Francis Cottet
Luc Desruelle
Michel Pinard

LabVIEW

Programmation et applications
Introduction à LabVIEW NXG

Préface de Jeff Kodosky,
créateur de LabVIEW

4e édition

DUNOD

National Instruments grants the nonexclusive right to use the LabVIEW logo on the cover and the National Instruments Logo on the back cover of the book. Designs and images representing NI ideas and concepts are the sole property of National Instruments Corporation.

National Instruments accorde le droit non exclusif d'utiliser le logo LabVIEW et le logo National Instruments sur la couverture du livre. Les dessins et les images représentant les idées et concepts de NI sont la propriété exclusive de National Instruments Corporation.

Photo de couverture : Interfaces LabVIEW

© Dunod, Paris, 2001, 2009, 2015, 2018
11, rue Paul Bert, 92240 Malakoff
ISBN 978-2-10-078283-3

Préface à la 4e édition

LabVIEW will be 29 years old this autumn, and when I reflect on its history, I am gratified by the success it has enjoyed, bemused by how much of my life I have invested in its development, and sobered at the realization that a number of the original ideas are still being worked on. Granted, these were the more far out and fuzzier ideas, but the fact that it has taken more than 15 years to reach the point where we can begin exploring them seriously, inevitably makes me wonder about how much time is left because, after all, I am a grandfather now. On the other hand, to be back in an inventing mode, with the possibility of making another advance as significant as LabVIEW 1 was in its day, is exhilarating. Well, perhaps that can be the topic of a future foreword. For now, let me briefly describe how we have gotten to where we are today.

In 1983, National Instruments was a small company making GPIB interfaces and driver software for a variety of mini and microcomputers and operating systems, but we were growing faster than the market and it was clear that we had to do something more if we were going to continue our growth rate. The logical step was to tackle the next higher level problem beyond connecting instruments to computers, namely writing the software to automate the measurement systems.

There were lots of things we could have done as evidenced by the variety of extant products based on Basic or Forth, or consisting of libraries to be used with C or Fortran, or products which were simply configurable but specialized applications. But we wanted to do something that was different, dramatically better, revolutionary. In fact, the precedent we wanted to emulate was the spreadsheet. The spreadsheet made the computer accessible to everyone. It put the power and flexibility of the computer squarely in the hands of the user as no other application had ever done before. We wanted to do the same thing for our customers, engineers automating product testing and scientists automating laboratory experiments.

We could have set out a less grandiose task for ourselves, something more easily achievable, for instance, a modest incremental improvement over existing products. But that wasn't the kind of software we wanted to create, and that wasn't the kind of job we wanted to do. So we pursued the grandiose goal.

After almost a year of occasional useful insights but no major breakthroughs, the frustration was getting unbearable. And then the unexpected occurred. The Macintosh was introduced. It was immediately clear that the Mac's mouse and interactive graphics had to be part of the solution we were seeking. The mouse and graphics made it much easier to interact with software, but could they be used to actually construct software more productively? We were optimistic and naive enough to believe they could.

Once again it was many months of mounting frustration before the last major insight arrived. I finally saw a way to combine the rigor of structured programming with the simplicity and inherent parallelism of dataflow to create a graphical structured dataflow language. The rest, as they say, is history.

LabVIEW 1 took over a year and a half to implement on the Mac. It was an interpreter, so it was slow. It had a painful lack of editing features and only minimal I/O and computational capability. Yet it inspired people and it really was accessible to engineers and scientists, and even doctors.

It was also clear to the development team as well as customers that LabVIEW should have a compiler so it would execute as fast as conventional text languages. It ought to be much more convenient and smarter about editing. There should be a way to undo an edit. It should support more types of I/O, such as DAQ cards. It should support color monitors. There should be a way to compare different versions of a program. The structured dataflow language is parallel so it should run on multithreaded systems and multiprocessors and distributed systems. A loop with a timer should be deterministic if the operating system was realtime. And why not program an embedded processor using the same graphical language?

All these things were obvious to everyone concerned, but no one guessed it would take as long as it has to implement them. And in the meantime, the Web was invented, FPGAs were invented, machines became 1 000 times faster with 1 000 times the RAM and disk space, all adding to the development challenge and opportunity. However, as I mentioned at the start, there were many other ideas, not as obvious and not as well defined, but still part of the overall virtual instrumentation vision, which we have yet to complete. So stay tuned. LabVIEW is not finished. In fact, I think it's still just getting started.

Jeff KODOSKY, Austin, Texas
February 4th, 2015

Traduction de la préface de Jeff KODOSKY par Fabrice LEMAINQUE

LabVIEW fêtera son 29e anniversaire cet automne. Lorsque je me penche sur son histoire, je me réjouis du succès qu'il a rencontré, suis abasourdi par le temps que j'ai pu lui consacrer mais reste désolé en constatant qu'un certain nombre d'idées originales demeurent encore à implémenter. D'accord, il s'agit essentiellement des idées les plus extrêmes et les plus tordues, mais le fait qu'il nous ait fallu quinze ans pour atteindre le point où nous pouvons commencer à les explorer sérieusement me fait m'interroger sur le temps qui nous reste. Après tout, je suis désormais grand-père. D'un autre côté, revenir au mode invention, avec la possibilité d'aboutir à une réalisation aussi innovante que l'était LabVIEW 1 à son époque est exaltant. Peut-être cela devrait-il faire l'objet d'une autre préface. Je vais ici me borner à décrire brièvement comment nous en sommes arrivés au point actuel.

En 1983, National Instruments était une petite société réalisant des logiciels d'interfaces et de pilotes GPIB pour différents mini et micro-ordinateurs et systèmes d'exploitation. Nous grandissions toutefois plus vite que le marché et il était évident que nous devions passer à quelque chose de nouveau pour maintenir notre taux de croissance. L'étape logique était de s'attaquer au problème de niveau supérieur à celui de la simple connexion d'instruments à un ordinateur, en pratique d'écrire un logiciel d'automatisation des systèmes de mesure.

Nous aurions pu accomplir bien des choses, en regardant les différents produits existants fondés sur les langages Basic ou Forth, s'appuyant sur des bibliothèques destinées à être employées en C ou en Fortran, ou constituant des applications spécialisées faciles à configurer.

Nous souhaitions toutefois réaliser quelque chose de différent, fantastiquement mieux, révolutionnaire. En fait, nous cherchions à suivre l'exemple du tableur. Les tableurs ont rendu l'ordinateur accessible à chacun. Ils plaçaient la puissance et la souplesse de l'informatique entre les mains de l'utilisateur comme jamais aucune application ne l'avait fait auparavant. Nous voulions accomplir la même chose pour nos clients, des ingénieurs effectuant des tests automatisés de produits et des scientifiques réalisant des expériences de laboratoires automatisées.

Nous aurions pu nous fixer un objectif moins ambitieux, quelque chose de plus facile à atteindre : par exemple, une amélioration incrémentielle mineure des produits existants. Ce n'était toutefois pas le type de logiciel que nous souhaitions concevoir, ni le type de travail que nous voulions accomplir. Nous avons donc maintenu notre but grandiose.

Après presque un an de percées occasionnelles utiles mais sans réalisation majeure, la frustration devenait insupportable. Puis survint l'inattendu. Le Macintosh apparut sur le marché. Il nous sembla évident que la souris et les graphismes interactifs du

Mac devaient être intégrés à la solution que nous cherchions. La souris et les icônes facilitaient grandement l'interactivité avec les logiciels, mais pouvaient-elles être employées pour améliorer la productivité en matière de construction logicielle ? Nous étions suffisamment naïfs et optimistes pour le croire.

Une fois encore, il s'écoula des mois de frustration croissante avant la dernière grande inspiration. J'ai finalement découvert un moyen de combiner la rigueur de la programmation structurée avec la simplicité et le parallélisme inhérent aux flux de données pour créer un langage de flux de données graphique structuré. Le reste, comme on dit, c'est de l'histoire.

Il m'a fallu plus de dix-huit mois pour implémenter LabVIEW 1 sur Mac. C'était un interpréteur, donc lent. Il présentait un manque cruel de dispositifs d'édition et des capacités d'entrées-sorties et de calcul limitées. Il a pourtant plu au public et était réellement accessible aux ingénieurs et scientifiques, et même au monde médical.

Il était également évident tant pour l'équipe de développement que pour les clients que LabVIEW devait disposer d'un compilateur pour pouvoir s'exécuter aussi rapidement que les langages textuels conventionnels. Il lui fallait de même être plus pratique et plus intelligent en matière d'édition. Il fallait pouvoir annuler une modification, et il devait prendre en charge d'autres types d'entrées-sorties, comme les cartes DAQ. Il devait prendre en charge les écrans couleur. Il devait être possible de comparer différentes versions d'un programme. Puisque le langage de flux de données structuré était parallèle, il devait pouvoir s'exécuter sur des systèmes multithreads, multiprocesseurs et distribués. Une boucle cadencée devait être déterministe si le système d'exécution était Temps réel. Et pourquoi ne pas programmer un processeur embarqué à l'aide du même langage graphique ?

Toutes ces choses étaient évidentes pour chaque personne impliquée, mais personne ne pouvait deviner qu'il faudrait aussi longtemps pour les mettre en œuvre. Et, pendant ce temps, le Web était inventé, de même que les circuits FPGA, tandis que la vitesse des processeurs, la taille de la RAM et de l'espace disque se voyaient multipliées par 1 000, augmentant d'autant les défis et opportunités de développement.

Toutefois, comme je l'ai mentionné au début, il subsiste de nombreuses idées, pas aussi évidentes ni aussi bien définies, mais appartenant néanmoins à la vision globale de l'instrumentation virtuelle, et qu'il nous reste à implémenter. Alors, restez à l'écoute. LabVIEW n'est pas achevé. En fait, j'incline même à penser qu'il commence juste.

Jeff KODOSKY, Austin, Texas
4 février 2015

Introduction

LabVIEW est un environnement de développement complet, graphique, compilé et particulièrement bien adapté au domaine de l'acquisition et de la mesure. Son approche totalement graphique offre une souplesse et une dimension intuitive inégalée. Comparativement aux langages textuels il offre la même puissance de programmation mais sans le côté abstrait et complexe lié à la syntaxe.

Orienté tests et mesures, il dispose de nombreuses fonctions permettant de piloter facilement des cartes d'acquisition et autres instruments, mais aussi de filtrer, d'analyser et de présenter les données. Ce langage est également appelé code G. Le code est représenté par un schéma composé de fonctions, de structures et de fils qui propagent les données. L'approche visuelle, l'interface entre le code et le développeur s'opposent sur la forme et la philosophie aux langages textuels, comme le C. LabVIEW est redoutable de puissance et n'a rien à envier aux autres langages.

Une des différences fondamentales de LabVIEW est que ce langage suit un modèle de flux de données, et non de flux d'instructions. Cela signifie que pour un langage textuel ce sont les instructions qui ont la priorité, alors qu'avec LabVIEW ce sont les données. Une fonction s'exécutera donc uniquement si elle dispose à ses entrées de toutes les données dont elle a besoin. Lorsqu'un langage classique est ainsi séquentiel, LabVIEW est naturellement prédisposé au parallélisme. Ce qui augmente encore sa puissance et la rapidité d'exécution du code.

En lisant ce livre, vous pourriez avoir le sentiment que LabVIEW est un outil uniquement dédié à l'acquisition et à la mesure. Certes, il propose un environnement qui facilite grandement cela, et les auteurs de ce livre ont fait délibérément le choix de mettre cette facilité à l'honneur. Mais ne vous y trompez pas, LabVIEW est infiniment plus que cela. C'est un langage à part entière manipulant une grande variété de concepts tels que le flux de données, la programmation objet, la gestion événementielle, l'encapsulation, la modularité, le typage des données, etc. Il peut vous permettre d'implémenter n'importe quel type d'application.

En l'utilisant vous pourrez mesurer les données d'un capteur relié à une carte d'acquisition, analyser les informations provenant d'un instrument connecté par une liaison série, réaliser un rapport de test sous Microsoft Office, implémenter un PID sur une cible FPGA, contrôler précisément un procédé physique complexe dans

une tâche déterministe sur une cible Temps Réel, trouver par le calcul matriciel la distance de Mahalanobis... mais également solutionner un Rubik's cube, un Sudoku et même jouer aux échecs (un exemple de jeu d'échecs implémenté en langage G est abordé dans ce livre).

Ce langage s'adresse à tous les techniciens ou ingénieurs qui désirent développer une application de mesures, de tests, de supervision ou de contrôle/commande avec une interface utilisateur de très grande interactivité. Mais plus généralement à tous les développeurs qui désirent utiliser un langage intuitif, puissant et ouvert sur la totalité du génie logiciel.

Historique

LabVIEW, contraction de Laboratory Virtual Instrument Engineering Workbench, a été développé par la société National Instruments à partir de 1983. Initialement conçu pour une plate-forme MacOS dans la première version distribuée en 1986, le langage devient compilé en 1990. L'environnement de programmation LabVIEW est porté sur plate-forme Windows dès 1992. La version 3.1 en 1994 intègre le composant « Application Builder » qui permet de générer un exécutable autonome, à partir du code source. En 1998 la version 5 intègre la gestion multitâche, des assistants logiciels pour les cartes d'acquisition (DAQ) et le contrôle d'instruments, le portage sous Linux et le module pour cible Temps Réel (RT). La version 7.1 de 2004 marquera de grandes évolutions avec les modules pour cible FPGA et Windows CE, la structure événementielle et la traduction de la plate-forme de développement en français. La version 8 introduit la gestion par projet, les XControls et surtout la programmation objet (OOP). À partir de 2009, National Instruments décide de générer une version de LabVIEW chaque année et dont le nom sera celui de l'année. Dans sa version 2018, le logiciel est actuellement disponible sur différents systèmes d'exploitation : Windows, Mac OS, et Linux. La portabilité des applications développées sous LabVIEW est totale entre les différentes plates-formes. LabVIEW peut générer du code compilé sur ces systèmes d'exploitation mais également pour des cibles Temps Réel (RT), FPGA et embarquées. En 2017, National Instruments a dévoilé deux nouvelles versions : LabVIEW 2017, la version « Standard », et LabVIEW NXG 1.0, la « Nouvelle génération » (NeXt Generation). Au fil des années, LabVIEW NXG doit devenir une version plus aboutie, plus ergonomique et plus performante de la version actuelle de LabVIEW. Pour des raisons évidentes le nom LabVIEW a été conservé. Mais nous sommes sur deux versions indépendantes et incompatibles de l'environnement de développement. Le code de la version « standard » doit être migré pour être utilisé sur la version « NXG ». Par contre, il est impossible de faire le chemin inverse. Cette première version ne présentait pas encore toutes les fonctionnalités de LabVIEW Standard. Sous le slogan « aussi productif que LabVIEW,

la programmation en option », elle focalise l'attention sur le premier pilier de cette version qui est une nouvelle approche de l'automatisation des mesures, dans un seul logiciel et sans programmation. Cette première version est incomplète. Elle ne possède pas l'ensemble des fonctionnalités nécessaires à la programmation, ni l'ensemble du support matériel. Elle n'est qu'une première étape. En 2018, elle sera suivie par la version NXG 2.0 et surtout la 2.1. Cette deuxième version amène véritablement les outils de développement nécessaires à la programmation en code G. C'est le deuxième pilier de la révolution. De nouvelles fonctionnalités sont dévoilées, comme les WebVIs. Cette révolution est tellement ambitieuse, qu'elle se déroule par étapes successives. Il faudra attendre 2019 pour avoir une grande partie des concepts et *toolkits* équivalents à la version standard, dans cet environnement innovant, modernisé et résolument tourné vers le futur de LabVIEW.

LabVIEW dans le monde de l'instrumentation

LabVIEW est un des premiers langages de programmation graphique destiné au développement d'applications d'instrumentation. Un logiciel d'instrumentation pourrait être défini comme un programme permettant de contrôler depuis un ordinateur, un système allant du capteur à la chaîne d'acquisition ou de commande jusqu'à l'édition du rapport final. Couplé à des cartes d'entrées/sorties, il permet de gérer des flux d'informations numériques ou analogiques et de créer ou de simuler des instruments de mesures (oscilloscope, compteur, multimètre, etc.). Le temps nécessaire à l'assemblage des composants de ce type de système est négligeable par rapport à celui nécessaire à sa programmation dans un langage classique (C, Pascal, Ada, etc.). Les interfaces utilisateurs développées avec ces langages sont le plus souvent obscures et incompréhensibles. Les utilisateurs disposent avec LabVIEW d'un puissant outil intégré d'acquisition, d'analyse et de présentation des données, une solution qui entraîne un gain notable de productivité comparable à celui obtenu par l'introduction des tableurs dans les logiciels financiers. Pour cela, le langage utilise toute la convivialité des interfaces interactives des ordinateurs actuels en fournissant des objets proches graphiquement des objets réels (voyants, curseurs, interrupteurs, boutons, graphes, etc.) mais aussi des commandes systèmes (pour une représentation plus standard) ainsi que des outils familiers et simples d'utilisation pour la programmation (structures de programmation, fonctions arithmétiques, fonctions logiques, comparateurs, etc.).

Remarques

La version française de LabVIEW 2017 est l'environnement de référence de cet ouvrage. Afin de permettre aux utilisateurs de versions anglaises d'utiliser ce livre, une traduction est associée aux termes importants dans la mesure où cela

ne nuit pas à la lisibilité du texte, par exemple « mode animation (*execution highlighting*) ». Cet ouvrage peut aussi être utilisé pour des versions précédentes du logiciel étant donné que les versions successives apportent des enrichissements de l'environnement tout en gardant une compatibilité.

De plus pour permettre au lecteur une meilleure compréhension des concepts expliqués dans le livre, le code présenté dans les exemples est disponible en téléchargement sur la page dédiée à cet ouvrage sur le site Web de Dunod (www.dunod.com). Il est classé par chapitre et il est utilisable à partir d'une version de développement LabVIEW 2010.

Le langage LabVIEW propose un environnement très orienté instrumentation où nous retrouvons les quatre bibliothèques de base nécessaires à ces applications industrielles (figure 1) :

- **acquisition et génération des données** : contrôle d'instruments (GPIB, série, Ethernet, USB, PCI, PXI, VXI), gestion de cartes d'entrées/sorties numériques/analogiques (DAQmx), gestion de cartes d'acquisition d'images, commande d'axes moteurs, etc. ;
- **analyse et traitement des données** : traitement du signal (génération, filtrage, FFT, etc.), traitement statistique (régression, lissage, moyenne, etc.) ;
- **présentation et sauvegarde des données** : affichage (courbes, graphiques 2D, etc.), stockage des données (archivage, base de données), génération de rapports (impression, rapport Word et Excel) ;
- **publication et échange des données** : échange de données entre application (ActiveX, .NET, etc.), partage par le réseau (TCP/IP, Internet, moteur de variables partagés, DataSocket, etc.).

Figure 1 – Bibliothèques logicielles proposées dans l'environnement LabVIEW.

De nombreuses applications professionnelles et industrielles très variées ont été développées avec l'environnement LabVIEW. Pour informer le lecteur nous pouvons

citer quelques-unes de ces applications qui font en outre l'objet de fiches descriptives détaillées et disponibles sur le site Web de National Instruments (Annexe) :

- nanocaractérisation de matériaux par un système de spectrométrie par diffusion d'ions pour le CEA de Grenoble ;
- baie de tests de culasses de moteur de F1 ;
- analyse de la parole pour l'identification de locuteurs dans le cadre d'expertises judiciaires ;
- système d'acquisition de mesures destiné à tester des moteurs d'avions ;
- système de surveillance de l'isolation des transformateurs de mesures pour tester l'état du réseau électrique ;
- simulateur d'ILS (Instrument Landing System) pour la formation d'ingénieurs des systèmes de sécurité aérienne ;
- système automatisé des tests de téléphones sans fil ;
- test des systèmes de freinage pour les wagons de fret ;
- etc.

Organisation du livre

Cet ouvrage est divisé en sept chapitres.

Le premier présente les concepts et l'intérêt de la programmation graphique. Le flux de données qui est une notion fondamentale à comprendre sous LabVIEW est défini, ainsi que les éléments nécessaires à sa propagation et son exploitation.

Le chapitre deux permet de décrire les éléments de base de l'environnement de programmation, illustrés avec des exemples simples. Le lecteur apprend à utiliser les objets pour réaliser une interface utilisateur mais aussi les structures de programmation et les bibliothèques de fonctions. Le langage de développement graphique LabVIEW est intuitif, mais cela ne doit pas faire oublier que, comme avec tous les langages, il faut respecter des règles.

Le chapitre trois aborde des aspects plus avancés de l'utilisation de l'environnement en définissant des techniques permettant au code d'être maintenable, évolutif, performant et compréhensible par d'autres utilisateurs. Il permet de décrire par des exemples concrets et détaillés les structures de programme à utiliser afin de permettre au lecteur de réaliser une application. L'utilisateur sera capable à la suite de ce chapitre de préparer la certification LabVIEW développeur.

Les trois chapitres suivants abordent les capacités spécifiques de LabVIEW pour l'acquisition, l'analyse et la présentation des données. LabVIEW est le langage de développement le plus efficace pour réaliser un tel système. Dans le chapitre quatre, le lecteur apprendra à construire un système de mesure ou de génération de signal, à base de carte d'acquisition

(DAQ), d'instrument (IVI, VISA), de système Temps Réel (RT) ou de FPGA. Différentes méthodes de connexion seront envisagées (Série, GPIB, Ethernet, PCI, etc.), ainsi que la réalisation de drivers dans les règles de l'art. Après cette collecte des données, dans le chapitre cinq l'utilisateur pourra à partir de bibliothèques intégrées réaliser un traitement du signal, une analyse mathématique ou un filtrage de l'information. Dans le dernier chapitre, nous illustrerons comment archiver et échanger les données ainsi traitées. Nous verrons comment finaliser son développement, en générant un exécutable autonome et en réalisant un rapport professionnel sous Microsoft Office, pour présenter ses données.

Le chapitre sept est consacré à LabVIEW NXG, la nouvelle génération de l'environnement de développement LabVIEW. Les principes de la programmation en code G restent identiques entre les deux versions de LabVIEW. Mais l'éditeur a été révolutionné pour être plus intuitif, plus moderne, plus ergonomique et s'éloigner du concept de l'instrument physique. De puissants nouveaux concepts apparaissent, comme les interfaces interactives pour réaliser des mesures sans programmation, la génération de code HTML dans les WebVIs ou la capture des données. Le développeur devra donc changer quelques habitudes par rapport à la version LabVIEW classique. Les nouvelles fonctionnalités sont progressivement détaillées, illustrées avec des exemples simples et complétées par des glossaires. Le lecteur apprend progressivement à faire le lien entre les deux environnements et à comprendre les différences. Au final, la démarche à suivre pour la migration du code de LabVIEW Standard vers NXG est décrite concrètement par un développeur, à partir d'un exemple de projet qui a été illustré dans le chapitre 3.

Ce livre n'a pas pour but d'être un ouvrage exhaustif. En effet il existe la documentation en ligne sur le site de National Instruments, également fournie avec le logiciel, qui est une description complète de l'ensemble des possibilités de LabVIEW. Mais cet ouvrage permet de s'initier aux bonnes règles de développement du logiciel LabVIEW, de les comprendre et les mettre rapidement en œuvre par des exemples concerts. Il permet aussi d'appréhender différents domaines d'applications du langage pour l'acquisition, l'analyse et la présentation des données.

Remarques de lecture

Dans l'environnement de développement LabVIEW, la **couleur** est un paramètre important ayant une sémantique très précise concernant en particulier le typage des données. Aussi, pour permettre au lecteur une meilleure compréhension, l'ensemble des figures sont en couleur.

Les conventions d'écriture suivantes figurent dans ce livre :

- police `courier New` : mots utilisés dans l'environnement LabVIEW (exemple : « commandes ») ;
- mots en italiques et entre parenthèses : traduction des mots utilisés dans l'environnement LabVIEW en version anglaise (exemple : « (*controls*) »).

Remerciements

Remerciements de M. Cottet

Je tiens à remercier ici toutes les personnes à qui je dois d'avoir pu réaliser cet ouvrage.

Il me vient naturellement à l'esprit Louis-Paul Doco (National Instruments France) dont la confiance et l'amitié tout au long de ces années m'ont permis de concrétiser de nombreux projets et, en particulier, celui de ce livre. Je remercie Jeff Kodosky (National Instruments), créateur de LabVIEW, d'avoir accepté de préfacer ce livre.

Les diverses collaborations fructueuses que j'ai pu avoir avec Patrick Renard (National Instruments France) au cours des dix dernières années m'ont fourni la matière à cet ouvrage. Pour cela, je l'en remercie chaleureusement. Je ne saurais citer toutes les personnes de la société National Instruments qui ont participé de près ou de loin à ce livre lors de discussions ou par leurs écrits et leurs exposés.

Je remercie vivement Emmanuel Geveaux (Tecatlant), qui par ses travaux de thèse, ses commentaires et ses remarques constructives, m'a apporté une aide dans la rédaction de ce livre. Qui mieux que lui n'illustre la phrase « l'élève a dépassé le maître ».

De nombreuses conversations avec mes collègues et particulièrement Laurent David et Emmanuel Grolleau m'ont permis d'enrichir et d'améliorer ce livre. Ces derniers ont de plus relu avec beaucoup d'attention les premières rédactions du présent ouvrage, je leur suis très reconnaissant de tout ce travail.

Je souhaite également remercier mon père qui a consacré beaucoup de temps à toutes ses relectures attentives. Enfin, j'entends dédier ce livre à ma femme et mes deux enfants, qui ont dû passer de longues journées seuls, sans rien voir d'autre de moi qu'une tête fixée sur un écran d'ordinateur.

« Les racines de l'éducation sont amères, mais ses fruits sont doux. »

Aristote

Remerciements de M. Pinard

J'ai commencé à utiliser le logiciel LabVIEW il y a pratiquement une dizaine d'années, lorsque j'ai voulu traiter les signaux issus de capteurs placés sur des montages électroniques ou électrotechniques.

Comme je l'ai indiqué dans un autre ouvrage paru chez Dunod, *La commande électronique des machines*, j'ai pu utiliser les acquisitions des courants, tensions, vitesse relatifs à un moteur, et ainsi obtenir par calcul un tracé en temps réel du couple par estimation ou par transformation mathématique. Et ceci, grâce aux systèmes d'acquisition, aux traitements numériques mettant en œuvre le logiciel LabVIEW.

En développant des applications de LabVIEW dans un but pédagogique, mes étudiants se sont très vite intéressés à l'utilisation des boucles, des flux de données, des tableaux, des contrôles, des commandes.

Par ailleurs, j'ai eu l'occasion de demander à des étudiants étrangers stagiaires (chinois, grecs, singapouriens, sud-africains…) de développer sur LabVIEW des projets divers : présentation virtuelle de maquette de jeu, d'un convertisseur de puissance, d'un contrôleur de signaux, d'un indicateur d'état pour un lanceur de satellite… Là encore, un très vite engouement pour les boucles, les indicateurs, l'utilisation des capteurs…

Il me semble que c'est le côté « emboitement à l'infini » du logiciel qui séduit les utilisateurs et qui les amène à pousser leur travail et leur recherche parfois même au-delà de leur propre limite.

Je remercie tout particulièrement Monsieur Frédérick Drappier, Directeur général de National Instruments France, qui m'a encouragé dans mon travail.

Et ma gratitude va à Monsieur Alexandre Lubino qui m'a aidé à exploiter au mieux les dernières versions du logiciel.

Je tiens aussi à remercier mes deux coauteurs qui m'ont apporté beaucoup, chacun avec leur curiosité et leur expérience, aussi bien pour développer des sujets de réflexion et d'étude que pour écrire un livre original et riche en exemples pédagogiques.

En espérant que le lecteur pourra lui aussi y découvrir quelques richesses…

Remerciements de M. Desruelle

L'écriture d'un livre est une fabuleuse aventure, dont je savoure d'autant plus le plaisir que celle-ci touche à sa fin. Je tiens à remercier chaleureusement les personnes qui m'ont guidé sur ce chemin. J'ai débuté le développement LabVIEW en lisant la première version du livre et c'est aujourd'hui comme architecte certifié (CLA) et LabVIEW Champion que je collabore à cette nouvelle version. J'espère que ce livre continuera à inspirer d'autres développeurs.

Tout d'abord merci à toute l'équipe de mon éditeur pour son professionnalisme lors de la finalisation et la promotion du livre. Merci aussi aux deux premiers auteurs pour m'avoir proposé de faire partie de cette aventure depuis la troisième édition.

Je ne saurais citer toutes les personnes de la société National Instruments qui ont été une source d'inspiration au cours de ces dernières années. Grâce à leurs présentations sur Internet, leurs réponses sur le forum francophone, les challenges de la communauté, et surtout l'excellence des conférences, j'ai appris beaucoup au fil des années de ce que j'ai voulu transmettre dans ce livre. Je remercie plus précisément Jeff Kodosky pour avoir inventé le logiciel graphique LabVIEW que j'utilise avec passion chaque jour, et d'avoir accepté de préfacer le livre. Il me vient également à l'esprit toute l'équipe de NI France et plus particulièrement Marie Remondière, Marie Weill et Emiliana Tedesco. J'ai une pensée particulière pour Emmanuel Roset qui m'a aidé techniquement et pour ses précieux conseils lors de l'écriture du chapitre sur LabVIEW NXG.

Je veux remercier tous ceux qui m'ont donné l'envie d'écrire ce livre, notamment tous mes amis spécialistes du codage textuel lors de nos débats sur les performances des différents langages. J'espère (avec beaucoup d'humour) leur donner des arguments pour passer de la syntaxe textuelle séquentielle à l'intuitif parallélisme graphique, pour passer de l'ombre à la lumière.

Je veux exprimer mes sincères remerciements à tous mes clients qui me proposent depuis de nombreuses années de relever des défis passionnants avec LabVIEW. J'espère que cela durera encore longtemps. Leurs projets ont été la matière première de ce livre. Certains sont devenus bien plus que des clients, comme une certaine équipe « monitoring ». J'ai une gratitude particulière pour Guillaume Granelli pour sa lecture attentive et ses remarques pertinentes.

Merci à tous mes collaborateurs de l'équipe MESULOG qui m'ont apporté conseils et aide précieuse. Leur grande expérience et expertise du logiciel LabVIEW ont été le ciment pour consolider les informations du livre (merci à toutes vos « Beta gold »). Relecteurs compétents et persévérants, je remercie Jérémy Chiffe, Maxime Renaud, Mathieu Reyrolle, Micaël Da Silva, Julien Massé et Damien Lamy. J'ai une pensée

particulière pour Jean-Louis Schricke qui a été le premier à me faire confiance et à m'intégrer dans une entreprise spécialisée dans le développement professionnel sous LabVIEW.

À tous ceux qui m'ont consacré de leur temps, m'ont conseillé et m'ont soutenu. Je pense à Pierre dit « ouadji » développeur confirmé et exigeant qui m'a soutenu sans faille dès le début, il a relu une grande partie de mon travail et a permis d'améliorer le livre. À Olivier Jourdan pour son aide sur les outils indispensables à utiliser avec LabVIEW, notamment les sondes personnalisées.

Pour terminer, j'ai une pensée pour toute ma famille qui m'a soutenu et encouragé, mon père et mon frère qui ont relu avec une grande attention ce livre qui est si technique, alors qu'ils ne sont pas du métier. Je dédie le livre à ma compagne et mes deux fils, et je sais maintenant qu'ils comprennent comment piloter un instrument à partir d'une image. Merci à Maxime et Simon pour leur aide indispensable et inoubliable pour illustrer les acquisitions du chapitre sur LabVIEW NXG.

Pour vous tous qui m'avez consacré de votre temps, j'espère simplement que vous aurez du plaisir à prendre ce livre entre vos mains.

Table des matières

Le code des exemples est téléchargeable sur dunod.com.

1 Les concepts et l'environnement de programmation LabVIEW

Ce chapitre présente les concepts et l'intérêt de la programmation graphique, qui permet de représenter du code par un schéma, par opposition aux mots du langage textuel. Le code obtenu est ainsi plus intuitif et plus naturel. Le flux de données, qui est une notion fondamentale à comprendre sous LabVIEW, est défini, ainsi que les éléments nécessaires à sa propagation et son exploitation.

LabVIEW étant un langage de développement à part entière, nous décrirons dans ce chapitre les éléments de base de l'environnement de programmation. Notamment, nous détaillerons la notion d'instrument virtuel (VI) sur ordinateur, de « diagramme », de « face-avant » et plus généralement des outils permettant de prendre en main l'environnement LabVIEW.

Pour finir, nous présenterons en fin de chapitre un exemple très simple pour illustrer la réalisation d'un code graphique.

1.1 Concepts de base de la programmation graphique

1.1.1 Programmation graphique flux de données

La programmation à l'aide d'un langage graphique date des années 1980. Les langages de programmation textuels conventionnels impliquent un mode de conception séquentiel contraint par l'architecture même de la machine, l'exemple le plus simple étant l'assembleur puisqu'il reflète l'architecture du processeur. La possibilité de dessiner un diagramme ou le contrôle/commande d'un processus 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. À cet égard, la représentation textuelle

est moins performante puisqu'elle ne permet qu'une représentation séquentielle de l'information. D'ailleurs, les efforts mis en œuvre pour rendre un texte plus rapidement compréhensible reposent sur des moyens graphiques, tels que le surlignage, les italiques, les caractères gras, la couleur ou l'indentation d'un texte. Lorsque des objets peuvent être manipulés graphiquement, la communication homme/machine est grandement facilitée, ce qui a conduit par exemple au développement des systèmes d'exploitation graphiques (MacOS ou Windows). Les personnes ont tendance à dessiner des schémas pour expliquer le fonctionnement de mécanismes, de circuits, etc. Nous retrouvons alors la représentation naturelle de conception sous forme de « blocs fonctionnels » d'une application de contrôle/commande. Le but de la programmation graphique est donc de faciliter la mise en œuvre d'applications informatiques. On peut dégager les avantages suivants :

- 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 a ses inconvénients. Ainsi, les principaux problèmes liés à la programmation graphique sont les suivants :

- difficulté de visualisation pour les programmes de taille importante nécessitant une architecture modulaire et hiérarchique ;
- ambiguïté dans l'interprétation des graphismes ;
- nécessité de disposer d'un environnement de développement efficace (éditeurs, outil de mise au point, outil de test, etc.) comme dans le cas de l'environnement LabVIEW ;
- complexité de plus en plus grande des images au fur et à mesure de l'introduction de nouveaux concepts et de nouvelles fonctionnalités. L'utilisation de la notion de projet devient indispensable pour gérer le code et les applications.

La représentation graphique d'un programme peut être basée sur deux formalismes déterminant le mode d'exécution du programme : flux de données ou 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 calcul connectés par des arcs spécifiant quel calcul doit être effectué ensuite : les réseaux de Pétri ou le GRAFCET sont, par exemple, basés sur ce modèle. Au contraire, dans les graphes 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. Bien qu'il existe des langages flux de données textuels tels que LUSTRE (1991),

SIGNAL (1993) en partie ou bien encore Lucid (1986), nous nous intéresserons dans la suite de cet ouvrage à la représentation graphique de programmes dont l'exécution est basée sur le flux de données (*bibliographie* : Halbwachs 1991, Rutten 1993).

Un diagramme flux de données permet de décrire un programme de manière graphique, constitué de nœuds de traitements ou calculs interconnectés par des arcs qui spécifient le flux des données transitant entre les nœuds producteurs de données et les nœuds consommateurs de données. Le diagramme flux de données est un graphe acyclique qui peut être composé des trois éléments différents suivants (figure 1.1) :

- **terminaux :** ce sont les liens avec l'extérieur qui représentent la production et la consommation de données ;
- **nœuds :** ce sont les traitements à effectuer qui sont représentés par une figure géométrique pouvant contenir une image illustrant leur fonctionnalité. Ils possèdent des connexions d'entrée et des connexions de sortie ;
- **arcs orientés (ou fils de liaison sous LabVIEW) :** ils relient nœuds et terminaux et permettent d'indiquer le passage de données d'un nœud vers un autre. Un arc orienté peut se séparer en plusieurs branches, ce qui correspond à une duplication de la donnée. En revanche des arcs orientés ne peuvent pas se regrouper (convergence interdite).

Il est possible d'illustrer les données transitant sur les arcs (ou fils de liaison) par des jetons. Ce symbolisme de propagation de données par mouvement d'un jeton est utilisé dans un outil de mise au point appelé mode animation (chapitre 2). La production et la consommation des jetons dans le diagramme flux de données sont régies par les règles d'évolution suivantes :

- lors de l'initialisation, chaque terminal d'entrée produit un jeton sur son ou ses arc(s) de sortie ;
- lorsqu'un nœud possède un jeton sur chacun de ses arcs d'entrée, alors le nœud peut être exécuté : chaque jeton d'entrée est consommé et un jeton est produit sur chacun des arcs de sortie du nœud.

De ces règles, nous déduisons que plusieurs nœuds peuvent être exécutés simultanément. Dans le diagramme flux de données de la figure 1.1, les deux premiers nœuds « x » peuvent être exécutés en même temps en consommant les données *a*, *c* et deux fois la donnée *b*. La programmation flux de données intègre donc implicitement la notion de **parallélisme**. Ensuite, le troisième nœud « x » peut s'exécuter et enfin le dernier nœud « - » est exécuté en produisant la donnée D.

Un diagramme flux de données peut être encapsulé afin d'être réutilisé, en tant que nœud ou « macronœud », par d'autres diagrammes flux de données. Les terminaux du diagramme deviennent alors les connexions d'entrées/sorties de ce nouveau nœud. Ce mécanisme d'encapsulation est un mécanisme d'abstraction, bien connu

du génie logiciel : il permet de présenter des fonctionnalités sans décrire leur fonctionnement interne. La figure 1.2 montre ce mécanisme pour le diagramme flux de données de la figure 1.1. Formellement, cette encapsulation revient à donner un nom à un diagramme flux de données. Une fois encapsulées, les variables internes au nœud et les résultats des calculs intermédiaires ne sont plus visibles de l'extérieur. Il est important de noter que certains terminaux d'entrée (constante ou paramètre avec valeur par défaut) ou éventuellement de sortie (variable de test) peuvent être non visibles et donc non accessibles lors de l'utilisation de l'icône du programme flux de données ainsi encapsulé.

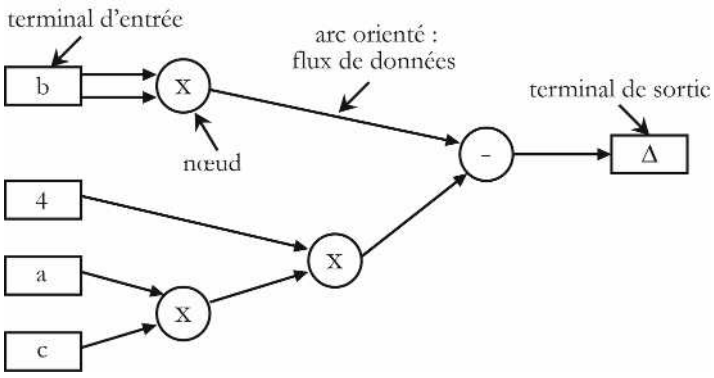

Figure 1.1 – Diagramme flux de données représentant le calcul du discriminant d'une équation du second degré $\Delta = b^2 - 4\ a\ c$.

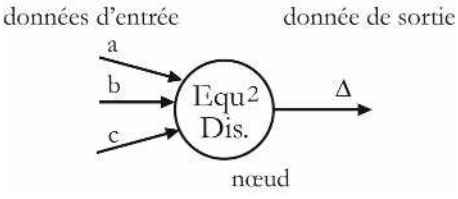

Figure 1.2 – Encapsulation du diagramme flux de données représenté à la figure 1.1.

1.1.2 Extension du concept « flux de données »

Le modèle flux de données, présenté dans le paragraphe précédent, est trop simple pour être utilisé en tant que langage de programmation. En effet, cette programmation flux de données pure souffre de deux manques :

- mémorisation momentanée de données afin de pouvoir réutiliser des données déjà calculées ;
- structures de programmation pour exprimer des choix ou des itérations.

La première extension apparaît essentielle afin de répondre au problème de **causalité** que l'on trouve dans un flux de données où il est nécessaire de réutiliser une donnée déjà produite (figure 1.3). En effet, le nœud « n1 » ne peut être exécuté que si la donnée *a* est présente sur son entrée ; or cette donnée ne sera produite que si le nœud « n2 » qui s'exécute est lui-même conditionné par l'exécution du nœud « n1 » : la boucle est bouclée. La mise en place d'une fonction de mémorisation pour casser ce flux de données résout le problème de causalité en distinguant la donnée $a_{(t-1)}$ à un instant t–1 et la donnée $a_{(t)}$ à un instant t. Il ne faut pas oublier d'affecter une valeur initiale à la donnée $a_{(t=0)}$ avec laquelle l'exécution du programme flux de données peut commencer.

Cette mémorisation de données est implémentée de différentes manières dans l'environnement LabVIEW : registres à décalage associés aux structures de programmation, variables locales ou globales. Ce problème de causalité est la seule difficulté conceptuelle de la programmation flux de données qui est immédiatement détectée et déclarée comme erreur lors de l'édition d'un programme flux de données.

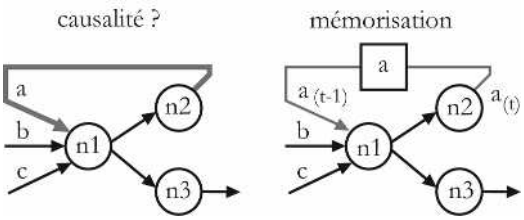

Figure 1.3 – Le problème de causalité dans un flux de données pur et sa solution par insertion d'une fonction de mémorisation.

Afin d'exprimer tous les algorithmes, il est nécessaire de disposer d'une structure itérative et d'une structure de choix. De plus, étant donné le parallélisme implicite de la programmation graphique flux de données, des structures permettant d'imposer des séquences dans les traitements semblent indispensables. Ainsi, dans l'environnement LabVIEW, nous trouvons : les deux structures d'itération boucle

« Pour » (*For*) et boucle « Tant que » (*While*), la structure « Condition » et la structure « Séquence » (chapitre 2).

En résumé, l'environnement LabVIEW propose un cadre de programmation graphique, basé sur le flux de données et enrichi des deux extensions mémorisation et structures de programmation. Ce langage de programmation, appelé **langage G** pour Graphique, possède la même puissance d'expression que les langages textuels classiques (*bibliographie* : Kodosky 1989, Kodosky 1991, Johnson 1997, Cottet 2001).

Remarque

Dans la suite de ce livre le terme LabVIEW désignera aussi bien l'environnement de développement d'applications que le code (appelé code G). Cette notion sera reprise en détail dans le chapitre 3.

1.2 Environnement de la programmation LabVIEW

1.2.1 Instrument virtuel (VI)

Si nous réalisons qu'un logiciel de mesure est un instrument de mesure contrôlé depuis un ordinateur à la place des boutons sur sa face-avant, cela conduit logiquement à la notion d'instrument virtuel (instrument réel contrôlé depuis un ordinateur). Un instrument virtuel est un programme qui présente une interface sous forme graphique pour l'apparenter à un instrument physique. Dans LabVIEW, les utilisateurs manipulent des instruments depuis l'ordinateur (virtuels) comme s'il s'agissait d'instruments physiques sur « étagère » (réels). Une application, développée sous LabVIEW, est donc appelée **Instrument Virtuel** (*Virtual Instrument* : VI).

Un programme ou VI, développé dans l'environnement LabVIEW, se compose principalement de deux éléments étroitement associés et regroupés sous le même nom « nom_application.vi » (l'extension **.vi** permet une reconnaissance immédiate par l'environnement LabVIEW). Ainsi nous avons :

- La Face-avant (*Front panel*) qui représente le panneau de contrôle de l'instrument virtuel composé d'objets variés (boutons, indicateurs, graphes, etc.). Cette fenêtre est l'**interface utilisateur** du programme au sens génie logiciel : définition des entrées/sorties de données accessibles par l'utilisateur du programme (figures 1.4, 1.5 et 1.18). En imitant l'interface classique des appareils de mesures (générateur, oscilloscope, etc.), LabVIEW apporte une continuité pour les utilisateurs des appareils d'instrumentation. La réalisation de cette interface peut être également effectuée de manière indépendante par rapport au programme.

Ainsi, le langage LabVIEW peut être utilisé aussi comme un outil d'aide à la conception ou un outil de prototypage. En lisant ce livre, vous pourriez avoir le sentiment que LabVIEW est un outil uniquement dédié à l'acquisition et à la mesure. Certes, il propose un environnement qui facilite grandement cela, et les auteurs de ce livre ont fait délibérément le choix de mettre cette facilité à l'honneur. Mais LabVIEW est un langage de développement à part entière qui peut vous permettre d'implémenter n'importe quel type d'application. Cette notion sera reprise en détail dans le chapitre 3.

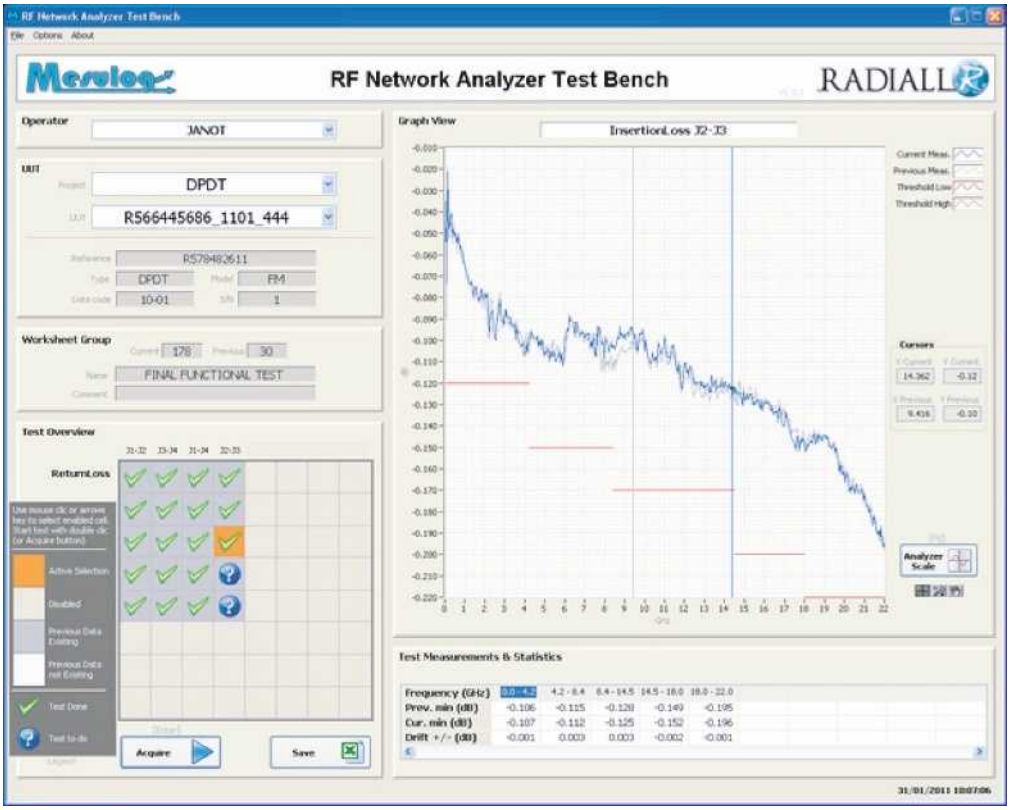

Figure 1.4 – Exemple de « Face-avant » ou interface utilisateur développée dans l'environnement LabVIEW : « pilotage d'un analyseur de réseau RF ».

- Le Diagramme (*diagram*) qui est le **programme** de l'application ou code source. Il écrit sous la forme d'un diagramme flux de données en langage G : ensemble des icônes et des liaisons entre ces icônes utilisées (figure 1.19). Le diagramme contient les fonctions de l'instrument virtuel. Contrairement à la procédure qui consistait pour le technicien ou le scientifique à dessiner le schéma d'une

application et ensuite à le convertir en un code propre au langage choisi, pour LabVIEW, le diagramme est le programme. Représenté en image, le programme s'explique de lui-même et est donc facile à adapter et à comprendre lorsque la représentation graphique garde une taille acceptable. Cette partie de l'application est ce que l'on appelle le code source par opposition à l'interface utilisateur.

Dans l'exemple de la figure 1.4, le logiciel sert à valider des commutateurs hyperfréquences pour lesquels un plan de test comprenant des étapes manuelles, automatiques et semi-automatiques est exécuté séquentiellement.

Dans l'exemple de la figure 1.5, LabVIEW permet l'automatisation d'un système de nanocaractérisation pour développer la recherche sur les matériaux au CEA. Le logiciel permet également une analyse déportée des résultats d'acquisition.

Ces notions seront reprises en détail dans les concepts de programmation LabVIEW dans le chapitre 3.

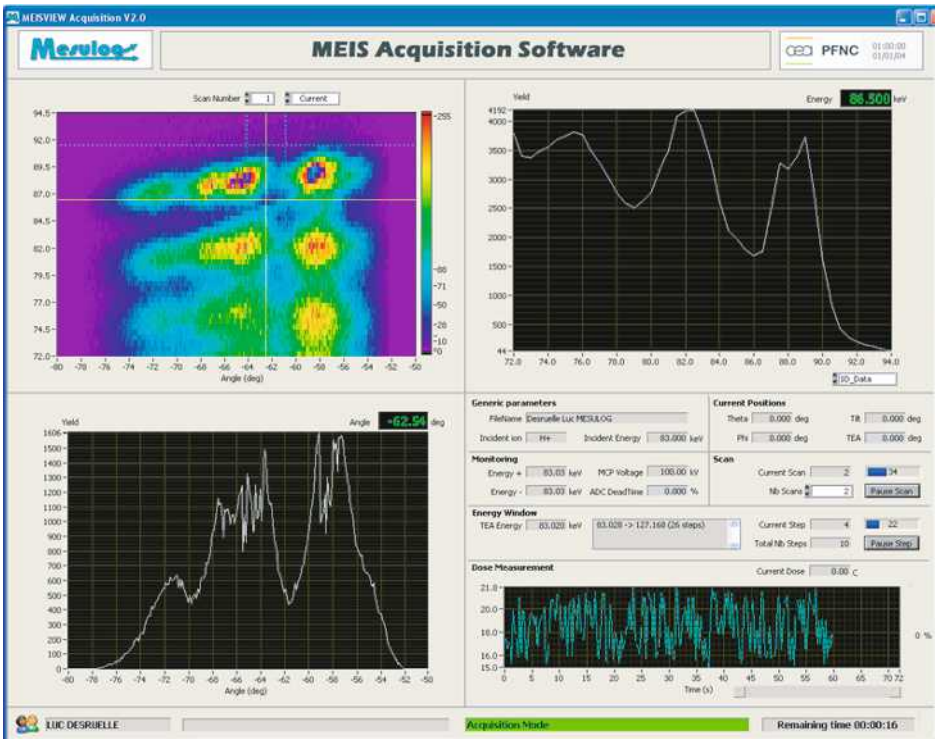

Figure 1.5 – Autre exemple de « Face-avant » : interface utilisateur pour la « nanocaractérisation » de matériaux.

Ces deux fenêtres, « face-avant » et « diagramme », bien que totalement disjointes au niveau de l'écran de l'ordinateur et de nature informatique très différente (interface utilisateur et programme), sont indissociables et constituent les deux aspects d'une seule et même fonction ou VI.

Remarque

En mode de développement, lors de la réalisation du code G, les deux fenêtres « face-avant » et « diagramme » peuvent être ouvertes par le développeur. En revanche, suite à la création d'un exécutable autonome, appelé environnement d'exécution ou Run-Time, seulement la « face-avant » du programme principal et des boîtes de dialogue seront visibles. Le code G du diagramme n'est plus visible. Ces notions seront reprises en détail dans le chapitre 3.

Une application LabVIEW ou VI est en fait un module logiciel que l'on peut soit exécuter, soit inclure dans une autre application. En effet, lorsqu'une partie de l'application est réalisée et testée, il est possible de créer un nouvel objet graphique ou une icône qui représente alors une fonction bien définie par son interface utilisateur (variables en entrée/sortie et fonctionnalité). Cette caractéristique montre que LabVIEW intègre le concept de programmation modulaire. Ainsi, une autre composante d'un VI est la présentation graphique de la hiérarchie (*VI Hierarchy*) de l'ensemble des modules utilisés, appelés aussi sous-VI (figure 1.6).

En résumé, une application développée dans l'environnement LabVIEW est caractérisée de façon complète avec les quatre éléments suivants : « Face-avant » ou interface utilisateur, « Diagramme » ou programme, structure hiérarchique des sous-VI utilisés et l'icône de l'application elle-même.

Remarque

Une application peut aussi être insérée dans un projet qui va regrouper plusieurs VI (Voir le chapitre 2).

1.2.2 Présentation générale de l'environnement

Démarrage de l'environnement LabVIEW

Après avoir lancé le démarrage du logiciel, apparaît l'écran présenté à la figure 1.7. La fenêtre est divisée en deux parties :

- Fichiers
- Ressources

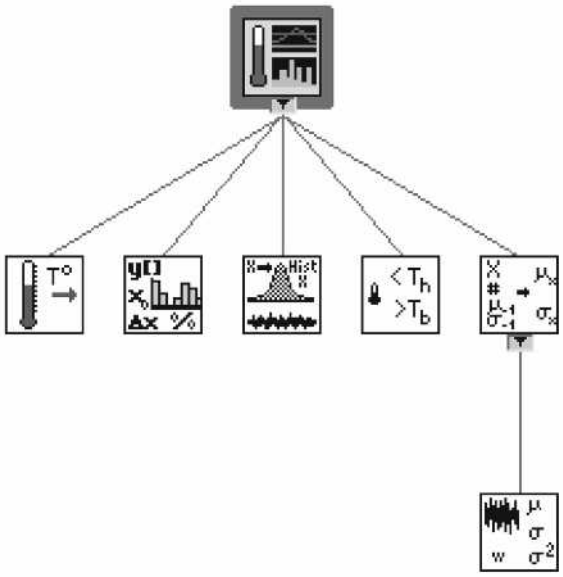

Figure 1.6 – Exemple de structure hiérarchique des modules utilisés pour une application développée dans l'environnement LabVIEW : « mesure et analyse d'une température ». L'icône positionnée en haut de la hiérarchie correspond à l'icône du VI développé.

Les fichiers peuvent être ceux d'un VI, d'un projet ou pris à partir d'un modèle. Ils sont soit nouveaux, soit déjà conçus et placés dans un répertoire.

Les ressources sont là pour aider le concepteur qu'il soit nouveau ou déjà habitué à des versions précédentes de LabVIEW.

Plusieurs rubriques sont présentées dans les ressources :

- **`Nouvel utilisateur de LabVIEW ?`** Le nouvel utilisateur est aidé de diverses manières, soit par une initiation consistant en un document disponible sur Adobe Reader, soit par un Tutorial, soit par un guide, soit par une aide.
- **`Mise à jour de LabVIEW ?`** Cette rubrique s'adresse plutôt à des utilisateurs déjà habitués à des anciennes versions de LabVIEW. Il est précisé les apports de cette version, en particulier pour la création et la mise au point des projets, des fusions de VI, une amélioration de l'utilisation des boucles FOR (voir *paragraphe 2.2.2*) et la liste des nouvelles fonctionnalités.
- **`Liens Internet`** sur la toile, l'utilisateur est relié à d'autres utilisateurs grâce au forum de discussion. Il peut suivre des cours de formation en ligne (en anglais ou en français) et même tester ses compétences dans l'apprentissage du logiciel.

Enfin, la « zone LabVIEW » lui donne accès à de nombreuses applications du logiciel.

- **`Exemples`** Il est conseillé à l'utilisateur de rechercher des exemples déjà disponibles avec l'installation du logiciel pour mieux comprendre son utilisation.

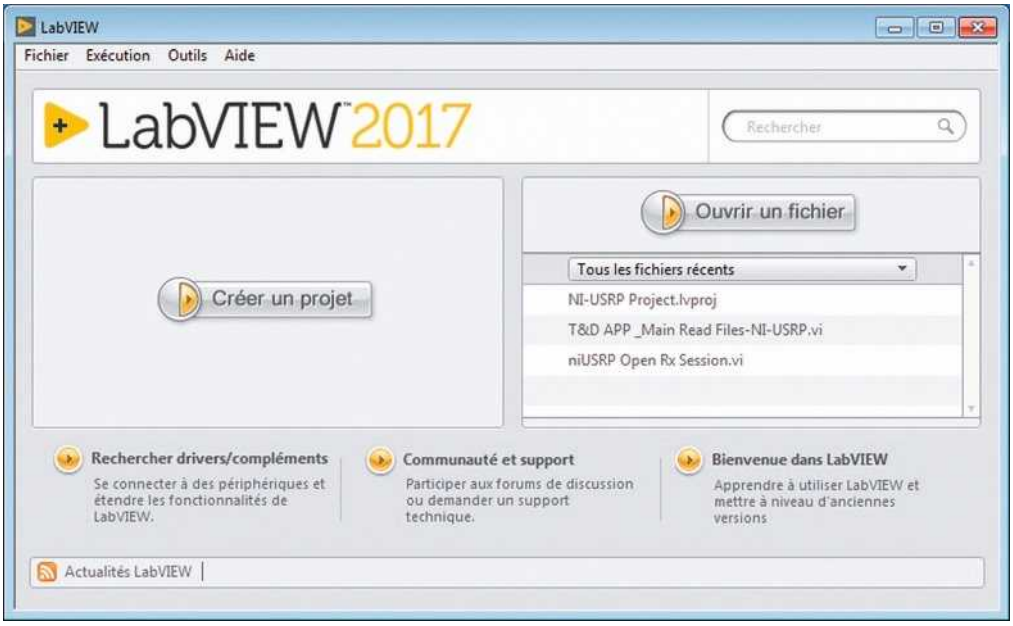

Figure 1.7 – Fenêtre de lancement de l'environnement LabVIEW.

Après avoir démarré le logiciel, apparaît l'écran présenté à la figure 1.7.

La fenêtre est divisée en deux parties :

- Fichiers (Nouveau ou à Ouvrir),
- Projets (Nouveau ou à Ouvrir).

D'autre part, diverses informations sont disponibles pour l'utilisateur :

- Pour trouver des Drivers,
- Pour communiquer avec des utilisateurs de LabVIEW,
- Il a aussi le LabVIEW News, qui est connecté à la communauté francophone du site de National Instruments, et affiche les dernières nouvelles.

Fenêtres « Face-avant » et « Diagramme »

Comme nous l'avons vu précédemment, le travail de développement avec LabVIEW s'effectue dans deux fenêtres différentes : interface utilisateur ou « Face-avant »

et programme ou « Diagramme ». Au lancement du logiciel LabVIEW, après avoir désactivé la fenêtre de démarrage ou en cliquant sur **`Nouveau VI`** de la fenêtre de démarrage, il apparaît une fenêtre active « Face-avant » vierge nommée **`Sans titre 1`** (*Untitled 1*) (figure 1.8). La deuxième fenêtre « Diagramme » apparaît cachée en arrière-plan et non active.

Intégrée au bord supérieur de cette fenêtre (***W***) ou externe (***M***), nous trouvons la barre des menus de LabVIEW qui est composée de sept menus. Les différentes fonctions associées à ces menus sont détaillées au fur et à mesure des besoins dans la suite de ce livre et dans l'*annexe*. Nous pouvons décrire brièvement les fonctions principales de ces différents menus :

- **`Fichier`** (*File*) : les fonctions accessibles par ce menu restent très classiques avec la gestion des fichiers LabVIEW (nouveau, ouvrir, enregistrer) et l'impression ;
- **`Édition`** (*Edit*) : ce menu est aussi classique et permet de faire les éditions de base des fenêtres LabVIEW (copier, coller, supprimer) ;
- **`Affichage`** (*Display*) : ce menu présente la palette des commandes, des outils de la fenêtre « Face-avant », des fonctions et celle des outils de la fenêtre « Diagramme ». Il présente par ailleurs des fenêtres « Hiérarchie du VI », « Hiérarchie de classes LabVIEW », Navigateur de classes, Navigateur de propriétés « ActiveX » (voir chapitres 5 et 6) ;
- **`Projet`** (*Projet*) : ce menu présente la même commande « Nouveau projet » et « Ouvrir un projet » sur la fenêtre « Face-avant » ou sur la fenêtre « Diagramme » (voir chapitre 6) ;
- **`Exécution`** (*Operate*) : ce menu permet de gérer l'exécution et le contexte d'exécution d'un VI, en particulier les données par défaut de la « Face-avant » ;
- **`Outils`** (*Tools*) : ce menu permet d'accéder à des assistants de haut niveau pour configurer les cartes d'entrées/sorties, choisir les drivers de ces cartes, gérer une variable partagée (voir chapitre 4), etc. ;
- **`Parcourir`** (*Browse*) : ce menu très spécifique de l'organisation hiérarchique des programmes LabVIEW permet de gérer les différents modules de cette architecture ;
- **`Fenêtre`** (*Window*) : ce menu permet d'activer ou de désactiver les nombreuses fenêtres utilisées dans l'environnement LabVIEW en phase de développement ;

- **Aide** (*Help*) : ce menu permet d'accéder à toute l'aide en ligne, du commentaire associé à une icône d'un programme au document « Manuels de référence de LabVIEW » au format PDF.

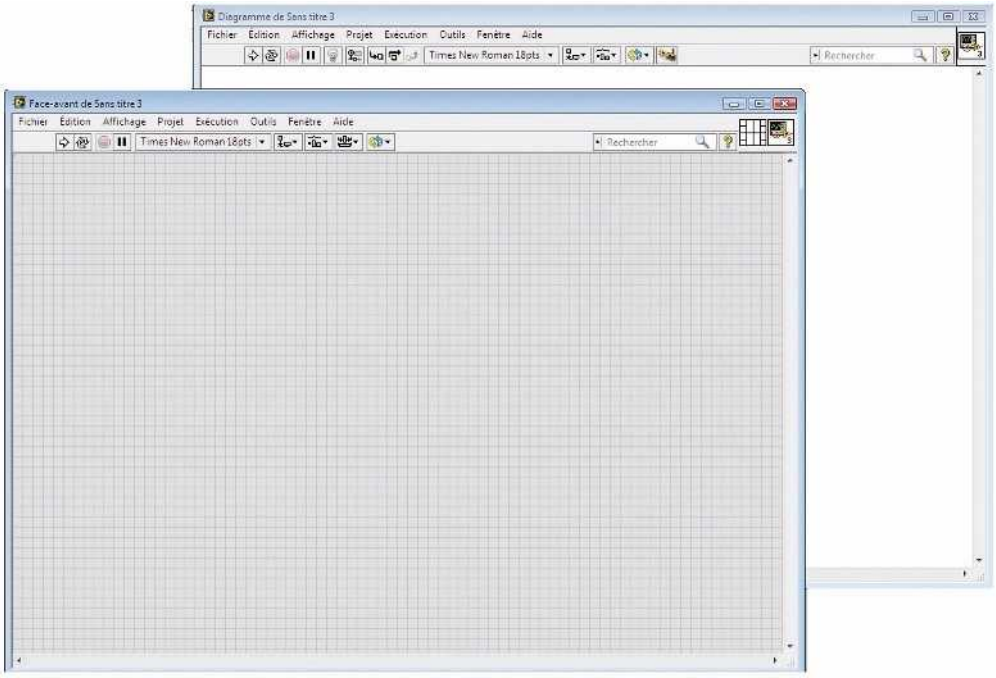

Figure 1.8 – Fenêtres « Face-avant » *(Front Panel)* et « Diagramme » *(Block Diagram)* après le lancement de l'environnement du logiciel LabVIEW avec la commande Nouveau VI.

Pour rendre la deuxième fenêtre « Diagramme » active, il suffit de cliquer sur la partie apparente ou de faire appel à l'action suivante :

Ainsi, une fenêtre « Diagramme » est activée avec le nom **Diagramme de Sans titre 1** (*Untitled 1 Diagram*). Pour travailler simultanément en

parallèle dans les deux fenêtres (visualisation dite mosaïque (*tile*)), ce qui est l'utilisation de développement la plus logique, on fait appel à la commande suivante du même menu :

Nous obtenons ainsi les deux fenêtres de travail qui peuvent être activées l'une après l'autre par un simple clic dans la fenêtre choisie (figure 1.9). L'ensemble de ces deux fenêtres constitue l'Instrument Virtuel en création. Nous pouvons immédiatement noter, en dessous de la barre des menus décrite ci-avant, la présence d'une barre de boutons, et en particulier, quatre boutons semblables dans les deux fenêtres « Face-avant » et « Diagramme » de LabVIEW destinés à la gestion de l'exécution d'un VI (figure 1.10).

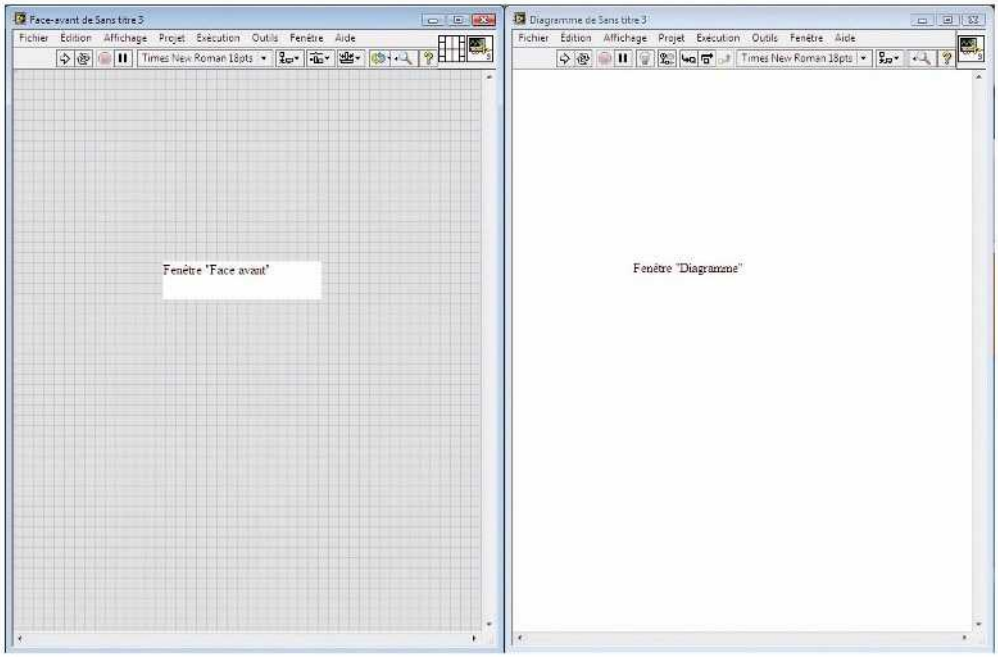

Figure 1.9 – Les deux fenêtres en mode mosaïque du logiciel LabVIEW avec la commande Nouveau VI.

Attention

La fermeture de la fenêtre « Face-avant » entraîne également la fermeture de la fonction, ou VI, en cours de développement alors que la fenêtre « Diagramme » peut être fermée même lors de l'exécution de la fonction.

Pour réaliser le développement d'un VI LabVIEW au sein des deux fenêtres « Face-avant » et « Diagramme », nous avons besoin des trois palettes ou menus contextuels suivants :

- palette **`Outils`** (*Tools*) : le curseur de la souris peut avoir différents effets selon la présentation choisie dans cette palette (sélection, écriture, câblage, coloration, etc.) ;
- palette **`Commandes`** (*Controls*) : cette palette, accessible uniquement dans la fenêtre « Face-avant », permet de créer cette interface utilisateur avec l'ensemble des objets représentant les entrées/sorties du programme ;
- palette **`Fonctions`** (*Functions*) : cette palette, accessible uniquement dans la fenêtre « Diagramme », offre l'ensemble des « instructions » du langage graphique LabVIEW du plus bas niveau (opérateurs mathématiques) au plus haut niveau (gestionnaire de cartes d'entrées/sorties).

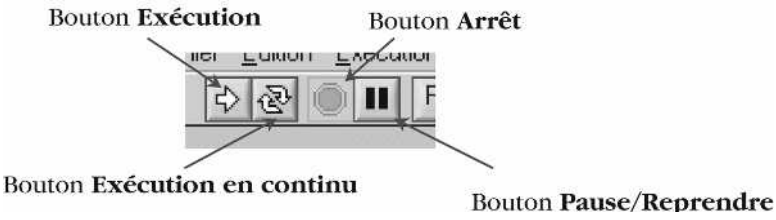

Figure 1.10 – Boutons de gestion de l'exécution du VI dans les deux fenêtres « Face-avant » et « Diagramme » de LabVIEW.

Remarque

L'affichage de ces différentes palettes est obtenu à l'aide du menu `Fenêtre`. Mais, pour les deux dernières palettes associées à une fenêtre spécifique, elles peuvent apparaître momentanément (accès contextuel) par un clic du bouton droit de la souris.

Pour rendre permanent cet accès contextuel, il suffit de faire glisser le curseur de la souris vers la « punaise » située en haut à gauche de cette fenêtre. Cette opération est possible pour toutes les fenêtres ou palettes possédant cette représentation « punaise ».

Palette « Outils »

La palette **Outils** est accessible à partir du menu « Affichage » existant aussi bien dans la fenêtre « Face-avant » que dans la fenêtre « Diagramme » (figure 1.11).

Un outil est un mode de fonctionnement spécifique associé au curseur de la souris. Les différents outils sont disponibles dans la palette flottante des outils qu'il est possible d'afficher en faisant :

Les outils sont les suivants (figure 1.11) :

- outil **Doigt** (*Operating tool*) : dans la phase d'édition, il permet de changer la valeur d'une commande ou de sélectionner et modifier les textes associés à un objet de la « Face-avant ». Lorsque cet outil passe dans les zones d'écriture des variables (fenêtre « Face-avant ») ou des constantes (fenêtre « Diagramme ») d'entrée, il change d'aspect pour se transformer en un curseur de type traitement de texte. Dans la phase d'exécution, cet outil est le seul outil accessible et permet de gérer complètement l'exécution de l'application en agissant sur les objets entrées de la « Face-avant » de cette application ;
- outil **Flèche** (*Positioning tool*) : il permet de positionner, de dimensionner et de sélectionner les objets utilisés dans les deux fenêtres « Face-avant » et « Diagramme ». Pour la fonction de redimensionnement des objets, l'outil change d'aspect lorsqu'il passe sur l'un des coins des objets. Cet outil peut aussi être utilisé pour la duplication des objets créés dans les deux fenêtres comme nous le verrons dans le chapitre 2.
- outil **Texte** (*Labeling tool*) : il permet soit d'éditer des textes libres (amélioration de la présentation ou commentaires de programme), soit d'écrire des textes dans les zones prévues à cet effet dans les différents objets des deux fenêtres de travail. Dans les deux cas, l'aspect du curseur est différent : pour le texte libre et pour le texte dans les zones des objets avec un curseur de type traitement de texte ;
- outil **Bobine** (*Wiring tool*) : il permet de relier les objets par des fils pour créer les flux de données dans la fenêtre « Diagramme » et d'affecter les objets commandes aux cases du connecteur lors de la création de programmes encapsulés ;

- outil **`Menu local`** (*Object Shortcut Menu tool*) : il permet d'ouvrir le menu local d'un objet pour en modifier les caractéristiques ;
- outil **`Main`** (*Scroll tool*) : il permet de déplacer la zone de visualisation de la fenêtre sans utiliser les barres de défilement ;
- outil **`Point d'arrêt`** (*Breakpoint tool*) : il permet de placer des points d'arrêt dans les programmes des VI ;
- outil **`Sonde`** (*Probe tool*) : il permet de poser des sondes ou afficheurs (valeurs des variables à un instant donné) sur les fils du diagramme flux de données ;
- outil **`Pipette`** (*Color Copy tool*) : il permet de copier une couleur pour ensuite l'appliquer avec l'outil Pinceau ;
- outil **`Pinceau`** (*Color tool*) : il permet d'appliquer une couleur à une zone ou à un objet dans les deux fenêtres de travail pour améliorer la lisibilité, en particulier l'apparence de l'interface utilisateur. Il est possible de définir une couleur de fond (*background*) et une couleur de premier plan (*foreground*) qui sont affectées aux objets créés. Ces couleurs sont alors affichées à côté de l'outil de coloration (figure 1.11).
- Outil **`Automatique`** : c'est la LED, qui, si elle est allumée (verte), permet une sélection automatique de l'outil selon la position de la souris sur une icône. **C'est l'outil le plus utile du développeur**.

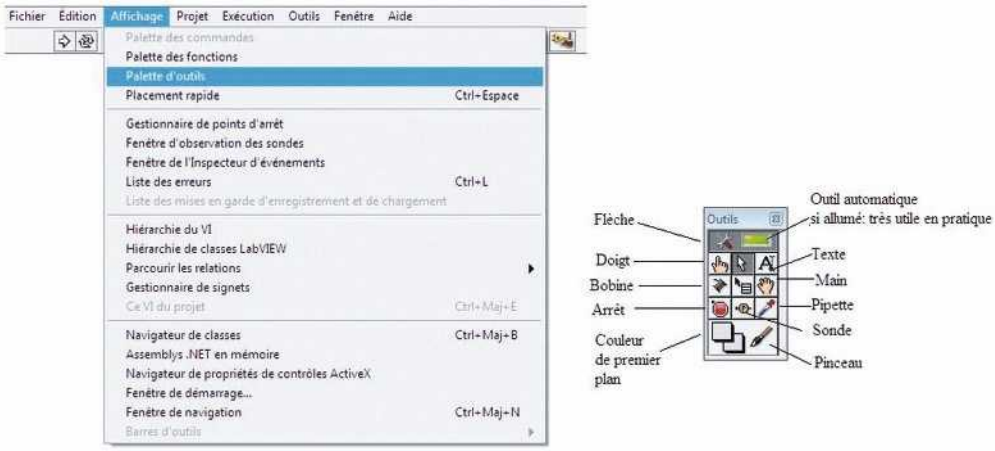

Figure 1.11 – Fenêtre de sélection des différents outils de LabVIEW.

Remarque

Pour accéder plus rapidement à ces fonctionnements spécifiques liés au curseur de la souris, il est important de noter qu'il est possible de passer de façon circulaire à quatre fonctions principales à l'aide de la touche de tabulation (») :

- soit dans la fenêtre « Face-avant » : Doigt » Flèche » Texte » Pinceau ;
- soit dans la fenêtre « Diagramme » : Doigt » Flèche » Texte » Bobine ;
- soit dans la fenêtre « Diagramme » en mode exécution : Doigt » Sonde » Point d'arrêt.

Palette Commande de la fenêtre « Face-avant»

Cette palette **commandes** (*Controls*) est uniquement accessible dans la fenêtre Face-avant. Elle permet d'accéder à l'ensemble des objets représentant les entrées/sorties du programme. L'affichage permanent de cette palette peut se faire par l'accès menu suivant (figure 1.12 (a)) :

Figure 1.12 (a) – Accès à la palette des commandes.

Les différentes catégories d'objets accessibles à l'aide de cette palette sont présentées sur la figure 1.12 (b). Ces différents objets associés à cette palette sont détaillés au fur et à mesure des besoins dans la suite de ce livre. Nous pouvons décrire brièvement ces types d'objets en parcourant la palette de gauche à droite et de haut en bas :

- **Numérique** (*Numeric*) : objets pour définir des entrées ou des sorties de données de type numérique entier ou réel ;
- **Booléen** (*Boolean*) : objets pour définir des entrées ou des sorties de données de type booléen ;
- **Chaîne & chemin** (*String & Path*) : objets pour définir des entrées ou des sorties de données de type chaîne de caractères ou des chemins d'accès à des fichiers (chaîne de caractères formatée en utilisant la syntaxe de la plate-forme d'exécution) ;
- **Tableau & cluster** (*Array & Cluster*) : objets pour définir des entrées ou des sorties de données de type tableau (ensemble de données homogènes en type) ou de type cluster (ensemble de données hétérogènes ou de type différent) ;
- **Liste & Table** (*List & table*) : objets pour définir des entrées ou des sorties de données homogènes en type, structurées et représentées en liste ou table ;

- **`Graphe`** (*Graph*) : objets pour définir uniquement des sorties de données sous forme de courbes à deux ou trois dimensions ;
- **`Menu déroulant & Enum`** (*Ring & Enum*) : objets pour définir des entrées ou des sorties de données homogènes présentées sous la forme de menu déroulant ;
- **`E/S`** (*I/O*) : objets pour définir des entrées ou des sorties faisant référence à des données de type « mesures » spécifiques à l'environnement LabVIEW (Waveform, numéro de voie DAQ, noms de ressource VISA, session IMAQ, etc.) ; liens entre composants DSP (*Digital Signal Processeur*) et l'ordinateur grâce à un menu spécifique du logiciel ; développement d'applications avec des cartes FPGA ; cartes d'acquisition et de commande grâce au port USB (voir chapitre 4) ;

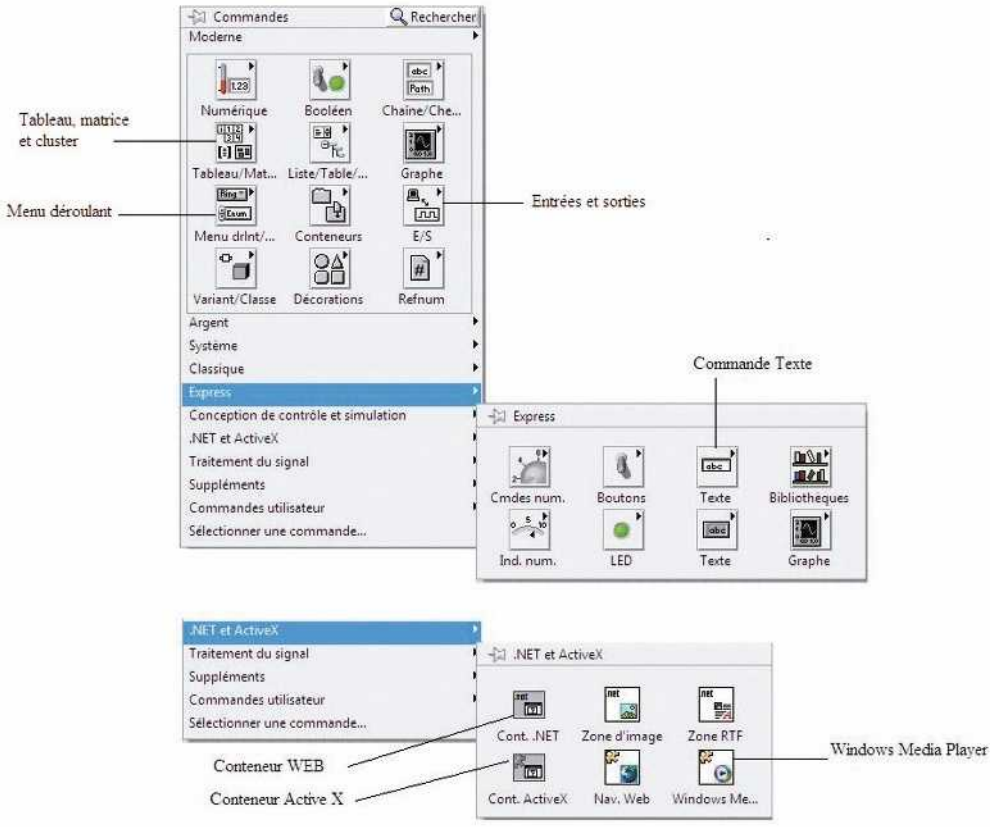

Figure 1.12 (b) – Palette des commandes disponibles dans la fenêtre « Face avant » d'un VI.

- **Décorations** (*Decorations*) : objets graphiques passifs pour améliorer la présentation de la « Face-avant » (zones rectangulaires avec effet 3D et flèches) ;
- **Refnum** (*Refnum*) : objets pour définir des entrées ou des sorties de référence unique à un objet tel qu'un fichier, un périphérique ou à une connexion réseau ;
- **commande numérique** (*Numeric control*) : commande permettant de fixer des valeurs numériques pour commander des systèmes ;
- **commande texte** (*Text control*) : commande d'un texte placé sur la fenêtre « Face-avant » ;
- **Indicateur texte** (*Text informer*) : indication d'un texte placé sur la fenêtre « Face-avant » ;
- **Boutons et commutateurs** (*Boolean control*) : commande en « tout ou rien » ;
- **ActiveX** (*ActiveX*) : sous-menu qui permet l'incorporation d'objets ActiveX dans la Face-avant d'un VI (n'est possible que sous Windows) ;
- **Navigateur WEB** (*WEB Navigator*) : le logiciel LabVIEW offre de nombreuses possibilités d'applications nouvelles grâce à Internet.

Il est évident que nous allons trouver dans les instructions de la palette `Fonctions` de la fenêtre Diagramme (*paragraphe suivant*), toutes les fonctions nécessaires aux traitements des différents types de données produites ou consommées par ces objets d'entrées/sorties.

Palette Fonctions de la fenêtre « Diagramme »

Cette palette **Fonctions** (*Functions*) est uniquement accessible dans la fenêtre Diagramme. Elle permet d'accéder à l'ensemble des objets représentant les instructions ou fonctions du langage permettant de constituer le programme LabVIEW. L'affichage permanent de cette palette peut se faire par l'accès menu suivant (figure 1.13 (a)) :

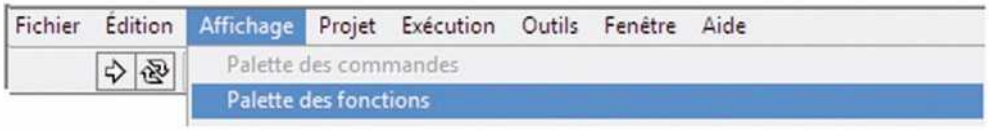

Figure 1.13 (a) – Accès à la palette des fonctions.

Les différentes catégories de fonctions ou instructions accessibles à l'aide de cette palette sont présentées sur la figure 1.13 (b). Ces différentes fonctions associées à cette palette sont détaillées au fur et à mesure des besoins dans la suite de ce livre.

Nous pouvons décrire brièvement ces types de fonctions en parcourant la palette de gauche à droite et de haut en bas :

- **Structures** (*Structures*) : structures de programmation, constantes numériques, variables locales et globales ;
- **Numérique** (*Numeric*) : fonctions de traitement des données de type numérique ;
- **Booléen** (*Boolean*) : fonctions de traitement des données de type booléen ;
- **Chaîne** (*String*) : fonctions de traitement des données de type chaîne de caractères ;
- **Tableau** (*Array*) : fonctions de traitement des données de type tableau (ensemble de données homogènes en type) ;
- **Cluster** (*Cluster*) : fonctions de traitement des données de type cluster (ensemble de données hétérogènes en type) ;
- **Comparaison** (*Comparison*) : fonctions de comparaisons et de tests ;
- **Synchronisation** (*Synchronization*) : assure la synchronisation des tâches s'exécutant en parallèle et pour la transmission des données en parallèle ;
- **Informations temporelles** (*Time Information*) : on peut ainsi fixer la vitesse d'exécution des VI et recueillir les informations date et heure venant de l'ordinateur ;
- **Traitement du signal** (*Signal Processing*) : les fonctions principales intervenant en traitement du signal sont disponibles (figure 1.14) ;
- **E/S sur fichiers** (*File I/O*) : fonctions de gestion des fichiers (format tableur, format texte, format binaire) ;
- **Waveform** (*Waveform*) : associées à un nouveau type de données sous forme d'un cluster intégrant (mesures, t_0 et Dt), ces fonctions spécifiques de ce type sont identiques à beaucoup de fonctions disponibles dans les sous-menus précédents (opérations mathématiques, traitements, etc.) ;
- **E/S d'instruments** (*Instrument I/O*) : gestionnaires d'instruments (liste très importante livrée sous forme d'un CD-ROM spécifique avec l'environnement LabVIEW), gestion du bus GPIB 488, de la liaison série et gestion d'instruments avec le « langage » VISA de pilotage de différents types d'instruments (oscilloscope, multimètre, etc.) ;
- **Mathématiques** (*Mathematics*) : fonctions mathématiques avancées (lissages et interpolations, probabilités et statistiques, opérations sur les tableaux, opérations sur les matrices, etc.), optimisation, détermination de zéros de fonctions, etc. ;

Figure 1.13 (b) – Palette de programmation des fonctions disponibles dans la fenêtre « Diagramme ».

- **`Contrôle d'applications`** (*Application Control*) : fonctions permettant de contrôler l'application en mode exécution (redéfinition de ses propriétés, arrêt, appel de l'aide en ligne, impression de la face-avant, etc.) ;
- **`Graphisme et son`** (*Graphics & Sound*) : fonctions de traitements de graphiques (tracés d'objets géométriques, modifications des formats de tracés, sauvegarde du graphe dans un format spécifique de type image, etc.) et fonctions de gestion du son ;

- **`Génération de rapport`** (*Report Generation*) : fonctions permettant de créer un rapport en cours d'exécution d'un VI (création de texte, insertion d'image, etc.).

À la figure 1.14, on présente plus en détail les fonctions accessibles grâce au logiciel soit pour déterminer les entrées/sorties, soit pour définir des fonctions mathématiques, soit pour effectuer un traitement du signal.

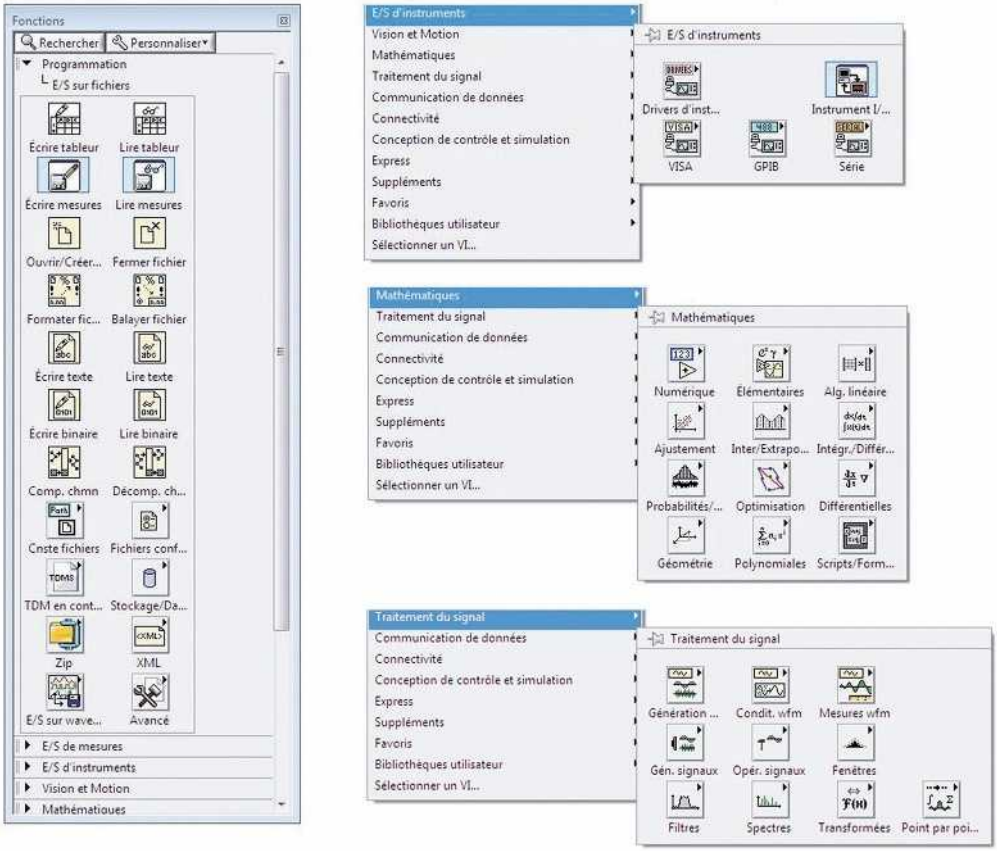

Figure 1.14 – Détails des fonctions disponibles en E/S sur fichiers, en E/S d'instruments, en mathématiques et en traitement du signal.

Boutons de gestion des palettes Commandes et Fonctions

Lorsqu'on accède aux palettes **`Commandes`** et **`Fonctions`** de façon contextuelle (apparition momentanée et localisée à l'endroit du clic souris sur l'écran), il est possible que, pour atteindre la fonction recherchée, de nombreuses palettes

spécifiques se déroulent les unes à côté des autres (figure 1.15). Cette procédure d'accès à une commande ou à une fonction peut être assez lourde et délicate à gérer (maintien ou non du bouton de la souris). Il est donc plus facile de travailler à l'édition du programme en rendant ces fenêtres accessibles en permanence (utilisation de la punaise ou du menu **Fenêtre**).

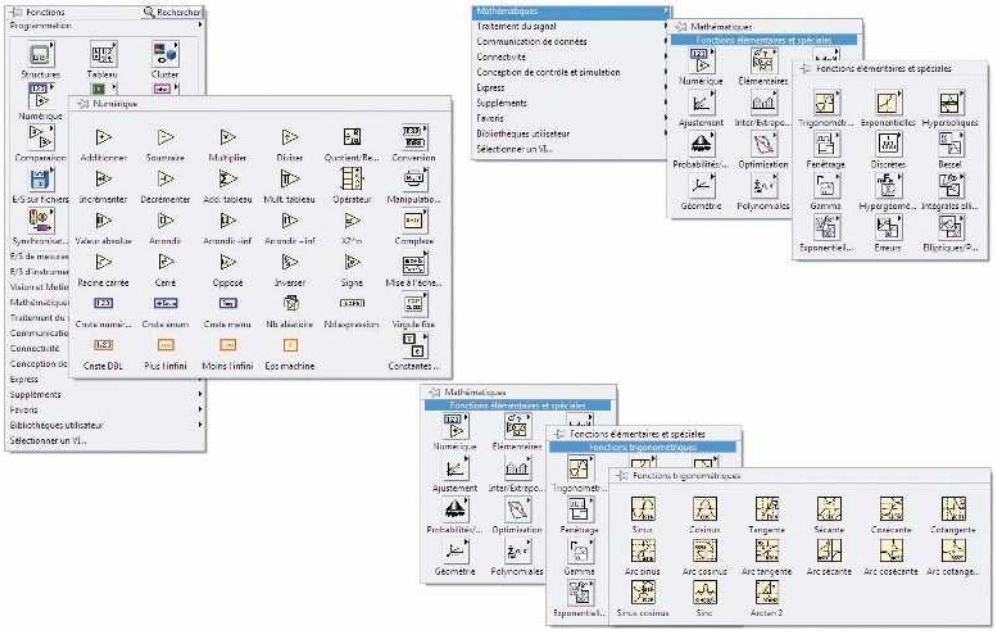

Figure 1.15 – Accès aux fonctions numériques et mathématiques avec des ouvertures de palettes successives.

Remarque

Il est possible d'avoir à l'écran plusieurs fenêtres de fonctions ouvertes simultanément offrant ainsi un champ des fonctions à la fois plus spécifique et plus large.

Par exemple, si on désire développer un programme de traitements mathématiques sur des données, les fenêtres `Numérique`, `Tableau` et `Mathématiques` peuvent être disponibles en même temps.

Dans ce contexte, les palettes spécifiques se substituent les unes aux autres lors des choix successifs des éléments désirés comme la palette **Fonction** suivie de la palette **Numérique**, ou bien la palette **Mathématiques**, puis la palette **Fonctions élémentaires et spéciales**, puis la palette **Fonctions**

trigonométriques. Par contre, dans le cas d'un choix erroné ou du besoin d'un autre élément, la palette initiale n'est plus visible. Mais ces palettes à affichage permanent disposent de deux boutons de gestion qui sont très efficaces pour résoudre ce problème (figure 1.16) ;

- **Affichage et personnalisation de l'affichage** (*Affichage*) : il est ainsi possible de définir les présentations des icônes souhaitées par l'utilisateur ;

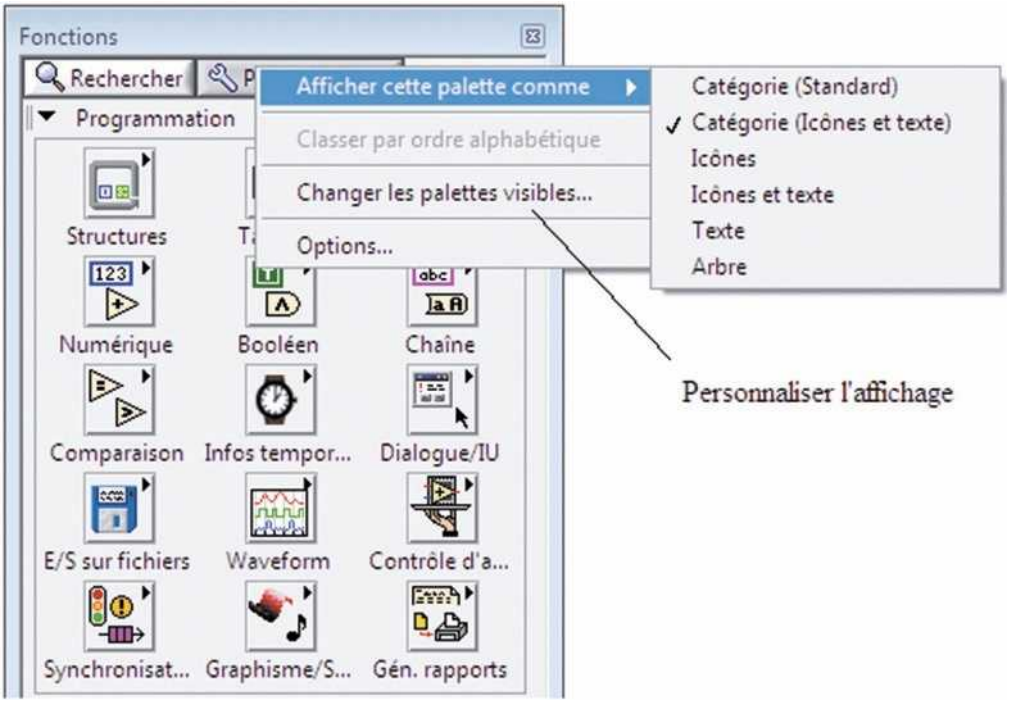

Figure 1.16 – Permet de modifier l'apparence de la palette des fonctions dans le diagramme. Sera repris en détail dans le chapitre 3.

- bouton **Recherche** (*Search*) : il permet de trouver très rapidement la palette contenant l'élément recherché à partir de son nom ou d'un mot contenu dans son nom. Cette recherche, définie par défaut pour l'ensemble des éléments commandes et fonctions, peut être restreinte soit à l'ensemble des commandes, soit à l'ensemble des fonctions (figure 1.17).

D'autre part, certaines de ces palettes, y compris les palettes initiales **Commandes** ou **Fonctions**, sont trop complètes pour le développement d'une application spécifique. Il est possible de réduire les éléments accessibles d'une palette donnée à partir du bouton appelé **Options** (chapitre 3).

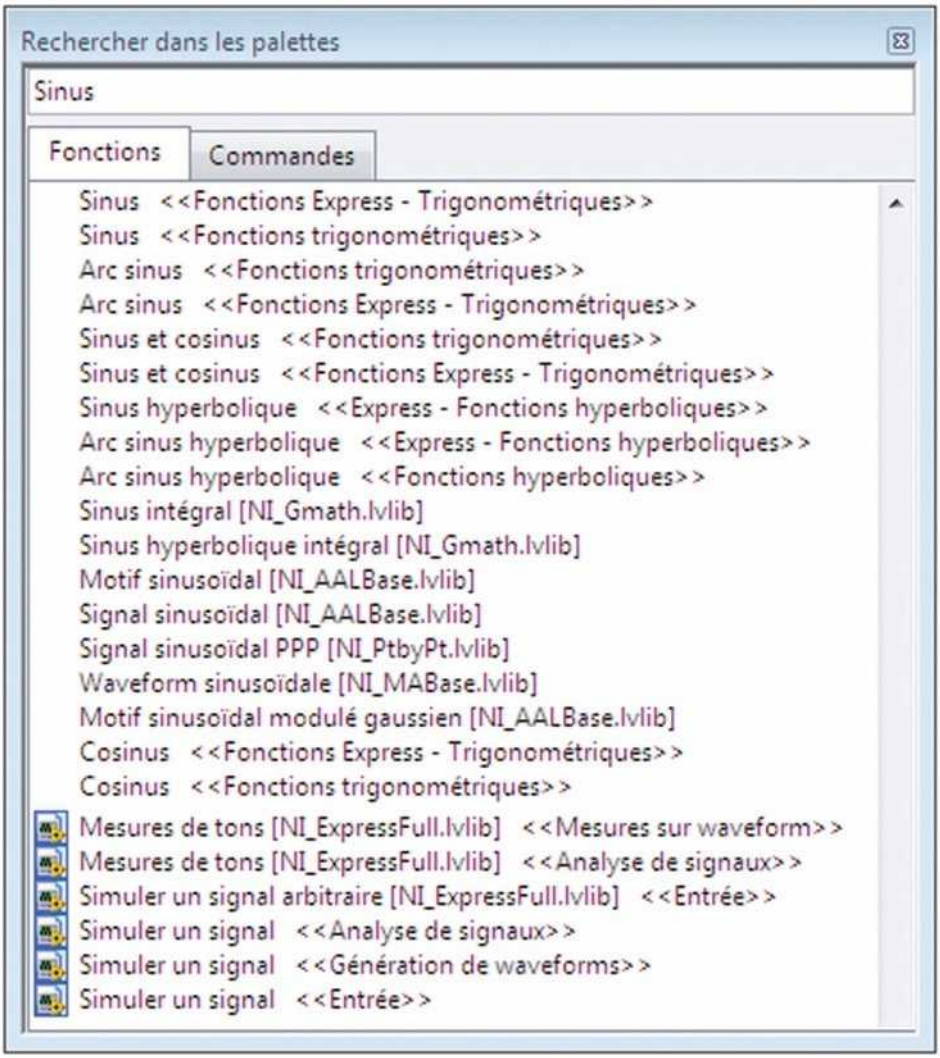

Figure 1.17 – Fenêtre de recherche d'un élément dans les palettes Commandes et Fonctions.

Remarque

Ajoutons la fonction de recherche la plus utilisée par les développeurs, la fonctionnalité appelée « quick drop » ou « placement rapide », qui facilite le travail du développeur, accessible via « ctrl + espace ». Beaucoup plus utilisé que la fenêtre présentée en figure 1.17.