Rapport de TER
Automatisation des protocoles de communication

UFR Sciences et Techniques de Nantes

Master 1 ALMA - 2017/2018

auteurs : Fortin Guillaume et Brohan Romain
encadrant : André Pascal

I - Introduction

Dans le cadre de notre master, on nous a demandé de travailler sur un projet de recherche encadré (TER) par un enseignant/chercheur sur la période du 15 janvier 2018 au 23 mai 2018 (date de rendu du rapport). Il s'est réalisé en parallèle de nos autres enseignements, cependant, une journée entière par semaine (le vendredi) a été libérée pour être consacrée à ce projet.

Dans un premier temps, nous avions remarqué un sujet qui nous semblait intéressant : "Automatiser la production logicielle de contrôleurs d'automates", malheureusement ce sujet avait déjà été pris par un autre groupe d'étudiants. Nous avons donc contacté Mr Pascal André, encadrant de ce sujet, pour lui demander s'il était possible de nous fournir un sujet similaire. Sa réponse a été positive et il nous a fait parvenir un nouveau sujet : "Automatiser la production logicielle de protocoles de communications". C'est donc ce sujet que nous avons sélectionné pour ce TER.

Nous avons alors établi un rendez vous avec Mr Pascal André afin de discuter des tenants et aboutissants de ce projet. Il en est sortit que l'objectif final de ce projet était de réussir à transformer automatiquement des modèles (type diagramme UML) en code source. Plus concrètement, pour nous il s'agit de retranscrire en code ce qui est de l'ordre de la communication dans le modèle en entrée. Par exemple, prenons un modèle qui représenterait une voiture et une tablette et sur lequel il y a une flèche partant de la tablette en direction de la voiture avec la mention "avancer". Dans ce cas, il est facile de comprendre le modèle, mais est-ce aussi simple de le retranscrire en code ? Comment la tablette communique avec la voiture ? Quels prétraitements faut-il faire au niveau de la connexion ou au niveau du code pour que la voiture comprenne ce qui est demandé ?
Voilà une partie des question auxquelles nous devons répondre pour réussir à automatiser la production logicielle.

Bien entendu, il n'est pas ici question de donner une réponse à l'objectif final, le sujet de recherche est assez récent et la question complexe. Il s'agit plutôt d'appréhender l'étendu du domaine et de sa complexité tout en apportant un début de réponse ou des pistes de recherche.

Pour mener à bien ce projet nous avons à notre disposition une brique EV3 lego mindstorm, il s'agit d'un appareil programmable sur lequel on peut brancher différents éléments comme des moteurs ou des capteurs :

Dans notre cas, une fois assemblées, les différentes pièces doivent former un véhicule qui sera contrôlé à distance à l'aide d'une application android/java. Ceci dans le but de réaliser manuellement le code nécessaire à la communication entre ces appareils, comprendre comment la communication fonctionne et de quoi elle a absolument besoin et comment sont transmis les informations entre les appareils. L'idée est qu'il nous faut réaliser manuellement une transformation de code une première fois avant de pouvoir l'automatiser.

En cours de projet nous avons été informé que nous allions devoir travailler en collaboration avec un groupe d'étudiants en MIAGE, de ce fait il a été décidé que notre groupe s'occuperait de la partie communication et code sur la machine alors que le groupe de MIAGE s'occuperait du code sur la machine et de l'interface d'une application pour communiquer avec la machine. La partie code est donc commune aux deux groupes, tout comme la construction du véhicule qui doit être adaptée pour les besoins de tous.

II - Le véhicule

Pour commencer, nous devions donc appréhender l'EV3, construire le modèle et réaliser le programme pour faire bouger le véhicule.

La construction

Ici il faut rappeler que l'EV3 est un objet LEGO, la construction n'a donc rien eu de compliqué.

Nous sommes parti sur un modèle de véhicule présenté dans le manuel d'assemblage (voir image ci-dessous), l'idée est de rassembler les deux moteurs avec les deux roues et d'avoir un support pour poser l'EV3 sur la structure et le brancher aux moteurs. Ces simples éléments suffisaient pour les besoins de notre projet. Néanmoins, un capteur qui permet d'éviter au véhicule de se cogner dans les murs a été ajouté par la suite.

modèle de construction du manuel

Par rapport à ce modèle nous avons retiré les éléments qui ne nous servaient pas.

Le code

Une fois la construction effectuée il a fallu nous attaquer à l'EV3. Tout d'abord, il faut noter que nous travaillons avec une brique EV3 tournant sous leJOS, c'est à dire que l'EV3 est boot sur une carte sd sur laquelle est installé leJOS.

Le choix de ce boot a été fait par notre encadrant, leJOS a l'avantage d'utiliser du code java au lieu du c++ et d'être bien plus simple d'utilisation que le programme de base, ce qui permet de simplifier le côté programmation pour mieux s'attacher à la communication.

Concernant la prise en main de la brique sous leJOS, elle a été assez simple, le menu est intuitif et suffisamment bien pensé sur la globalité (nous verrons dans la partie communication que certains points étaient quand même obscurs).
Pour faire tourner un programme sur l'EV3 il suffit que celui-ci soit stocké dans le dossier Programs de la carte sd (si celle-ci est sous leJOS) et de le lancer via le menu des programmes. D'ailleurs, un programme lancé via connexion usb, bluetooth ou wifi sera enregistré sur l'EV3 (pas besoin donc de sortir la carte sd).

Bien que le but du TER ne soit pas le codage en lui même, nous devions tout de même réaliser un code ayant pour but de faire fonctionner l'EV3 comme une voiture.
Pour ce faire, notre encadrant nous a fournis une étude de cas sur le sujet [1].

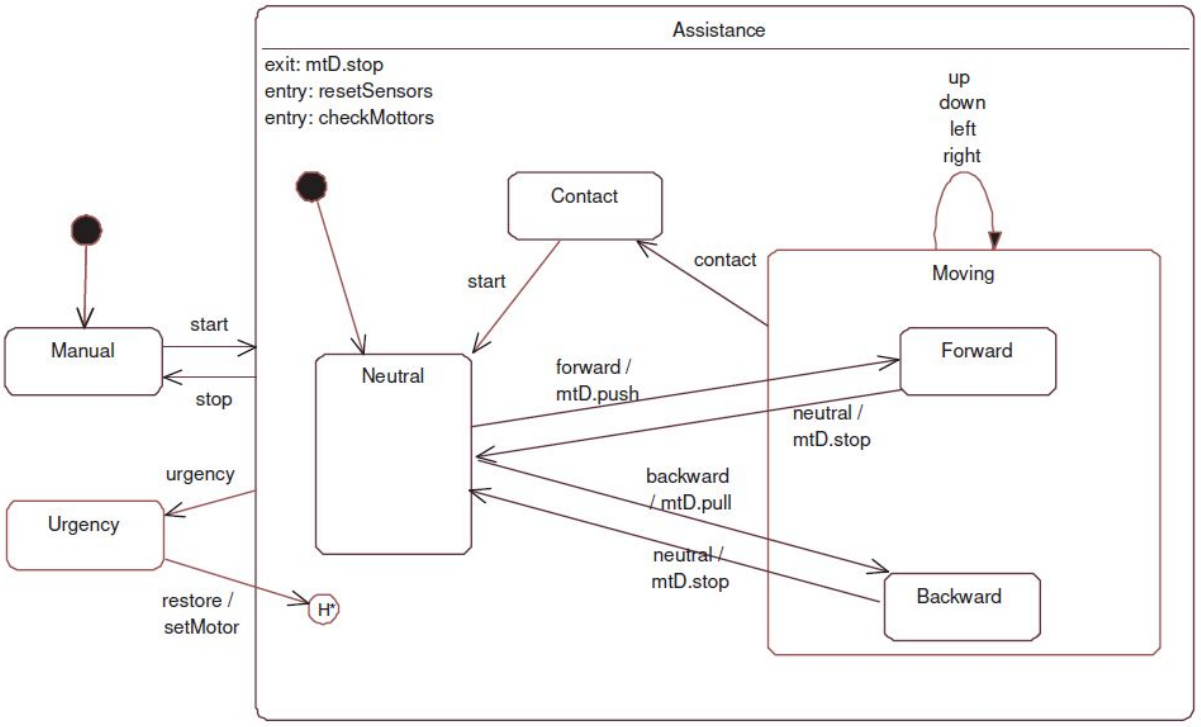
Dans un premier temps, nous avons simplement voulu faire fonctionner l'EV3 sans forcément coller aux diagrammes, nous avons donc implémenter les fonctions forward() et backward() qui permettent respectivement de faire avancer et reculer l'EV3.
Pour nous aider, nous avions à notre disposition le site de leJOS[2] qui regroupe les installations nécessaire et surtout la documentation[3], à partir de celle-ci nous avons trouvé les classes servant aux moteurs et aux boutons : lejos.hardware.Button et lejos.hardware.motor.
Il nous suffisait alors de déclarer deux moteurs et d'utiliser les fonctions:
setSpeed(int val) qui définit une vitesse pour le moteur
forward() qui fait avancer le moteur
backward() qui fait reculer le moteur
stop() qui arrête le moteur
La fonction Button.waitForAnyPress() permet d'attendre qu'on ait cliqué sur un bouton avant d'exécuter la suite du code.

La principale difficultée de ce code de départ a été de réaliser la synchronisation entre les moteurs, en effet, il fallait que les deux moteurs avancent, reculent ou s'arrêtent en même temps. Nous avons donc utilisé les fonctions synchronizeWith(T), startSynchronization() et endSynchronization() qui ont été trouvé dans la documentation leJOS.

Dans cette première ébauche nous avons aussi utilisé la fonction Button.waitForAnyEvent() qui permet d'attendre n'importe quelle action sur un bouton (l'appui ou le relâchement) avant d'exécuter la suite du code, nous l'avons utilisé pour faire avancer ou reculer le véhicule tant qu'on maintenait le bouton enfoncé, cette idée a été

oublié par la suite car nous nous sommes rendu compte que cela serait difficile à mettre en place avec la connexion à distance et que ça ne représentait pas tout à fait le modèle de l'étude de cas.

Dans un second temps, le groupe de MIAGE avait également réalisé un code assez similaire au nôtre à ceci près qu'ils n'utilisaient pas Button.waitForAnyEvent(), nous avons donc retenu leur code. Néanmoins, il nous fallait un code respectant l'étude de cas vu plus haut, dans cette étude on peut voir un diagramme d'état:



Au départ, nous avons donc décidé d'inclure au code créé le pattern State, en plus d'ajouter les fonctions turnRight() et turnLeft(). Il s'agissait d'avoir un état Stop et un état Moving : dans le premier il était seulement possible d'avancer et d'accélérer alors que dans le deuxième on pouvait accélérer, décélérer, tourner à gauche et à droite. En fait, les méthodes accélérer et avancer étaient confondues et dépendaient de l'état, idem pour reculer et décélérer. Néanmoins, nous nous sommes rendu compte que le pattern State n'était utile que pour des commandes restreintes aux boutons de la brique, car sur l'application qui allait servir pour commander l'EV3 il était demandé d'avoir distinctement un bouton pour avancer et un autre pour accélérer. Nous avons donc réaliser le code sans pattern State mais en respectant le diagramme d'état.

Nous avons donc une classe EV3Car définissant la construction de l'EV3, cet EV3Car est constitué de deux moteurs et d'une vitesse pour chacun d'entre eux.

La classe est définie par les méthodes suivantes:
EV3Car(EV3LargeRegulatedMotor leftMotor, EV3LargeRegulatedMotor rightMotor), le constructeur qui initie la vitesse des moteurs à 0.
up(int value) qui permet d'accélérer, la méthode modifie la vitesse des deux moteurs en même temps.
down(int value) idem que up mais permet de décélérer.

left(int ratio) permet de tourner à gauche, on met à jour la vitesse du moteur droit en multipliant la précédente vitesse par le ratio.
right(int ratio) idem que left mais modifie la vitesse du moteur gauche.
forward() permet de lancer la rotation des moteur vers l'avant de manière simultanée.
backward() permet de lancer la rotation des moteurs vers l'arrière de manière simultanée.
stop() permet de stopper simultanément les deux moteurs.

Cette classe est utilisée dans une classe principale mainClass. Dans cette classe on utilise simplement un switch/case pour déterminer quelle action effectuer suivant le signal reçu. Un booléen permet de vérifier si le véhicule est en état d'arrêt ou de marche.

C'est également cette classe qui sera modifiée suivant la méthode pour envoyer les instructions (directement sur l'EV3 via les boutons ou à distance).

III - La communication avec l'EV3

Connexion Bluetooth

Une fois le code fait nous devions chercher les différents moyens de communication possibles avec l'EV3. Nous avons donc commencé avec le bluetooth puisqu'au début nous n'avions pas de clé wifi et donc la possibilité de se connecter en wifi.

Avant toute chose, il faut s'apparier avec l'EV3 et bien que cela semble simple, ça ne l'a pas vraiment été. En effet, il existe de nombreux tutoriels sur internet pour se connecter en bluetooth à l'EV3 mais beaucoup ne sont pas pour leJOS et ne fonctionnent donc pas. Les quelques tutos trouvés ne présentent qu'une seule façon de se connecter en bluetooth et cette méthode est censé fonctionner au vu des captures d'images présentées sur les sites. Malheureusement ça n'est pas tout le temps le cas, nous n'avons pas compris pourquoi mais parmis les deux méthodes trouvées et fonctionnelles, une seule a fonctionné pour nous. Nous l'avons donc signalé dans la documentation consacrée à la connexion bluetooth[4].

Nous étions censé être le seul groupe sur la connexion, néanmoins le groupe des MIAGE avait inclut une fonction de connexion en bluetooth dans leur code. Nous avons donc commencé par étudier cette fonction: Depuis un programme lancé sur l'EV3 celui-ci se met en attente de recevoir un signal de connexion, il faut donc qu'une application java ou android se connecte à l'EV3 à l'aide de l'adresse mac pour débloquer l'attente et établir la communication.

De la même manière, nous avons trouvé dans la documentation une fonction où c'est l'EV3 qui tente de se connecter à un appareil en bluetooth mais nous ne l'avons pas utilisé.
Cette méthode fonctionnait très bien, mais deux problèmes se posaient à nous:

- Le premier étant que cette connexion n'était pas du tout pratique, on est obligé de lancer un programme sur l'EV3 pour pouvoir s'y connecter, on ne peut pas le faire entièrement à distance.
- Le second, nous avons travaillé avec eclipse et sous eclipse il existe un module leJOS qui lance une application java permettant de chercher un ev3, s'y connecter (par usb, bluetooth et wifi), naviguer dans les menus, lancer un programme, etc. Le

problème étant qu'à aucun moment Eclipse ne demande son autorisation à l'EV3 pour se connecter.

A partir de là, la question de la sécurité s'est posé, on s'est dit que la connexion établie à partir d'un programme de l'EV3 était sécurisée comparée à la connexion d'Eclipse. Malgré ça, nous avons quand même cherché de ce côté pensant que nous pourrions nous même établir une sécurité si elle n'était pas présente.

C'est à ce moment que nous avons commencé à utiliser wireshark, pour essayer de mieux comprendre les messages entre Eclipse et l'EV3 mais malheureusement cela ne fut pas fructueux.

Nous avons alors cherché sur internet et dans le code source du module leJOS d'eclipse installé sur nos machines. Si le premier n'a pas donné grand chose, le second était très intéressant mais assez difficile à comprendre, d'autant plus au regard du nombre de fichiers sources assez conséquent. Nous avons donc passé beaucoup de temps à chercher comment Eclipse se connectait à l'EV3 sans trouver.

C'est pourquoi nous avons décidé de passer à la connexion wifi en attendant vu qu'on avait reçu la clé wifi nécessaire à la connexion.

Connexion WiFi

Nous nous sommes donc intéressés au fonctionnement et paramétrage de la connexion WiFi. Un des avantages du wifi par rapport au bluetooth est qu'il nous permettrait de contrôler le véhicule à une distance bien plus grande.

Nous avons tout d'abord cherché à établir la connexion entre l'EV3 et l'ordinateur. Pour cela nous avons utilisé la clé USB WiFi que nous avons réçu pour permettre à la brique d'émettre un signal. Ceci fait, nous avons établit la connexion WiFi en nous basant sur les tutoriels trouvés sur internet, que nous avons ensuite résumés dans un manuel[5] et notre EV3 s'est ainsi retrouvé connecté à notre ordinateur. Il faut noter qu'aucune difficulté spécifique n'a été rencontrée dans la connexion wifi avec l'EV3.

Suite à cela, nous avons cherché à envoyer des ordres au robot en utilisant les méthodes fournis par leJOS et en nous basant sur le code utilisé en Bluetooth. Cependant, il s'avère que l'aspect WiFi de leJOS est très peu développé : il n'existe que deux ou trois méthodes spécifiques au Wifi, et aucune d'entre elles ne permet de gérer les connexions de l'EV3. Par conséquent, il était impossible de faire attendre une connexion wifi à l'EV3 comme cela était possible en bluetooth.

C'est à ce moment qu'on s'est dit qu'il nous serait alors obligatoire de comprendre comment le module leJOS du logiciel Eclipse dirige le robot à distance. C'était alors le seul moyen que l'on avait pour faire bouger l'EV3 en WiFi. A force de recherches, nous avons fini par trouver une classe qui avait l'air de répondre à nos besoins: RemoteEV3[6], qui prend en paramètre l'adresse IP d'une brique disponible en WiFi pour créer un objet EV3 (la méthode pouvant aussi être utilisée en bluetooth). Cet objet représente alors la brique à laquelle le programme est connecté, et toute action sur la brique peut être effectuée via celui-ci. Cette méthode permet donc de contrôler l'EV3 même sans qu'un programme soit lancé sur celui-ci. Nous avons réussi à trouver ce que nous cherchions.

Il nous restait alors à vérifier si notre hypothèse de départ sur la sécurité était la bonne, nous abordons ce point dans la partie *IV - La sécurité*.

Au cours de nos recherches sur cette connexion, nous avons utilisé wireshark afin d'analyser l'échange entre l'ordinateur et l'EV3 et vérifier quel protocole est utilisé par leJOS. Il s'agit du protocole TCP/IP comme on peut le voir ici.

Pour conclure cette partie communication, après avoir étudié la sécurité du système de communication (voir Partie IV), nous conseillons d'utiliser un programme ayant recours à RemoteEV3 comme contrôleur. Pour cela nous avons commencés à suivre l'article de leJOS News sur le sujet[7] , qui permet de contrôler l'EV3 sans avoir de programme lancé dessus. Malheureusement nous avons rencontrés plusieurs problèmes avec Eclipse et des erreurs non indiquées, nous avons réussi à réaliser l'application mais elle n'est pas entièrement conforme à notre cas.

IV - La sécurité

Après avoir réussi à se connecter à distance à l'EV3, nous avons dû aborder la question de la sécurité. Est-il possible, et si oui, comment, de sécuriser entièrement la communication entre l'EV3 et un appareil tiers de telle sorte à ce qu'il ne soit pas possible que deux appareils contrôlent le robot en même temps ?

Nous savions déjà qu'il était possible, via le module Eclipse, d'avoir plusieurs appareils connectés au même moment sur un même EV3. Néanmoins, il nous était alors impossible de le faire bouger en même temps.

C'est comme ça que nous avons constaté que l'EV3 acceptait toutes les communications qui lui étaient demandées depuis le module leJOS.

Néanmoins, nous nous demandions toujours si la méthode de connexion en bluetooth où l'EV3 attend la connexion n'était pas plus sécurisée vu qu'elle nécessitait qu'un programme soit lancé. Nous avons alors testé de nous connecter à l'EV3 grâce au module Eclipse pendant qu'un programme était lancé: C'était possible de se connecter mais impossible de prendre le contrôle de l'EV3, si ce n'est de pouvoir modifier la couleur des LED. Nous aurions pu nous dire que ça n'était pas grave de pouvoir se connecter simultanément tant qu'un seul programme était apte à contrôler l'EV3, mais le fait de pouvoir modifier les LED nous a fait chercher plus loin.

C'est ainsi que nous avons implémenter nous même la méthode RemoteEV3 pour pouvoir faire des test. Là nous nous sommes rendu compte d'une chose importante, quand l'un de nous avait déjà exécuté son code, l'autre recevait automatiquement une erreur lors de la tentative d'exécution. Nous avons trouvé cela étrange étant donné qu'on avait déjà réussi, comme vu plus haut, à nous connecter en simultané.

C'est alors que nous ai venu l'idée de ne déclarer qu'un seul port pour les moteur, chacun un différent. Il nous a alors été possible d'exécuter simultanément nos codes, mieux, il nous a été possible de contrôler en même temps les moteurs. Nous avons alors recommencer le test mais cette fois en lançant un programme sur l'EV3, résultat impossible d'exécuter notre programme.

Nous en avons donc conclu que lorsqu'un port de l'EV3 est déclaré via une connexion, celui-ci le réserve et il est alors impossible pour quiconque d'autre d'utiliser ce port. De même lorsqu'un programme est lancé sur l'EV3 il réserve automatiquement tous ses ports pour l'usage du programme. On pourrait alors penser qu'exécuter un programme

sur l'EV3 permet d'empêcher son contrôle par un autre appareil (puisque pour un programme il n'y a qu'un seul appareil), mais ça n'est pas le cas. Le module éclipse implémente deux méthodes: l'une permet de stopper un programme en cours d'exécution sur l'EV3 et l'autre permet d'éteindre l'EV3, et nous avons constaté que quelque soit l'état de l'EV3 celui-ci va exécuter la méthode.

Cela pose un problème puisque nous souhaitons une connexion unique entre les deux appareils, et que rien ne puisse interférer entre les deux. Or, ici il est possible d'interférer en stoppant le programme en cours et ainsi prendre le contrôle de l'EV3.

Nous avons donc cherché des moyens de limiter le nombre de connexion à l'EV3 à une seule client à la fois. Nous n'avons trouvé aucun paramètre s'apparentant au nombre de connexions dans le code leJOS, il ne semble donc pas possible de régler cela avec leJOS.

Comme la programmation de l'EV3 fait qu'il sert de serveur et le reste de client, nous avons tenté une autre approche : celle de créer un serveur http en utilisant du code java exploitable par l'EV3 via leJOS[8], qui gèrera les communications. Mais même cela n'empêche pas la commande RemoteEV3 d'être utilisée par un appareil et d'éteindre celui-ci de force.

Pour vérifier s'il n'était pas possible d'empêcher la méthode d'intervenir, nous avons observé les échanges TCP/IP entre les appareils avec wireshark. Pour cela, nous avons lancé un programme sur le robot et avons tenté de le contrôler avec le module leJOS d'Eclipse. Nous avons bien remarqué que, quelque soit l'état dans lequel se trouve l'EV3 (en train d'exécuter un programme, dans les menus, etc), il reçoit les messages et y répond[9].
Il semble donc impossible de complètement sécuriser un EV3 en Wifi sans modifier le code source.

Conclusion

Ce projet fut pour nous le premier projet de recherche sur lequel nous avons eu à travailler. Ne sachant pas comment cela allait se passer, nous étions un peu perdu au début mais notre encadrant Pascal ANDRÉ a su nous orienter sur les objectifs à atteindre et les étapes à suivre. La mise en relation avec une seconde équipe (l'équipe des MIAGE) a permis de répartir les tâches et s'est bien déroulée dans l'ensemble, bien que cela aurait été plus pratique si nous avions eu les mêmes plages de travail.

Actuellement, le travail que nous avons effectué correspond au pré-traitement du sujet. En effet nous n'avons même pas eu le temps d'aborder la modélisation de l'automatisation des protocoles de communication, il nous a tout d'abord fallu comprendre comment le système fonctionnait afin de pouvoir réussir, ou non, à l'automatiser. Nous nous sommes donc fixé comme objectif de finir la partie communication et sécurité. Ce que nous avons réussi pour la première, mais pour la sécurité nous n'avons pas eu le temps de tout explorer. Néanmoins, nous pensons la sécurité limitée puisque nous n'avons pas trouvé de moyen d'empêcher l'arrêt du robot ou d'un programme par un autre client. Le travail que nous laissons ici a été pensé pour être être repris par une autre équipe plus tard pour continuer sur ce sujet.

La suite du travail à effectuer pour la prochaine équipe serait alors de s'attaquer à la modélisation des protocoles de communication wifi et bluetooth. Le travail sur la sécurité

peut aussi être poursuivi, il est possible qu'il existe une méthode que nous n'avons pas exploré pour protéger la connexion.

Les problèmes qui nous ont ralentis dans le projet étaient principalement des problèmes techniques dus au manque de documentation de leJOS et aux erreurs de code.

Dans l'ensemble ce projet a été une bonne expérience pour nous concernant le travail d'équipe, et la découverte du domaine de la recherche applicative.

Annexe :

Annexe 1 :

Capture de la communication entre l'ordinateur et l'ev3 alors que l'ev3 exécute un programme.