# Initiation à LabVIEW

## Les bases de la programmation

**Francis Cottet**
est professeur des universités à l'École nationale supérieure de mécanique et d'aérotechnique (ISAE-ENSMA) de Poitiers.

DUNOD

Illustration de couverture : © Andrey_A – Fotolia





11, rue Paul Bert, 92240 Malakoff
www.dunod.com
ISBN 978-2-10-078646-6

# Table des matières

# Avant-propos

L'outil de programmation graphique LabVIEW pour « *Laboratory Virtual Instrument Engineering Workbench* » est né dans l'environnement du « test et mesure » destiné principalement à l'industrie et aux laboratoires scientifiques. Les outils ou instruments adaptés à ce domaine sont devenus de plus en plus complexes à régler et à piloter pour exploiter les données traitées. La société National Instruments, commercialisant ces instruments de « test et mesure », s'est rapidement retrouvée confrontée à la difficulté croissante des utilisateurs vis-à-vis de ces matériels de plus en plus performants.

Ainsi, le premier objectif de ce langage de programmation LabVIEW a été de répondre au besoin de réaliser la simple connexion d'un instrument de « test et mesure » à un ordinateur, c'est-à-dire d'écrire un logiciel d'automatisation des applications/expérimentations de mesure. Apparu sur le marché en 1986, le langage de programmation graphique LabVIEW a été immédiatement conçu comme un environnement de programmation à caractère universel, mais particulièrement bien adapté à la mesure, au test, à l'instrumentation et à l'automatisation. Cette origine de la conception de ce langage va se retrouver dans la structuration même des programmes comme nous le verrons. Un logiciel d'automatisation pourrait être défini comme un programme permettant de contrôler et commander un processus physique externe allant du simple capteur ou actionneur à la chaîne de fabrication.

La complexité et la spécificité de ce langage ont rapidement nécessité la rédaction d'un ouvrage en français. Un premier ouvrage sur la programmation et les applications de LabVIEW existe depuis 1993. Afin de suivre les évolutions de cet environnement de programmation, des éditions successives ont été faites et le dernier ouvrage publié (4e édition) en 2018, est adapté à la version 2017 de LabVIEW. Ce livre est très complet et s'adresse de façon préférentielle aux utilisateurs finaux des laboratoires ou des entreprises industrielles utilisant ce progiciel.

Ce nouvel ouvrage a comme objectif d'être un manuel très didactique, progressif et émaillé de beaucoup d'exercices. Il a vocation à s'adresser à tous les types d'utilisateurs sans préjuger de l'application finale visée avec un large domaine d'exemples traités. Il ne s'agira nullement d'un ouvrage exhaustif sur les possibilités de ce langage LabVIEW qui sont extrêmement vastes, mais plutôt d'une prise en main pas à pas de l'environnement complexe de programmation LabVIEW avec les différentes étapes suivantes : bases de la programmation graphique, environnement spécifique de programmation de LabVIEW, création de l'interface utilisateur (les entrées et les sorties), bases de la programmation (structures de programmation, traitements alphanumériques, traitements des tableaux…), édition, exécution et mise au point d'un programme LabVIEW…

La première partie de l'ouvrage comprend quatre chapitres qui présentent les bases de la programmation dans cet environnement LabVIEW. La deuxième partie décrit de façon détaillée l'ensemble des objets utilisés pour ces programmes. Chaque chapitre du livre est assorti d'exercices permettant de mettre en pratique les notions abordées en se basant sur des exemples simples. Une troisième partie est entièrement consacrée aux corrigés des exercices proposés à la fin de chacun des chapitres de l'ouvrage.

En utilisant des matériels très simples (cartes d'entrées/sorties internes et externes), un chapitre est consacré à l'aspect « acquisition et restitution de données », application naturelle avec ce langage présentant des exemples simples d'utilisation : mesure de température, mesure de tension…

Pour l'ensemble des notions de traitement du signal et d'acquisition de données utilisées dans cet ouvrage, des informations plus complètes et plus détaillées sont présentes dans le livre *Traitement des signaux et acquisition de données*, Francis Cottet, 4e éd., Dunod, 2015.

Je souhaiterais vivement remercier toutes les personnes du Laboratoire d'informatique et d'automatique pour les systèmes de l'ISAE-ENSMA qui m'ont apporté leur soutien de diverses façons : conseils, aides techniques, mise à disposition de matériels, documents d'enseignement, exemples de programmation LabVIEW, etc.

Francis Cottet

# Partie 1

# Les bases de la programmation avec LabVIEW

Après la présentation des principes généraux de la programmation graphique, cette partie va introduire l'environnement de programmation de ce langage graphique dans toutes ses particularités.

Le premier chapitre est consacré à la programmation flot de données utilisée dans l'environnement LabVIEW en montrant ses spécificités en termes de contrôle d'exécution et de propagation des données. Il met en exergue la limitation de cette programmation qui impose l'ajout de structures de contrôle et d'éléments permettant la mémorisation des données.

Le deuxième chapitre présente l'environnement de développement très particulier avec ses deux facettes : l'interface utilisateur et le programme flot de données lui-même. Cela permet de découvrir d'une part les deux palettes différentes des objets qui peuvent être utilisées dans le développement de l'interface utilisateur et du programme et d'autre part la palette des outils qui permettent les actions sur ce développement et sur les objets manipulés.

Le troisième chapitre est consacré à la réalisation d'un programme au sens propre de la construction de ce flot de données. Le câblage associé à cette réalisation du programme est présenté de façon détaillée afin d'offrir des bases très solides d'« écriture » d'un programme LabVIEW à un nouvel utilisateur.

Le quatrième chapitre qui termine cette prise en main de l'environnement de développement LabVIEW traite de l'exécution d'un programme réalisé, mais surtout de la mise au point de ce programme. La richesse de ces outils de mise au point répond ainsi à la complexité de cet environnement. Ce chapitre se termine par la création d'un sous-programme, c'est-à-dire l'encapsulation d'un programme qui est la base de la réalisation de programmes simples.

Chapitre 1

# Les grands paradigmes de la programmation graphique

## 1 La naissance des langages graphiques

Un langage informatique est une notation textuelle ou graphique qui permet d'exprimer un ensemble d'actions ou fonctions, appelé algorithme, traduisant l'application souhaitée par l'utilisateur. Ces fonctions agissent sur des données (variables ou constantes) spécifiques de l'application qui est désignée par « programme informatique ». Ainsi, les applications ou programmes sont d'une très grande variété, allant du programme de gestion (*i.e.* gestion financière) au programme de calcul scientifique (*i.e.* programme de simulation météorologique) en passant par des programmes de contrôle/commande (*i.e.* pilotage d'expérimentations). Dans cet ouvrage, nous nous intéresserons plus particulièrement à ces derniers étant donné l'orientation du langage de programmation LabVIEW.

Afin de mieux comprendre ce mode spécifique de programmation, il est intéressant de situer ce nouveau paradigme du langage graphique flot de données, sur lequel LabVIEW est basé, dans l'ensemble des modes de programmation (Fig. 1.1). Les langages de programmation sont découpés en deux familles : langages impératifs (le programme consiste à exécuter les instructions ou les fonctions les unes après les autres) et langages déclaratifs (le programme, composé d'un ensemble de fonctions ou règles, consiste à les évaluer en fonction des données disponibles).

La première classe est la plus nombreuse et se compose des trois sous-familles suivantes :

- des langages à programmation procédurale comme Ada, Pascal ou C (l'exécution du programme suit « séquentiellement » la suite des ordres ou instructions) ;
- des langages à programmation orientée objet comme C++ ou Java (découpage du programme en entités « quasi indépendantes » et interconnectées en exécution avec des échanges de données) ;
- des langages à programmation concurrente comme Ada (exécution de parties du programme en parallèle).

La première catégorie de cette classe, qui est la plus fournie, la plus ancienne (*i.e.* langages assembleurs) et la plus classique, a vu aussi des développements de quelques langages graphiques spécifiques comme LUSTRE et Esterel entre autres.

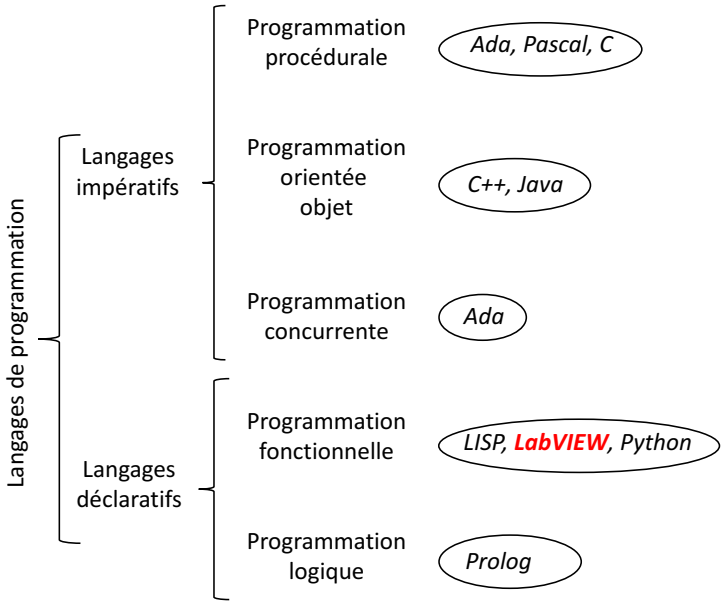


**Figure 1.1** – Classification des langages de programmation.

La deuxième grande famille de langages de programmation « langages déclaratifs » regroupe deux entités : les langages basés sur une programmation fonctionnelle, dont fait partie le langage graphique LabVIEW, et les langages basés sur une programmation logique (langages sources de l'intelligence artificielle).

Même si souvent des aspects graphiques ont été ajoutés à l'environnement de programmation comme une aide au développement afin de rendre un programme plus rapidement compréhensible avec par exemple pour un texte des moyens graphiques, tels que le surlignage, les italiques, les caractères gras, la couleur ou l'indentation d'un texte. Pourtant, ce qui a souvent rendu complexe la compréhension, la mise au point et la correction (debuggage) des programmes est le fait qu'ils soient ensuite généralement compilés donnant des programmes exécutables par la machine (code du processeur) qui peut s'éloigner fortement de l'idée initiale du concepteur.

Cela a conduit au développement de la programmation basée sur un environnement graphique qui date des années 1980 et qui a vu le jour grâce à l'apparition d'ordinateur proposant cette capacité liée à une interface graphique (Macintosh d'Apple™ en 1984). La possibilité de dessiner son programme (la liste des actions souhaitées) permet au concepteur d'exprimer ses idées d'une manière plus intuitive et plus naturelle. L'esprit humain perçoit et comprend les concepts compliqués plus rapidement lorsque ceux-ci sont représentés graphiquement. Lorsque des objets peuvent être manipulés graphiquement, la communication homme-machine est grandement facilitée. Ce principe a

conduit au développement des systèmes d'exploitation graphiques de tous les ordinateurs actuels.

Le but de la programmation graphique est donc de faciliter la mise en œuvre d'applications informatiques. On peut dégager les avantages suivants :

- la facilité d'utilisation par les non-programmeurs : les images sont en général préférées aux mots, la programmation devient alors plus intuitive ;
- la sémantique des images est plus puissante que celle des mots (davantage de signification dans une unité d'expression plus concise) ;
- les images ne sont pas sujettes aux barrières des langues.

Il est évident que toute méthode à ses inconvénients. Ainsi les principaux problèmes liés à la programmation graphique sont les suivants :

- la difficulté de visualisation pour les programmes de taille importante nécessitant une architecture modulaire et hiérarchique ;
- l'ambiguïté dans l'interprétation des graphismes ;
- la nécessité de disposer d'un environnement de développement efficace (éditeurs, outil de mise au point, outil de test…).

Lorsque des personnes imaginent ou conçoivent une application, ils ont tendance à dessiner des schémas pour expliquer l'enchaînement et le fonctionnement de l'application qu'ils veulent réaliser. Nous retrouvons alors la représentation naturelle de conception sous forme de « blocs fonctionnels » d'une application. Chaque bloc fonctionnel décrit l'action à réaliser qui peut être plus ou moins complexe.

Ce mode de programmation s'applique parfaitement à la description du contrôle/commande des systèmes industriels (machine outils, véhicules de transport, appareils électroménagers, etc.). Prenons un exemple très simple d'un portail télécommandé (Fig. 1.2). Les trois blocs fonctionnels sont successivement : « Attendre », « Ouvrir portail », « Fermer portail ». Le schéma fonctionnel de cette application ouverture/fermeture automatique d'un portail amène à deux remarques :

- chacun des blocs fonctionnels intègre des actions complexes : commande du moteur électrique du portail, temporisation éventuelle, test de fin d'ouverture ou de fermeture, arrêt de l'action si réception d'un contre-ordre (cas réel de fonctionnement de ce type d'automatisme), etc. ;
- l'enchaînement de ces actions nécessite des évènements ou des données externes pour s'exécuter qui sont ici symbolisés par des « arrivées » entre les blocs fonctionnels. Le passage du dernier bloc fonctionnel à l'état « Attendre » est déclenché par la fin du bloc fonctionnel précédent « Fermer portail ».

La deuxième remarque, effectuée précédemment, conduit à deux formalismes déterminant le mode d'exécution du programme : soit orienté flux de données, soit orienté flux de contrôle. Les représentations orientées flux de contrôle ont longtemps été utilisées pour décrire des algorithmes (avant même l'apparition des premiers ordinateurs). Ces représentations décrivent les programmes comme étant des nœuds de

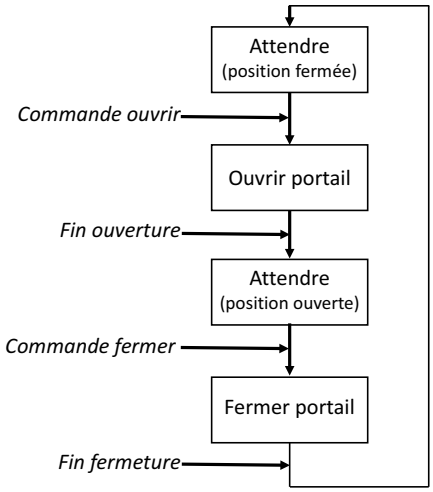


**Figure 1.2** – Exemple de la description fonctionnelle graphique d'une application « portail télécommandé ».

calcul connectés par des arcs spécifiant quel est l'ordre de calcul : exemple du modèle GRAFCET très utilisé dans les automatismes industriels réalisés souvent à base de relais électromécaniques (Fig. 1.3 et Encart 1.1).

Au contraire, dans les graphes pilotés par le flux de données, ce sont les données qui dirigent l'exécution du programme, analogie avec un circuit électronique et la propagation du signal à travers ce circuit. Ce type de fonctionnement est celui adopté par le langage de programmation graphique LabVIEW : la **programmation graphique flot de données**.

## Encart 1.1 Le GRAFCET, un modèle adapté aux applications de contrôle commande

Le GRAFCET, graphe fonctionnel de commande étape transition ou diagramme fonctionnel en séquence, a été créé et normalisé dans les années 1990. Il est destiné principalement à représenter des automatismes (machines outils, appareils électroménagers, etc.), c'est un moyen de communication adapté entre l'automaticien et le client pour la rédaction du cahier des charges de son application. Le graphe fonctionnel ainsi élaboré est appelé « grafcet » qui est graphe bipartite alterné, composé alternativement de transitions et d'étapes. L'étape représente un état du système qui est en général associé à une action ; elle est dite valide si le système est dans l'état représenté par cette étape. Les transitions sont associées à une condition de franchissement qui peut-être un événement ou une condition, appelée réceptivité.

Les règles d'évolution sont assez simples :

- une transition est validée si toutes les étapes amont sont actives ;
- une transition est franchissable si elle est validée (voir ci-avant) et si sa condition de franchissement est remplie ;