Année 2014-2015

Rapport de projet-EI4 AGI

PLCduino : Atelier Grafcet pour Arduino.

Projet réalisé par :

Enguerrand Belin.
David Debossu.
Nidal Naouis.

Projet encadré par :

Bertrand Cottenceau.

ISTIA-Université d'Angers

Remerciements

Nous souhaitons tout d'abord à remercier notre encadrant Bertrand Cottenceau, qui a encadré notre projet avec patience, s'est toujours montré à l'écoute toute au longue de la réalisation de ce projet et ses conseils nous ont été bien utiles.

Nos remerciements s'adressent également à l'ensemble du corps enseignant de la Filière Génie Automatique et Informatique pour nous avoir donné les moyens nécessaires à la réalisation de ce projet.

Enfin, nous adressons nos plus sincères remerciements à tous nos proches et amis, et à toute personne ayant contribué, de près ou de loin à la réalisation de ce projet.

Tables des matières

Introduction

Dans le cadre de la préparation de notre diplôme d'ingénieur à l'ISTIA option automatique et génie informatique, nous devrons réaliser un projet en quatrième année sous le thème :
«**PLCduino : atelier Grafcet pour Arduino**». Ce projet met en pratique les acquis reçus durant la formation.

Dans le cadre de ce projet nous devrons reprendre un projet développé par des anciens étudiants dont l'effort a été porté sur la Création d'un langage orienté automatisme industriel pour Arduino , afin de le compléter en développant une interface en C# pour éditer les grafcet et fournir une traduction adéquate avec le projet précèdent afin d'avoir le code C arduino et finalement voir le résultat sur la carte arduino.

Dans les systèmes d'automatisation industrielle, le grafcet permet de décrire graphiquement le comportement souhaité de la partie commande du système automatisé. C'est un outil puissant qui garantit la représentation fonctionnelle des automatismes industriels.

Nous avons conçu une application qui répond au mieux au cahier des charges proposé. Ce rapport après une présentation du projet, explicitera différentes étapes que nous avons effectuées afin de mettre cette application en place.

I- Présentation du projet

1- *Description du projet*

L'objectif de notre projet est d'implémenter une application permettant de concevoir une interface qui permettra à l'utilisateur de dessiner un grafcet en lui fournissant tous les blocs nécessaires (Etape, étape initial, actions, transitions, réceptivités…etc.) pour la construction d'un grafcet tout en respectant les normes du grafcet, ainsi que la traduction de ce graphe en langage C arduino.

2- *Cahier des charges*

Notre projet consiste trois phases de travail :

- L'implémentation de l'interface.
- La traduction en langage APJCGT.
- L'intégration du projet précédent.

L'interface développée doit répondre aux besoins suivants :

- **Boite à outils** ayant les contenus suivant :
  - un bouton étape initialisation.
  - un bouton étape.
  - un bouton transition.
  - un bouton divergent "ou".
  - un bouton divergent en "et".
  - un bouton de retour d'étape.
  - un bouton de convergence.

- **drag and drop** : permettre le déplacement de la boite à outil vers le dessin.
- **charger/sauvegarder un fichier** : en utilisant le concept de la sérialisation des objets, l'utilisateur pourra sauvegarder/charger un grafcet déjà dessiné.
- prendre en charge les mnémonique (attribué les entrés/sorties).
- **Traduction**: Cette étape permettra de reprendre le grafcet crée en forme textuelle.

  - Transcrire une étape en code.
  - Transcrire une transition en code.
  - Transcrire les timers en code.
  - Transcrire les actions en code.

3- Organisation du projet

Afin de comprendre le sujet de voir les possibilités nous avons commencé notre première séance par un brainstorming. Cette séance nous a permis de déterminer notre cahier des charges ainsi que de notre interface graphique.

Le projet de fin d'études a débuté le 10 Décembre 2014 et a duré 4 mois. La figure suivante présente le diagramme de GANTT représentant le planning prévisionnel pour mener à bien le projet.

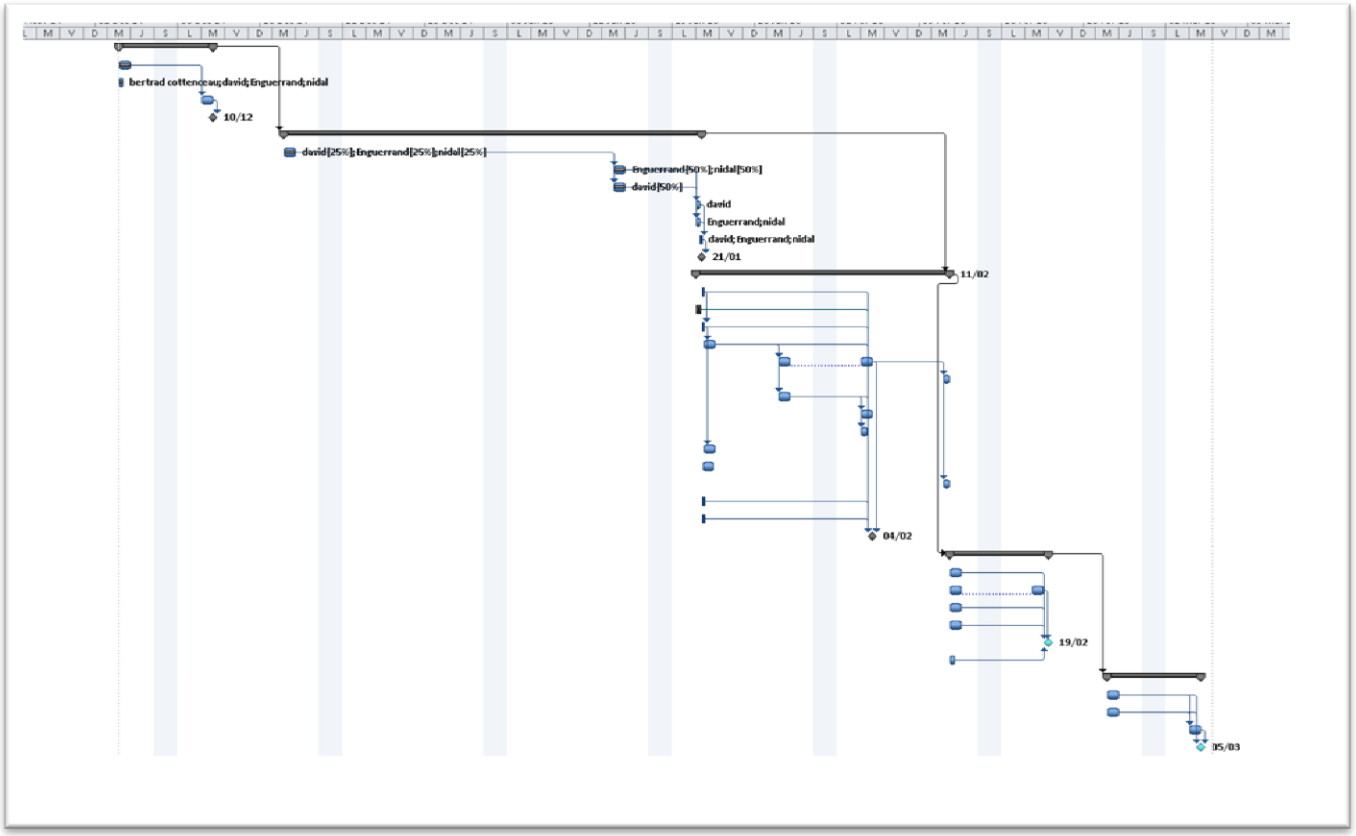


Figure 1 Diagramme de Gantt du projet

Comme on peut le remarquer personne n'avait de tache précise dans le projet. De plus quelques fonctionnalités étaient d'importance moindre compte tenu du planning projet. D'autres étaient plus utiles pour le débug comme par exemple exemple les fonctions de vérification ou encore charger/sauvegarder dans un fichier. Lorsque l'un de nous avait fini une tache (fonctionnalité), il en commence une autre tâche au quelle est disponible (en priorisant sur les fonctionnalités principales).

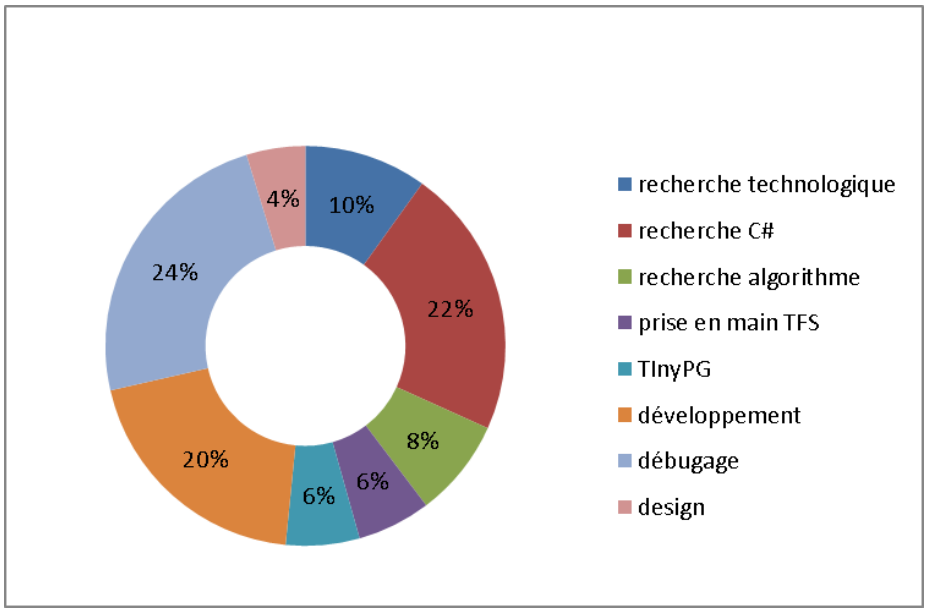


Figure 2 Répartition du travail

On peut constater que nous avons passé beaucoup de temps sur de la recherche puis sur le débugage. De plus notre projet est la création d'une interface graphique, cependant on peut le remarquer qu'on a passé peu de temps à créer le design comparé au développement.

4- Team Foundation server (TFS)

Dans la perspective d'un développement collaboratif, nous avons découvert TFS, qui nous a permis de mettre notre code sur un serveur qui fait office de git amélioré.

Le coté pratique de TFS est que l'on peut accéder au code depuis n'importe quelle machine et le fusionner si jamais plusieurs personnes ont modifié le projet.

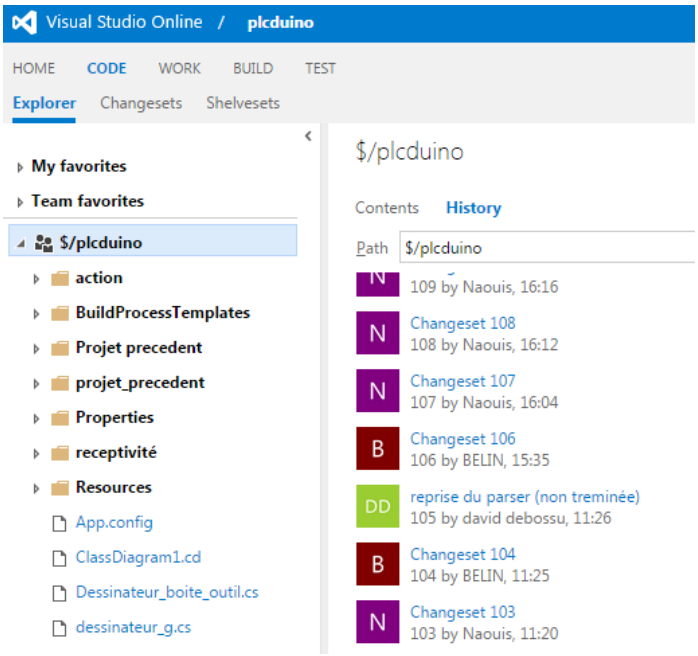


Figure 3 Aperçu de l'interface TFS web

Le coté un peu moins pratique est le prise en main de TFS qui a été un peu longue (environ 7h), puisqu'un compte Microsoft online était nécessaire et il a fallu également affecter les participants au projet pour leur donner le droit de modification et de lecture. Une fois cela fait, il ne restait plus qu'à se connecter depuis Visual Studio pour bénéficier d'un partage dédié sur le serveur.

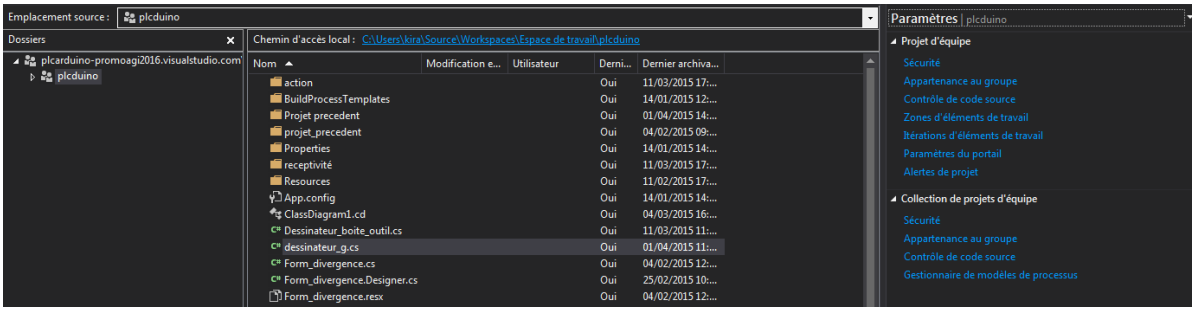


Figure 4 Aperçu de l'interface

Nous avons utilisé qu'une infime partie de TFS, qui permet bien d'autres choses comme de la planification, des gestions de taches, des contrôles de code source et des tests directement sur le serveur.

Durant le projet, nous avons eu un seul problème de fusion qui à faire perdre ½ journée de travail à un de nous trois. Depuis ce problème nous faisions plus attention lors d'un « commit ». Or mit ce problème, les auto-merges (fusion automatique par le logiciel) se sont toujours bien passé même quand on travailler sur un même fichier des même parties de codes, ou encore une restructuration du code (changer une variable en une classe avec des propriétés et des méthodes) en même temps que le développement d'une fonction (qui utilise cette ancienne variable).

II- Développement

1- *Deux solutions pour le développement de l'interface*

Lors de nos recherches sur le projet, nous avons remarqué que l'on peut le faire de deux manières différentes:

- Une première version dans laquelle chaque objet serait une agglomération de conteneurs (templates) ayant leurs propres attributs et ses propres contrôles.
- Une deuxième version où l'on crée les objets directement dans l'interface, en les 'dessinant' dans un panel de réception dédié.

A- Méthode du conteneur

➢ *Les avantages :*

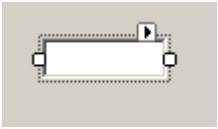


Image d'une Template d'étape

Chaque objet serait dans un conteneur, il serait plus simple de modifier le design, la disposition, aussi de rajouter des objets spécifiques, et déplacer ce conteneur. De ce fait il y aurait eu des 'modèles' de conteneurs pour définir chaque composant du grafcet.

➢ *Les inconvénients :*

L'un des inconvénients est la performance ainsi que la complexité à mettre en place, car chaque Template ouvre une fenêtre gérée par Windows. De plus, nous n'avions pas assez de connaissance en début de projet pour gérer ce type de contrôle personnalisé, assez complexe à créer.

B- Méthode du dessin

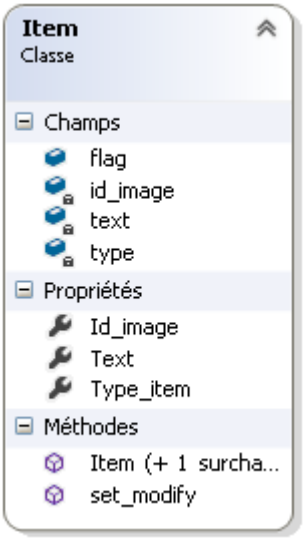
➢ *Les avantages :*

Le logiciel gère moins de fenêtre, moins d'objets ce qui en théorie permet de prendre moins de ressources. De plus, nous connaissons mieux la technologie que les conteneurs, ayant fait un Tetris avec cette méthode.

➢ *Les inconvénients :*

Nous devons associer tous les objets visuels à leur objet ce qui peut devenir compliqué.

2- *Notre solution*



Suite à plusieurs débats, un prototype de test dans chaque choix et un tableau décisionnel, nous avons pris le choix du "dessin". Une fois la solution choisie nous avons rapidement eu l'idée d'avoir une seule classe pour tous les items[1]:

Nous avons eu l'idée aussi de stocker ces items dans une liste de liste permettant de simplifier l'affichage du grafcet et de ne pas gérer la taille du grafcet.

Le fait d'utiliser une liste de liste d'items a facilité le développement du programme du fait que toutes les données étaient dans une seule et même variable, dans le cas présent un objet Grafcet.

Cet objet contient donc le grafcet en lui-même ainsi que ses données de configuration. Une telle disposition permet de pouvoir sauvegarder le tout dans un fichier en sérialisant les données, puis de récupérer les données depuis un fichier externe.

L'interface développée (voir ci-après) permet de créer manuellement un grafcet en drag&drop[2], de le modifier à souhaits, de configurer l'interface arduino et de remplir des textbox avec les algorithmes souhaités, tout en ayant une vérification / alerte en temps réel.

[1] Un item défini une partie du grafcet, composé d'une image et éventuellement d'une zone de texte et d'attributs (type [action, réceptivité, …], texte…)
[2] Phénomène de glisser - déposer

3- L'interface

Nous avons été rapidement d'accord sur cette interface, dans laquelle nous avons une boite à outils, une grille, deux débogueurs (dont un activé dans certains cas), deux boutons pour traduire le grafcet et deux menus.

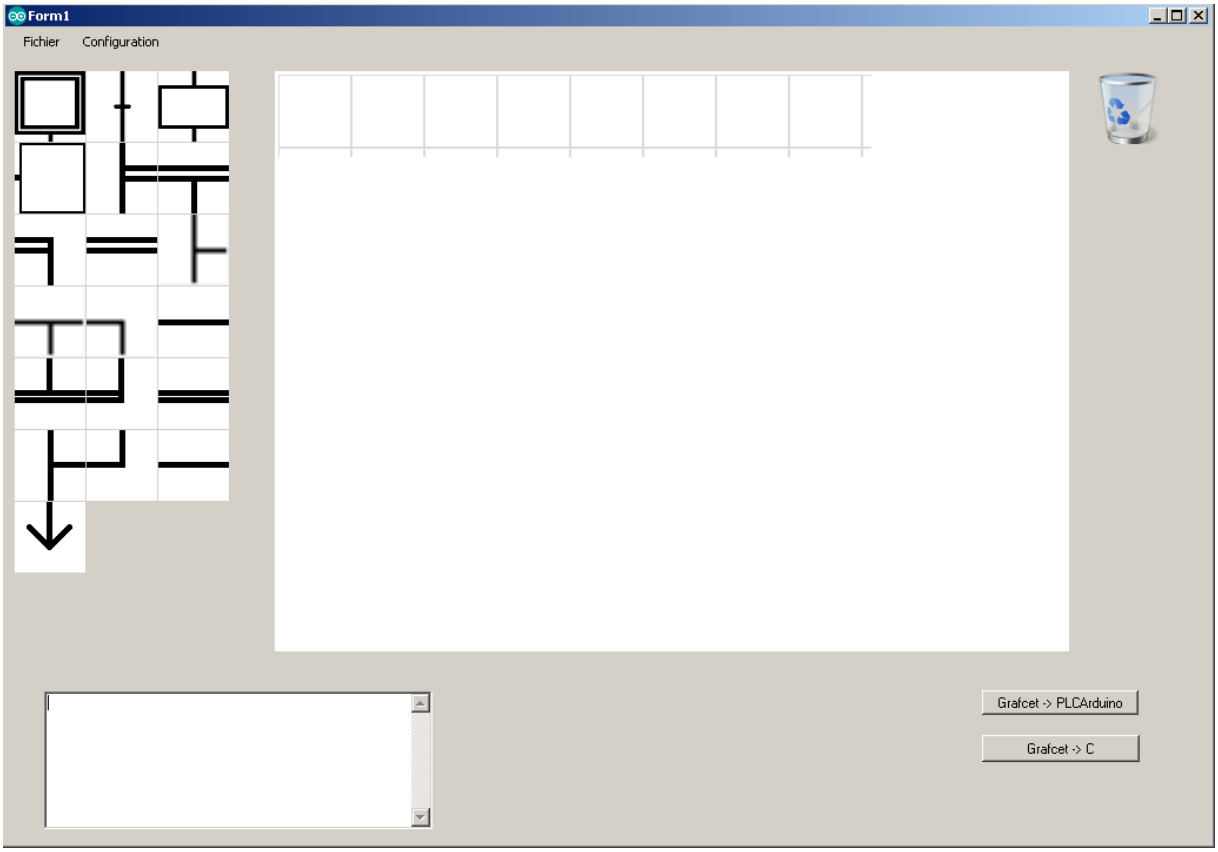


Figure 5 l'interface développée en C#

La boite à outils contient tous les éléments qui peuvent apparaitre dans un grafcet.

La grille permet d'afficher le grafcet dessiné.

Dans le menu « Fichier » nous avons la possibilité de charger/sauvegarder un grafcet et dans « Configuration » de configurer le grafcet en cours.

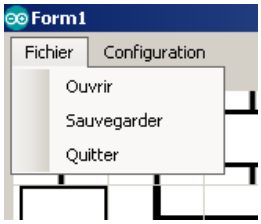


Figure 6 Menu de l'interface

Ci-dessous un aperçu de la partie débug du logiciel, comportant deux zones de textes. Chacune permet un débeug personnalisé ; il est montré à gauche les incohérences et erreurs par rapport aux normes Grafcet et à droite les erreurs quant aux algorithmes des réceptivités et actions. Cette dernière apparait seulement lors du focus de la textbox correspondante.

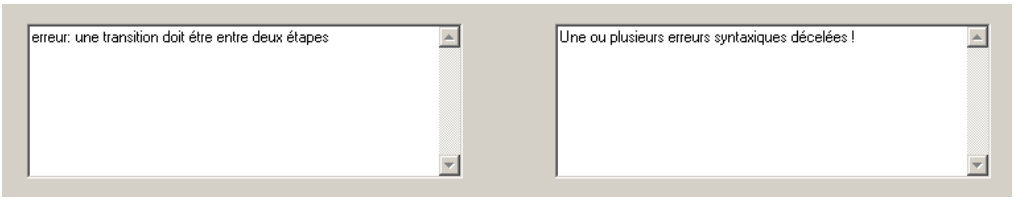


Figure 7 Débogueur de l'interface

A- L 'Ajout des éléments au grafcet

Pour mettre des éléments dans le grafcet, nous avons réalisé deux méthodes, la première utilise le concept du Drag&Drop vers la zone de dessin, et la deuxième qui se résume dans un simple clic sur l'élément que l'on souhaite ajouter au grafcet.

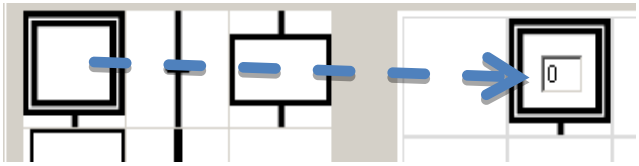


Figure 8 Ajout des éléments du grafcet

B- La suppression d'un élément

Pour supprimer un élément, une corbeille a été rajoutée. Il suffit de faire un Drag&Drop de l'élément que l'on désire supprimer vers la corbeille pour le supprimer.

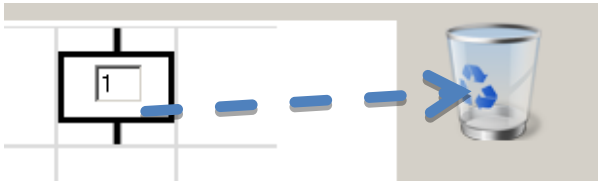


Figure 9 Suppression d'un élément

C- Les transitions

Suite à plusieurs testes nous avons remarqué que les transitions complexes étaient illisibles. Donc nous avons rajouté un message de dialogue en cas d'un double clic sur une transition. Cette boite de dialogue permet de modifier la transition et permet de la voir en entier.

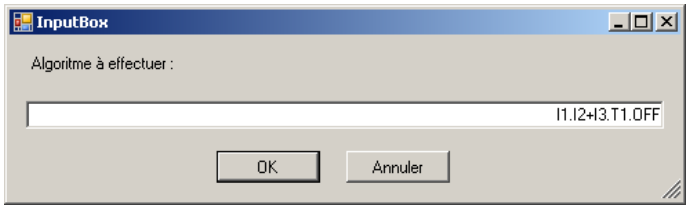


Figure 10 Message de dialogue pour les transitions

D- Les divergences

Lors d'un ajout de divergence, le logiciel demande le nombre de branches que l'on veut créer :

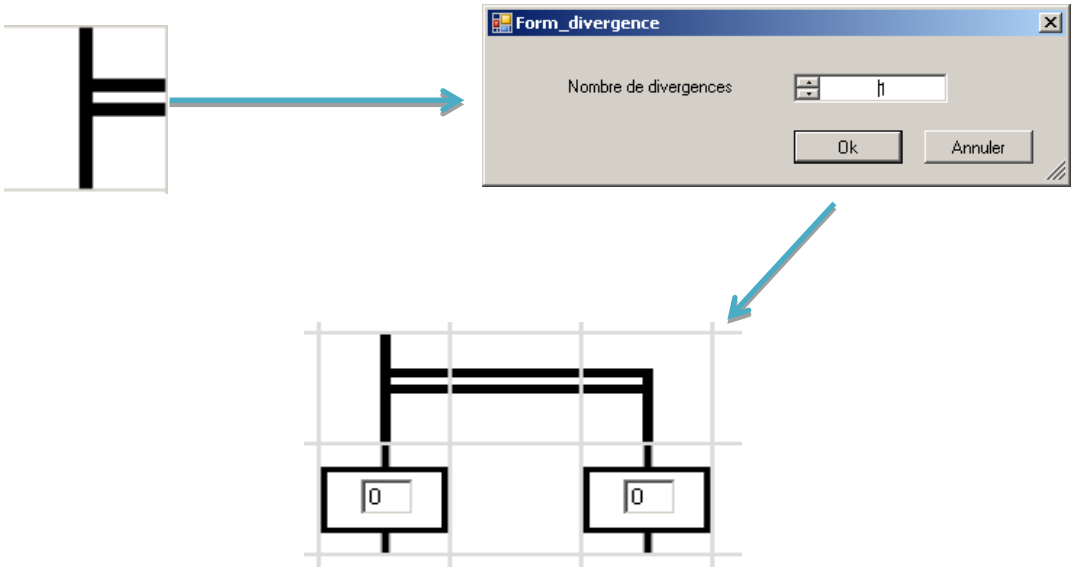


Figure 11 Création des divergences.

Lors de la validation, le logiciel se charge de créer la divergence en lieu et place du drop.

E- La configuration des Entrées/Sorties

Dans la fenêtre de configuration, on peut configurer chacune des broches de la carte Arduino en entrée ou en sortie. On peut aussi configurer les timers. Une fois la configuration faite, le logiciel nous indique la syntaxe adéquate pour l'algorithme.

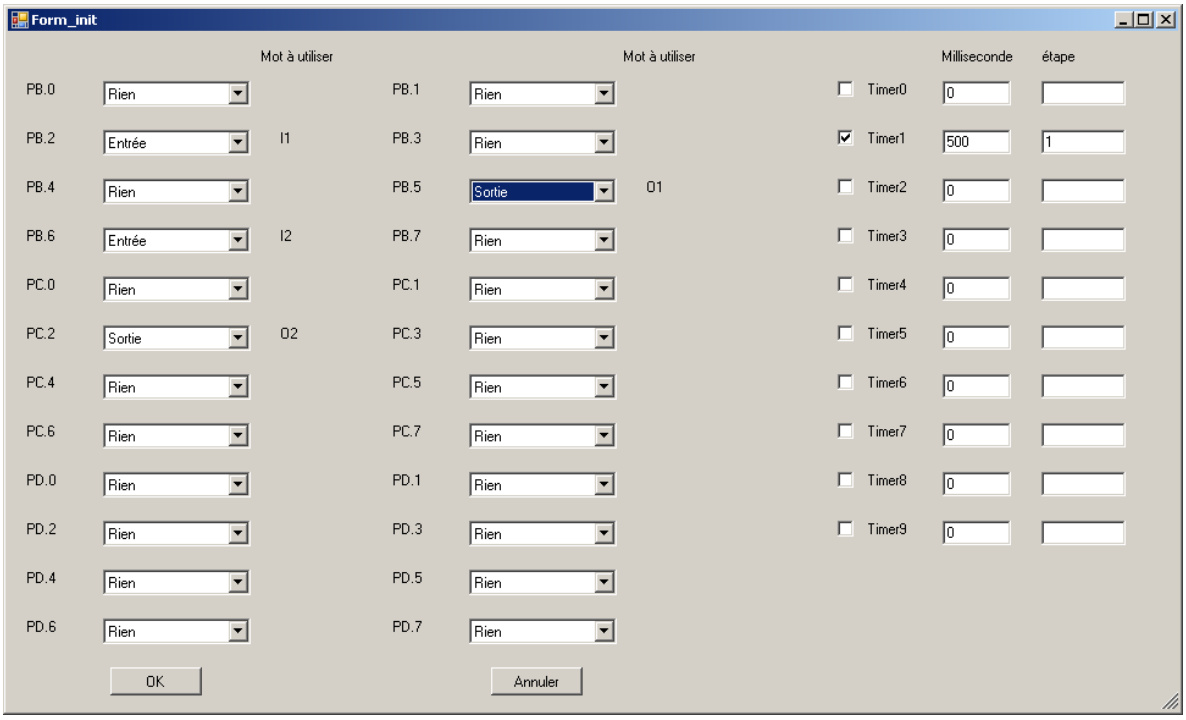


Figure 12 Configuration des Entrées/Sorties

4- *La traduction*

Une fois l'interface développée, nous avons fait la traduction de notre grafcet en langage APJCGT, le langage créé par le groupe précédent (APJCGT sont les initiales des créateurs).

Quatre étapes nécessaires doivent être réalisées pour avoir une traduction complète : la configuration, les réceptivités, les transitions et les actions.

A- La configuration

Le logiciel récupère les étapes du grafcet ainsi que la configuration et le formate pour qu'il soit compatible avec le langage APJCGT.

Cette étape est possible par une sauvegarde de la configuration dans le grafcet. Ainsi lors d'un chargement d'un fichier de sauvegarde la configuration est restaurée.

B- Les réceptivités

A chaque fois que l'utilisateur modifie une réceptivité celle-ci est vérifiée régulièrement par un analyseur syntaxique. Lors de la traduction on reformate ces réceptivités (on transforme le '.' en '&') puis on les ajoute directement dans le fichier.

C- Les actions

En analysant le langage APJCGT, nous avons remarqué qu'il était plus simple de ne pas mettre de "set" et de "Reset", mais simplement d'utiliser le "=". Cela permet d'avoir un algorithme simple, qui génère un fichier légèrement plus volumineux.

D- Les objets 'étape' et 'transition'

Le programme analyse chaque transition et cherche les étapes avant et après par des algorithmes récursifs. C

5- *Intégration du projet précédent*

A la fin de notre projet nous avons intégré le projet précédent. Nous avions quelques petites erreurs mais une fois corrigées, nous arrivions à compiler avec leur langage.

Avec une carte Arduino, deux boutons poussoirs et deux LED, nous avons fait un test simple : nous avons fait le grafcet ci-contre et après compilation et intégration du code source dans la carte Arduino, les LED ont réagi comme nous le souhaitions.

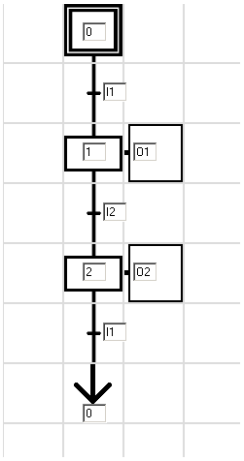


Figure 13 Grafcet de test

Nous avons fait un deuxième test avec des timers. Le logiciel compile mais cependant les LED ne réagissent pas comme on le voulait. Nous avons vérifié que le code qu'on a créé était en accord avec leurs règles, ce qui était le cas. Nous avons analysé le code C qu'il générait, nous voyons que nos timers ont été programmé, et ils sont dans les conditions de franchissement des étapes.

Cette erreur est pour le moment encore inconnue et inexplicable, car le code généré paraissait correct. Néanmoins cette partie a besoin d'être plus développée pour être entièrement fonctionnelle.

6- Vérification

A- TinyPG

Afin de pouvoir vérifier la qualité des algorithmes entrés dans les zones de texte du grafcet, nous avons réutilisé la technologie de tinyPG, déjà utilisée par le groupe précédent.

TinyPG est un mini-programme qui permet de créer un parseur à partir d'expressions régulières et de règles de grammaire personnalisées.

Le groupe précédent a utilisé TinyPG pour créer le code Arduino à partir d'un langage créé par leurs soins.

Nous l'avons utilisé pour vérifier en temps réel l'algorithme de l'utilisateur.

Pour l'instant le parseur ne prend en compte que les erreurs flagrantes d'algorithmique, mais ne propose pas de contenu d'erreur détaillé ni de coloration syntaxique.

Nous avons développé deux règles syntaxiques pour vérifier les réceptivités et les actions.

B- Vérification des réceptivités:

Voici la structure d'une réceptivité :

```
Start               -> (OrExpr)+ ;

OrExpr  -> AndExpr (PLUS AndExpr)*;

AndExpr             -> Atom (POINT Atom)*;

Atom                ->(FrontMontant |FrontDescendant)? (_rtimer | PARENTOUV OrExpr PARENTFER |
input | etape_on | CaracRECEPT)+;
```

C- Vérification des actions:

```
Start               -> (Expr)+ ;

Expr                -> (_timer | output)(espace  (_timer | output))*;

//receptivite timer

_rtimer -> NumTimer POINT OFF;
```

Les deux règles utilisent les expressions régulières suivantes (liste non exhaustive):

```
input        -> @"(I|i)([0-9]){1,2}";

output        -> @"(O|o)[0-9]+";

CaracRECEPT     -> @"(\!|\+|\-|\<|\>|\=)+";
```

Ces règles sont traduites en code C# pour pouvoir les intégrer dans le projet. L'intégration se fait alors par implémentation du parser, scanner et du parseTree.

```
Receptivite.Scanner scan_T = new Receptivite.Scanner();
Receptivite.Parser parse_T = new Receptivite.Parser(scan_T);

Action.Scanner scan_A = new Action.Scanner();
Action.Parser parse_A = new Action.Parser(scan_A);
// ajout du parser en fonction du type d'item
```

```
if (ana_syntaxe.Type_item == Type.action)
{
  Action.ParseTree result = parse_A.Parse(ana_syntaxe.Text);
```

Les parseurs sont déclenchés en fonction du type de textbox (réceptivité ou bien action) par un écouteur d'évènement lorsque celui-ci a le focus (Handler 'gotFocus')

La vérification de l'algorithme utilisateur se fait régulièrement pendant l'utilisation de l'interface grâce à un timer et une zone de texte indique s'il y a erreur ou non.

D- Cohérence du grafcet

Afin d'aider l'utilisateur à faire son grafcet nous avons fait une vérification du grafcet toutes les 2 secondes et avant chaque traduction.

Ceci permet de vérifier que :

- Une étape doit être placée entre deux transitions
- Une transition doit être placée entre deux étapes (initiale ou non)
- Deux étapes ne doivent pas avoir le même numéro.
- Une étape est identifiée par un numéro.

La vérification « une transition doit être placée entre deux étapes» réalise la structure du grafcet en cas de succès. Les deux fonctions ont quasiment le même algorithme donc nous avons décidé de les rassembler en une.

Conclusion

Ce projet nous a permis de mettre en pratique ce que nous avons acquis tout au long de notre formation et aussi de faire face à des problématiques. En effet nous sommes partis d'un réel besoin, nous avons pu découvrir toutes les grandes étapes d'un projet, depuis la rédaction du cahier des charges jusqu'à la mise en place de l'interface graphique.

Or ce travail était une occasion pour approfondir nos connaissances en langage C#, en plus de cela étudier les différentes normes du grafcet pour mettre en place une interface qui permettra à l'utilisateur la création des graphes. Afin de mener à bien ce projet, plusieurs phases ont été réalisées pour atteindre les objectifs préalablement fixés. Avant de se lancer dans la phase du développement de l'interface, il fallait comprendre le projet existant, le langage APJCGT qui décrit un grafcet d'une manière textuelle. Cette phase était importante pour garantir l'intégration du projet précèdent et la traduction du grafcet dessiné afin de construire le code C qui sera exécuté sur la carte Arduino.

Nous avons essayé de réunir le maximum des fonctionnalités pour répondre au cahier de charge. Pourtant, ceci n'exclut pas l'existence d'autres fonctionnalités que nous n'avons pas pu les mettre à cause de la contrainte du temps, nous aurions pu écarter notre projet en ajoutant un raccourci clavier qui va permettre un rajout plus rapide des objets, l'interface pourra aussi avoir un menu contextuelle sur un ajout (liste des items possibles lors d'un clic droit).

PLCduino : Atelier Grafcet pour Arduino

Projet réalisé par : Belin Enguerrand, Debossu David, Naouis Nidal.
Projet encadré par : Cottenceau Bertrand**.**

Résumé

Le projet EI4 nous a permis de mettre en pratique ce que nous avons acquis tout au long de notre formation.

En effet nous sommes partis d'un réel besoin ainsi, nous avons pu découvrir toutes les grandes étapes d'un projet, depuis la rédaction du cahier des charges jusqu'à la mise en place de l'interface graphique.

L'effort a été porté sur la réalisation d'une interface graphique développée en C#, qui va permettre à l'utilisateur de créer un grafcet, le traduire en un langage personnalisé afin d'obtenir le code C qui va être exécuté sur la carte Arduino. Ce travail ne pourra pas être complet sans la compréhension du projet précèdent et une bonne maitrise des normes grafcet.

Ce projet nous a permis de mélanger le développement objet (c#) à l'automatisme (grafcet) et à l'open source (arduino et tinypg) pour aboutir à un logiciel permettant de gérer complètement un grafcet et l'envoyer compilé vers une carte Arduino.

Mots-clés : programmation C#, grafcet, carte arduino, interface IHM, analyse syntaxique.

Abstact

The EI4 project allowed us to practice what we have learned throughout our training.

In Fact we started by a real need, and discovered all the important stages of a project, from writing the specifications to the implementation of the graphical interface.

The effort has been made on the construction of a graphical interface developed in C #, which will allow the user to create a grafcet, translate it to a personalized language in order to get the C code that will be executed on the Arduino board. This project will not be complete without understanding the previous project and having a good knowledge of the grafcet norms.

This project allowed us to work on both the Development (C #) and the automatism (grafcet), also the open source (Arduino and tinypg), in order to achieve software that will manage a grafcet, compiled and run it on the Arduino board.

Keywords: programming C#, grafcet, Arduino board, interface HMI, parsing.