MASTER EN SCIENCES INFORMATIQUES

Définition d'un langage de programmation visuelle générique pour la programmation de prototypes de systèmes embarqués

Reiland, Olivier

Award date:
2015

Awarding institution:
Universite de Namur

UNIVERSITE DE NAMUR
Faculté d'informatique
Année académique 2014-2015

Définition d'un langage de programmation visuelle générique pour la programmation de prototypes de systèmes embarqués

Olivier Reiland

Promoteur : _____ (Signature pour approbation du dépôt – REE art. 40)
Vincent Englebert

Co-promoteur : Bruno Dumas

Mémoire présenté en vue de l'obtention du grade de
Master en Sciences Informatiques

RESUME

Le développement de projets électroniques s'est considérablement ouvert au grand public depuis l'apparition de composants électroniques intelligents tels que les Lego Mindstorms, les Phidgets ou encore les composants .Net Gadgeteer. Ces composants sont aisément manipulables par des programmes qui peuvent exploiter les capacités de capteurs et de moteurs sans que l'utilisateur ne dispose pour autant de notions pointues en électronique et en soudures.

La programmation de ces projets est encore aujourd'hui majoritairement faite via des langages de programmation textuels traditionnels qui restent plus difficiles à maîtriser pour des novices en programmation. Des environnements visuels sont apparus mais ne proposent pas tous une programmation visuelle dans laquelle les programmes sont exclusivement écrits par assemblage d'éléments graphiques. De plus, certains langages ne proposent de ne gérer qu'une seule famille technologique de composants. Si la syntaxe visuelle du langage n'est pas suffisamment expressive, le programmeur débutant risque d'éprouver des difficultés et de renoncer facilement à l'apprentissage de ce langage.

Ce mémoire se propose de présenter un modèle de programmation visuelle qui se veut le plus expressif possible que ce soit aussi bien à la compréhension qu'à la mise en place du code et qui permette au débutant de pouvoir réaliser ses projets sans éprouver de frustration due à une syntaxe visuelle complexe. Ce langage se veut générique afin de faciliter la manipulation des composants de diverses familles technologiques au sein d'un même projet.

Mots clés

Lego Mindstorms, Phidgets, .Net Gadgeteer, programmation visuelle, prototypage électronique, syntaxe visuelle générique

ABSTRACT

The development of electronic projects has been considerably spread to the public since the emergence of intelligent electronic components easily manipulated by programs. Those programs are used to exploit the capabilities of sensors and motors without the need for user to dispose of advanced abilities in electronics and welding.

The programming of these projects is still mainly done via traditional text-based programming languages that are more difficult to learn for novices in programming. Visual environments have emerged but many of them do not offer a visual programming in which programs are written exclusively by assembling graphic elements. Moreover some languages do offer to only handle one technological family of components. If the visual syntax of the language is not expressive enough the beginner programmer may experience difficulties and risks to give up learning this language.

This thesis aims to present a visual programming language that aims to be as expressive as possible for the understanding and for the implementation of the code. It allows the beginner to carry out their projects without experiencing frustration due to a complex visual syntax. This language aims to be generic in order to facilitate handling of various technological components families within the same project.

Keywords

Lego Mindstorms, Phidgets, .Net Gadgeteer, visual programming, electronic prototyping, generic visual syntax

REMERCIEMENTS

Je tiens à remercier les Professeurs Vincent Englebert et Bruno Dumas pour le temps qu'ils m'ont consacré, l'aide qu'ils m'ont apportée tout au long du mémoire et pour leur motivation à dispenser bons conseils et éclaircissements lors de la préparation de ce travail.

Je remercie les professeurs de l'Université de Namur pour la qualité de leur enseignement, leur disponibilité et leur professionnalisme.

Je remercie particulièrement ma compagne et mes enfants pour leur support à toutes épreuves dans les bons moments comme dans les moments difficiles.

Je remercie mes amis d'Incubhacker[1], le hackerspace de Namur, pour leurs conseils, leur bonne humeur et pour m'avoir transmis la fibre robotique qui m'a aidé à trouver divers bouts d'inspiration lors de l'écriture de ce travail.

[1] http://www.incubhacker.be/

Table des matières

Table des illustrations

Glossaire

API	« Application Programming Interface », Pilote logiciel propre à une certaine technologie.
GPIO	« Global Purpose Input Output » sont les connecteurs permettant d'ajouter des cartes d'extension aux mini-ordinateurs Raspberry Pi.
RAD	« Rapid Application Development » environnement graphique de programmation rapide d'applications.
DSL	« Domain Specific Language » est un petit langage focalisé sur un domaine applicatif spécifique.
DSML	« Domain Specific Modeling Language » est un langage utilisé pour modéliser un DSL.
GPL	« General Purpose Language » désigne un langage qui n'est pas spécifique à un domaine particulier mais peut convenir à un ensemble de domaines applicatifs différents.
RFID	« radio frequency identification » *est une méthode pour récupérer des données à distance en utilisant des marqueurs appelés ' radio-étiquettes ' ou 'tags rfid'* [2]
PWM	« Pulse Width Modulation » *est une technique d'électronique utilisée pour synthétiser des signaux continus à l'aide de circuits à fonctionnement tout ou rien (à états discrets).* [3] Il est possible de faire varier la vitesse d'un moteur s'il est connecté en PWM.
I2C	« Inter-integrated Circuit » *est un bus de données qui permet de relier facilement un microprocesseur et différents circuits* [4]

[2] https://fr.wikipedia.org/wiki/Radio-identification
[3] https://fr.wikipedia.org/wiki/Modulation_de_largeur_d%27impulsion
[4] https://fr.wikipedia.org/wiki/I2C

1. Introduction

La programmation visuelle est un sujet traité depuis de nombreuses années mais il n'existe cependant aucun cadre formel qui définisse des règles claires et précises de bonnes pratiques dans la conception d'un langage de programmation visuelle. [Moody, 2009] nous recommande toutefois de faire attention à certains aspects lors de la conception d'une syntaxe graphique pour ce genre de langage. L'importance des différentes variables visuelles, comme la forme ou la couleur, constitue l'un de ces aspects.

La robotique est une discipline mélangeant l'informatique et l'électronique. Elle a longtemps été réservée à un public de programmeurs et de férus en électronique. Elle consiste en un assemblage de composants électroniques dans un même projet pour lequel une programmation de ces composants est réalisée. Ceux-ci peuvent soit capturer des valeurs ou traduire une valeur en une modification de l'état d'un composant physique appelé actuateur (moteur).

Avec le temps, certains langages classiques ont évolué vers des langages plus visuels mêlant pour certains une programmation textuelle à une programmation visuelle consistant dans un premier temps en du placement de composants visuels manipulés via des références textuelles dans du code écrit en langage traditionnel.

La robotique a également évolué pour s'étendre à un public plus large et plus jeune. Les composants de base sont toujours utilisés mais des composants plus évolués ne nécessitant ni connaissance en électronique ni soudures sont apparus et avec ceux-ci les langages visuels se sont adaptés pour gérer totalement ou partiellement la programmation graphique exploitant ces composants dans des projets de prototypage électronique.

Des langages sont apparus au sein d'environnements de programmation permettant ainsi d'offrir un contexte complet de travail pour les amateurs de projets de prototypage. La plupart des langages visuels existants permettent seulement une programmation visuelle réservée uniquement à une certaine famille de composants électroniques. Cependant très peu de ces langages permettent de bénéficier dans un même projet d'un large éventail de composants appartenant à différentes familles technologiques sans que l'environnement ne devienne difficile à manipuler, à comprendre et ne donne l'impression de n'être utilisable que par un public de professionnels.

Dans ce travail, nous avons voulu rechercher une syntaxe générique visuelle de composants visuellement expressifs et utilisables quelle que soit la famille technologique des composants électroniques employés de telle sorte que cette syntaxe puisse être utilisable dans un langage qui serait adapté à la compréhension des plus jeunes et des débutants dans les disciplines informatiques et électroniques. Ce langage serait employé avec des composants électroniques évolués tels que ceux présentés dans l'état de l'art de ce travail.

La méthode suivie pour la réalisation de ce travail a impliqué d'établir en premier lieu un état de l'art. Sur base de cet état de l'art, nous avons appliqué le mode opératoire expliqué au chapitre 3.

2. Etat de l'art

2.1 Présentation

Cet état de l'art va présenter les différentes architectures de composants électroniques ainsi que les différents langages de programmation visuelle pouvant être utilisés dans la mise en œuvre de projets de prototypages. Il vous donnera un aperçu des points forts et des faiblesses de ces langages et mettra en évidence les lacunes et les manquements dans les fonctionnalités et dans l'utilisabilité de ceux-ci.

Sur base de cet aperçu, une piste sera élaborée afin de dégager des améliorations possibles dans l'expressivité des concepts fondamentaux de ces langages, tenter de diminuer les points négatifs de ceux-ci et proposer des solutions permettant une meilleure prise en main de ce genre de langages de développement en particulier pour un public de jeunes utilisateurs.

2.2 Le prototypage

La robotique est une discipline faisant appel à l'électronique et à la programmation. Robotique et prototypage sont des concepts liés.

Le prototypage électronique de bas niveau est la réalisation d'un projet de prototype électronique qui consiste souvent à mettre en relation une carte contrôleur, des capteurs et des déclencheurs afin de réaliser des actions en réaction à des évènements évalués par ces capteurs. Outre la carte contrôleur, on retrouve également des plaques pré-trouées où « breadboard » dans les projets de prototypage qui permettent d'agencer des câbles et divers composants sans avoir à effectuer de soudure. Une fois qu'un prototype est fonctionnel, on peut passer à l'étape de création du circuit imprimé et des soudures. La figure 1 présente un exemplaire de plaque pré-trouée de prototypage.

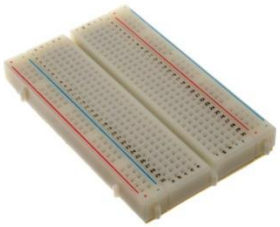


Figure 1 : Breadboard de prototypage

Les langages utilisés pour la programmation de ces prototypes sont souvent Python ou le langage C.

Selon le projet choisi, le cœur de l'application va fonctionner soit sur un ordinateur, soit via le téléversement d'un programme généralement compilé dans une puce sur un circuit embarqué ou dans un boîtier intelligent lié à la technologie du capteur.

En fonction des connaissances de l'utilisateur et de la complexité du projet, celui-ci va parfois préférer un ensemble de composants qui nécessite plus de base en électricité ou en électronique (Arduino, Raspberry Pi [5]) ou utiliser les composants électroniques intelligents d' autres systèmes pour lesquels il va pouvoir s'affranchir de connaissances en électronique et sera alors à même de se focaliser sur le

[5] https://www.raspberrypi.org/help/what-is-a-raspberry-pi/

cœur logiciel de son projet de prototypage (Phidgets, Lego Mindstorms NXT ou EV3, FlowPaw, Vex IQ , .Net Gadgeteer ou encore les composants Flotilla).

Le prototypage ludique ou éducatif permet de réaliser des prototypes par assemblage de composants électroniques intelligents autour d'un contrôleur sans qu'il n'y ait besoin de soudures et de connaissances en électronique.

Un projet de robotique est caractérisé par un ensemble de composants physiques. On retrouve par exemple des capteurs et des actuateurs qui sont agencés pour les besoins d'un projet particulier et qui par l'intermédiaire d'un autre composant dit "gestionnaire" ou "contrôleur" vont permettre à l'application du prototype de transmettre et de recevoir des informations.

En effet, une carte contrôleur dans un projet de robotique est une carte qui est responsable de la gestion des entrées et des sorties de données de et vers cette carte. Selon la technologie utilisée, un programme compilé sera injecté dans un microcontrôleur ou sera simplement exécuté sur le mini-ordinateur. Les cartes Arduino et Raspberry Pi sont des cartes très utilisées dans les projets de prototypage. Ces cartes vous sont présentées dans la figure 2.

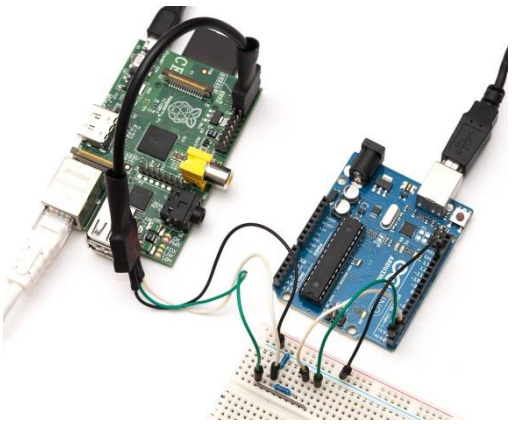


Figure 2 : Exemple de projet avec Raspbery Pi, Arduino et une breadboard

«Arduino est une plateforme open-source d'électronique basée sur des logiciels et matériels spécifiques à une prise en main facile. Arduino ressent l'environnement par la réception d'informations provenant de plusieurs capteurs et affecte son environnement proche par le contrôle de lumières, de moteurs et d'autres actuateurs».[6]

Il existe plusieurs modèles de cartes Arduino. Ces cartes ont différentes tailles, sont équipées de microprocesseurs aux puissances diverses et disposent d'un certain nombre de connecteurs métalliques appelés « pin » sur lesquels se dérouleront les entrées et sorties d'informations analogiques ou digitales de et vers ces cartes. Le microprocesseur est le cœur d'une carte contrôleur. Nous devrons préciser à la carte Arduino ce qu'elle doit faire par la réalisation d'un programme spécifique qui sera ensuite compilé pour être injecté dans le microcontrôleur de cette même carte.

Il est possible d'étendre les fonctionnalités d'une carte Arduino par l'ajout d'autres mini-cartes, appelées «shield», comportant des composants et des circuits imprimés. Ces cartes d'extensions

[6] https://www.arduino.cc/

apportent alors à la carte Arduino la possibilité de pouvoir contrôler divers nouveaux capteurs et/ou actuateurs. Une liste de différents shields peut être vue sur le site de « generationRobots ».[7]

Les cartes Arduino et les composants de robotique bon marché associés permettent d'accéder au monde du prototypage et de la robotique mais cela nécessite déjà des notions en électronique sans lesquelles l'utilisateur hasardeux débutant et donc non-averti encourt le risque de griller ses composants si des contraintes ne sont pas respectée. Par exemple, le montage d'un circuit électronique contenant une diode mais ne contenant aucune résistance pour la protéger sera considéré comme à risque car la diode ne sera pas protégée et aura dès lors de très fortes chances de griller lorsque le courant la traversera.

Le Raspberry Pi est un micro-ordinateur sur lequel est installé un système d'exploitation Linux [8] qui permet de réaliser et de piloter divers projets de prototypage par l'utilisation de ses connecteurs électriques appelés GPIO (General-Purpose Input Output), par l'exploitation de ses ports USB ainsi que par l'ajout de cartes d'extensions qui se positionnent sur les broches GPIO et qui permettront d'étendre les fonctionnalités de base du Raspberry Pi. Des shields existent aussi pour le Raspberry Pi afin d'étendre les capacités de ce mini-ordinateur.

A la différence des cartes Arduino, un programme écrit pour un projet de prototypage Raspberry Pi ne sera pas injecté dans un microcontrôleur mais il sera néanmoins compilé pour y être exécuté. Raspberry Pi étant un ordinateur sur lequel un service d'accès distant peut-être installé, il pourra être accédé de manière distante par l'utilisateur afin que ce dernier lance de nouveaux processus ou récupère diverses informations en provenance de capteurs placés dans l'application robotique. On retrouve de nombreux projets associant un Raspberry Pi et une carte Arduino. Un exemple de ces projets vous a été présenté dans la figure 2.

Le téléversement est le fait d'injecter un programme, compilé pour être exécuté par un microprocesseur bien précis, dans le microcontrôleur d'une unité de contrôle pour projets de robotique. Une fois ce code téléversé, l'unité de contrôle réceptrice sera autonome et se comportera de la façon décrite par le programme téléversé. Ce principe est employé pour communiquer la logique de programmation aux briques programmables Lego Mindstorms[9] ainsi qu'aux cartes Arduino (technologie plus difficile pour les projets de robotique mais permettant la réalisation de projets robotiques en restant très proche de la manipulation de composants).

Les utilisateurs d'Arduino utilisent régulièrement le logiciel propre à Arduino afin d'écrire leur solution logicielle, de la compiler et de la charger dans le contrôleur de la carte. L'extension Ardublock[10] est utilisée pour étendre les fonctionnalités de base du logiciel propre d'Arduino et ainsi permettre une programmation plus graphique pour les programmes devant exploiter cette carte.

[7] http://www.generationrobots.com/fr/174-shield-arduino
[8] https://fr.wikipedia.org/wiki/Linux
[9] http://www.lego.com/en-us/mindstorms/?domainredir=mindstorms.lego.com
[10] http://blog.ardublock.com/

2.3 Les familles de composants électroniques évolués

2.3.1 Les composants Phidgets

Les composants Phidgets sont des petits circuits électroniques pleinement opérationnels et dédiés à des expériences particulières comme la prise de température, la détection de distance, la mise en route de servomoteurs, la détection de pression sur une plaque ou encore la reconnaissance de matériel RFID. La société qui développe ces composants est Phidgets Inc©[11].

Ces éléments sont principalement utilisés afin d'étendre les interactions homme-machine en recueillant des informations obtenues par des interactions avec le monde extérieur, par des captations d'évènements et en exprimant un comportement propre en réaction dans ce même monde extérieur via le contrôle d'activateurs tels que des moteurs et/ou des lumières. L'environnement Phidgets dispose donc d'un ensemble de composants dédiés pouvant être utilisés au choix et en fonction du but du projet de robotique que l'on désire réaliser. Chaque Phidget est adressable dans le code du programme via un numéro de série qui permet de l'identifier. Un exemple de ce type de composant est présenté à la figure 3.

Figure 3 : Carte Phidget GPS avec connectique USB

Cet écosystème permet d'affranchir l'utilisateur des notions de base en électronique et lui permet de se concentrer sur la problématique logicielle de son projet plutôt que sur le côté technique de celui-ci et donc de privilégier la conception et la programmation. En effet, les utilisateurs n'ont pas besoin de manipuler les composants de base ni de devoir effectuer des soudures ou encore d'avoir à s'occuper de la mise à jour de programmes dans des microcontrôleurs lorsqu'il s'agit d'exécuter l'application relative à leur projet d'électronique.

Chaque ajout d'un composant ne nécessite tout au plus qu'une simple connexion de la fiche du composant sur l'un des ports USB d'un ordinateur ou sur une carte contrôleur de Phidgets, appelée « bridge » ou « interface kit » , associée au projet de prototypage et elle-même connectée à l'un des ports USB de l'ordinateur. Cette carte contrôleur agit comme un gestionnaire de Phidgets et représente le point de départ d'un prototype basé sur une architecture de ces composants.

La figure 4 présente une carte contrôleur Phidgets sur laquelle est connecté un composant Phidget capteur.

[11] http://www.phidgets.com/

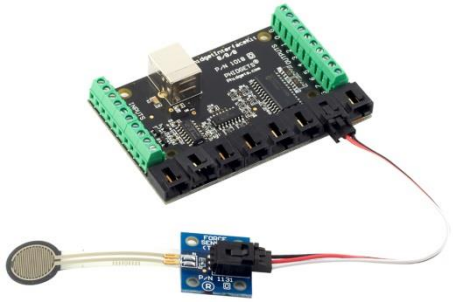


Figure 4 : Phidget capteur de force connecté à une carte Interface Kit (Bridge)

Il est également possible d'étendre la connectique Phidgets avec des composants tiers bon-marché souvent utilisés pour des projets d'électronique ludique tels que des capteurs, des lumières LED, des bras articulés, des servomoteurs, des afficheurs LCD, des afficheurs 7 Segments ou encore des caméras. La figure 5 présente une carte Phidget sur laquelle sont connectés 4 servomoteurs.

Figure 5 : carte Phidget avec plusieurs servomoteurs connectés

De par leur faible consommation électrique, l'emploi de mini-ordinateurs tels que le Raspberry Pi © facilite la portabilité de certains projets Phidgets. Il suffit que les pilotes Phidgets soient compilés et renseignés auprès du système d'exploitation du Raspberry Pi (Raspbian[12], un système d'exploitation de la famille Debian sous Linux, par exemple) pour que ce dernier puisse faire fonctionner des programmes permettant d'envoyer, de recevoir et d'interpréter des informations de et vers les divers capteurs.

Selon le modèle de Raspberry Pi choisi et selon les éléments connectés, la consommation électrique de ce petit appareil va débuter à partir de 200 mAh (milliampères par heure) pour un modèle A+. Si on le combine avec une batterie portable, comme celles utilisées pour recharger des téléphones portables, on peut faire fonctionner le prototype pendant une durée de plus ou moins dix heures avec un boîtier d'une capacité de 2200 mAh.

Sachant que la consommation peut varier mais également qu'il existe des batteries portables allant jusqu'à plus de 22000 mAh, on peut alors espérer une forte portabilité des solutions de prototypage exploitant la technologie des Raspberry Pi. La figure 6 présente une carte Raspberry Pi sur laquelle est connectée une carte Phidget « lecteur » de badges RFID ainsi que deux ampoules LED (composants tiers).

[12] https://www.raspbian.org/

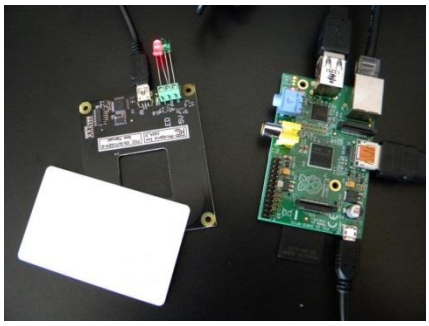


Figure 6 : Carte Phidget RFID (à gauche) connectée à un ordinateur Raspberry Pi (à droite)

L'emploi de l'écosystème Phidgets en conjonction avec des outils de développement visuel va permettre de mettre des projets de robotique dans les mains d'un large public qui, par le biais d'une programmation graphique se voulant aisée et intuitive, va permettre d'initier et de faire comprendre les bases de la programmation aux plus jeunes ainsi qu'aux débutants en électronique et en robotique. Grâce à la mise à disposition de nombreux pilotes dédiés à la technologie des Phidgets, ces composants sont directement manipulables par du code pouvant être écrit dans un nombre important de langages de programmation ainsi que par diverses interfaces graphiques d'environnements de programmation permettant de les manipuler. Les Phidgets peuvent, de cette façon, être utilisés via les systèmes d'exploitation classiques Windows, Mac OSX ou Linux.

API signifie "interface de programmation applicative"; c'est par le biais de fonctions contenues dans des librairies d'API que les programmes peuvent recevoir et envoyer des informations au départ et vers les composants Phidget. A un plus bas niveau, c'est ce qui permet au système d'interagir avec ces composants. Selon Wikipédia, les langages suivants disposent de libraires (API) permettant de commander et recevoir des informations de et vers les Phidgets[13]. Ces langages sont : [14]

- Adobe Director
- Autolt
- C# (sous Visual Studio)
- C, C++
- Cocoa
- Delphi
- Flash AS3
- Flex AS3
- Java
- MATLAB
- Python Modules
- REAL Basic
- Visual Basic .NET (sous visual studio)
- Visual Basic 6.0, Visual Basic for Applications VBA, Visual Basic Script
- Visual C/C++/Borland
- Max6 (Max/MSP)
- Microsoft Robotics Studio (aujourd'hui déprécié)
- **Ni LabView**
- **FlowStone (Flowbotics Studio)**

La plupart de ces langages ne permettent pas de réaliser de programmation graphique mais seulement de réaliser une programmation textuelle traditionnelle. Nous discuterons par après des langages de programmation graphique que sont Ni LabView et Flowstone.

[13] http://www.phidgets.com/
[14] https://en.wikipedia.org/wiki/Phidget#Phidget

2.3.2 Les composants Lego Mindstorms

L'intermédiaire physique principal de tout projet Lego Mindstorms est la P-Brick ou brique de programmation. Cette brique, représentée dans la figure 7, est le composant principal et le cerveau contrôleur des projets Lego Mindstorms. Les différents capteurs et moteurs du projet sont reliés à cette brique via les ports dédiés de celle-ci. La P-Brick dispose de toute une connectique propre ainsi que de ses propres actuateurs intégrés comme par exemple un écran, un haut-parleur et 4 boutons rétro-éclairés programmables. Chaque élément pouvant se retrouver intégré dans les flux de comportement de l'application via l'utilisation de blocs d'action s'y rapportant.

Figure 7 : La brique programmable Lego Mindstorms EV3

Lorsqu'un de ces capteurs est connectés à la brique, il est repéré et affiché de telle sorte que l'utilisateur, dans l'environnement de développement de Lego Mindstorms EV3, ne perde pas de vue quel élément est connecté et à quel port de la brique. Il est possible de connecter plusieurs briques en série. Les différentes briques seront alors automatiquement identifiées dans le code visuel du programme. Par exemple, la valeur 3 qui définirait un port de moteur connecté à une première brique pourrait également s'écrire 103. Le programme comprend alors que l'on traite avec le port 3 de la brique 1.

Les composants Lego Mindstorms se basent sur un concept similaire à celui des Phidgets mis à part que toute la logique du programme doit être au final téléchargée sur la brique intelligente P-Brick afin que cette brique se comporte en cerveau intelligent pour le projet en développement et puisse alors gérer les composants Lego Mindstorms© selon la manière décrite dans le programme réalisé. La programmation de ces composants peut être réalisée par le biais de différents langages de programmation mais il n'existe qu'un seul langage graphique dédié : EV3-G dans l'environnement de développement Lego Mindstorms EV3.

Il est à noter que l'environnement de développement de Lego Mindstorms EV3 comporte notamment une zone représentant la brique programmable dans sa partie inférieure droite, une zone présentant la palette des composants dans sa partie inférieure gauche et une zone centrale dans laquelle l'utilisateur peut programmer dans le langage de programmation EV3-G. La figure 11 présente l'environnement de développement Mindstorms avec un exemple de programme dans sa partie centrale.

Les composants Lego Mindstorms peuvent être vus de deux manières :

- Les éléments physiques : la brique programmable (afficheur, haut-parleur, boutons), les moteurs et capteurs connectés à la brique ainsi que les câbles permettant de relier chaque composant à la brique ou une brique à une autre. On peut observer ces capteurs et actuateurs dans la figure 8.

- Les éléments logiques dédiés aux actions sur la brique programmable, sur les moteurs, à la récupération des données obtenues par les capteurs ou encore à l'utilisation de concepts avancés comme par exemple l'exploitation d'une connexion Bluetooth dans un programme la nécessitant.

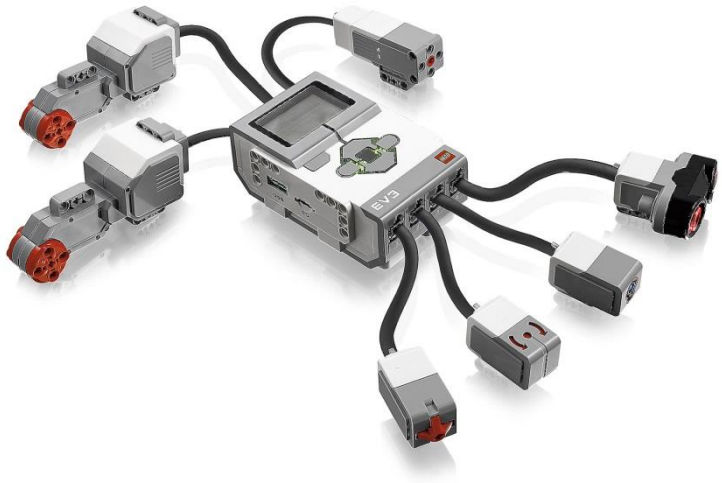


Figure 8 : Capteurs et actuateurs connectés à une brique Lego Mindstorms EV3

Les composants de l'ancienne génération Mindstorms, les NXT, sont encore utilisés pour des projets de robotique et disposent de leur propre application à présent désuète mais similaire à celle utilisée pour la technologie EV3.

2.3.3 Les composants .Net Gadgeteer

Les composants .Net Gadgeteer, dont l'un d'eux est présenté en figure 9, sont des composants similaires aux composants Phidgets mais qui exploitent une technologie concurrente (circuits électroniques, câblages, cartes contrôleurs, microprocesseurs et API de programmation différentes). Aucune soudure n'est requise pour l'utilisation de la technologie Gadgeteer. Leur mise en œuvre se fait soit par l'utilisation de langages de programmation tels que C# et Visual Basic .Net proposés sous l'environnement Microsoft Visual Studio avec le plugin « .Net Gadgeteer », soit avec le seul langage de programmation visuelle pouvant être utilisé avec ces composants dans le cadre d'une programmation graphique : le langage Flowstone sous l'environnement de programmation Flowbotics Studio.

Figure 9 : Potentiomètre .Net Gadgeteer (résistance variable)

L'architecture Gadgeteer nécessite l'utilisation d'un élément central faisant fonction de composant contrôleur ou gestionnaire de composants auquel se connecteront les différents éléments capteurs et actuateurs Gadgeteer (capteur de distance, moteurs, etc…). La carte GHI Fez Spider mainboard, présentée dans la figure 10, est l'une de ces cartes contrôleur. Elle exploite le .Net Micro Framework de Microsoft [15] pour l'exécution de programmes réalisés via les langages C# et Visual basic .Net.[16] Le Micro framework permet aux développeurs de bénéficier de la technologie .Net adaptée pour la programmation de petits composants électroniques.

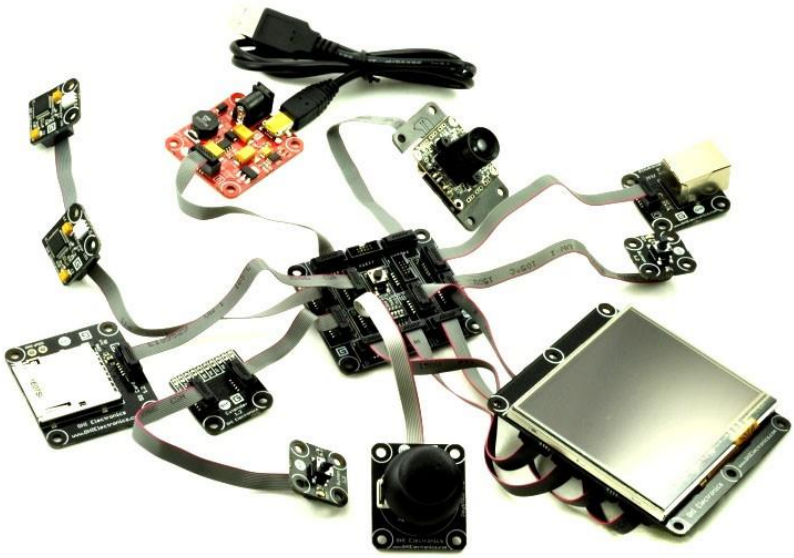


Figure 10 : Composants .Net Gadgeteer connectés à une carte GHI Fez Spider board

[15] https://www.ghielectronics.com/technologies/netmf
[16] https://www.ghielectronics.com/catalog/product/432

Tout comme l'infrastructure Phidgets, il est possible d'étendre les fonctionnalités du système par l'ajout de composants tiers servant pour les projets d'électronique bon-marché tels que les projets Arduino et / ou Raspberry Pi. Les composants Gadgeteer constituent une alternative intéressante et plus aisée que celle rencontrée dans les projets classiques de prototypage (Arduino) ; leur utilisation est similaire à celle des Phidgets pour ce qui concerne la connexion de composants aux cartes contrôleurs.

Il est à noter que tous les composants Gadgeteer ne peuvent pas fonctionner sur n'importe quelle carte contrôleur Gadgeteer. En effet, seulement un sous-ensemble de composants Gadgeteer peut être connecté sur une carte contrôleur car il peut y avoir certains soucis (différences de type de courant entre cartes, différences de firmwares, etc...). De plus les différents connecteurs d'une carte contrôleur ne permettent pas de connecter n'importe quel élément n'importe où sur la carte. Les éléments à connecter doivent appartenir à certaines familles technologiques telles qu'I2C ou PWM. Certains modules ne peuvent être uniquement connectés que sur certaines cartes de certains fabricants.[17]

[17] http://forums.netduino.com/index.php?/topic/4055-will-all-of-the-gadgeteer-modules-work-with-the-netduino-go/

2.3.4 Les autres familles de composants

Il existe d'autres familles de composants pour lesquelles on retrouve le même principe général de connectique : une carte principale contrôleur sur laquelle viennent se connecter les composants choisis pour des projets de robotique.

Les composants Vex IQ représentent une technologie proche de celle des Lego Mindstorms EV3 et peuvent être gérés par des projets réalisés sous l'environnement / langage de programmation ModKit micro for Vex.

Les composants FlowPaw représentent une technologie similaire à celle des .Net Gadgeteer et peuvent être intégrés dans des projets développés via le langage Flowstone sous l'environnement Flowbotics Studio.

Les composants Flotilla représentent une technologie similaire à celle des Phidgets. Ils seront commercialisés d'ici le mois d'Octobre 2015.

Etant donné que tous ces composants ont de fortes similitudes avec les composants déjà présentés, nous ne les détaillerons pas plus dans ce travail.

- Vex IQ (http://www.vexrobotics.com)
- FlowPaw (http://www.flowpaw.com)
- Flotilla (http://shop.pimoroni.com)

2.4 La programmation visuelle

Selon la définition de Wikipédia, « Un langage de programmation graphique ou visuelle est un langage de programmation dans lequel les programmes sont écrits par assemblage d'éléments graphiques. Sa syntaxe concrète est composée de symboles graphiques et de textes qui sont disposés spatialement pour former des programmes. Généralement un langage de programmation est associé à un environnement graphique de programmation. Il n'est pas toujours possible de les dissocier ». [18]

Selon [Slany, 2012], "La programmation visuelle , ce n'est pas simplement jeter et déposer des blocs dans l'interface mais plutôt une affaire de motivation qui est obtenue par le fait d'éviter des frustrations dues à des structures syntaxiques compliquées dans le flux d'exécution des programmes ou à des erreurs difficilement identifiables comme souvent dans les principaux langages de programmation".

« Un langage dans un environnement de développement visuel rapide (RAD, Rapid Application Development) est un générateur de code source en langage de base qui derrière chaque action visuelle (dépôt de contrôle, clic de souris, modifications des propriétés, etc...) engendre des lignes de code automatiquement et d'une manière transparente pour le programmeur ». [19]

Selon [Marttila-Kontio, 2011], «un langage de programmation visuelle se compose d'un environnement visuel et d'une syntaxe visuelle ».

En général, un composant visuel représente un capteur, une action sur un composant, un comportement, une redirection du flux ou un traitement de données. La programmation visuelle a l'avantage de donner une vision plus explicite du développement à l'utilisateur.

Les langages de programmation visuelle permettent d'élaborer des programmes en disposant des blocs les uns à la suite des autres, en spécifiant des propriétés et occasionnellement en plaçant du code personnalisé pour ces composants de telle manière à canaliser le flux de traitement d'informations et ainsi à pouvoir interagir avec des éléments perceptibles dans le monde extérieur comme des activations de matériaux électroniques connectés tels que des moteurs et cela en réaction à un événement reçu par des capteurs disposés dans l'architecture du projet de prototypage.

Certains de ces langages permettent de mettre en évidence le code caché derrière la boîte visuelle et donc d'éditer ce code comme par exemple dans le langage Flowstone (langage de l'environnement Flowbotics Studio) avec Ruby, le langage textuel de scripts, utilisé pour créer de nouvelles boîtes ou pour éditer le comportement d'une boîte déjà existante. Pour d'autres langages visuels, ces boîtes visuelles sont des boîtes noires où il n'est possible de changer que certaines propriétés de celles-ci. Les boîtes de ces langages sont une encapsulation d'un programme de plus bas niveau proche des fonctions de pilotage de base de ces composants (API).

La programmation visuelle convient parfaitement pour des utilisateurs non avertis. Elle permet de développer l'intérêt et la curiosité des plus jeunes pour les technologies de l'informatique.

Certains langages de développement sont inspirés du modèle "pipe and filter", un modèle d'architecture logicielle conçu par Ken Thomson et provenant du monde Unix dans lequel les filtres symbolisent le traitement d'une information en provenance d'une fonction ou « Pipe » pour en sortir

[18] https://fr.wikipedia.org/wiki/Langage_graphique
[19] http://rmdiscala.developpez.com/cours/LesChapitres.html/Cours5/Chap5.4.htm

une autre information qui sera redirigée sur un autre fil ; l'utilisateur relie en fait les connecteurs entre les blocs utilisés dans son application. [20]

Il existe différents langages de programmation visuelle pour composants physiques. Certains langages ne s'adressent qu'à une famille particulière de composants : Scratch / ModKit micro pour Arduino, Lego Mindstorms EV3-G ou encore ModKit micro pour Vex IQ robotics. D'autres sont spécifiques à des solutions intégrées de robotique telle que celle pour le robot NAO (Choregraphe).

Selon [Vasek, 2012], "Les deux formes les plus communes de langages de programmation visuelle sont les langages par flux (dataflow) et les langages de programmation par blocs".

Il est à noter que le terme « programmation visuelle » a été employé par erreur dans le passé pour désigner des environnements de programmation qui permettaient de placer des composants graphiques d'une palette de composants vers un espace de travail et qui nécessitaient absolument une programmation textuelle manuelle pour une grosse partie du codage des applications à réaliser. Ces composants servent en fait d'aide à la programmation dans ces projets pour lesquels une référence s'y référant est manipulée dans le code des programmes. (Visual Basic 6[21], .Net Gadgeteer (plugin pour Visual studio)[22]). Ce type de programmation n'est en rien une programmation graphique ou visuelle.

En effet, comme le confirme [Marttila-Kontio, 2011], "un langage de programmation visuelle est souvent confondu comme étant synonyme d'un langage qui a un environnement de programmation visuelle alors qu'en fait, bien qu'il ait un environnement de programmation, il signifie surtout un langage qui consiste en une syntaxe visuelle".

Selon [Marttila-Kontio, 2011], l'un des gros problèmes des langages de programmation visuelle est le scaling-up problem ou problème de mise à l'échelle. « L'abstraction visuelle est la solution au problème de mise à l'échelle qu'ont les langages de programmation visuelle».

[Hils, 1992] nous parle d'abstraction procédurale et nous propose une solution similaire pour les langages par flux (dataflow) : « un graphe entier peut être considéré comme une procédure et peut être condensé dans un simple nœud ».

Selon [Koitz et Slany, 2014], "Bien que les langages de programmation visuelle diminuent la charge syntaxique et donc permettent aux enfants et aux adolescents de programmer, les bénéfices ne semblent pas être très apparents pour ce qui concerne la composition de formules mathématiques ... mais une approche hybride avec du texte semble être le bon choix".

[20] http://www.dossier-andreas.net/software_architecture/pipe_and_filter.html
[21] https://fr.wikipedia.org/wiki/Visual_Basic
[22] http://research.microsoft.com/en-us/projects/gadgeteer/

2.5 Les langages de programmation visuelle

2.5.1 Mindstorms EV3-G (Environment Mindstorms EV3)

Le langage de programmation visuelle Lego Mindstorms EV3-G (langage de l'environnement Lego Mindstorms EV3) permet de programmer les composants Mindstorms de manière facile, intuitive et efficace. Ce langage basé sur NI LabView[23] est dédié à la technologie EV3 (Troisième évolution des composants Lego Mindstorms) est cependant bien visuellement différent de NI LabView et permet de réaliser une vraie programmation visuelle en n'utilisant quasi exclusivement que la souris et en se servant de boîtes décrivant des actions, des concepts de contrôle de flux ou des comportements.

D'autres langages permettent également la programmation de ces composants de manière visuelle en exploitant les API fournies par Lego. Flowstone[24], utilisé sous l'environnement Flowbotics Studio[25], est l'un de ces langages. Le langage Mindstorms EV3-G est lui uniquement ciblé pour ne contrôler que des éléments Lego Mindstorms de la technologie EV3.

La zone de travail de l'environnement Mindstorms EV3 est très épurée et se compose de l'écran principal au centre, de la zone des composants en bas ainsi que de la zone informant l'utilisateur de l'état des connectivités de l'infrastructure physique Mindstorms avec l'ordinateur sur lequel s'exécute cet environnement de développement. Les connexions à la brique programmable y sont, de cette manière, représentées.

Une fois le nom du projet défini, le reste du développement de l'application peut entièrement être réalisé par l'usage unique de la souris. Occasionnellement l'utilisation du clavier permet de régler plus précisément les valeurs des paramètres lorsque par exemple l'utilisation d'un curseur vertical ne permet pas de préciser facilement une valeur spécifique.

Le corps du programme est constitué d'un ensemble de flux de comportements constitués de boîtes représentant des capteurs, des moteurs ou des modifications de flux comme par exemple les constructions itératives et conditionnelles. La programmation est en fait réalisée par sélection d'une catégorie de boîtes et ensuite par placement d'instances de boîtes de ces catégories dans l'une des lignes de flux du programme Mindstorms.

Les dessins des blocs du programme, tout comme les codes couleurs utilisés pour les catégoriser, se veulent explicites. Les propriétés de ces composants peuvent être éditées de façon directe dans les composants eux-mêmes ; des petites bulles de messages informent l'utilisateur de la nature des options lorsque ces dernières sont survolées par la souris.

En effet, des couleurs sont employées pour la classification des groupes de boîtes et permettent de bien répertorier celles-ci parmi les diverses catégories suivantes :

- Les blocs d'actions (**verts**) dans lesquels on retrouve les blocs de gestion de moteurs, de gestion de l'affichage sur la P-Brick et de gestion sonore,
- Les blocs de contrôle de flux de programmation (blocs **orange**),
- Les blocs de représentation des capteurs (blocs **jaune**),
- Les blocs responsables des opérations sur les données (blocs **rouge**),

[23] http://www.ni.com/labview/f/
[24] http://www.dsprobotics.com/flowstone.html
[25] http://www.flowbotics.com/

- Les blocs permettant d'effectuer une programmation plus avancée (blocs **bleu**) comme par exemple l'accès aux fichiers, la messagerie ou encore la gestion de connexions en Bluetooth,
- Les blocs personnalisables où MyBlocks (blocs **cyan**) qui permettent de créer de nouveaux blocs.

Le langage Mindstorms EV3-G n'autorise aucune édition du code des boîtes. Il est seulement possible de modifier les propriétés visuelles des divers blocs du programme. L'interface graphique permet de relier une valeur lue par un composant capteur et de créer un lien entre ce composant et une autre boîte ayant le rôle d'opérateur mathématique ou de sélecteur de flux selon la valeur qui lui est transmise.

L'écran principal de l'application peut devenir rapidement brouillon au vu du nombre grandissant de composants. L'utilisateur a alors la possibilité de créer ses propres composants (blocs MyBlock de couleur Cyan) et de factoriser de cette façon les parties de traitement occupant beaucoup de place à l'écran. Ces éléments personnalisés pourront alors être réutilisés dans de nouveaux projets.

Ce langage de programmation visuelle est assez intuitif et permet une prise en main de la technologie en un temps très court. Néanmoins un certain temps d'adaptation est nécessaire afin de bien comprendre les divers modes de fonctionnement de chaque brique ainsi que les diverses interactions possibles entres les composants visuels. Le fait de disposer de tutoriaux bien fournis permet de s'immerger plus facilement dans ce langage de programmation et permet de s'assurer une prise en main progressive de la technologie Lego Mindstorms EV3.

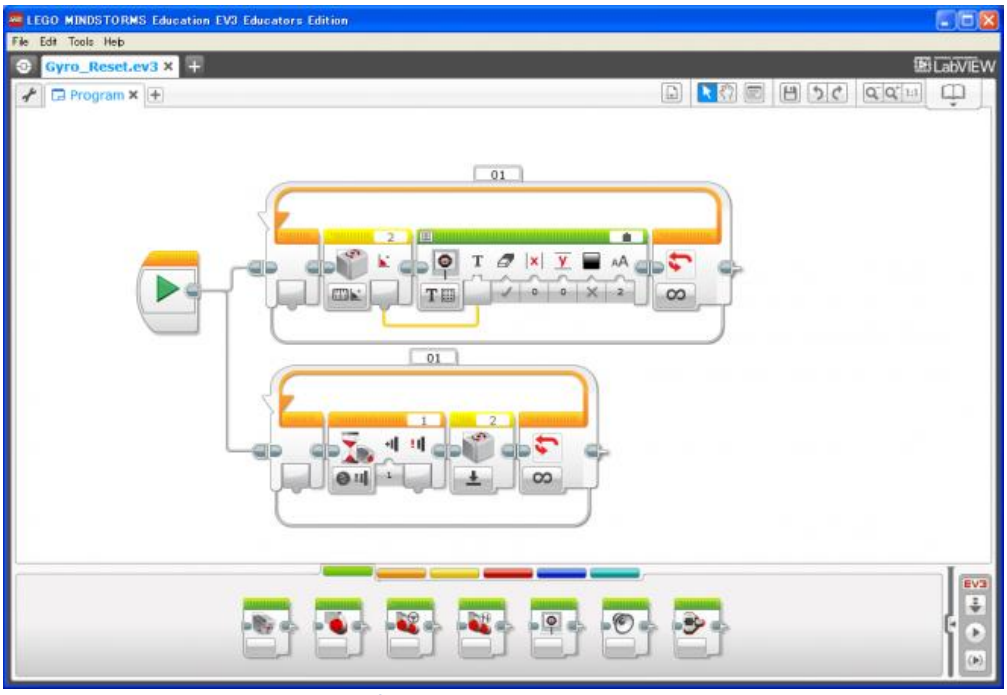


Figure 11 : Environnement / langage de développement Lego Mindstorms EV3-G

2.5.2 Scratch (Environnement Scratch)

Scratch[26] , présenté dans la figure 12, est un environnement / langage de programmation visuelle dans lequel on glisse les composants de structure, de contrôle de flux, d'instructions et de mathématique du panneau reprenant l'ensemble des boîtes disponibles jusque dans le centre de l'interface afin d'y écrire le code du programme qui sera ensuite transformé en Python[27]. Le langage Scratch est en fait l'assemblage des différents blocs dans le centre de l'interface graphique. Toute l'activité de développement dans l'environnement Scratch se fait par le biais de la souris.

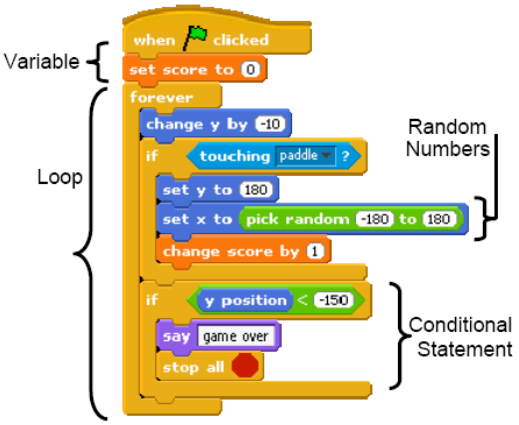


Figure 12 : langage de développement Scratch

Ce style de langage et d'environnement de programmation favorise l'écriture de code mais est malheureusement impuissant quant aux mauvais montages électroniques réalisés par l'utilisateur dans son projet de prototypage matériel associé ; il est cependant utilisable pour faciliter l'écriture du code pour des projets de prototypage Arduino et/ou Raspberry Pi et cela via des variantes dédiées pour Arduino et Raspberry Pi.

En effet, des environnements similaires dédiés à l'électronique existent aussi : ArduBlock (plugin de l'interface utilisateur standard d'arduino est dédié à une programmation visuelle de type Scratch pour les projets de robotique), Catroid ou ModKit Micro (décliné en 2 variantes : Arduino ou Vex IQ robotics).

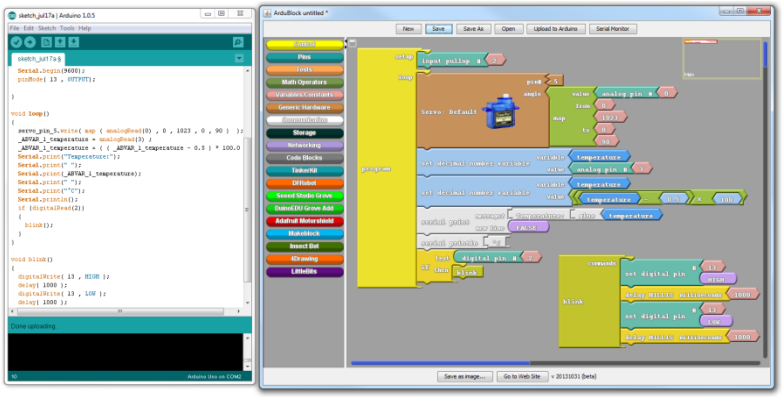


Figure 13 : Extension logicielle d'arduino : ArduBlock

[26] https://scratch.mit.edu/
[27] https://www.python.org/

Selon [Slany, 2012], "Catroid, basé sur Scratch, permet également de contrôler à distance des composants matériels externes tels que des robots Lego Mindstorms via Bluetooth, des carte arduino Bluetooth ou des drones populaires comme le quadcopter de AR.Drone via une connection Wifi".

Le langage Scratch est souvent employé pour faire découvrir la programmation aux plus jeunes ; il permet de manipuler du son, de l'image et de faire des animations. Scratch est un langage de programmation par blocs de type « Event-driven » ou « conditionné par des évènements ». Ardublock, présenté dans la figure 13, propose une syntaxe visuelle similaire à celle proposée par le langage Scratch et permet de gérer les évènements de composants électroniques tangibles connectés à une carte Arduino.

L'environnement Scratch dispose tout comme Lego Mindstorms EV3-G d'un ensemble de blocs répartis en diverses catégories. Certaines catégories de Scratch peuvent être différentes selon le support ou le système d'exploitation sur lequel est exécuté l'environnement de programmation. Les catégories « capteur » et « senseurs » sont habituellement celles qui sont affectées par l'emplacement où s'exécute l'environnement Scratch (Windows, Linux, Android ou via la version interactive web).

L'une des principales forces de Scratch réside dans le fait que chaque bloc dispose d'une forme particulière jouant le rôle de détrompeur visuel qui permet de faire comprendre facilement aux utilisateurs que tel ou tel emboitement inter-blocs ou intra-bloc est impossible. La figure 14 présente un aperçu de l'environnement / langage Scratch.

Figure 14 : Environnement / langage de développement Scratch

Comme le souligne [Vasek, 2012], Scratch représente ses types de données en utilisant des formes différentes pour chaque type de donnée. Il explique notamment que les fonctions, retournant un certain type de donnée, ont la même forme que ce type de donnée. Selon [Koitz et Slany, 2014], "En plus de leur couleur, la forme des blocs dans Scratch, permet de les distinguer car seulement des blocs syntaxiquement compatible peuvent être connectés entre-eux."

Selon [Conversy, 2014], « Scratch est un langage visuel disposant de connecteurs sur ses boites ; ces connecteurs suggèrent comment placer les blocs. Les connecteurs sont similaires aux flèches dans un langage Dataflow car ils indiquent la direction de la séquence d'instructions».

2.5.3 Flowstone (Environnement Flowbotics Studio)

Flowbotics Studio est un environnement qui utilise Flowstone comme langage de programmation ; bien que Flowbotics dispose déjà en interne d'un nombre important de composants liés aux périphériques standards des ordinateurs, il permet l'interfaçage avec tous les composants de robotique standards pour autant qu'il dispose des librairies d'API relatives à ces composants. (Comme par exemple, les servomoteurs, les cartes d'acquisition de données, cartes sonores, webcam, Phidgets ou même les Lego Mindstorms NXT et EV3).

Flowstone est présenté dans les figures 15 et 16.

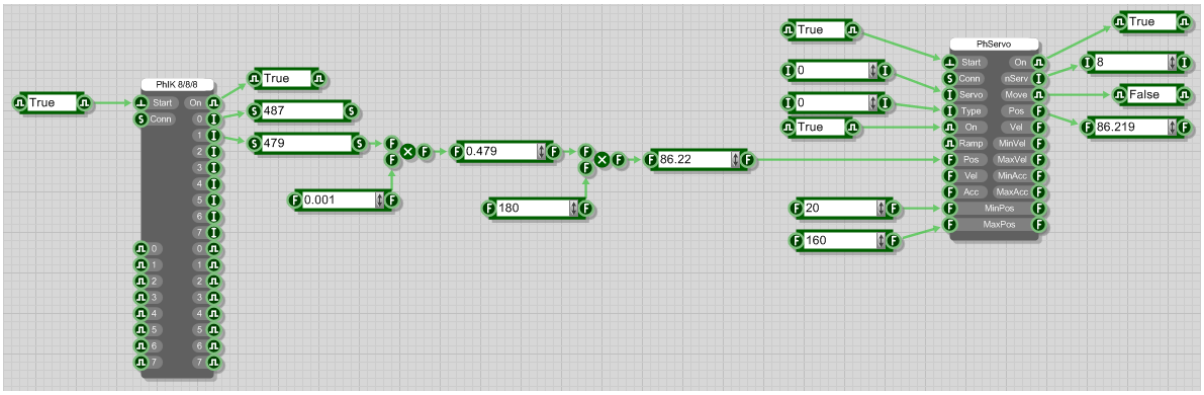


Figure 15 : Langage de programmation Flowstone (Exemple n°1)

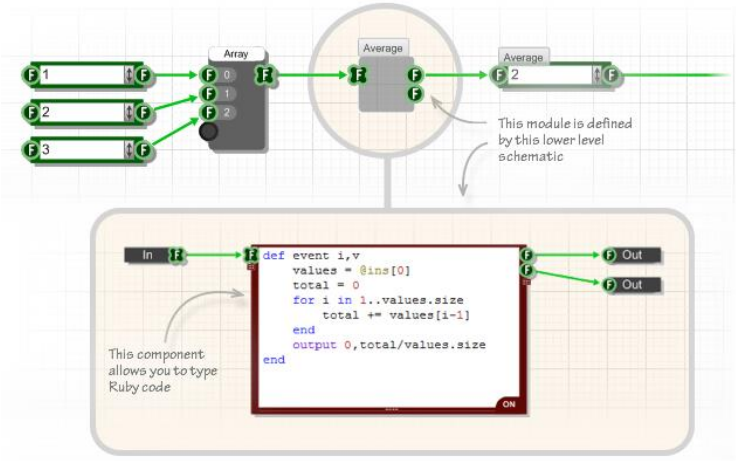


Figure 16 : Langage de programmation Flowstone (Exemple n°2)

Flowbotics Studio permet l'exécution en temps réel des projets sans avoir besoin d'attendre la compilation globale du projet. Il permet même d'écrire des composants personnalisés directement en script Ruby (langage procédural classique). En fait, on retrouve dans Flowstone deux types de boîtes ou constructions visuelles : les modules et les primitives. Tout module de Flowstone peut voir son propre code interne en Ruby édité ou simplement visualisé par l'utilisateur exploitant ce module dans son application.

La programmation visuelle avec Flowstone est assez simple car il suffit de positionner les composants voulus dans la zone de programmation et de faire ensuite les liens entre les sorties d'une boîte et les entrées d'autres boîtes. Dans cette application, on accède en temps réel aux signaux analogiques et

digitaux des composants. Cet aspect particulier de l'environnement Flowbotic Studio le rend éligible pour certains types de projets nécessitant l'écoute en temps réel de signaux analogiques et/ou digitaux.

Flowbotics Studio dispose entre autres d'un gestionnaire permettant de disposer et d'enregistrer ses propres modèles de comportement afin de réduire les temps de développement pour les futurs projets issus de ce même environnement. De cette manière, le code visuel peut être factorisé pour une réutilisation de celui-ci. Il met aussi à la disposition de l'utilisateur un moteur graphique puissant afin de pouvoir utiliser toutes sortes d'images et/ou de créer ses propres images ou graphes et en les intégrant dans les solutions. Ce moteur permet au langage Flowstone de disposer des fonctionnalités de lecture, d'enregistrement vidéo via webcam et d'intégrer la reconnaissance visuelle ou encore la détection de mouvements dans les projets. [28]

Flowbotics Studio permet d'exploiter la communication par port série afin de se connecter à divers éléments externes de robotique bon marché comme par exemple des bras commandés. Il permet également de gérer les flux de streaming, de visualiser en temps réel l'impact de l'utilisation des applications développées sur les variables et autres composants de la solution développée et en tant qu'outil de développement rapide de développer des solutions en très peu de temps. Cet environnement et son langage sont des outils parfaits pour l'éducation car il est simple d'utilisation et peut donc être utilisé pour un apprentissage par des enfants. Il aide également à favoriser l'apprentissage et la performance dans les domaines des sciences, de la technologie, de l'ingénierie et des mathématiques.[29]

« Flowstone supporte un grand nombre de matériaux électroniques tels que ceux provenant de Lynxmotion, phidgets, Pololu, robot electronics mais également ceux implémentant des protocoles comme Bluetooth, Xbee, ModBus, TCP/IP, UDP/IP, RS232, I2C entre autres. » [30]

[28] http://www.robotshop.com/ca/fr/flowbotics-studio-lien-telechargement.html
[29] http://www.robotshop.com/ca/fr/flowbotics-studio-lien-telechargement.html
[30] http://www.lynxmotion.com/p-883-flowbotics-studio-v2-cd.aspx

2.5.4 Choregraphe (Environnement Choregraphe)

Choregraphe est un environnement / langage de programmation visuelle exclusivement utilisé pour l'animation des robots NAO de Aldebaran Software [31] : des robots évolués, équipés de multiples capteurs, de servomoteurs et de caméras.

Dans ce logiciel, présenté dans les figures 17 et 18, on glisse les blocs de comportement et d'actions dans l'espace de programmation. Chaque bloc ainsi placé permet d'être connecté à la suite d'un autre par l'intermédiaire d'un lien et peut être lui-même le précédent d'un autre bloc également. On crée de cette façon un ensemble de comportements qui vont permettre au robot de réagir face à diverses contraintes. L'ensemble de la programmation se fait via la souris et l'interaction avec le clavier est quelque fois nécessaire pour la définition de noms de modules et/ou de textes personnalisés ainsi que pour certains réglages de propriétés dans certains blocs. La programmation NAO ne nécessite pas de disposer de connaissances en électronique ni en programmation.

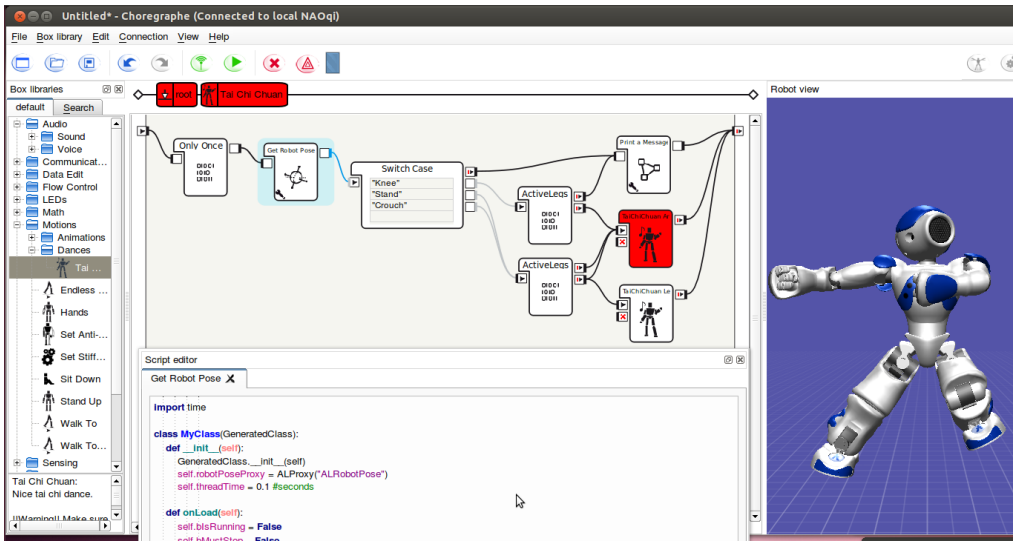


Figure 17 : Environnement / langage de développement Choregraphe (exemple n°1)

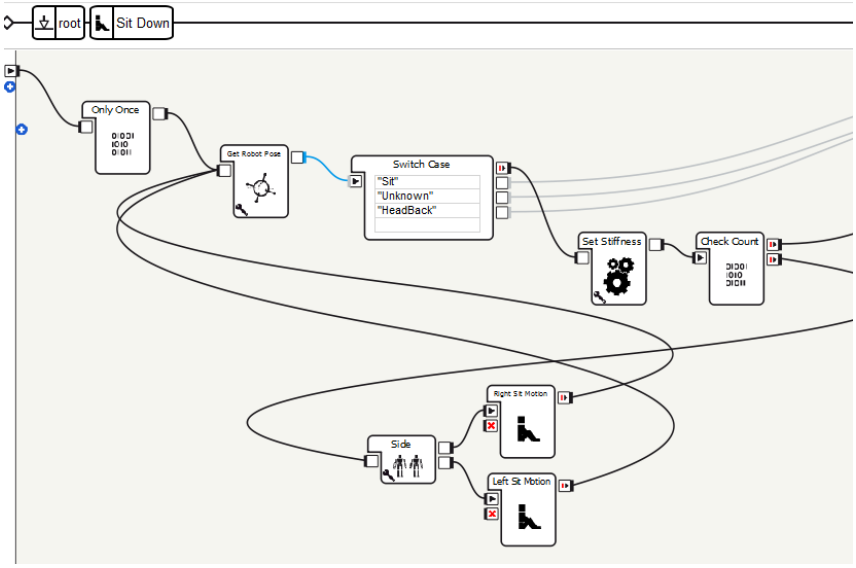


Figure 18 : Environnement / langage de développement Choregraphe (exemple n°2)

[31] https://www.aldebaran.com/fr/robots

L'environnement permet de créer ses propres blocs afin de factoriser un ensemble de blocs décrivant un certain comportement. Cela permet d'éviter d'avoir un programme visuellement brouillon et de pouvoir, de cette façon, bien s'organiser dans le développement des solutions. L'abstraction procédurale peut ainsi être appliquée au code du programme. La plupart des composants ne sont pas très expressifs et ne sont identifiables aisément que par un nom et une image qui leur est propre ; d'autres composants comme les boîtes contenant plusieurs possibilités sont plus expressifs car l'utilisateur voit tout de suite qu'il s'agit d'une alternative de plusieurs choix ou d'états possibles.

L'utilisateur a la possibilité de choisir parmi un nombre important de fonctionnalités toutes liées à la gestion du robot NAO. La grosse difficulté est de sélectionner les bonnes fonctionnalités dans une liste importante de choix textuels associés de petites images.

Dans Choregraphe, les boîtes disposent d'un port d'entrée et d'un ou de plusieurs ports de sortie ; les boîtes disposent aussi d'un port qui sera utilisé en cas de problème dans l'exécution d'un composant.

2.5.5 Le langage « G » (Environnement NI LabView)

L'environnement LabView, illustré dans la figure 19, dispose de son propre langage appelé « le langage G » qui se compose d'arcs, de liens de données « datawire » et de boîtes appelées « Vi » ; les « Vi » sont des fonctions intégrées au langage « G » qui permettent d'effectuer des opérations algébriques, de l'acquisition de données, de la gestion de communication et des opérations en base de données.

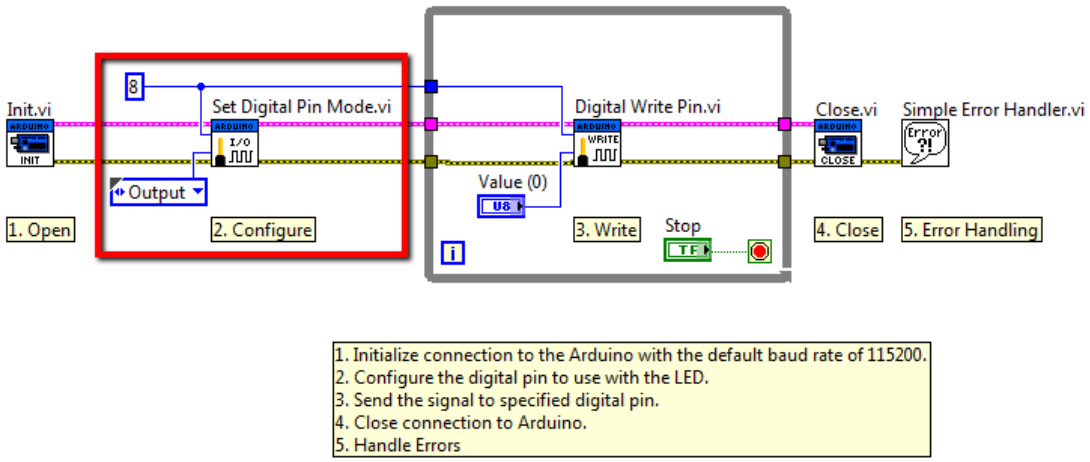


Figure 19 : Langage « G » dans LabView

[Cox et Dang, 2010] nous disent que « LabView est structuré, spécifique à un domaine et conçu pour l'acquisition de données et le contrôle d'instruments virtuels (Vi)".

«Les composants de LabView sont appelés 'Virtual Instrument' (Vi) car leur apparence et leur fonctionnement imitent de vrais instruments physiques. Un programme LabView est composé de nœuds. Chaque nœud peut être une fonction, un subVi ou une structure. Le SubVi est similaire à une fonction dans un langage de programmation textuelle. Chaque Vi est représenté avec une petite icône dans le coin supérieur droit dans les panneaux visuel 'Panneau Frontal' et 'Diagramme de blocs'.»[32]

Le diagramme est réalisé dans la fenêtre de commande via des composants pris, par l'utilisateur, du panneau des composants et "lâchés" dans la fenêtre de commande.

Ce langage permet de mettre en évidence plusieurs types de données en leur assignant à chacun une couleur spécifique. Il se range dans la famille des langages de programmations par Flux.

On retrouve notamment les constructions logicielles suivantes : « Boucles While » et « répétition for » ; les conditionnelles « If-Then-Else » et les alternatives « Switch Case ».

[Boubaker, 2011] nous dit que «LabView a d'autres types de structures d'exécution telles que les structures d'évènement utilisées pour gérer certaines tâches lancées en parallèle, les structures de séquences utilisées pour forcer une séquence de tâches à exécuter, des registres à décalage (shift register) utilisés dans les boucles pour passer des valeurs d'une précédente itération de boucle à l'itération suivante, des tunnels utilisés pour alimenter ou sortir des valeurs de structures. LabView suit un modèle de type flux de donnée (dataflow) pour l'exécution de ses Vi. Un nœud du diagramme de blocs s'exécute quand il a reçu toutes les données pour ses entrées. »

[32] http://www.ni.com/labview/

Le langage « G » n’est pas un langage interprété mais un langage compilé en arrière-plan, il dispose de trois palettes de composants : les fonctions, les contrôles et les outils. Ce langage utilise les types de données suivantes : numérique, chaîne de caractères, type de donnée dynamique, booléen, énumération, variant ou encore TimeStamp (date).

2.5.6 App Inventor pour Android (Environnement App Inventor)

Cet environnement / langage, présenté dans les figures 20 et 21, est inspiré du langage Scratch et est propre au développement d'applications devant s'exécuter sur plateforme Android.

Selon [Honig, 2013], « App Inventor est un outil visuel de programmation permettant de développer ses compétences par du raisonnement symbolique, de la résolution de problèmes et de la prédiction de conséquences. Les changements dans l'application sont directement visibles sur le téléphone Android connecté car il n'y a pas d'étape de compilation. App Inventor est limité par rapport à un environnement traditionnel de programmation car il ne permet pas la création de nouveaux composants (nouvelles classes) et ne permet pas de contrôler la priorité des évènements. Il possède cependant beaucoup de constructions et rend possible la construction d'une application mobile sophistiquée».

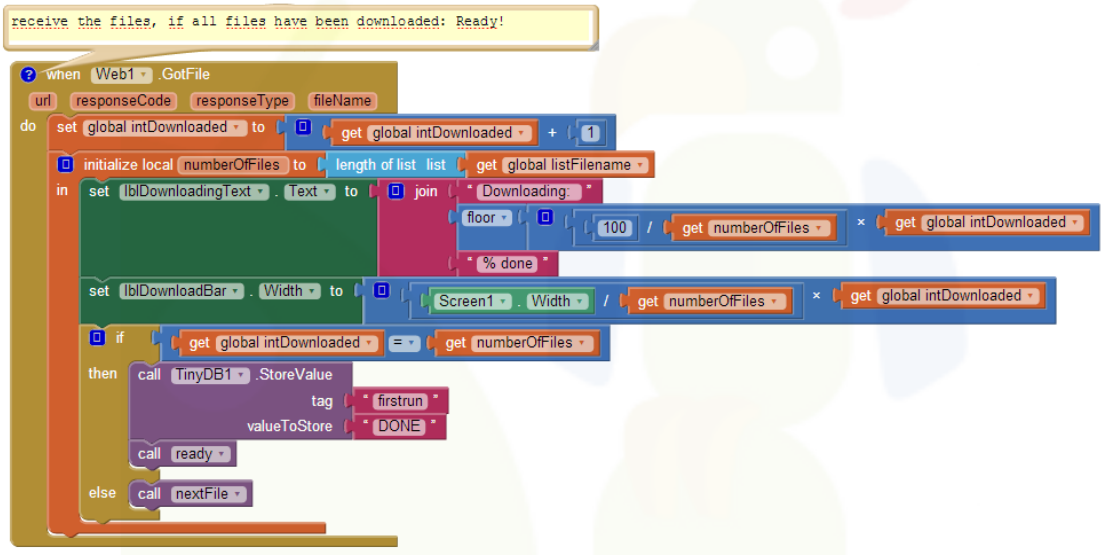


Figure 20 : Environnement / langage de développement App Inventor (exemple n°1)

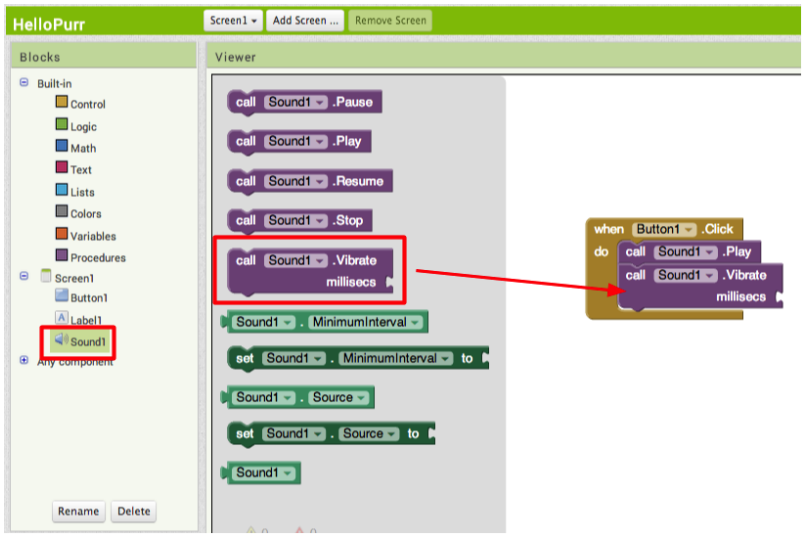


Figure 21 : Environnement / langage de développement App Inventor (exemple n°2)

2.5.7 Microsoft Kodu (Environnement Microsoft Kodu)

Environnement / langage développé par Microsoft afin d'initier les plus jeunes aux joies du développement d'applications. Il permet d'écrire des programmes composés d'une série de règles. Chaque règle est composée d'une condition ainsi que d'une action à réaliser si la condition est vérifiée. Les conditions et actions sont définies par une succession de blocs dessinés appelés "tuiles". Les applications ainsi développées doivent être exécutées sur des machines utilisant le système d'exploitation Windows. La figure 22 illustre un programme Kodu.

Figure 22 : Environnement / langage de développement Microsoft Kodu

Selon [Stolee, 2010], « Kodu est un langage visuel interprété de haut-niveau. Il a largement été inspiré par la robotique de telle sorte que chaque personnage et objet dans Kodu est programmé interactivement pour réagir avec le monde tel un agent intelligent».

« Kodu a été pensé comme un support à l'apprentissage. En effet, il permet de travailler la créativité, la programmation, la résolution de problèmes ainsi que le récit dont l'utilisateur devient le metteur en scène. Son fonctionnement a été étudié pour qu'il soit simple et accessible à un public de débutants »[33].

[33] http://edutechwiki.unige.ch/fr/Kodu

2.5.8 KTechLab (Environnement KTechLab)

Environnement / langage de programmation permettant la création de programmes selon le paradigme « FlowChart », paradigme fort proche des dessins que l'on retrouve dans les solutions de principes de programmation. Ce langage, dont un exemple est présenté dans la figure 23, est très expressif dans le sens où il informe quasi directement toute personne qui le lit de ce qu'il fait, de son mode de fonctionnement mais par contre il souffre de grosses lacunes comme par exemple, la gestion des variables.

« KTechlab est un environnement de simulation de circuits électroniques et de microcontrôleurs. Il est sous licence libre et est conçu pour l'environnement KDE (Linux). Il simule une variété de composants (logiques, circuits intégrés, linéaires, non-linéaires, composants actifs) »[34].

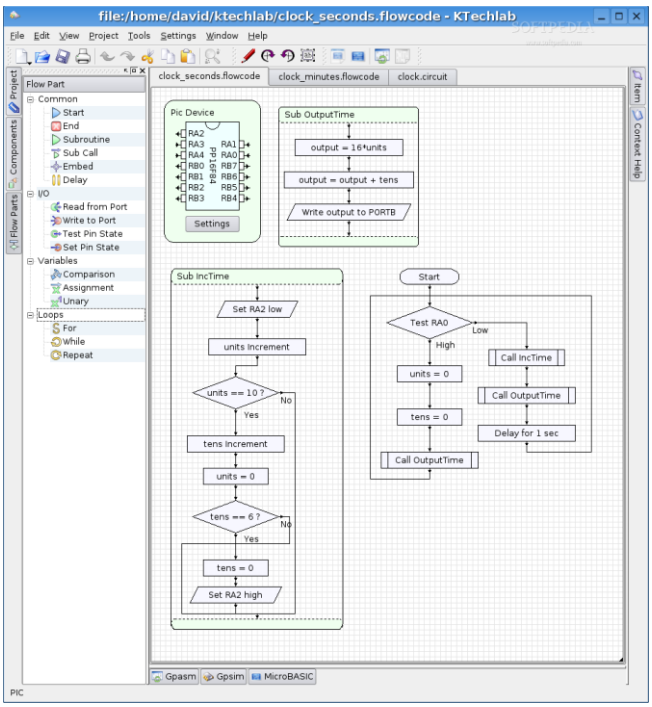


Figure 23 : Environnement / langage de développement KTechLab

[34] https://fr.wikipedia.org/wiki/KTechlab

2.6 Forces et faiblesses des langages de programmation visuelle

Nous citerons ci-après les forces et faiblesses des différents langages de programmation afin de mettre en évidence les points positifs et négatifs de ceux-ci et de se rendre compte des améliorations possibles qu'il serait intéressant de souligner.

2.6.1 Lego Mindstorms EV3-G

Forces

- Simplicité et intuitivité apparente.
- Programmation par placement de boîtes.
- Les boîtes sont colorisées et catégorisées en fonction de leur nature :
  (Action, comportement, capteurs, moteurs, concepts avancés).
- Edition des propriétés au sein même des composants sans passer par des écrans intermédiaires d'éditions de propriétés : puissance visuelle.
- Certaines boîtes, comme celles des moteurs, bénéficient d'un paramètre définissant le mode dans lequel ce composant doit fonctionner. Du mode choisi dépend la présence d'autres paramètres visuels pour ce composant.
- L'utilisateur peut lier une information recueillie via une boîte capteur pour la diriger vers l'entrée de données d'une autre boîte. Si l'assignation d'un lien de données d'une boîte à une autre s'avère impossible alors l'application nous le signale.
- L'environnement EV3 nous permet de savoir rapidement si la brique de commande P-Brick est bien connectée au pc ; il nous informe également des capteurs et autres éléments Lego Mindstorms connectés à la brique ainsi que les ports concernés par ces connexions.
- L'utilisateur est guidé par l'interface. Une aide en ligne est également disponible.
- Application gratuite mais inutile si l'on ne dispose pas des composants Mindstorms.
- Parmi les constructions du langage, on retrouve les boucles itératives pouvant être paramétrées selon plusieurs modes (boucle finie, infinie, finie et en attente qu'un certain évènement se déclenche), la construction conditionnelle du sélecteur permettant de gérer plusieurs alternatives de conditions ainsi que les boîtes d'attente d'évènements particuliers permettant de déclencher des processus.
- Une application Mindstorms est une application multi-threadée.
- L'application bénéficie de l'abstraction procédurale. Il est en effet possible de définir ses propres blocs en déplaçant un ensemble de blocs composant un flux de contrôle Mindstorms dans un nouveau bloc personnalisé (couleur cyan).
- Le langage nous permet de laisser des notes à côté des boîtes Mindstorms afin d'améliorer la compréhension du code. Cela permet plus de clarté à la relecture du code.

Faiblesses

- Ce langage n'est pas forcément intuitif pour tout le monde.
- Il n'est pas toujours aisé de comprendre facilement les modes, variables et autres propriétés des blocs Mindstorms et donc de pouvoir disposer d'une prise en main rapide de l'application. L'écriture guidée d'un programme - exemple est nécessaire pour un bon apprentissage du langage.
- Bien que basé sur NI LabView, Lego Mindstorms EV3-G ne lui ressemble en rien et est uniquement dédié à la technologie Lego Mindstorms EV3.
- Dans cet environnement, il n'y a pas de notion de débogage, d'exécution pas à pas ou d'exécution en temps réel possible vu que le programme doit d'abord être téléversé sur la brique programmable Lego EV3, la P-Brick.
- De nouveaux éléments peuvent être ajoutés à la palette de composants Mindstorms mais les sociétés autorisées à fournir de nouveaux composants sont peu nombreuses.
- La forme des boîtes ne leur confère pas forcément la propriété de détrompeur et dès lors ne permet pas de s'assurer d'un certain contrôle de la syntaxe.
- Lorsqu'il faut affiner certaines valeurs dans les propriétés des boîtes, l'usage du clavier s'avère indispensable car le curseur fourni pour déterminer une valeur numérique ne permet pas facilement de s'arrêter sur une valeur numérique précise.
- Il n'y a pas de prise en compte d'une éventuelle erreur de fonctionnement dans le comportement interne d'un composant lego Mindstorms ; en effet, aucune gestion d'erreur n'existe dans cet environnement de programmation.
- Les liens de données sont très pratiques pour relier une valeur de sortie d'une boîte à une valeur d'entrée d'une autre boîte mais au bout d'un certain moment, l'environnement visuel est saturé par ces liens et il devient difficile de se retrouver dans la compréhension de ceux-ci.
- Le visuel des formes et des composants n'est pas toujours très expressif.
- L'utilisation exclusive de la souris peut ne pas plaire à tout le monde.

2.6.2 Scratch

Forces

- Simple d'utilisation et visuellement très expressif.
- Programmation par placement de blocs, de variables, de flux ou d'actions.
- Les blocs sont catégorisés et colorisées selon leur catégorie.
- Scratch est un logiciel open-source et gratuit qui permet d'écrire rapidement du code fonctionnel et qui permet d'éveiller les plus jeunes aux joies de l'informatique.
- Environnement / langage disponible sur plusieurs plateformes et systèmes d'exploitation.
- La forme des blocs leur confère des particularités de détrompeurs car ils permettent de s'assurer d'un certain respect de la syntaxe concrète en refusant visuellement certains emboîtements impossibles syntaxiquement.
- Factorisation possible du code grâce à la création de nouveaux blocs personnalisables. (Le langage permet la création de nouveaux blocs avec définition de paramètres ; il nous permet donc de définir de nouvelles fonctions et de les appeler.)
- Le langage est multi-threadé, permet la gestion et la réaction à des évènements.
- Parmi les constructions du langage, on retrouve les blocs de répétitions et les blocs conditionnels.
- Scratch permet de gérer des variables et des listes en fournissant les blocs de gestion pour ces éléments.
- La mise à jour du code est directement visible dans la fenêtre de rendu de création sans qu'une compilation soit nécessaire.

Faiblesses

- Trop coloré, peut perturber le programmeur débutant.
- N'est spécifique au prototypage Raspberry Pi que via une version dédiée.
- Aucun guidage de l'utilisateur n'est proposé.
- Scratch n'est pas exploitable pour de gros projets.
- Difficulté de relecture du code car l'espace visuel est très vite fort encombré et cela même en s'aidant des ascenseurs verticaux et horizontaux de l'interface de programmation.
- L'interface générale conserve un aspect brouillon à la relecture du code.
- N'est pas conçu pour être utilisé avec des composants électroniques intelligents comme les Phidgets (aucune variante n'existe).
- Scratch ne permet pas de programmer une carte Arduino dans sa version de base ; ardublock, qui s'inspire de Scratch, permet de programmer une carte Arduino.
- Ne permet pas le débogage de code (la possibilité d'exécution du code en mode pas-à-pas est inexistante).
- Même si le langage nous permet de créer de nouveaux blocs, l'espace visuel est de plus en plus chargé et l'usage d'ascenseurs, bien que pratique, fait perdre au programmeur la vue d'ensemble de l'application.
- Seulement 3 types de variables sont gérés : les entiers, les chaînes de caractères et les booléens. Il est cependant à noter que le langage permet la gestion de listes de variables.
- Selon [Koitz et Slany, 2014], "des langages comme scratch peuvent être améliorés pour ce qui concerne la présentation de formules par l'utilisation d'une méthode hybride mélangeant le visuel de Scratch et une représentation textuelle des formules."

2.6.3 Flowstone (Flowbotics Studio)

Forces

- Simple d'utilisation.
- Utilisation assez intuitive de l'application.
- Permet d'interfacer un grand nombre d'éléments électroniques tels que les Phidgets ou encore d'autres éléments propres aux prototypages de base comme ceux réalisés avec des cartes Arduino.
- Programmation par le placement de boîtes et d'autres éléments visuels, par des liens que l'on établit d'une boîte où d'un élément graphique à un autre élément graphique.
- Parmi les boîtes que l'on place, il y a les primitives non-éditables et les modules éditables ; les primitives sont des fonctionnalités intégrées au langage. L'édition des modules permet la visualisation et la modification du programme écrit en langage textuel de script Ruby.
- Possibilité d'exécution du code en temps réel.
- Les types de données sont appelés « connecteurs » et sont représentés par des symboles (en entrées et sorties des modules et des primitives).
- L'environnement dispose d'un puissant moteur vidéo, d'un outil d'édition sonore, d'éléments de reconnaissance vocale et visuelle, d'un éditeur graphique pour gérer ses propres animations dans le code visuel du programme.[35]
- L'environnement met également à disposition un gestionnaire de modèles permettant entre autres la factorisation du code.
- Possibilité de faire le maximum rien qu'avec le contrôle souris.
- Convient pour l'apprentissage pour les plus jeunes, les débutants comme pour une utilisation par des personnes confirmées.
- Tout comme NI LabView, Flowstone permet de faire de la programmation « bas-niveau » mais à la différence de NI LabView, est beaucoup plus facile d'utilisation que ce dernier.

Faiblesses

- Application payante ; beaucoup d'éléments sont d'ailleurs payants comme des codes pré-écrits pour des composants dédiés à certaines technologies telles que celle des Phidgets.
- Flowstone n'est pas aussi joli et avenant que ne l'est Lego Mindstorms EV3-G.
- Il n'y a pas d'indicateur pour signaler que les composants sont bien connectés physiquement à l'ordinateur ou à une carte contrôleur.
- Lors de projets de prototypage, il est nécessaire que le programmeur dispose néanmoins de quelques notions d'électronique afin de disposer les bons composants ainsi que les bons liens entre ceux-ci dans l'interface.
- Vu le nombre important de composants avec lesquels il est possible d'interagir, l'utilisateur doit avoir une idée plus ou moins précise de ce qu'il recherche dans la palette des composants graphiques. Il a néanmoins la possibilité de faire une recherche textuelle des éléments qu'il souhaite placer dans le code de son programme.
- Faible diversité des couleurs dans l'environnement de programmation ; il y a une forte dominance du noir et du vert.

[35] http://www.robotshop.com/en/flowbotics-studio-download.html

2.6.4 Choregraphe

Forces

- Facilité d'utilisation et interface aisée.
- Programmation par placement de boîtes. Chaque boîte représentant soit un certain comportement du robot, soit un certain mécanisme pour contrôler le flux d'informations.
- L'utilisateur lie une boîte à une autre, réalise des flux appelés comportements en spécifiant les débuts et fin de flux.
- Chaque boîte dispose d'un ou de plusieurs ports de sorties: certains sont employés quand l'action de la boîte s'est bien déroulée ; d'autres dans le cas contraire (gestion d'erreur).
- L'utilisateur peut factoriser le code de son programme en créant des composants personnalisés.
- L'utilisateur à la possibilité d'éditer le code Python des boîtes afin de consulter ou modifier la programmation de ces boîtes.

Faiblesses

- Logiciel propriétaire spécifique à la programmation des robots NAO.
- L'utilisateur ne se repère pas facilement dans le projet de développement car il doit choisir les boîtes d'actions parmi une liste d'options texte accompagnées de petites imagettes.
  En effet, il est parfois difficile de trouver la bonne boîte pour l'action souhaitée.
- L'édition des propriétés des boîtes se fait via une fenêtre qui s'ouvre lors d'un clic sur la boîte. Ces propriétés ne sont pas immédiatement visibles et le dessin n'est pas toujours très explicite. Il y a dès lors utilisation de plusieurs boites de propriétés.
- Toutes les actions ne sont pas réalisables par la souris, certaines nécessitent l'utilisation d'un clavier ; l'usage du clavier est souvent requis pour la configuration des propriétés de boîtes.
- Pas de débogage possible du code mais il est possible de faire démarrer le flux d'exécution à partir de divers endroits dans le programme (dessin de flèche noire remplie dans un port d'entrée d'une boîte).
- Factorisation de code possible mais étape un peu laborieuse ; de plus, le développeur perd la vue d'ensemble du flux du programme.
- Application payante mais il y a possibilité de disposer d'une application d'évaluation du produit.
- L'espace visuel est très vite encombré et le code devient vite brouillon ; ce qui entraine une relecture difficile du code visuel.
- La richesse des couleurs n'est pas employée dans l'interface de programmation. Les boîtes sont en noir et blanc.

2.6.5 Langage « G » (NI LabView)

Forces

- Permet de programmer de manière “bas-niveau” des applications de simulation ou de prototypage.
- Les types de données sont colorisés.
- Lorsque l’on désire placer un composant, l’interface nous propose diverses possibilités de placement pour ce composant.
- Le langage est extensible et peut être augmenté d’éléments tiers comme les composants dédiés aux lego Mindstorms NXT et EV3.
- Les boucles bénéficient d’une possibilité de cadenceurs qui leur permettent de synchroniser le passage dans la boucle avec une certaine fréquence de passage.
- Le langage permet aux fonctions récursives de pouvoir accéder aux valeurs de l’itération précédente.
- LabView n’est pas un pur langage de type « Data Flow » ou « flux de données » car il contient des constructions comme les boucles For et While ou encore des conditionnelles comme la boîte Select Case.
- L’environnement LabView permet la manipulation du code de certaines boîtes et emploie des variables locales et globales afin de rendre le langage plus pratique et expressif.
- Il dispose aussi de constructions plus originales comme les structures Evènement, Séquence, cluster et shift register.
- Ce langage permet l’abstraction procédurale car il permet de factoriser du code (création de sous-schémas).
- NI LabView est extensible à des composants de sociétés tierces.

Faiblesses

- Il existe de nombreux écrans “pop-up” permettant d’accéder et de paramétrer les propriétés des boîtes.
- Les boîtes “select-case” ne permettent de gérer que 3 alternatives possibles.
- L’espace visuel est très vite encombré visuellement.
- Le langage « G » un langage GPL (General Purpose Langage) ou Langage à but général ; il est plus complet que d’autres langages mais est dès lors plus complexe à manipuler que d’autres langages plus « grand public ».
- Il n’y a pas de contrôle de la syntaxe visuelle.
- La complexité d’utilisation du langage s’apparente à la conception d’un schéma électrique.
- Langage très orienté laboratoire.

2.6.6 App Inventor

Forces

- Langage fonctionnant sous plateforme Android, inspiré du logiciel Scratch et bénéficiant de la simplicité d'utilisation de celui-ci. L'application développée peut ainsi avoir accès au téléphone, à l'écran, au GPS et aux autres composants intelligents du smartphone ou de la tablette. Des modifications effectuées dans l'application sont directement visibles et perceptibles dans l'application.
- L'environnement / langage permet l'écriture de fonctions personnalisées afin de factoriser du code et permet de définir le code devant s'exécuter lorsqu'un évènement est déclenché sur un des composants utilisés par l'application.
- App Inventor permet l'écriture de fonctions, la gestion de leurs paramètres et les appels vers ces fonctions.
- L'environnement permet un débogage (exécution pas à pas) des applications développées ; les variables peuvent être observées dans ce mode.
- App Inventor permet une gestion des variables et des évènements.
- Ce langage permet de stocker et récupérer des valeurs dans une mini base de données (tinyDB).
- Tout comme Scratch, les boîtes sont catégorisées et colorisées selon leurs catégories et disposent de détrompeurs visuels permettant à App Inventor de s'assurer d'un certain respect de la syntaxe visuelle (syntaxe basée sur l'assemblage de boîtes).
- Parmi les constructions du langage, on retrouve la conditionnelle IF-THEN-ELSE et la répétition finie FOR EACH.

Faiblesses

- Seulement pour Android.
- Peu d'ouverture à d'autres matériels électroniques tiers.
- L'utilisation de variables globales s'avère obligatoire.
- App Inventor est mono-threadé à la différence de Scratch qui lui est multi-threadé.
  En effet, la particularité d'une « single-threaded application » ou « programme avec un simple fil d'exécution » est qu'il n'y a pas d'entrelacement d'évènements entre les diverses actions dans les boucles.
- Dans App Inventor, il n'existe pas de construction « structure de contrôle » de type While (tant que). Il n'existe que la répétition finie For Each (Pour chaque).
- Dans ce langage, il est inutile de connaître par cœur les noms des évènements des objets car ceux-ci sont directement proposés au programmeur lors de l'écriture du programme.
- Même si App Inventor n'est pas un langage orienté objet, il est nécessaire de disposer des notions d'orienté objet afin de développer avec celui-ci.

2.6.7 Microsoft Kodu

Forces

- Application réalisée via une succession de règles simples basées sur des associations de tuiles (dessins). Une règle est définie selon le modèle suivant : CONDITION : ACTION. [36]
- Le fait que des répétitions sont effectives dans le code montre qu'il est possible de faire de la réutilisabilité de code et donc de la factorisation de code.
- Selon [Stolee et Fristoe, 2011], les règles peuvent être réparties sur plusieurs pages de codes (avec un maximum de 12) ; une page définit un état d'un personnage et est assimilable à un nœud dans un graphe de type « Flux de contrôles » ou « control Flow ». Il est donc possible d'exécuter du code de manière non-linéaire en passant d'une page de code à une autre.
- Tout comme Scratch, ce langage est totalement Event-driven (basé sur les évènements).
- Il existe plus de 500 tuiles différentes.
- Il est possible d'indenter successivement plusieurs règles et ainsi créer des conjonctions locales de conditions, des disjonctions.

Faiblesses

- Les conjonctions de conditions sont parfois difficiles à lire et à comprendre.
- Le fait qu'il soit nécessaire de dessiner sur le côté un graphe afin de s'assurer de bien comprendre les répétitions dans l'algorithme n'est pas une chose pratique.
- Après analyse, plusieurs constructions se retrouvent parmi les constructions virtuelles de ce langage mais effectivement il n'y en a qu'une seule : WHEN + DO.
- Les tuiles sont liées à Windows et aux technologies vendues par Microsoft comme par exemple : la manette XBOX 360 pour pc.
- La limitation du nombre de pages à 12.
- Il est difficile de développer autre chose qu'une application de jeu pour enfants.
- Il n'existe pas de mécanisme d'abstraction procédurale tout comme il n'y a pas de concept de fonctions, de paramètres et d'appels de fonctions.
- Pas de possibilité d'exécution pas à pas du code du programme.

[36] http://research.microsoft.com/en-us/projects/kodu/

2.6.8 KTechLab

Forces

- Le programme est présenté de façon claire à la manière d'une résolution de principe de programmation. Les conditions, répétitions et appels de fonctions sont bien visuellement distincts.
- Ce style de programme visuel permet une relecture du code assez aisée.
- Selon [Terbuc, Globacnik et Majhenic, 2007], "Il permet la construction et l'analyse de circuits analogiques dans un mode graphique, son interface est très facile à utiliser". Ils nous disent aussi que « dans les schémas, l'utilisateur peut utiliser des microprocesseurs PIC virtuels qui peuvent être programmés de telle façon qu'il est possible de tester la bonne exécution du code programmé via ces composants virtuels ».
- [Nehra, 2014] nous précise que « ce langage peut être utilisé pour une simulation mixte de composants analogiques et de petits processeurs digitaux ».

Faiblesses

- Il n'y a pas de gestion de variables.
- Il n'y a pas de gestion d'erreurs.
- L'espace visuel est vite encombré.
- Les différentes constructions du langage ne sont pas colorisées et ressemblent toutes à des figures géométriques très similaires les unes des autres.
- Le style de programmation bien qu'expressif nécessite beaucoup d'interventions au clavier.
- La couleur verte utilisée pour les programmes n'est pas très attractive.
- Ce langage permet d'écrire et de compiler des petits programmes exclusivement pour des microprocesseurs PIC.

2.7 Synthèse de l'état de l'art

D'après la définition de Wikipédia, « Un langage de programmation graphique ou visuelle est un langage dans lequel les programmes sont écrits par assemblage d'éléments graphiques. Sa syntaxe concrète est composée de symboles graphiques et de textes qui sont disposés spatialement pour former un programme. » [37]

Un environnement de programmation est un logiciel qui permet de réaliser des programmes dans un ou plusieurs langages de programmation, qui met à disposition un ensemble d'outils intégrés, autorise l'ajout de modules complémentaires et permet au programmeur d'être plus productif et efficace dans le développement de ses projets. Il n'est pas toujours aisé de pouvoir dissocier un environnement de programmation du langage de programmation associé à celui-ci.

Ce chapitre a présenté plusieurs langages / environnements de programmation visuelle pour projets de robotique :

- EV3-G (Lego Mindstorms EV3)
- Choregraphe
- Flowstone (Flowbotics Studio)
- Le langage G (NI LabView)

Nous avons également évoqués, dans ce chapitre, d'autres environnements / langages de programmation visuelle :

- Scratch
- App Inventor
- Kodu
- KTechLab

Des programmes inspirés de Scratch et spécifiques à une programmation de projets électroniques ont aussi été brièvement présentés : Ardublock et Modkit Micro pour Arduino.

Lego Mindstorms EV3-G est gratuit mais est lié exclusivement aux composants Lego Mindstorms EV3. Ce langage de programmation est un bon exemple de programmation impérative.

Choregraphe est payant et dédié à la programmation unique de robots tels que NAO ; il nous donne néanmoins une bonne idée de ce que à quoi un langage de type « control flow » ou «flux de contrôles » ressemble.

Flowbotics Studio (et son langage Flowstone) est une application payante, très intuitive, ergonomique, permettant l'interfaçage avec une multitude de composants intelligents tels que les Phidgets, les Lego Mindstorms NXT et EV3, les composants .Net Gadgeteer ou encore les composants FlowPaw pour ne citer que les plus connus.

Flowbotics est plus puissant que LabView et est beaucoup plus facile d'utilisation que ce dernier ; la programmation et l'exécution en direct de l'application se font via la même interface graphique et il est possible pour l'utilisateur de modifier la programmation de la plupart des modules utilisés dans un

[37] https://fr.wikipedia.org/wiki/Langage_graphique

programme visuel. Flowstone est un mélange de programmation par flux (data flow) et de programmation impérative.

NI LabView et son langage G sont également une bonne alternative pour le développement d'applications de simulation ou de prototypage. L'une de ses forces est qu'il est capable d'incorporer des composants de diverses sociétés tierces dans un même projet.

Néanmoins, il s'adresse plus à un public d'experts de par la complexité de manipulation de son environnement. Dans cet environnement l'utilisateur doit travailler avec une interface utilisateur pour la manipulation des composants « jouables » et une autre interface appelée « diagramme de blocs » qui lui permet d'écrire visuellement le programme ; l'utilisateur ne peut seulement modifier que certaines boîtes appelée « formula boxes ». Le langage utilisé alors est une forme de C++.

Scratch est un langage visuel extrêmement explicite et est très adapté pour les débutants en informatique. D'autres langages, inspirés par Scratch, comme App Inventor permettent d'accéder aux divers capteurs et senseurs des téléphones mobiles et tablettes tournant sous la plateforme Android. Scratch est un autre exemple de programmation impérative. Il est très expressif car ses blocs correspondent à une utilisation harmonieuse de plusieurs variables visuelles comme cela est préconisé par [Moody, 2009].

Il existe également d'autres environnements / langages aux concepts similaires (boîtes et liens) mais pas forcément aussi puissants que les cinq pour la robotique cités plus haut.

Parmi les divers langages de programmation visuelle étudiés, on retrouve ceux appartenant aux paradigmes de flux (data flow / control Flow) et ceux représentants du paradigme de programmation impérative. Les langages de programmation déclarative indiquent l'ordre d'exécution des composants tandis que la programmation impérative explique comment réaliser les actions de ceux-ci.

Selon [Marttila-Kontio, 2011], le langage « G » de NI LabView n'est pas, malgré les apparences, un langage « data flow » pur car il a dû intégrer quelques adaptations afin de disposer d'une vraie efficacité de programmation : Un outil intégré lui permet d'éditer certaines boîtes ; il fait également usage de variables locales et globales (ces variables sortent complètement de l'idée du paradigme de « data flow » pur).

En programmation « data flow », un nœud s'exécute dès qu'il y a des données à chacune de ses entrées. Ensuite le résultat de ce nœud va dans un autre nœud/composant. Le cœur du « data flow » est la mobilité des données d'une source vers une destination en passant par diverses transformations. Certains composants réorientent le flux de données tout comme de l'eau qui s'écoule dans des tuyaux.

Selon [Vasek, 2012], "Les programmes « dataflow » représentent des programmes comme des graphes dirigés pour lesquels des nœuds représentent des opérations et un lien entre le bord d'un nœud à un autre représente le chemin parcouru par la donnée d'une opération à l'autre".

Pour un programme Control Flow, cela se passe comme dans un « Data Flow » sauf qu'ici le flux d'exécution est le cœur du programme. Les contraintes de précédence contrôlent le flux d'exécution basé sur la complétude des tâches. Une tâche ne peut s'exécuter si la précédente ne s'est pas terminée[38]. Dans un langage « control flow », les flux d'exécutions opèrent sur des données externes. L'exécution conditionnelle, les redirections ou appels de procédures changent le flux d'exécution du

[38] http://www.bradleyschacht.com/control-flow-vs-data-flow/

programme. On peut imaginer que des données transitent par les flèches et que certaines de ces données sont des appels de procédure pour lesquels il n'y a pas de mouvement d'informations.[39]

Dans les programmes Data Flow et Control Flow, la programmation employée dans un nœud est une programmation de type impérative. Le parallélisme d'exécutions des boîtes y est implicite.

Dans l'ensemble, les environnements de programmation visuelle sont très difficilement indissociables de leurs langages de programmation dédiés et il n'en existe qu'un seul qui permette d'interfacer les projets de robotique avec un très grand nombre de composants électroniques intelligents ou de bas niveau : Flowstone, un langage Esperanto de programmation visuelle.

La programmation s'en trouve intuitive, d'une complexité minimale, abordable au plus grand nombre, limitée à l'usage de la simple souris et se résumant à un agencement visuel de boîtes aux rôles de composant actif, de capteur, de comportement ou d'action ainsi que par le placement d'arcs permettant d'établir des liens entre boîtes ou entre des sorties et des entrées de données pour certaines boîtes spécifiques.

Il semble très difficile de trouver un environnement de développement visuel à faible coût et permettant l'interfaçage avec une majorité de composants électroniques tels que les architectures de composants intelligents présentées dans cet état de l'art mais il existe néanmoins un grand nombre de langages textuels permettant le développement de projets en utilisant diverses librairies contenant les API d'accès (pilotes) aux fonctionnalités de composants tels que celles des Phidgets.

L'expressivité de certains langages visuels est fortement associée à des DSL (Domain Specific Language) et les rends dès lors plus faciles de manipulation ; ces langages sont néanmoins limités. Des langages moins DSL mais plus GPL (General Purpose Language) comme le langage « G » offrent moins de confort d'utilisation mais sont plus riches d'expressivité.

Nous avons relevé que les concepts d'abstraction procédurale et d'abstraction des données sont intéressants ainsi que le fait que la saturation de l'espace visuel que cherche à résoudre l'abstraction procédurale rend difficile la relecture du code.

Nous avons écarté tout un ensemble de langages appartenant à divers paradigmes de programmation visuelle ou graphique car nous avons estimé qu'ils n'apportaient pas d'éléments nouveaux ou pertinents par rapport aux langages déjà présentés où qu'ils étaient trop spécifiques à des domaines très particuliers et étaient dès lors difficilement généralisables.

[39] http://stackoverflow.com/questions/461796/dataflow-programming-languages

3. Méthode employée pour la suite du travail

L'état de l'art nous a permis d'explorer le terrain de l'existant, a mis en évidence les composants évolués et les langages permettant de les exploiter, nous a éclairé sur les forces et faiblesses de ces langages et nous a donné une bonne idée des éléments intéressants dont il serait intéressant de reprendre les concepts lors de l'élaboration d'un nouveau langage visuel. Les définitions suivantes correspondent bien à la méthode qui a été employée sur base de l'exploitation des informations récoltées dans l'état de l'art.

D'après [Moody, 2009], "Une notation visuelle consiste en un ensemble de symboles graphiques, un ensemble de règles de composition et des définitions du sens de chaque symbole (sémantique visuelle). Le vocabulaire visuel et la grammaire visuelle forment ensemble la syntaxe visuelle ou concrète."

Selon [Cho et Gray, 2011], "un langage est généralement développé par la conception et l'implémentation de trois éléments de langage : la syntaxe concrète, la syntaxe abstraite et la sémantique. Quand on développe des DSML (langage de modélisation spécifique à un domaine) et plus précisément des langages de modélisation visuelle, la syntaxe concrète correspond aux éléments de modélisation qui symbolisent les concepts du domaine et la syntaxe abstraite est définie comme les relations entre les concepts de modélisation tels que définis dans le métamodèle. La sémantique qui gouverne les propriétés structurelles et comportementales du DSML est souvent associée au métamodèle."

[Cho, 2011] nous dit que "le développement de DSML commence par l'identification des besoins pour un problème dans un domaine. Une fois les besoins identifiés des experts du domaine et du développement vont collaborer pour concevoir la syntaxe concrète. Les experts en développement se baseront sur cette syntaxe pour élaborer la syntaxe abstraite à laquelle ils attacheront une certaine sémantique."

3.1 Description de la méthode

- Repérer les caractéristiques d'expressivité des différents langages retenus dans l'état de l'art et essayer de dégager le génome de ces langages en mettant en évidence les constructions et points forts de ceux-ci. Sur base de ces résultats, le métamodèle de ces langages sera réalisé afin de mettre en évidence la syntaxe abstraite des langages.
- Imaginer et présenter deux cas d'études pour lesquels une programmation visuelle pour débutants sera réalisée. Sur base des éléments visuels pertinents des langages retenus, imaginer une nouvelle syntaxe concrète originale et réaliser une nouvelle programmation visuelle expressive appliquée aux deux cas d'études imaginés.
- Grâce à la programmation visuelle (syntaxe concrète) des deux cas d'étude présentés, réaliser le métamodèle (syntaxe abstraite) du langage visuel présenté afin de mettre en évidence les différentes constructions de celui-ci.
- Eliciter de manière informelle les constructions et parties de constructions de ce nouveau langage. Documenter la sémantique du nouveau langage imaginé ainsi que les relations entre les différentes constructions de ce langage.
- Compléter le métamodèle du langage par de nouvelles constructions pouvant être utiles dans le développement visuel de futurs projets selon ce nouveau langage.

4. Syntaxe abstraite des différents langages retenus

Afin de mettre en évidence la syntaxe abstraite de ces langages, nous allons présenter leurs différentes constructions et les relations entre celles-ci sous la forme de diagrammes présentant le métamodèle de ces langages.

4.1 Scratch

La figure 24 présente le métamodèle de Scratch reprenant l'ensemble des constructions de ce langage et les relations entre ces constructions.

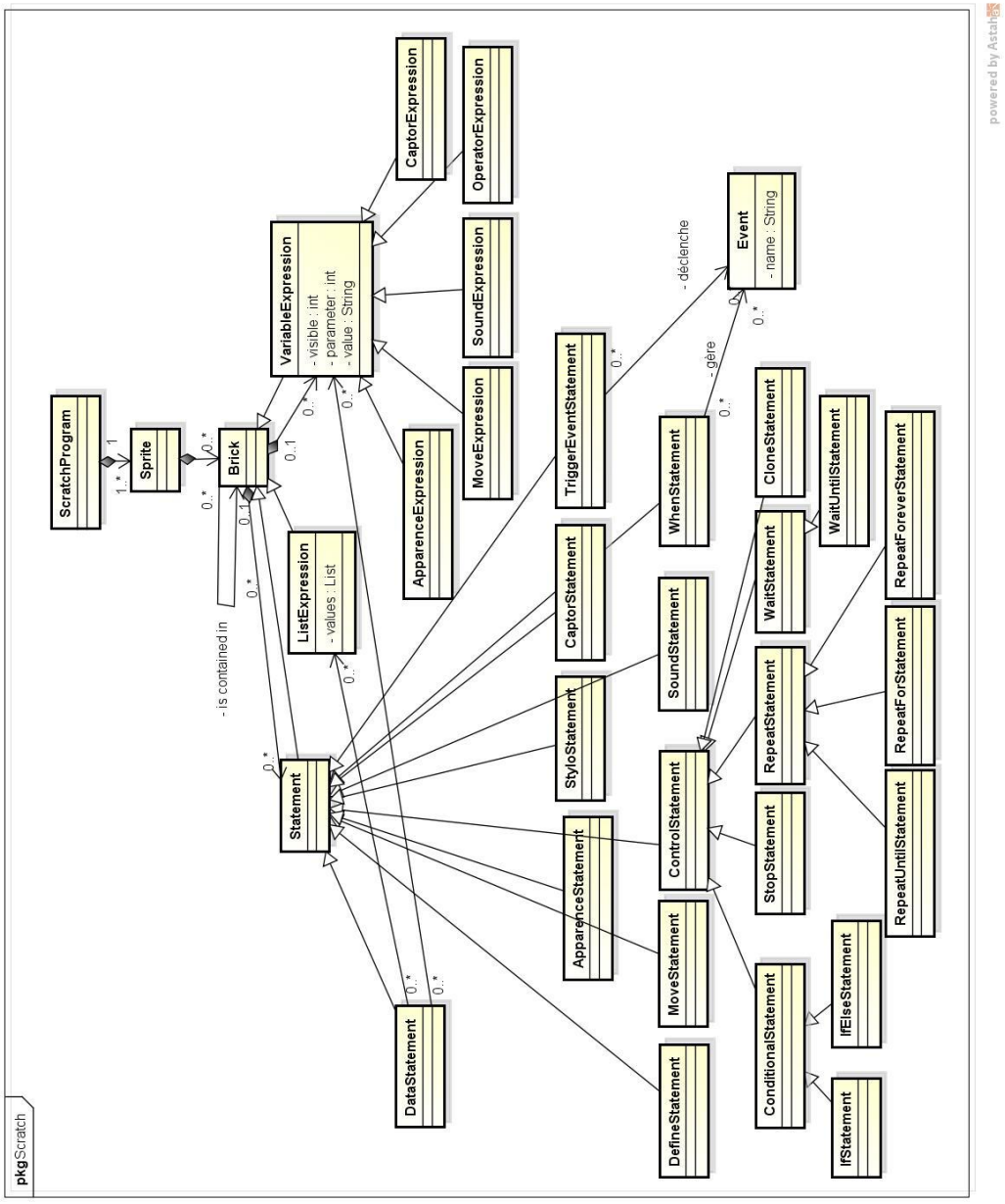


Figure 24 : Métamodèle du langage Scratch

4.2 Langage G (NI LabView)

La figure 25 présente le métamodèle du langage G de NI LabView reprenant les constructions du langage et les relations entre ces constructions.

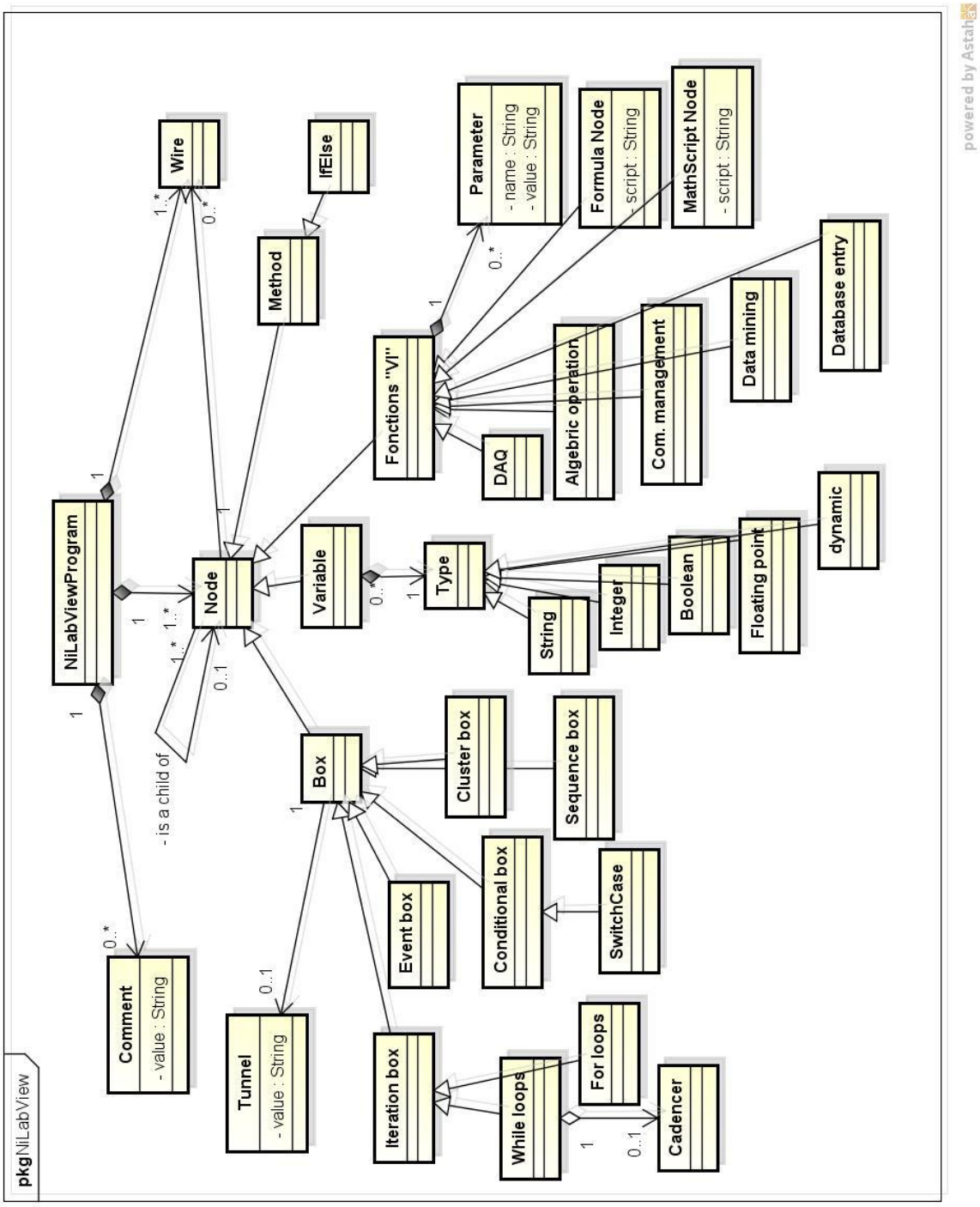


Figure 25 : Métamodèle du langage G de NI LabView

4.3 Lego Mindstorms EV3-G

La figure 26 présente le métamodèle du langage EV3-G de Lego Mindstorms reprenant les constructions du langage et les relations entre ces constructions.

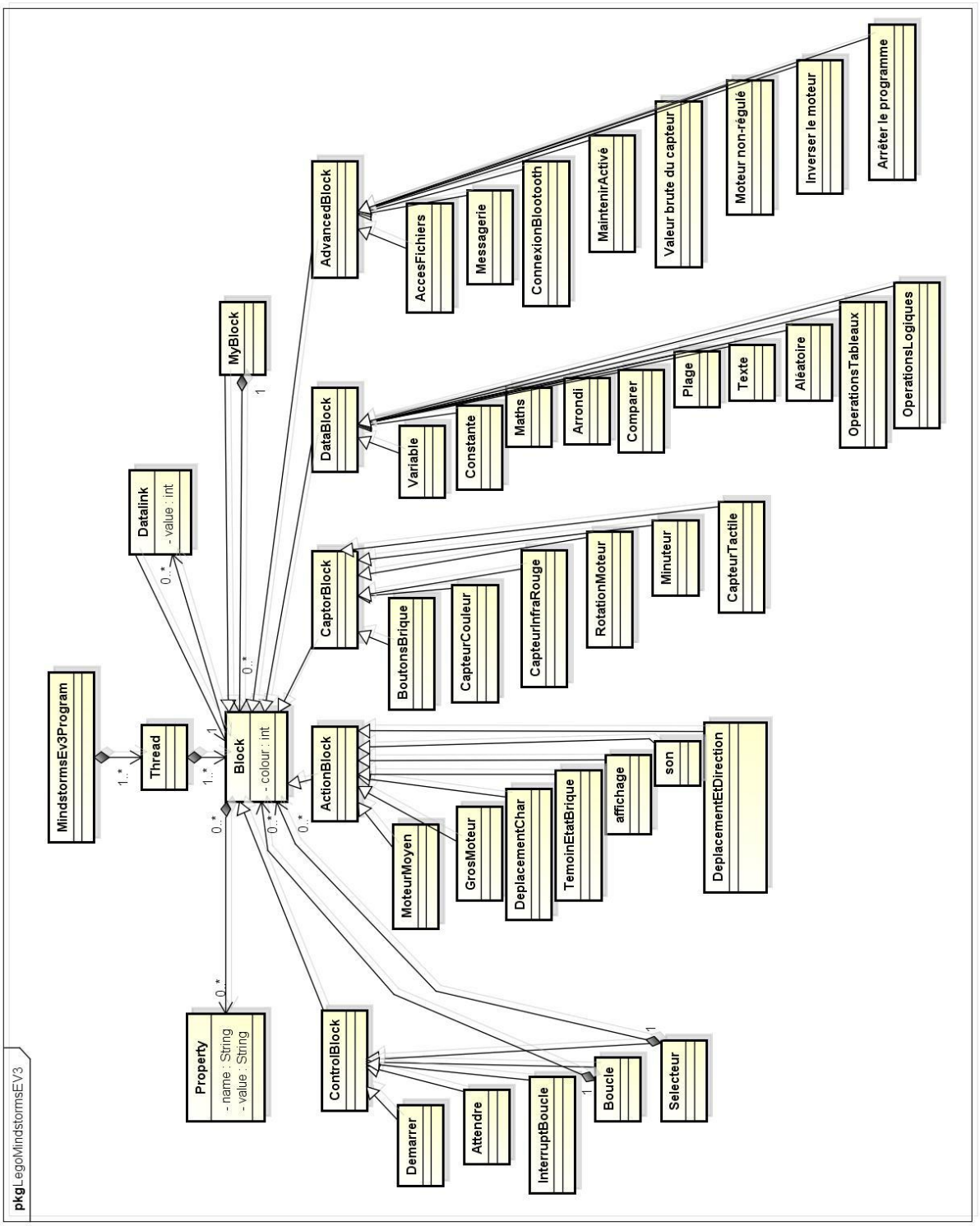


Figure 26 : Métamodèle du langage Mindstorms EV3-G

4.4 Flowstone (Flowbotics Studio)

La figure 27 présente le métamodèle de Flowstone reprenant les constructions du langage et les relations entre ces constructions.

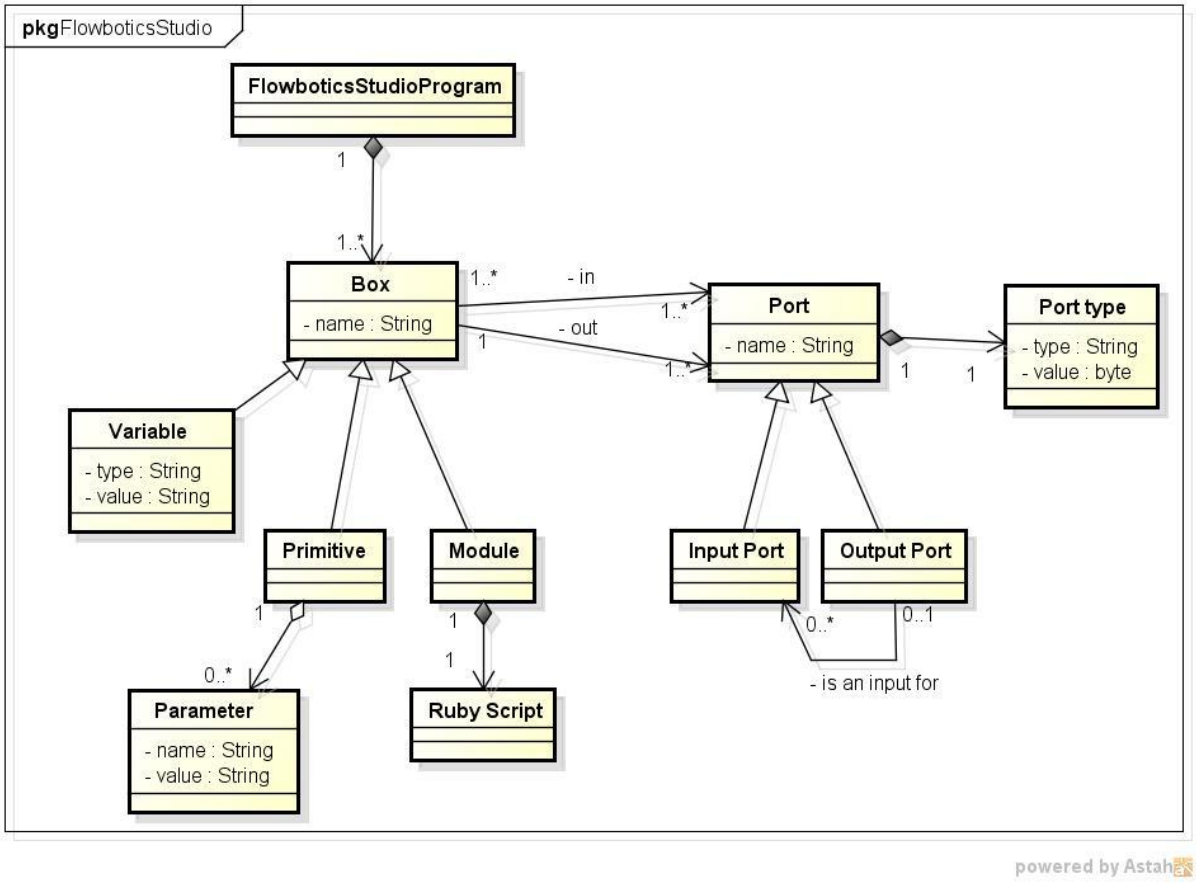


Figure 27 : Métamodèle du langage Flowstone de Flowbotics studio

4.5 Choregraphe

La figure 28 présente le métamodèle de Choregraphe reprenant les constructions du langage et les relations entre celles-ci.

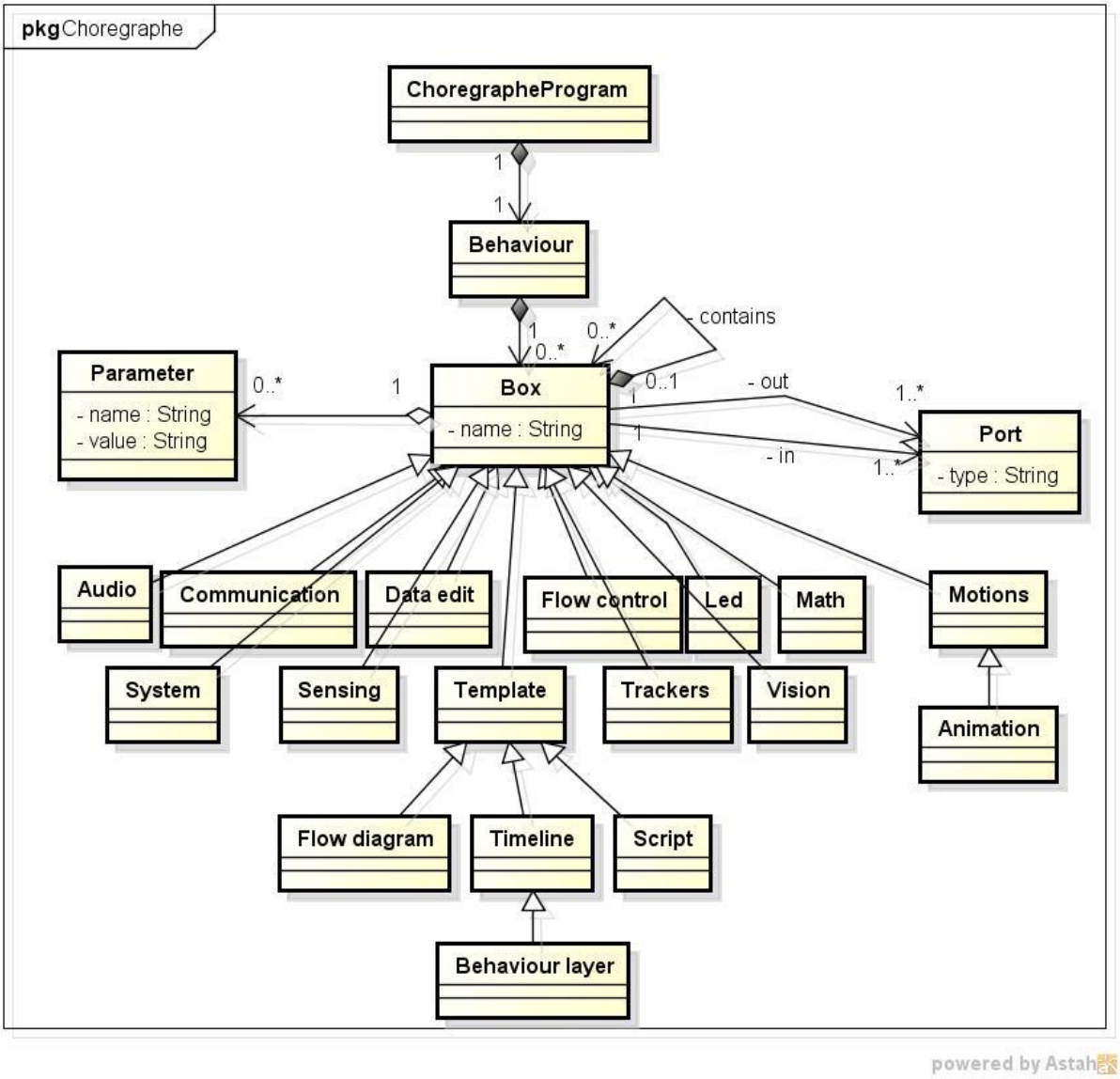


Figure 28 : Métamodèle du langage Choregraphe

5. Proposition d'un nouveau langage visuel

5.1 Introduction

Pour obtenir un langage expressif, nous avons choisi la présentation visuelle des langages de type DataFlow, le principe d'abstraction procédurale comme appliqué par Choregraphe, des couleurs et des formes de composants visuels tels que ceux présentés dans Scratch, l'utilisation de bulles externes comme paramètres de fonction et du typage de données tout comme cela est fait dans le langage Flowstone mais il a fallu faire des compromis. Il était par exemple difficile de garder le principe des détrompeurs de Scratch dans un langage de flux où les boites ne sont pas en contact les unes avec les autres.

5.2 Présentation des deux cas d'études

5.2.1 Cas d'étude n°1 : Le robot télécommandé

Un petit robot est télécommandé via une télécommande de télévision et reçoit ses ordres via un capteur infra-rouge. Il peut avancer, reculer, tourner à gauche ou à droite. Il est équipé d'un capteur de distance positionné à l'avant du robot. Si le robot est à moins de 10 centimètres d'un obstacle, il fait demi-tour automatiquement. Le robot est équipé de deux moteurs : un moteur pour la direction (port B) et un autre pour la propulsion (port A). Il est également équipé d'un capteur infra-rouge et d'un capteur de distance. Les moteurs et capteurs sont reliés à un élément contrôleur.

Un schéma du prototype est présenté en figure 29.

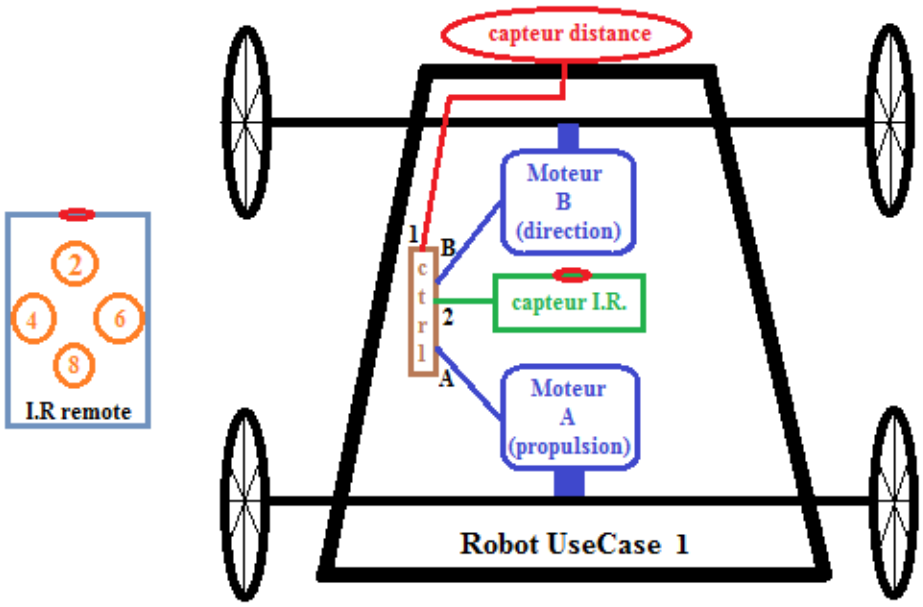


Figure 29 : Schéma du prototype de l'étude de cas n°1

Le programme visuel du cas d'étude n°1 est présenté dans les figures 30, 31 et 32.

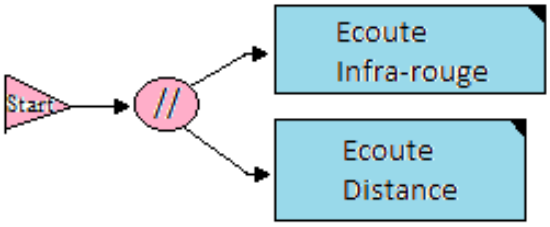


Figure 30 : Cas d'étude n°1 Exemple visuel (partie 1/3)

L'élément Start déclenche l'exécution du programme en appelant la construction (//) permettant de lancer l'exécution en parallèle de deux constructions CUSTOM : « Ecoute Infra-Rouge » et « Ecoute Distance ». La figure 30 nous montre un bon résumé de l'architecture générale du programme réalisé pour cette première étude de cas.

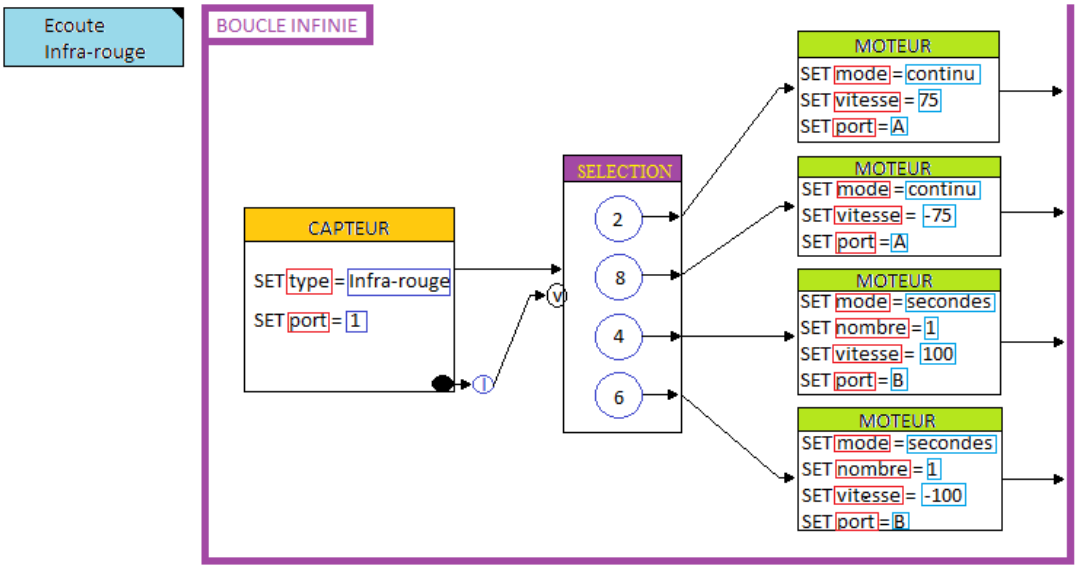


Figure 31 : Cas d'étude n°1 Exemple visuel (partie 2/3)

Dans la boucle infinie présentée en figure 31, le flux d'exécution se répète indéfiniment. Ce flux fait en premier lieu appel à un capteur "Infra-Rouge" connecté au port 1 de l'élément contrôleur. Ce capteur retourne une valeur entière qui sera évaluée dans la construction SELECTION suivante. Selon la valeur évaluée, une certaine construction de type MOTEUR sera exécutée.

Les deux constructions MOTEUR (en partant du haut) correspondent au moteur connecté au port A de l'élément contrôleur et ont le mode "continu" qui est défini. La valeur 2 de SELECTION permettra de faire tourner ce moteur, de manière continue, à une vitesse positive de 75 qui aura pour impact de faire avancer le robot; la valeur 8 de SELECTION fera tourner ce moteur à une valeur de -75 et fera tourner ce même moteur à la même vitesse mais fera, cette fois, reculer le robot.

Les deux constructions MOTEUR (en partant du bas) correspondent au moteur connecté au port B de l'élément contrôleur et ont le mode "secondes" qui est défini. La valeur 4 de SELECTION permettra de faire tourner ce moteur pendant une seconde à une vitesse de 100; ce qui aura pour effet de faire braquer instantanément les roues de direction du véhicule complètement vers la gauche (principe de direction avec retour au centre). La valeur 6 de SELECTION fera tourner ce même moteur à une vitesse de -100 pendant une seconde et fera alors braquer les roues du robot complètement vers la droite.

Une fois que l'exécution d'une construction MOTEUR s'est effectuée, le flux d'exécution revient au début de la boucle et refait le même scénario d'exécution.

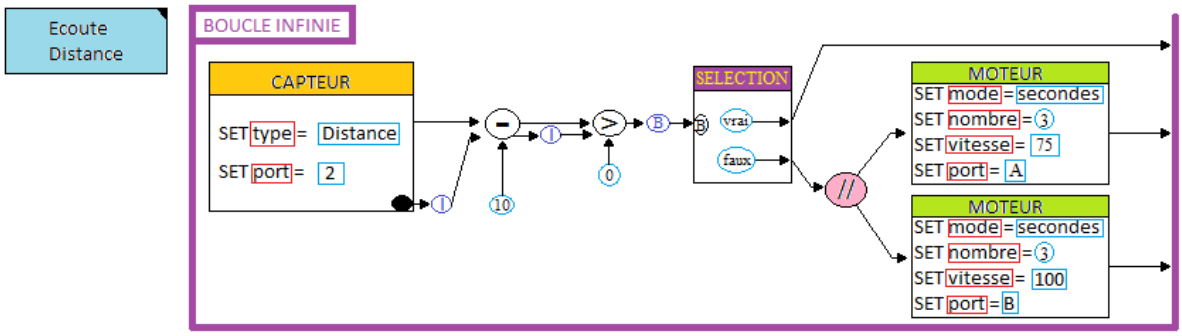


Figure 32 : Cas d'étude n°1 Exemple visuel (partie 3/3)

Dans la boucle infinie présentée en figure 32, le flux d'exécution se répète indéfiniment. Ce flux fait en premier lieu appel à un capteur "Distance" connecté au port 2 de l'élément contrôleur. Ce capteur retourne une valeur entière correspondante à la distance en centimètres qui sépare l'avant du véhicule d'un éventuel obstacle. Cette valeur est passée à la construction (-) ayant déjà (10) comme paramètre. Cette construction réalise la soustraction de dix de la valeur reçue (résultat = Valeur reçue - 10) et retourne le résultat.

Ce résultat est ensuite comparée à la valeur 0 via la construction (>) qui retourne une valeur booléenne indiquant si la valeur résultante est positive ou non. La valeur résultante booléenne est passée en paramètre à la construction SELECTION qui ne fera que rediriger vers la fin de la boucle dans le cas où la valeur transmise est positive (cas où le robot se situe à plus de dix centimètres d'un obstacle); par contre, si la valeur transmise est négative (le robot est à moins de dix centimètres d'un obstacle), il y aura déclenchement en parallèle de deux constructions MOTEUR qui mettront en route pendant trois secondes les moteurs branchés sur les ports A et B de l'élément contrôleur et cela respectivement à des vitesses de 75 et de 100; ce qui aura pour effet de faire avancer le robot tout en ayant constamment les roues avant braquées vers la gauche; le véhicule fera alors un demi-tour.

Une fois que ces moteurs ont tournés pendant trois secondes, le flux d'exécution retourne au début de la boucle et réitère le même scénario.

5.2.2 Cas d'étude n°2 : Le robot GPS

Un robot doit se diriger vers une certaine position donnée et une fois sur place, doit récupérer diverses informations environnementales (température, humidité, etc…). Le robot est équipé de deux moteurs : un moteur pour la direction (port B) et un autre pour la propulsion (port A). Il est également équipé d'un GPS, d'un capteur d'humidité et d'un capteur de température. Les moteurs et capteurs sont reliés à un élément contrôleur.

Un schéma du prototype est présenté en figure 33.

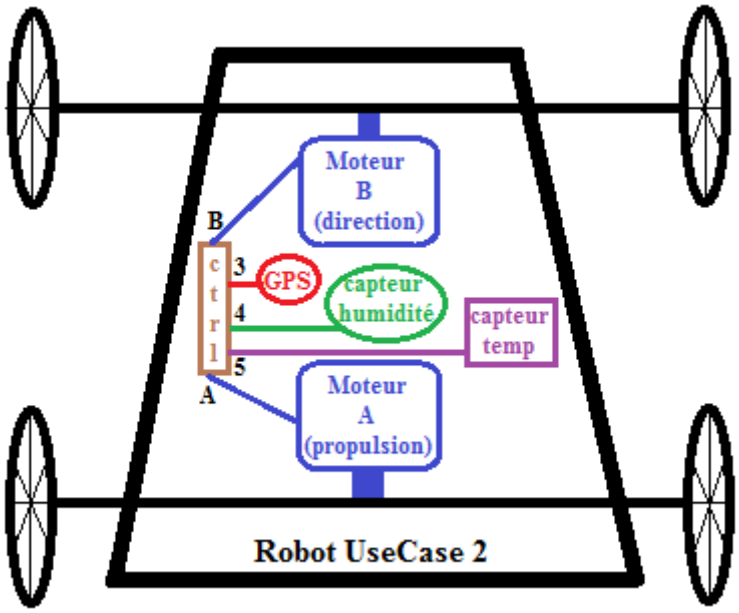


Figure 33 : Schéma du prototype de l'étude de cas n°2

Le programme visuel du cas d'étude n°2 est présenté dans les figures 34, 35, 36 et 37.

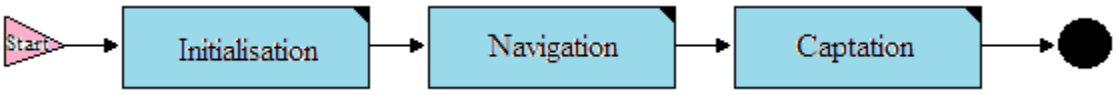


Figure 34 : Cas d'étude n°2 Exemple visuel (partie 1/4)

Dans la figure 34, l'élément Start déclenche l'exécution du programme en appelant en séquence la construction CUSTOM "Initialisation", la construction CUSTOM "Navigation", la construction CUSTOM "Captation" et enfin la construction STOP permettant d'arrêter l'exécution du programme.

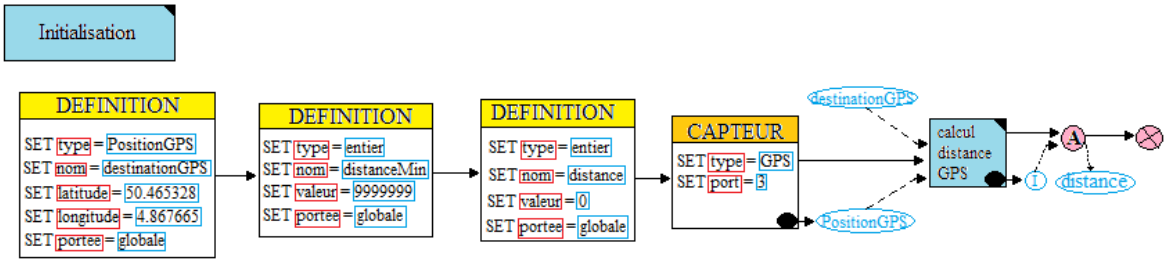


Figure 35 : Cas d'étude n°2 Exemple visuel (partie 2/4)

Dans la construction CUSTOM "Initialisation" présentée en figure 35, il y a exécution en séquence de trois constructions DEFINITION; la première permet de définir la variable "destinationGPS" comme étant de type "PositionGPS", de portée globale et ayant certaines valeurs pour sa longitude et sa latitude; la seconde construction définit la variable "distanceMin" comme étant de type entier, de portée globale et ayant une certaine valeur; la troisième construction permet de définir la variable "distance" comme étant de type entier, de portée globale et de valeur égale à zéro.

Le flux d'exécution exécute ensuite un appel au capteur GPS connecté au port 3 de l'élément contrôleur. Ce capteur retourne une valeur de type "PositionGPS"; la construction CUSTOM "calcul distance GPS" est alors appelée avec les paramètres "destinationGPS" et la valeur résultante de type "PositionGPS" du capteur; la construction CUSTOM retourne alors une valeur entière comme résultat.

Il y a ensuite appel de la construction d'affectation (A) que l'on utilise pour stocker la valeur entière résultante de la boîte CUSTOM dans la variable "distance; le flux d'exécution sort ensuite de la boîte CUSTOM "initialisation" et lance l'exécution de la boîte CUSTOM "Navigation".

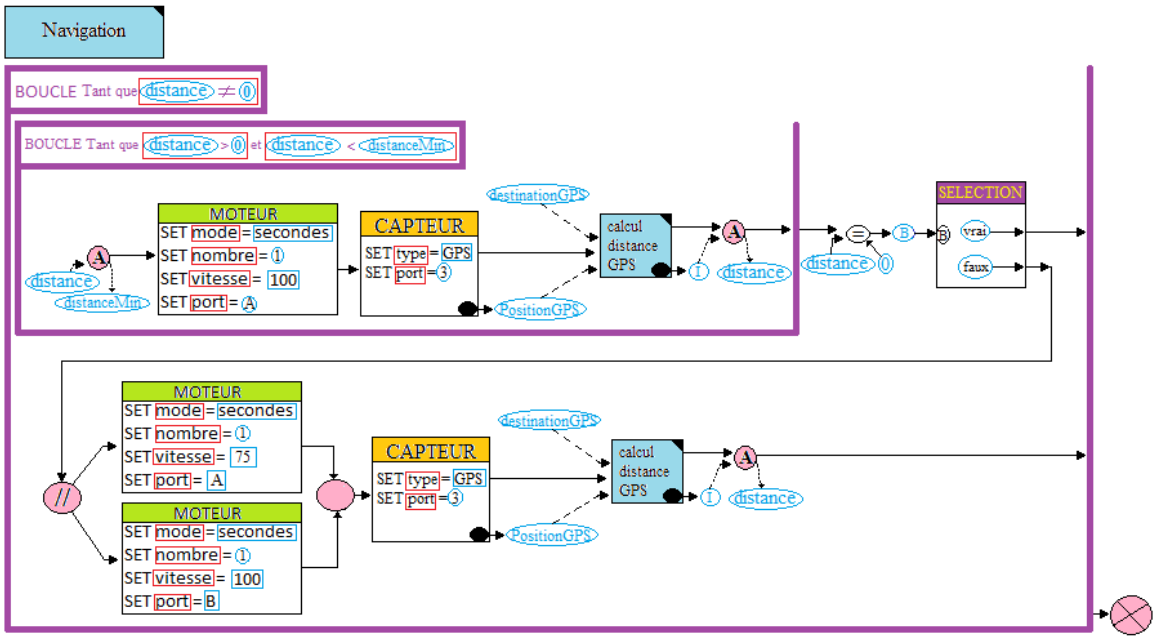


Figure 36 : Cas d'étude n°2 Exemple visuel (partie 3/4)

Dans la construction CUSTOM « Navigation » présentée en figure 36, on retrouve deux boucles finies imbriquées; la première boucle répète les différentes instructions jusqu'à ce que la condition (distance <> 0) ne soit plus vraie; l'ensemble des instructions de cette première boucle est alors répété un certain nombre de fois; la première construction exécutée est une deuxième boucle dans laquelle les

instructions sont également répétées tant que la condition suivante reste vraie : (distance > 0) et (distance < distanceMin).

Dans cette deuxième boucle, le flux d'exécution commence par assigner la valeur de la variable "distance" à la variable "distanceMin" via la construction d'affectation (A) puis fait tourner le moteur connecté au port A pendant une seconde à une vitesse de 100; le capteur GPS connecté au port 3 de l'élément contrôleur est ensuite appelé; ce capteur retourne une valeur de type "PositionGPS" qui avec la variable "destinationGPS" sont passés à la construction CUSTOM "calcul distance GPS" qui sort une valeur résultante de type entier, correspondante à la distance en mètres entre les deux coordonnées GPS, et qui est de suite assignée à la variable "distance"; le flux d'exécution arrive alors en fin de boucle, teste à nouveau la condition et sort éventuellement de la boucle ou répète encore une fois le scénario d'instructions. Le flux sort donc de la boucle interne si la variable « distance » n'est plus positive ou si sa valeur est supérieure ou égale à la valeur de « distanceMin ».

A la sortie de la deuxième boucle, le flux d'exécution exécute le test d'égalité (=) pour voir si la variable "distance" vaut zéro ou pas; dans le cas où la valeur booléenne retournée par ce test est vraie, le flux d'exécution sort de la boucle principale; dans le cas où la valeur booléenne retournée par ce test est fausse, le flux d'exécution va lancer en parallèle l'exécution de deux constructions MOTEUR qui vont mettre en marche les moteurs connectés aux ports A et B de l'élément contrôleur et vont fonctionner pendant une seconde respectivement à des vitesses de 75 et 100. Une fois que les moteurs se sont exécutés, la construction d'attente de flux va appeler la construction CAPTEUR GPS qui va donner une valeur de type "PositionGPS" comme résultat; ce résultat et la variable "destinationGPS" sont passés à la construction CUSTOM "calcul distance GPS" dont la valeur résultante va être assignée à la variable "distance" via l'opération d'affectation (A). Une fois cette affectation réalisée, le flux d'exécution va à nouveau évaluer la condition de la boucle principale qui, dans le cas où cette condition est toujours vraie, va reproduire une nouvelle fois le même scénario d'exécution.

Une fois que le flux d'exécution sort de la boucle principale, il sort également de la construction CUSTOM "Navigation" et va exécuter la construction CUSTOM "Captation" qui suit.

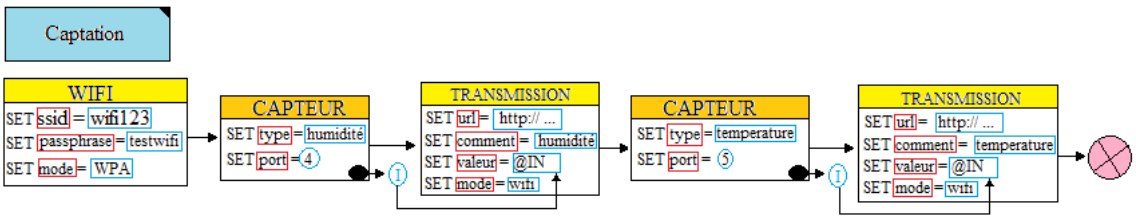


Figure 37 : Cas d'étude n°2 Exemple visuel (partie 4/4)

Dans la construction CUSTOM « Captation » de la figure 37, le flux d'exécution établit une connexion Wifi en appelant la construction WIFI puis fait appel à une construction CAPTEUR qui définit un capteur d'humidité connecté au port 4 de l'élément contrôleur et qui retourne une valeur entière comme résultat; cette valeur entière est passée comme argument à la construction TRANSMISSION suivante; Cette construction va transmettre, par internet, un commentaire définissant le type de capteur ("humidité") ainsi que la valeur observée par ce capteur (communiquée via le mot clé @IN).

Le flux d'exécution fait ensuite appel à un capteur de température et à une boîte TRANSMISSION afin de transmettre la valeur relevée par ce capteur d'une manière identique à celle utilisée pour la captation faite par l'autre capteur (humidité).

Le flux d'exécution sort ensuite de la construction CUSTOM et le programme s'arrête.

5.3 Le métamodèle du nouveau langage visuel

La figure 38 présente le métamodèle du nouveau langage visuel présenté pour la programmation des deux cas d'études.

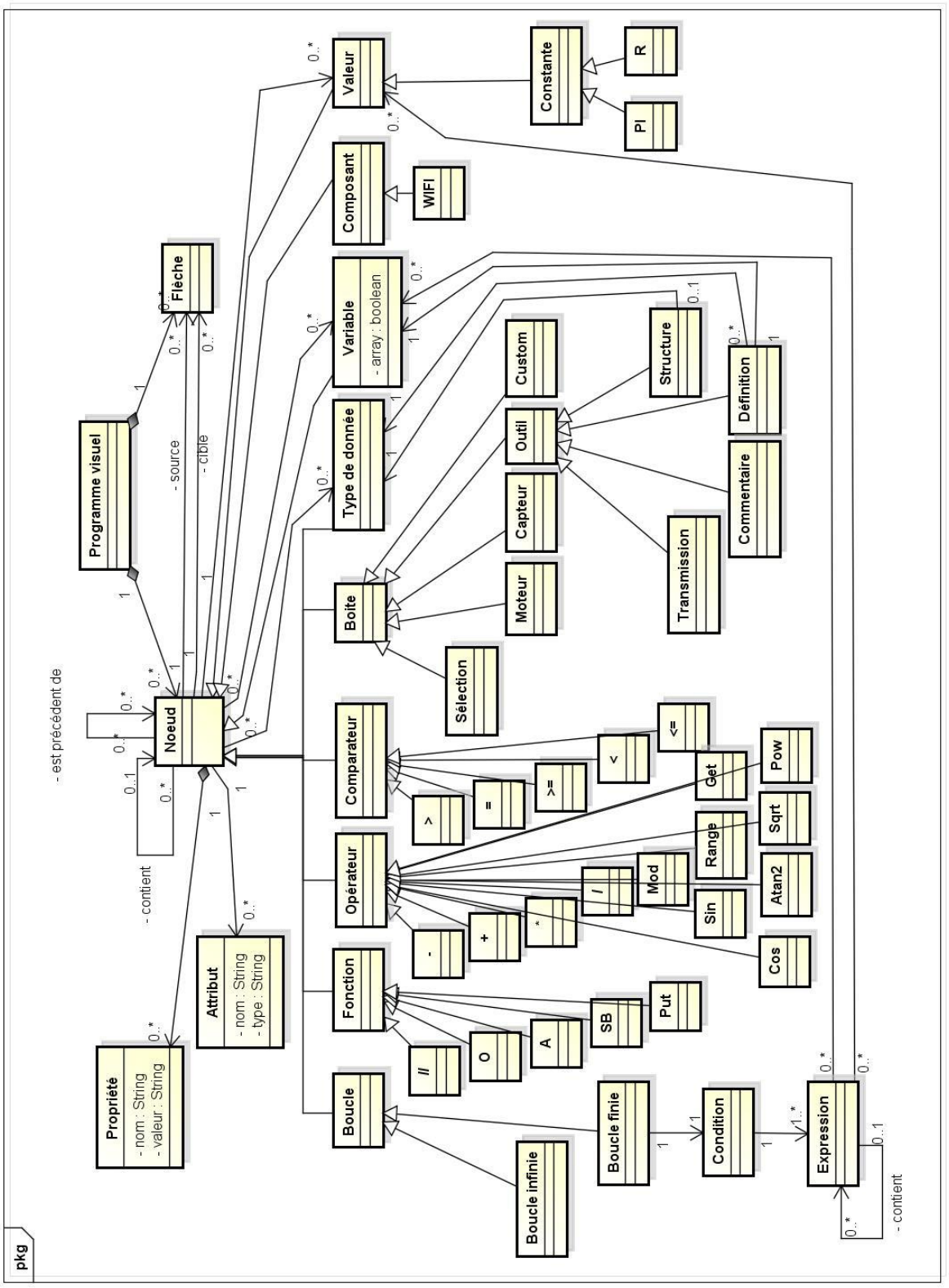
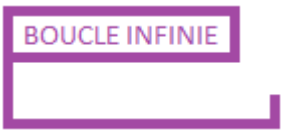
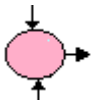
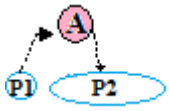
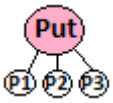


Figure 38 : Métamodèle d'un nouveau langage visuel

5.4 Elicitation des constructions du nouveau langage

 Figure 39 : Construction : boucle infinie	**Nom : Boucle infinie** (ou boucle finie avec condition jamais falsifiable : tant que vraie) Cette construction est composée d'une seule pièce. Pour sortir de la boucle, il faut utiliser la fonction de sortie de boucle. La séquence de blocs, contenue dans la boucle, sera reproduite à chaque itération dans la boucle et cela de façon infinie s'il n'y a aucune interruption volontaire de la structure répétitive. Étant un nœud comme un autre, une boucle peut parfaitement contenir d'autres boucles.
Boucle finie  Figure 40 : Construction : boucle finie	**Nom : Boucle finie** La boucle finie reprend les caractéristiques citées plus haut à la seule différence qu'il y a répétition tant que la condition de boucle est vérifiée à chaque itération. Une condition peut être composée de plusieurs expressions booléennes, séparées par des parenthèses et des opérateurs AND ou OR. Ces expressions peuvent inclure d'autres expressions et peuvent concerner diverses valeurs et / ou variables.
Sortie de boucle  Figure 41 : Construction : sortie de boucle	**Nom : SB** Permet de sortir d'une boucle finie ou infinie Est souvent employé après qu'une condition ait été vérifiée Est le seul moyen de sortir d'une boucle infinie Ne peut être utilisée que dans une boucle.
Lancement de plusieurs tâches en parallèle  Figure 42 : Construction : lancement simultané de plusieurs tâches	**Nom : //** Permet de préciser le lancement de plusieurs tâches en simultané. En entrée, nous avons une flèche. En sortie, nous avons autant de flèches qu'il y a de processus à exécuter. Cette construction est généralement suivie de  (construction d'attente de plusieurs processus lancés en parallèle pour faire repartir l'activité en un seul flux (figure 43)).

Attente de processus  Figure 43 : Construction : attente de processus	**Nom : O** Mise en attente de plusieurs processus afin de ramener plusieurs flux d'instructions à un seul flux. En entrée, plusieurs flèches de flux ; en sortie, nous aurons une seule flèche de flux.
(A) fonction d'affectation P1->P2  Figure 44 : Construction : fonction d'affectation	**Nom : A** Après l'exécution de cette fonction, P2 va valoir la valeur de P1. P2 est obligatoirement une variable ; P1 peut être une variable ou une valeur.
fin partielle de flux  Figure 45 : Construction : fonction de fin partielle de flux	**Nom : X** Notification de la fin partielle d'un flux. Le programme n'est pas terminé juste après ; d'autres flux peuvent encore se terminer après.
fin du programme  Figure 46 : Construction : fonction de fin de programme	**Nom : Stop** Stoppe l'exécution du programme
(Start)Début programme  Figure 47 : Construction : fonction de début de programme	**Nom : Start** Déclenche l'exécution du programme
Put  Figure 48 : Construction : fonction Put	**Nom : Put** P3 doit être une variable de type tableau de données. P2 est la clé du nouvel élément Après l'exécution de cette fonction P1, identifié par P2, sera le nouvel élément du tableau P3.

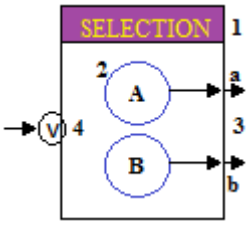


Figure 49 : Construction : boîte Sélection

Nom : Sélection

La boîte contient les différentes possibilités pour la valeur qui va lui être présentée en entrée de boîte. La valeur V donnée sera évaluée par rapport à diverses possibilités (A ou B) et la flèche déclencheur correspondante sera exécutée (a ou b).

1. La boîte conteneur des différentes possibilités de sélection
2. Les différentes options pour la sélection
3. Les flèches déclencheuses vers des parties de programme bien déterminées
4. Une valeur d'un certain type de donnée qui sera présenté en entrée de la boite et avec laquelle les différents choix seront évalués.

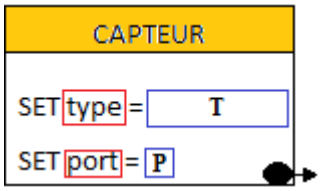


Figure 50 : Construction : boîte capteur

Nom : Capteur

Les capteurs sont d'un type T et sont connectés à un port P.

Il y a différents types de capteurs : Infra-rouge, Distance, GPS, Humidité, Température, Luminosité

Les ports sont numérotés de 1 à 8 sur le contrôleur.
Il n'y a pas de variable en entrée mais il y a une flèche, en sortie, pointant vers le type de la valeur retournée par le capteur (I) ou (PositionGPS)

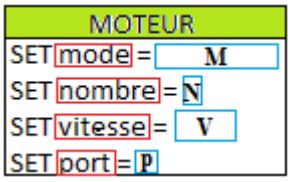


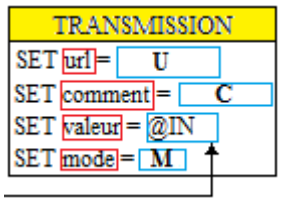
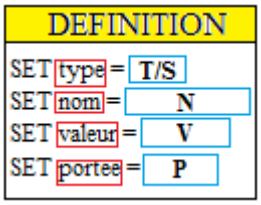
Figure 51 : Construction : boîte moteur

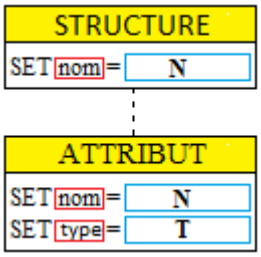
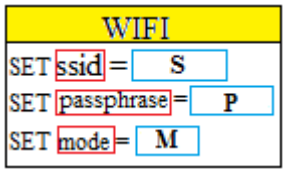
Nom : Moteur

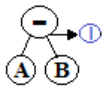
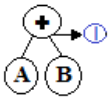
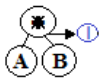
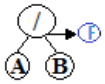
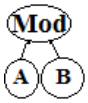
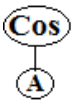
Cette construction représente le déclenchement d'un moteur, connecté à l'un des ports P de l'élément contrôleur, qui va tourner à une vitesse V selon un mode prédéfini M (continu, durant un nombre de secondes N, pour un certain nombre de rotations N).

Si ce mode est « durant-secondes » ou « rotation », il y a utilisation d'une nouvelle propriété « nombre » qui va indiquer le nombre de secondes ou de rotations. Si le mode est « arrêt », seule la propriété « Port » sera précisée.

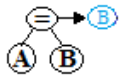
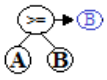
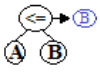
Si le mode est « continu » ; cela signifie que le moteur tournera en continu tant qu'on ne le stoppe pas ; « durant-secondes » signifie qu'il tournera pendant un certain nombre de secondes ; « rotation » signifie qu'il effectuera un certain nombre de rotations. « arrêt » permettra de stopper le moteur.

CUSTOM  Figure 52 : Construction : boîte Custom	**Nom : Custom** Boîte qui est employée pour définir de nouveaux blocs contenant des sous-diagrammes du programme. Le bloc Custom peut accepter des paramètres en entrée et peut retourner une valeur d'un certain type de données. Ils sont également employés pour définir les grandes lignes du programme afin que l'utilisateur dispose déjà d'une certaine synthèse du programme dès le début. Ce qui rendra la lecture du code plus compréhensive. Lors de l'appel à ce composant, les paramètres qui lui sont passés y sont reliés par des pointillés.
 Figure 53 : Construction : boîte Transmission	**Nom : Transmission** Permet de transmettre une valeur via une URL de type POST qui prendra comme paramètres un commentaire ainsi qu'une valeur à transmettre. La valeur sera envoyée vers un web service de type RESTful. Il y a toujours une flèche de données en entrée. La valeur en entrée sera assignée au paramètre @IN attribué à l'attribut « valeur » de la construction TRANSMISSION. U (URL) doit être une URL pour envoyer des données en mode POST. Le commentaire C permet de différencier les données envoyées. La valeur est la donnée à transmettre. Le mode M désigne le moyen de transport utilisé pour véhiculer les données ; wifi est l'un de ces modes.
 Figure 54 : Construction : boîte Definition	**Nom : Définition** Permet de définir une variable selon un certain type de données T ou selon une certaine structure S. Elle comprend les attributs nom N, type T/S et valeur V et est peut-être un élément tableau. Nous définissons également la portée P de la variable (locale ou globale). Une construction DEFINITION concerne une variable à définir pour un type normal de donnée ou pour une structure de type de donnée si celle-ci a précédemment été déclarée.

	Une structure de données est définie d'une manière similaire à celle présentée dans la figure 55. En effet, une structure de données comporte un nom et est associée à autant de constructions ATTRIBUT qu'il y a de données à définir pour cette structure. Une construction ATTRIBUT comporte deux propriétés : un nom et un type de donnée ; la déclaration d'une STRUCTURE dans un programme consiste à placer cette construction dans le programme avant son utilisation dans une structure DEFINITION. Il n'y a pas de flèche de données en sortie de la boîte DEFINITION mais une fois cette boîte exécutée, la variable devient accessible et modifiable dans le reste du programme.
COMMENTAIRE ||	**Nom : Commentaire** Permet de préciser des commentaires entre 2 paires de barres parallèles et cela n'importe où dans l'espace visuel du programme
 Figure 55 : Construction : Boîte Structure et boîte Attribut	**Nom : Structure** La construction STRUCTURE permet de préciser un nouveau type de données. Elle dispose d'une seule propriété qui permet de lui préciser un nom N. Chacun des attributs de ce nouveau type de donnée sera indiqué par une construction ATTRIBUT qui précisera un nom N et un type de donnée T. Ces attributs seront reliés à la construction STRUCTURE par des lignes pointillées. Par exemple, la structure de donnée « PositionGPS » est associée à 2 constructions ATTRIBUT comportant le même type de donnée « Float » et respectivement les noms "longitude" et "latitude". Cependant, le type de données « PositionGPS » est un type connu du langage et ne doit dès lors pas être défini en début de tout programme le mentionnant. Une fois la construction STRUCTURE définie dans le programme, le nouveau type de donnée peut être mentionnée dans une construction DEFINITION comme type de donnée de la variable à définir. Une STRUCTURE étant associée à un nouveau type de donnée.
 Figure 56 : Construction : Composant Wifi	**Nom : Wifi** Permet d'initier une connexion Wifi selon un SSID S, une phrase « mot de passe » P et un mode M. Une fois cette boite déclarée dans le programme, le mode wifi pourra être utilisé dans les boîtes TRANSMISSION qui suivront.

 Figure 57 : Construction : Opérateur de soustraction	**Nom : -** Opérateur qui effectue l'opération A - B et qui retourne un entier C
 Figure 58 : Construction : Opérateur d'addition	**Nom : +** Opérateur qui effectue l'opération A + B et qui retourne un entier C
 Figure 59 : Construction : Opérateur de multiplication	**Nom : *** Opérateur qui effectue l'opération A * B et qui retourne un entier C
 Figure 60 : Construction : Opérateur de division	**Nom : /** Opérateur qui effectue l'opération A / B et qui retourne un nombre décimal de type virgule flottante (float) C
 Figure 61 : Construction : Opérateur Modulo	**Nom : Mod** Opérateur qui effectue l'opération Modulo de A avec B et qui retourne un entier C
 Figure 62 : Construction : Opérateur Sinus	**Nom : Sin** Opérateur qui effectue l'opération Sin(A) et qui retourne un nombre décimal de type virgule flottante (float) C
 Figure 63 : Construction : Opérateur Cosinus	**Nom : Cos** Opérateur qui effectue l'opération Cos(A) et qui retourne un nombre décimal de type virgule flottante (float) C
 Figure 64 : Construction : Opérateur ArcTangente	**Nom : Atan2** Opérateur qui retourne une valeur entière comprise entre –Pi et Pi et qui représente l'angle Theta d'un point (x,y). Opérateur utilisé pour le calcul du nombre de mètres en deux positions GPS[40]. La première valeur est la coordonnée Y ; la seconde est X.

[40] https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Math/atan2

 Figure 65 : Construction : Opérateur Range	**Nom : Range** Opérateur qui prend une valeur entière E à l'échelle A-B et retourne cette valeur mise à l'échelle C-D
 Figure 66 : Construction : Opérateur Racine	**Nom : Sqrt** Opérateur qui calcule la racine A du nombre B et qui retourne le résultat C. Pour une racine carrée, A vaut 2.
 Figure 67 : Construction : Opérateur Puissance	**Nom : Pow** Opérateur qui effectue l'opération de A à la puissance B et qui retourne un entier C
 Figure 68 : Construction : Opérateur Get	**Nom : Get** Opérateur qui récupère l'élément identifié par A dans la collection B et qui retourne le résultat C
 Figure 69 : Construction : Comparateur d'égalité	**Nom : =** Opérateur qui teste l'égalité de 2 paramètres (A et B) et retourne une valeur booléenne vraie / fausse selon le résultat de la comparaison
 Figure 70 : Construction : Comparateur Plus grand que	**Nom : >** Opérateur qui teste A > B et retourne C, le résultat de la comparaison sous forme booléenne.
 Figure 71 : Construction : Comparateur Plus grand ou égal à	**Nom : >=** Opérateur qui teste A >= B et retourne C, le résultat de la comparaison sous forme booléenne.
 Figure 72 : Construction : Comparateur Plus petit que	**Nom : <** Opérateur qui teste A < B et retourne C, le résultat de la comparaison sous forme booléenne.
 Figure 73 : Construction : Comparateur Plus petit ou égal à	**Nom : <=** Opérateur qui teste A <= B et retourne C, le résultat de la comparaison sous forme booléenne.
 Figure 74 : Construction : Comparateur Différent de	**Nom : <>** Opérateur qui teste A <> B et retourne C, le résultat de la comparaison sous forme booléenne.

Types de données du langage

 Figure 75 : Type de donnée numérique (entier)	Type de donnée numérique (entier)
 Figure 76 : Type de donnée booléenne	Type de donnée booléenne
 Figure 77 : Type de donnée numérique (avec décimales)	Type de donnée numérique (avec décimales)
 Figure 78 : Type de donnée Chaîne de caractères	Type de donnée chaîne de caractères
 Figure 79 : Type de donnée PositionGPS	Type de donnée PositionGPS

5.5 Description de la sémantique du langage

Les commentaires peuvent être précisés n'importe où et doivent être repérés par deux barres parallèles (||).

(X), présenté dans la figure 45, est une construction indiquant la fin d'un flux d'exécution. Elle est placée en fin d'un flux d'exécution situé après la construction (Start) et avant la construction de fin de programme présentée dans la figure 46. Dans le programme, chaque flèche qui ne transmet pas de donnée, agit comme déclencheur de la boîte qui suit.

Le programme visuel exploite l'utilisation des variables globales qui une fois créées avec des blocs DEFINITION (figure 54) peuvent être modifiées ou utilisés dans le reste du programme. Les variables sont utilisées pour être passées en paramètres d'entrées de fonctions telles que « A », « Put » ou de blocs CUSTOM. Avant l'utilisation d'une variable, celle-ci doit avoir été déclarée avant via une construction DEFINITION.

La construction STRUCTURE (figure 55) permet de définir un nouveau type de donnée ; elle est entourée de plusieurs constructions ATTRIBUT se reliant à elle par des pointillés. Chaque construction ATTRIBUT définit un nom d'attribut pour ce type ainsi que le type de donnée de cet attribut.

La construction STRUCTURE doit être placée avant une construction DEFINITION s'y référant pour le type de la donnée qu'elle souhaite définir.

Le bloc CUSTOM (figure 52) contient un sous-diagramme ; il peut accepter des paramètres en entrée et peut retourner une valeur. L'utilisation des blocs CUSTOM permet de préciser rapidement l'organisation et la structure du programme.

Lors de la définition d'un bloc de ce genre, nous déclarons dans sa signature, les variables d'entrées ainsi qu'un éventuel type de donnée en sortie. Ces paramètres apparaîtront sur le bord du bloc CUSTOM.

Une construction TRANSMISSION (figure 53) sera employée pour autant qu'une variable ou une valeur calculée préalablement puisse lui être assignée. Si le mode Wifi est choisi, une construction WIFI (figure 56) devra avoir été déclarée avant l'exécution de la boite TRANSMISSION. Le symbole @IN sera employé pour signifier l'acquisition d'une valeur externe à la boîte.

Une construction CAPTEUR (figure 50) peut être placée partout ; Celle-ci, liée à un port bien précis d'un élément contrôleur, génère une valeur de résultat représentée par une flèche vers un certain type de donnée. Plusieurs propriétés sont associées à cette construction avec pour chacune, une valeur spécifique. Ces blocs seront souvent utilisés en conjonction avec d'autres blocs permettant d'évaluer la valeur retournée des capteurs et d'agir en conséquence.

MOTEUR (figure 51) est une construction pouvant être utilisée partout ; elle contient diverses propriétés ainsi que leurs valeurs. L'une de ces propriétés est le port de la brique auquel est connecté le capteur. Une boîte MOTEUR ne génère pas de valeur de sortie. Ces blocs seront généralement exécutés après qu'une certaine condition ait été vérifiée.

Les constructions BOUCLE (figures 39 et 40), SELECTION (figure 49), (||) et O (figure 43) sont les éléments gérant le flux d'exécution dans le programme.

Une boucle permet de contenir tout un ensemble de blocs d'instructions qui seront répétées plusieurs fois. Cette répétition sera infinie dans le cas d'une boucle infinie.

Une boucle finie verra son exécution dépendre de la validité d'une certaine condition ; une fois cette condition faussée, le processus sortira de la boucle et le fil d'exécution se poursuivra.

La boucle finie doit définir une condition qui sera ainsi la cause de sortie de cette boucle. Cette condition se compose d'une ou de plusieurs expressions booléenne imbriquées, d'opérateurs conditionnels comme AND ou OR et fait appel à des variables et/ou des valeurs définies précédemment dans le programme.

La boucle entoure l'ensemble des blocs qui vont devoir être répétés ; la fonction de sortie de boucle SB (figure 41) permet de faire une sortie de boucle de manière anticipative. Cette fonction ne peut être utilisée qu'à l'intérieur d'une boucle et ne concerne que la boucle conteneur.

Dans le cas d'une boucle finie, la condition est évaluée une fois avant l'entrée de la boucle et puis à chaque itération dans la boucle.

Dans une boucle, on peut placer toute sorte de blocs tels que d'autres boucles, boîtes de sélection, boîtes CAPTEUR, boîtes MOTEUR, des fonctions ou des opérateurs permettant d'exploiter le contenu de variables en redirigeant le flux en fonction des résultats, des valeurs, des éléments permettant de déclencher plusieurs actions à la fois, des assignations de variables ou encore des appels à des blocs CUSTOM existants.

Un bloc SELECTION évaluera une valeur, d'un certain type, en entrée et exécutera la flèche déclencheuse se rapportant à la valeur correspondante ainsi évaluée.

Un bloc (//) permettra de lancer plusieurs processus en parallèle et sera généralement suivi d'un bloc (0) qui attendra que l'ensemble des processus soit terminés avant de poursuivre un certain flux d'exécution. Si deux processus infinis sont lancés en parallèle, le bloc (0) peut ne pas être placé vu que le flux d'exécution ne l'atteindra jamais.

Les opérateurs **(-, +, *, /,** Mod, Atan2, Range, Put, Get, Sqrt et Pow) sont des opérateurs mathématiques nécessitant deux paramètres et qui retournent une valeur entière calculée. Les autres opérateurs Sin et Cos ne nécessitent qu'un seul paramètre.

Les comparateurs (=, >, >=, <, <=) nécessitent la présence de 2 autres nœuds pour leurs entrées et retournent une valeur booléenne ; un type de donnée sera employé pour préciser le type de la donnée résultante en sortie d'un bloc CUSTOM, d'un bloc CAPTEUR, d'un opérateur ou d'un comparateur.

Les opérateurs mathématiques et de comparaison utilisent deux paramètres en entrée de ceux-ci ; l'ordre des paramètres a donc de l'importance pour ceux-ci.

La fonction d'assignation A (figure 44) est une fonction d'assignation d'une valeur à une variable, qui attend deux paramètres et qui ne retourne rien ; elle nécessitera toujours, en second paramètre, un nom

de variable préalablement défini via une boite DEFINITION. Le premier paramètre peut être soit un type de données pour indiquer que l'on utilise une valeur de ce type, soit une variable ou une valeur.

Le bloc TRANSMISSION nécessite que la valeur à transmettre soit passée en paramètre via un nœud « type de donnée » et une ligne en pointillés le reliant vers ce bloc. Le bloc Transmission ne pourra préciser la propriété mode et la valeur WIFI uniquement si le bloc WIFI a été déclaré avant dans le programme.

Les blocs tels que DEFINITION, TRANSMISSION, COMMENTAIRE ou WIFI n'émettent pas de flèche de donnée.

La fonction « Put » permet de placer un nouvel élément dans une collection. Cet élément sera identifié par une clé unique passée en paramètre.

6. Conclusion et futurs travaux

Ce travail vous a présenté un état de l'art relatif aux projets de prototypages ainsi qu'aux langages visuels les plus représentatifs permettant de programmer des projets comprenant différents composants électroniques évolués. Dans cet état de l'art, nous avons évoqués les forces et faiblesses de ces différents langages et par le biais de métamodèles de ceux-ci, nous vous avons présenté un aperçu de leur syntaxe abstraite et avons mis en évidence leurs constructions respectives, les constructions d'un langage étant les éléments exprimant toute la force de ce langage. L'état de l'art a également critiqué l'aspect visuel des constructions de ces langages afin de donner un aperçu de leur syntaxe concrète.

Nous avons de cette manière extrait le génome de ces langages et de leur paradigme via leurs métamodèles. De cette analyse, nous avons mis en évidence les éléments positifs et négatifs de ceux-ci afin de ne garder que les éléments visuels positifs qui nous ont permis d'établir les bases graphiques d'un langage visuel se voulant expressif et générique.

Deux études de cas ont été imaginées puis traduites selon la syntaxe concrète visuelle de ce nouveau langage. Nous avons ensuite présentés le métamodèle ou syntaxe abstraite de ce nouveau langage ainsi qu'une présentation des différentes constructions de celui-ci et une explication de l'agencement des composants visuels dans ce langage (sémantique).

Nous avons finalement complété le métamodèle du langage imaginé avec d'autres constructions utiles pour de futurs projets de prototypage électroniques.

Nous pensons que le langage visuel proposé peut répondre aux attentes des utilisateurs débutants car ce langage différencie ses composants par des arrangements de formes et de couleurs et ne surcharge pas l'utilisateur avec un nombre important de constructions à l'aspect visuel complexe. L'utilisateur peut organiser très vite son programme et peut le communiquer à un tiers pour une relecture aisée de celui-ci. Toutefois les syntaxes (abstraite et concrète) proposées pourraient encore être améliorées afin de pouvoir être encore plus adaptables à des projets d'électroniques tout en gardant un caractère expressif et générique.

Futurs travaux

- Evaluer et mesurer l'expressivité du nouveau langage proposé.
- Réaliser une implémentation de ce nouveau langage.
- Illustrer d'autres études de cas.
- Définir des exemples liés à autre chose qu'à la robotique.

7. Bibliographie

[Slany, 2012] Wolfgang Slany, "Catroid: A Mobile Visual Programming System for Children" , Proceeding IDC '12 Proceedings of the 11th International Conference on Interaction Design and Children, Pages 300-303, 2012

[Vasek, 2012] Marie Vasek, "Representing Expressive Types in Blocks Programming Languages", 2012

[Marttila-Kontio, 2011] Marttila-Kontio Maija, "Visual data flow programming languages challenges and opportunities», Publications of the University of Eastern Finland. Dissertations in Forestry and Natural Sciences, no 30, 2011

[Koitz et Slany, 2014] Roxane Koitz , Wolfgang Slany , "Empirical Comparison of Visual to Hybrid Formula Manipulation in Educational Programming Languages for Teenagers" , Proceeding PLATEAU '14 Proceedings of the 5th Workshop on Evaluation and Usability of Programming Languages and Tools , Pages 21-30 , 2014

[Cox et Dang, 2010] Philip T. Cox, Anh Dang, "Semantic Comparison of Structured Visual Dataflow Programs" , Proceeding VINCI '10 Proceedings of the 3rd International Symposium on Visual Information Communication , Article No. 11 , 2010

[Boubaker, 2011] Olfa Boubaker , "National Instruments® LabVIEW™: Ultimate Software for Engineering Education" , 2011

[Moody, 2009] Daniel L. Moody, "The "Physics" of Notations: Towards a Scientific Basis for Constructing Visual Notations in Software Engineering", Published in Software Engineering, IEEE Transactions (Volume: 35, Issue: 6), pages 756 – 779, 2009

[Cho et Gray, 2011] Hyun Cho et Jeff Gray, "Design Patterns for Metamodels" , Proceeding SPLASH '11 Workshops Proceedings of the compilation of the co-located workshops on DSM'11, TMC'11, AGERE! 2011, AOOPES'11, NEAT'11, & VMIL'11, Pages 25-32 , 2011

[Cho, 2011] Hyun Cho , "A Demonstration-Based Approach for Designing Domain-Specific Modeling Languages" , Proceeding OOPSLA '11 Proceedings of the ACM international conference companion on Object oriented programming systems languages and applications companion, Pages 51-54 , 2011

[Hils, 1992] Daniel D Hils , "Visual languages and computing survey: Data flow visual programming languages" , Journal of Visual Languages & Computing , Volume 3, Issue 1, March 1992, Pages 69–101

[Stolee et Fristoe, 2011] Kathryn T. Stolee , Teale Fristoe , "Expressing Computer Science Concepts Through Kodu Game Lab" , Proceeding SIGCSE '11 Proceedings of the 42nd ACM technical symposium on Computer science education , Pages 99-104 , 2011

[Conversy, 2014] Stephane Conversy, "Unifying Textual and Visual: a Theoretical Account of the Visual Perception of Programming Languages», Proceeding Onward! 2014 Proceedings of the 2014 ACM International Symposium on New Ideas, New Paradigms, and Reflections on Programming & Software, Pages 201-212, 2014

[Honig, 2013] William L. Honig , "Teaching and Assessing Programming Fundamentals for Non Majors with Visual Programming" , Proceeding ITiCSE '13 Proceedings of the 18th ACM conference on Innovation and technology in computer science education Pages 40-45 , 2013

[Stolee, 2010] Kathryn T. Stolee , "Kodu Language and Grammar Specication" , August 27, 2010

[Terbuc,Globacnik et Majhenic, 2007] Martin Terbuc, Gregor Globacnik, Danilo Majhenic,"FeriX Live GNU/Linux for Education", 2007

[Nehra, 2014] Vijay Nehra,"Free Open Source Software in Electronics Engineering Education: A Survey",Published Online May 2014 in MECS (http://www.mecs-press.org/), 2014

8. Autres documents lus

[Kiper, Howard et Ames, 1997] J. D. Kiper, E. Howard et C. Ames, “Criteria for Evaluation of Visual Programming Languages,” Visual Languages and Computing, 8, 175–192, 1997

[Martin Erwig, 1998] Martin Erwig , "Abstract Syntax and Semantics of Visual Languages",Journal of Visual Languages & Computing, Pages 461–483, October 1998, Volume 9, Issue 5

[Do et Fekete, 2008] Thanh-Nghi Do , Jean-Daniel Fekete , "Un environnement de programmation visuelle pour la fouille de données", Numéro spécial de la revue RIA, Revue d'Intelligence Artificielle, Hermès, 2008, 22 (3-4), pp.503

[Clerici et Zoltan, 2004] Silvia Clerici, Cristina Zoltan , "A graphic functional-dataflow language" , 2004

[Sousa, 2012] Tiago Boldt Sousa, "Dataflow Programming Concept, Languages and Applications" , Doctoral Symposium on Informatics Engineering, 2012

[Bender, Laurin, Lawford, Ong, Postma et Pantelic, 2014] Marc Bender, Karen Laurin, Mark Lawford, Jeff Ong, Steven Postma and Vera Pantelic , "Signature Required: Making Simulink Data Flow and Interfaces Explicit" , Model-Driven Engineering and Software Development (MODELSWARD), 2014 2nd International , Pages 119-131

[Burnett, Atwood, Djang, Gottfried, Reichwein, Yang, 2001] Margaret Burnett, John Atwood, Rebecca Walpole Djang, Herkimer Gottfried, James Reichwein, Sherry Yang , "Forms/3: A First-Order Visual Language to Explore the Boundaries of the Spreadsheet Paradigm" , Journal of Functional Programming / Volume 11 / Issue 02 / 2001, pp 155-206

[Hossli, Pulliam, Vanvoorst et Yang, 1994] Richard Hossli, tim Pulliam , Brian Vanvoorst , Xiaoyang Yang , "Toward visual programming languages for steering scientific computations" , Published in Computational Science & Engineering, IEEE (Volume:1 , Issue: 4) , 1994

[Averbukh et Bakhterev, 2013] Vladimir Averbukh et Mikhail Bakhterev, "The analysis of visual parallel programming languages" , ACSIJ Advances in Computer Science: an International Journal, Vol. 2, Issue 3, No. , 2013

[Dearman, Cox et Fisher, 2005] David Dearman, Anthony Cox, Maryanne Fisher, "Adding Control-Flow to a Visual Data-Flow Representation", Published in Program Comprehension, 2005. IWPC 2005. Proceedings 13th International Workshop on 15-16 May 2005, pages 297 - 306

[Lee, Hamilton et Parastatidis, 2004] P.A. Lee, M.D. Hamilton, S. Parastatidis, "A Visual Language for Parallel,Object-Oriented Programming", 2004

[Lee et Webber, 2003] P. A. LEE et J. WEBBER, "Taxonomy for Visual Parallel Programming Languages", 2003

[Browne, Hyder, Dongarra, Moore et Newton, 1995] James C. Browne , Syed I. Hyder , Jack Dongarra, Keith Moore et Peter Newton , "Visual Programming and Debugging for Parallel Computing" , Published by the IEEE Computer Society , Issue No.01 - Spring (1995 vol.3) pages 75-83

[Auguston et Delgado, 1997] Mikhail Auguston, Alfredo Delgado, "Iterative Constructs in the Visual Data Flow Language" , page 152- , 1997

[Blackwell & Green, 1999] A.F. Blackwell et T.R.G. Green, "Does Metaphor Increase Visual Language Usability?" , Published in Visual Languages, 1999. Proceedings 1999 IEEE Symposium, pages 246-253, 1999

[Cox et Gauvin, 2011] Philip T. Cox, Simon Gauvin, "Controlled Dataflow Visual Programming Languages", Proceeding VINCI '11 Proceedings of the 2011 Visual Information Communication - International Symposium, Article No. 9 , 2011

[Dias et Oliveira, 2010] Pedro Dias, Sancho Oliveira, "Botbeans: a new educational visual programming tool with tangible results." , Proceeding OSDOC '10 Proceedings of the Workshop on Open Source and Design of Communication, Pages 43-44 , 2010

[Carlisle, Wilson, Humphries et Hadfield, 1995] Martin C. Carlisle, Terry A. Wilson, Jeffrey W. Humphries, Steven M. Hadfield, "RAPTOR: A Visual Programming Environment for Teaching Algorithmic Problem Solving" , Proceeding SIGCSE '05 Proceedings of the 36th SIGCSE technical symposium on Computer science education, Pages 176-180, 1995

[Manzo, Halper Matthew, Halper Michael, 2011] V. J. Manzo, Halper Matthew, Halper Michael, "Multimedia-Based Visual Programming Promoting Core Competencies in IT Education" Proceeding SIGITE '11 Proceedings of the 2011 conference on Information technology education , Pages 203-208 , 2011

[Konstantopoulos, Lydakis, Gkikakis, 2014] Stasinos Konstantopoulos, Andreas Lydakis , Antonios-Emmanouil Gkikakis , "Embodied Visual Programming for Robot Control" , Proceeding HRI '14 Proceedings of the 2014 ACM/IEEE international conference on Human-robot interaction , Pages 216-217, 2014

[Agrawal, Jain, Humar, Yammiyavar, 2014] Harshit Agrawal,Rishika Jain,Prabhat Humar,Pradeep Yammiyavar, "FabCode: Visual Programming Environment for Digital Fabrication" , Proceeding IDC '14 Proceedings of the 2014 conference on Interaction design and children, Pages 353-356, 2014

[Hu, Winikoff et Cranefield, 2012] Minjie Hu , Michael Winikoff , Stephen Cranefield , "Teaching Novice Programming Using Goals and Plans in a Visual Notation" , Proceeding ACE '12 Proceedings of the Fourteenth Australasian Computing Education Conference - Volume 123 , Pages 43-52, 2012

[Bouraqadi et Stinckwich, 2007] Noury Bouraqadi and Serge Stinckwich, "Bridging the Gap Between Morphic Visual Programming and Smalltalk Code", Proceeding ICDL '07 Proceedings of the 2007 international conference on Dynamic languages: in conjunction with the 15th International Smalltalk Joint Conference 2007, Pages 101-120, 2007

[Ingalls, Wallace, Chow, Ludolph et Doyle, 1988] Dan Ingalls, Scott Wallace, Yu-Ying Chow, Frank Ludolph,Ken Doyle ,"Fabrik: a visual programming environment" , Proceeding OOPSLA '88 Conference proceedings on Object-oriented programming systems, languages and applications , Pages 176-190, 1988

[Rojas-Galeano et Rodriguez, 2013] Sergio Rojas-Galeano , Nestor Rodriguez , "Goldenberry: EDA Visual Programming in Orange" , Proceeding GECCO '13 Companion Proceedings of the 15th annual conference companion on Genetic and evolutionary computation , Pages 1325-1332, 2013

[Krebs, Conrad et Wang, 2012] Dave Krebs, Alexander Conrad , Jingtao Wang , "Combining visual block programming and graph manipulation for clinical alert rule building" , Proceeding CHI EA '12 CHI '12 Extended Abstracts on Human Factors in Computing Systems, Pages 2453-2458, 2012

[Lucanin et Fabek, 2011] Drazen Lucanin , Ivan Fabek , "A Visual Programming Language for Drawing and Executing Flowcharts" , Published in MIPRO, 2011 Proceedings of the 34th International Convention , Pages 1679 - 1684, 2011

[Gómez-de-Gabriel, Mandow, Fernández-Lozano et García-Cerezo, 2010] Jesús M. Gómez-de-Gabriel, Anthony Mandow, Jesús Fernández-Lozano et Alfonso J. García-Cerezo , "Using LEGO NXT Mobile Robots With LabVIEW for Undergraduate Courses on Mechatronics", Published in Education, IEEE Transactions on (Volume:54 , Issue: 1) , Pages 41 - 47, 2010

[Krivanec, Petrackova, Thi Phuong Linh et Prusa, 2010] Stepan Krivanec, Alena Petrackova, Tran Thi Phuong Linh, Daniel Prusa, "Nao Robot Applications Developed In Choregraphe Environment", 2010

[MALONEY, RESNICK, RUSK, SILVERMAN et EASTMOND, 2010] JOHN MALONEY, MITCHEL RESNICK, NATALIE RUSK, BRIAN SILVERMAN and EVELYN EASTMOND published in Journal ACM Transactions on Computing Education (TOCE) TOCE Homepage archive , Volume 10 Issue 4, November 2010 , Article No. 16

[Mosconi, Porta, 2001] M. Mosconi, M. Porta, "Iteration constructs in dataflow visual programming languages", 2001