : consulter l’état de ces différents capteurs (batterie, …) et de lui envoyer des ordres.

La partie administrateur va alors permettre d’administrer l’application (comme ajouter de nouvelles fonctionnalités à faire réaliser à Nao ou bien gérer les périphériques pouvant se connecter au robot.

## Contraintes de développement

Afin de pouvoir communiquer directement avec le robot, nous devons utiliser le langage approprié afin de pouvoir lui envoyer les actions à exécuter sous forme de trames. Pour cela, nous avons décidé de communiquer avec Nao en utilisant ***le format JSON (modifier le format car on ne sait pas encore lequel utiliser). Expliquer le mode de fonctionnement de ce langage…***

## Caractéristiques de l’application

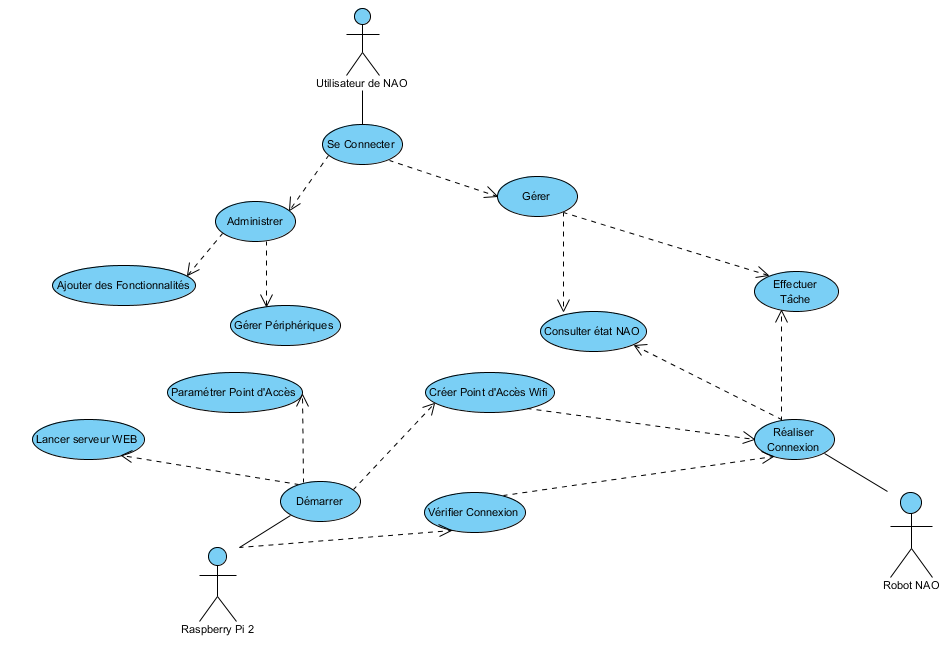
L’application devra être une application web capable d’être exécutée dans n’importe navigateur Web. Pour la développer, nous allons utiliser comme technologies principales : Le HTML5 et le CSS3 qui sont des langages de programmation Web s’exécutant côté client, du Javascript, du PHP qui lui s’exécute côté serveur… Quant au serveur, ce dernier tournera sous la version d’Apache.

## Contraintes de l’application

L’application devra être consultée par n’importe quel appareil composé d’un navigateur Web. Pour cela, les pages devront s’adapter automatiquement à n’importe quel type d’écran. Il faudra donc qu’elle soit en responsive design.

# Représentation UML

## D**iagramme de Cas d’Utilisation Général**



Les diagrammes de cas d'utilisation sont des diagrammes UML utilisés pour donner une vision globale du comportement fonctionnel d'un système logiciel. Ils sont utiles pour des présentations auprès de la direction ou des acteurs d'un projet, mais pour le développement, les cas d'utilisation sont plus appropriés.

|  |  |
| --- | --- |
| Cas  ‘’Démarrer’’ | Lorsque l’on branche le Raspberry Pi sous tension, ce dernier démarre automatiquement et doit lancer le serveur Web. |
| Cas  ‘’Vérifier Connexion’’ | Il permet de vérifier si la connexion entre le Raspberry Pi et le robot Nao est réussie grâce à un ping de l’adresse IP de Nao. |
| Cas  ‘’Se connecter’’ | Permet de réaliser la connexion de l’utilisateur sur l’application Web. Ce dernier peut alors s’identifier soit en tant qu’utilisateur soit en tant qu’administrateur. |
| Cas  ‘’Administrer’’ | Permet d’administrer l’application Web, permet d’ajouter des fonctionnalités pour la partie utilisateur. |
| Cas  ‘’Gérer Périphériques’’ | Permet de gérer les périphériques pouvant se connecter avec Nao (les autoriser ou bien les interdire). |
| Cas  ‘’Ajouter  des Fonctionnalités’’ | Permet à l’administrateur d’ajouter des fonctionnalités à la partie utilisateur. Il peut alors ajouter des actions à faire réaliser à Nao ce qui entraine une modification de l’IHM. |
| Cas  ‘’Gérer’’ | Permet de gérer Nao et de consulter des informations. Cela sera utiliser dans la partie utilisateur. |
| Cas  ‘’Consulter état Nao’’ | Permet de consulter l’état des différents capteurs du robot comme l’intensité du signal wifi, le pourcentage de batterie de Nao, … |
| Cas  ‘’Effectuer Tâche’’ | Permet de demander à Nao de réaliser une tâche particulière parmi une liste de tâches prédéfinies. |

## Diagramme de Classes Général

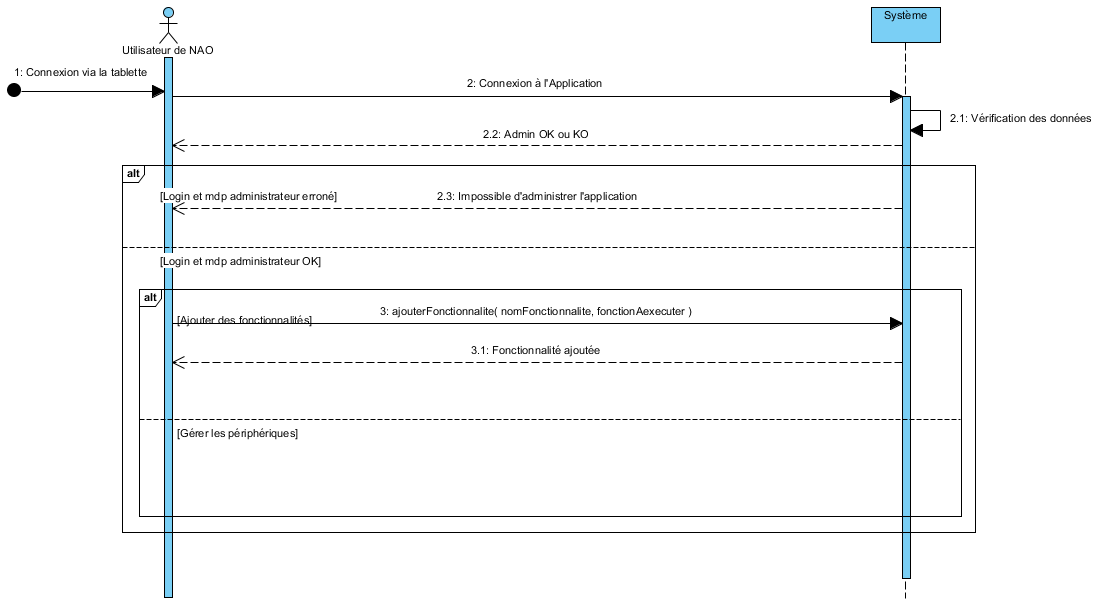
Le diagramme de classes est un schéma utilisé en génie logiciel pour présenter les classes et les interfaces des systèmes ainsi que les différentes relations entre celles-ci. Ce diagramme fait partie de la partie statique d'UML car il fait abstraction des aspects temporels et dynamiques.

|  |  |
| --- | --- |
| Classe raspberry | * ipPartenaire : variable de type entier permettant de stocker l’adresse IP du partenaire auquel le Raspberry va être associé (ici ça sera l’IP de Nao). * adresseMac : variable de type String permettant de stocker l’adresse mac du partenaire auquel Raspberry va être associé (ça sera l’adresse de mac de Nao ou de l’appareil autorisé à communiquer avec lui). * etatConnexion : variable de type booléen permettant de stocker un booleen (true or false) pour savoir si la connexion entre le Raspberry et Nao est correcte. * etablirConnexion(ipPartenaire) : méthode de type booleen permettant de réaliser la connexion entre le Rapberry et Nao. * deconnexion() : méthode de type void permettant de réaliser la déconnexion entre le Raspberry et Nao. * obtenirIP() : méthode de type entier permettant de récupérer l’adresse IP du partenaire. * obtenirAdresseMac() : méthode de type String permettant de récupérer l’adresse mac du partenaire. * getIP() : accesseur de type entier permettant aux autres classes de connaitre l’adresse IP du partenaire. * getMac() : accesseur de type String permettant aux autres classes de connaitre l’adresse mac du partenaire. |
| Classe Utilisateur | * login : variable de type String permettant de stocker l’identifiant de l’utilisateur. * motDePasse : variable de type String permettant de stocker le mot de passe de l’utilisateur. * etatConnexionUtilisateur : variable de type booleen permettant de stocker l’état de la connexion de l’utilisateur (true si l’utilisateur est connecté, false si ce dernier ne l’est pas). * profil : variable de type String permettant de stocker le profil de l’utilisateur utilisé (utilisateur ou administrateur). * connexion(login, motDePasse) : méthode de type booleen permettant de réaliser la connexion sur l’application (administrateur ou utilisateur en fonction du couple d’identifiants utilisés). * getProfil() : accesseur de type String permettant aux autres classes de connaitre le profil de connexion à l’application. |
| Classe Application | * nomFonctionnalite : variable de type String permettant de stocker le nom de la fonctionnalité que l’on souhaite ajouter dans l’application. * fonctionAexecuter : variable de type String permettant de stocker le nom de la fonction à exécuter (la fonction que Nao doit réaliser). * etatNao : variable de type booleen permettant de stocker l’état de connexion de Nao (true en ligne, false déconnecté). * ajouterFonctionnalite(nomFonctionnalite, fonctionAexecuter) : méthode de type booleen permettant d’ajouter une nouvelle fonctionnalité à l’application. * executerAction(nomFonctionnalite) : méthode de type booleen permettant de faire réaliser une action à Nao. * getNomFonctionnalite() : accesseur de type String permettant aux autres classes de connaitre le nom de la nouvelle fonctionnalité. * getFonctionAexecuter() : accesseur de type String permettant aux autres classes de connaitre la fonction à exécuter. * getEtatNao() : accesseur de type booleen permettant aux autres classes de connaitre l’état de connexion de Nao. |

## Diagrammes de Séquence

Les diagrammes de séquences sont la représentation graphique des interactions entre les acteurs et le système selon un ordre chronologique dans la formulation UML.

### Administrer



Afin de pouvoir administrer l’application, l’utilisateur de Nao se connecte par le biais de sa tablette, smartphone, PC… sur l’adresse du serveur. Il rentre alors son login et son mot de passe. Le système vérifie alors dans la base de données si les informations saisies sont cohérentes. Si le login et le mot de passe administrateur sont erronés, un message d’erreur apparait pour signaler à l’utilisateur qu’il s’est trompé. En revanche, si les identifiants de connexion sont corrects, l’utilisateur a alors accès au panel d’administration de l’application.

### Ajouter fonctionnalités

Dans la partie d’administration de l’application, l’utilisateur peut rajouter des fonctionnalités à la partie utilisateur. Pour cela, il se connecte en tant qu’administrateur sur l’application et clique sur le bouton pour ajouter une fonctionnalité. Le système va alors lister les fonctionnalités déjà existantes (elles sont stockées dans une base de données particulière) et les afficher à l’administrateur. Ce dernier pourra alors en ajouter une nouvelle à l’application, la base de données sera mise à jour et un nouveau bouton va apparaitre dans l’application.

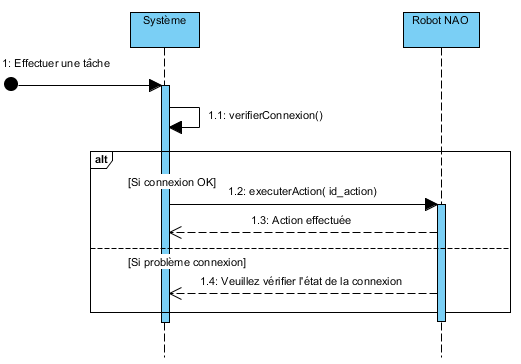
### Consulter Etat Nao

L’utilisateur de Nao peut consulter à tout moment l’état des différents capteurs de Nao via l’application. Le système envoi alors une requête Nao et ce dernier répond en indiquant son niveau de batterie, la force du signal et d’autres informations diverses.

### Démarrer Raspberry

Lorsque l’utilisateur réalise la mise sous tension du Raspberry, cela va provoquer la mise en route du système d’exploitation (Raspbian – Linux). Un batch de démarrage va alors lancer le serveur Apache préconfiguré. Deux cas sont alors possibles : soit le serveur démarre sans erreur et tout est ok. Soit le serveur n’arrive pas à se lancer, il faut alors redémarrer le Raspberry ou bien modifier le batch de lancement qui doit contenir une erreur.

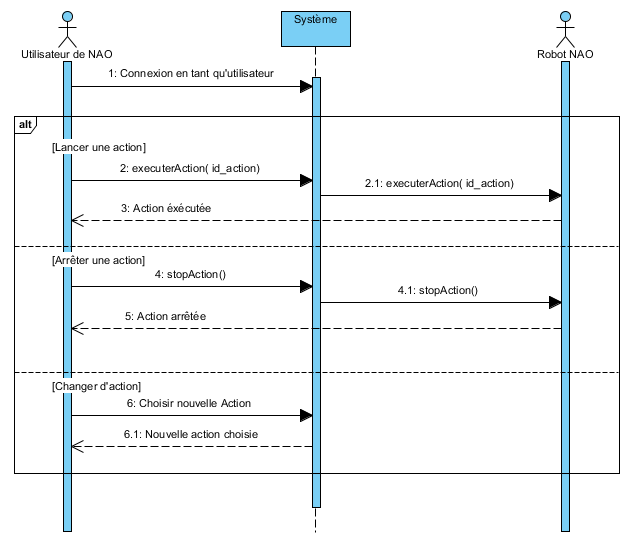
### Effectuer une tâche



Lorsque l’utilisateur souhaite que Nao réalise une tâche, il se connecte sur l’application via le navigateur Web. Il clique alors sur l’un des boutons indiquant l’action. Le système va alors vérifier si la connexion entre le Raspberry Pi et Nao est correcte. Deux cas sont alors possibles : la connexion est réussie, la méthode executerAction est alors appelée, Nao l’exécute alors.

Si la connexion n’est pas établie, un message d’erreur apparait à l’écran pour informer l’utilisateur de bien vouloir vérifier la connexion entre Nao et le Raspberry.

### Gérer Nao

L’utilisateur peut intervenir sur Nao à tout moment pour lui demander de réaliser une action, interrompre une action en cours ou bien demander à Nao de changer d’action. Pour cela, l’utilisateur se connecte sur l’application et il choisit parmi les différents choix qui lui sont proposés.

### Gérer Périphériques

L’utilisateur peut gérer les périphériques pouvant se connecter à Nao. Pour cela, lorsqu’il est connecté en tant qu’administrateur sur l’application, il y a un menu permettant de gérer les périphériques pouvant se connecter à Nao. Il doit alors renseigné le nom du périphérique qu’il souhaite ajouter ainsi que son adresse mac. De plus, ce dernier peut aussi blacklister certains appareils grâce à l’option refuser un périphérique. Toutes ces données sont stockées dans une base de données particulière : la base de données périphérique.

### Se Connecter

Pour qu’il puisse se connecter à l’application, l’utilisateur doit rentrer un couple d’identifiant : login et mot de passe. Si ce couple d’identifiants est erroné après vérification dans la base de données, l’utilisateur ne peut donc pas se connecter sur l’application.

### Vérifier Connexion Raspberry – Nao

Le Raspberry est capable de vérifier si la connexion avec Nao est établie ou non. Pour cela, il récupère l’adresse IP de Nao dans la base de données des périphériques et établie un ping sur cette adresse. Si le Raspberry reçoit une réponse à son ping, cela signifie que Nao est bien connecté sur le réseau Wifi du Raspberry. Sinon, cela affiche un échec de connexion à l’écran.

## Diagramme de déploiement

