1

Le nouveau Windows Runtime

WinRT (*Windows Runtime*) est un framework totalement nouveau pour Windows. Par son biais, les développeurs disposent d'une API multilangage pour créer des applications destinées à Windows 8. Les applications Windows Store sont des applications plein écran conçues pour des appareils spécifiques, des interactions tactiles et la nouvelle interface utilisateur Windows 8. Elles sont également appelées applications *ajustées* (*tailored*) car elles s'adaptent à l'appareil cible. Toutefois, rien n'interdit de développer des applications de bureau traditionnelles pour Windows 8. Dans cet ouvrage, l'expression "application Windows 8" désignera la version Windows Store (non de bureau) d'une application qui se fonde sur WinRT. L'arrivée de ce framework représente un changement profond dans le monde du développement pour Windows, aussi important que celui lié à la sortie de .NET à la fin des années 2000.

Dans ce chapitre, nous allons revenir sur les frameworks de développement pour Windows et expliquer pourquoi la popularité croissante des interfaces utilisateur naturelles a conduit Microsoft à proposer l'audacieuse plateforme Windows 8. Nous vous initierons aux applications Windows 8 et présenterons les différents langages dans lesquels elles peuvent être écrites. Vous découvrirez également comment les technologies fondées sur XAML, comme WPF et Silverlight, s'insèrent dans le nouveau Windows Runtime.

Retour en arrière : Win32 et .NET

> Je change d'avis plus rapidement que Bill Clinton.
>
> – Jay Leno, à propos de la nouvelle barre des tâches de Windows, lors de la soirée de lancement de Windows 95

En 1985, la première version de Windows est sortie sans grand tumulte. Il s'agissait non pas d'un système d'exploitation en soi, mais d'une couche appelée MS-DOS Executive (voir Figure 1.1) qui s'exécutait au-dessus du système en

ligne de commande MS-DOS. Une dizaine d'années plus tard, les choses ont énormément évolué avec l'arrivée de Windows 95. Bill Gates était alors monté sur une scène immense, devant le désormais emblématique "bouton Démarrer" de Windows, accueilli par Jay Leno[1], afin de faire la démonstration de la puissance du nouveau système d'exploitation. Il a été présenté sous les notes du morceau *Start Me Up* des Rolling Stones et a laissé ses principaux concurrents, comme Apple, loin derrière.

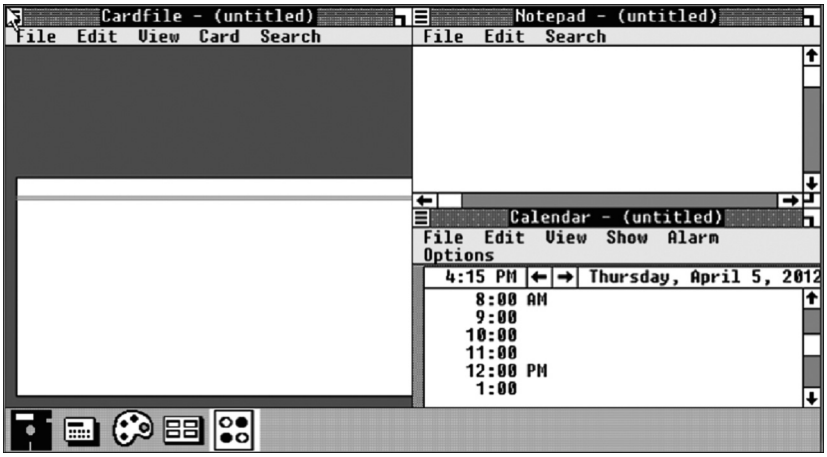

Figure 1.1
MS-DOS Executive.

Pour écrire des logiciels destinés à Windows 95 (voir Figure 1.2), les développeurs employaient *Win32*, une interface de programmation d'applications (API, *Application Programming Interface*) qui avait été conçue plusieurs années auparavant. À cette époque, Microsoft était en train de passer des anciens systèmes 16 bits aux nouvelles architectures 32 bits et leur prise en charge transparaissait dans le nom des API. Win32 se trouve encore aujourd'hui (sous le nom plus approprié d'API Windows) au cœur de tous les systèmes d'exploitation Windows, malgré l'arrivée de frameworks et de plateformes qui rendent son existence plus abstraite. Cette API était généralement considérée comme extrêmement puissante et souple au moment de sa sortie, mais elle imposait aux développeurs un important travail de prise en charge des opérations de bas niveau nécessaires à l'affichage d'un formulaire et aux interactions avec l'utilisateur.

1. N.d.T. : Jay Leno est un humoriste satirique américain qui officie sur la chaîne NBC dans son propre talk-show (source Wikipédia).

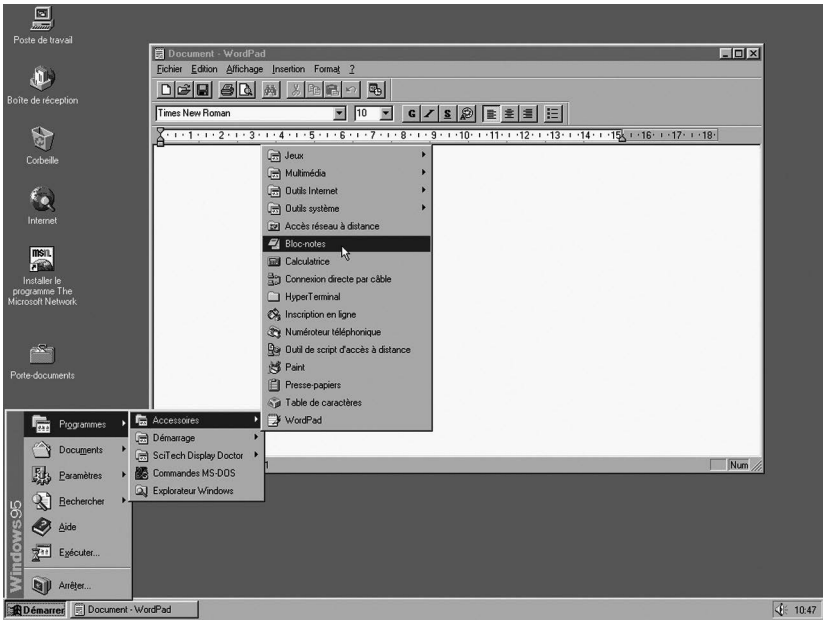

Figure 1.2
Windows 95.

Pour afficher le texte "Bonjour tout le monde !" en C++, le code suivant suffit :

```
#include<iostream.h>
int main()
{
    cout << "Bonjour tout le monde !" << endl;
    return 0;
}
```

Comparez-le à celui du Listing 1.1 qui réalise la même opération en C++ pour l'API Win32 ; cet exemple se fonde sur la bibliothèque de classes Microsoft Foundation (MFC, *Microsoft Foundation Class*).

Listing 1.1 : Le programme "Bonjour tout le monde" en version MFC/Win32

```
#include <afxwin.h>
class HelloApplication : public CWinApp
{
public:
 virtual BOOL InitInstance();
};

HelloApplication HelloApp;

class HelloWindow : public CFrameWnd
{
```

```
 CButton* m_pHelloButton;
public:
 HelloWindow();
};

BOOL HelloApplication::InitInstance()
{
 m_pMainWnd = new HelloWindow();
 m_pMainWnd->ShowWindow(m_nCmdShow);
 m_pMainWnd->UpdateWindow();
 return TRUE;
}

HelloWindow::HelloWindow()
{
 Create(NULL,
  "Bonjour tout le monde !",
  WS_OVERLAPPEDWINDOW|WS_HSCROLL,
  CRect(0,0,140,80));
 m_pHelloButton = new CButton();
 m_pHelloButton->Create("Bonjour tout le monde !",WS_CHILD|WS_
VISIBLE,CRect(20,
 20,120,40),this,1);
}
```

Cette API a été abondamment employée pendant des décennies pour écrire des logiciels Windows. Elle illustre la contrainte récurrente dans leur processus de développement : la possibilité de proposer une interface utilisateur élaborée implique un apprentissage ardu. Par ailleurs, il était obligatoire de maîtriser le langage C ou C++ car, bien que d'autres choix fussent possibles, ils dominaient tous deux le marché de ce que l'on nomme désormais du *code non géré*. Un code non géré est compilé directement en instructions natives que la machine cible est capable d'exécuter.

La première version du langage de programmation Visual Basic (plus fréquemment appelé VB) a été proposée en 1991. VB se fonde sur des concepts tirés d'un langage plus ancien, conçu pour l'enseignement et appelé BASIC, un acronyme de *Beginner's All-purpose Symbolic Instruction Code*. VB permet aux développeurs de structurer leur code en composants logiques qui coopèrent. Bien qu'il génère également du code non géré natif, aucune bibliothèque d'exécution complémentaire n'est requise pour créer l'application cible.

VB 4.0, sorti en 1995, est capable de produire des programmes Windows en version 16 bits et 32 bits. Son environnement de développement intégré (IDE, *Interactive Development Environment*), ainsi que l'existence de nombreux composants complémentaires et la possibilité de concevoir des interfaces utilisateur par simple glisser-déposer de contrôles, l'a rendu extrêmement populaire auprès des développeurs. VB permet également d'exploiter la technologie COM (*Common Object Model*), présentée en 1993 par Microsoft, qui permet de créer des composants logiciels.

COM autorise les communications entre les processus et permet de créer dynamiquement des composants logiciels (appelés objets) auxquels il est possible d'accéder depuis différents langages de programmation. Les développeurs peuvent utiliser des composants sans en connaître l'implémentation interne. La solution se fonde sur des interfaces externes. COM a également représenté un jalon important dans l'évolution de Windows et, comme vous le verrez plus loin, a aujourd'hui une influence sur la plateforme Windows 8.

L'architecture classique d'une application Win32 est illustrée à la Figure 1.3.

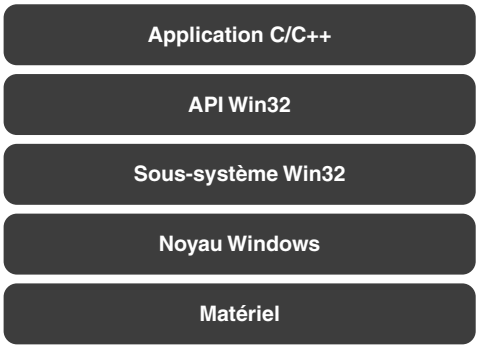

Figure 1.3
Architecture d'une application fondée sur l'API Win32.

Bien que l'API Win32 soit toujours présente, de nombreux développeurs n'ont pas conscience de son existence car, au milieu des années 1990, Microsoft a commencé à travailler sur un nouveau framework, nom de code NGSW (*Next Generation Windows Services*). La version bêta de ce framework est sortie à la fin des années 2000 sous son nom officiel : .NET 1.0. Il permet de créer un nouveau type de code appelé *code géré*.

Pour développer des applications, l'approche traditionnelle est dite non gérée car le code est compilé directement en instructions natives que la machine cible est capable de comprendre. Bien qu'elle présente plusieurs avantages, cette solution oblige le développeur à comprendre le fonctionnement de la machine hôte à un niveau très proche du noyau sous-jacent. Il doit maîtriser l'allocation explicite des zones de mémoire, ainsi que leur libération lorsqu'elles ne sont plus utiles. La mise en œuvre des opérations graphiques exige une compréhension des pilotes graphiques et du rendu des pixels dans les tampons internes utilisés pour l'affichage à l'écran.

Le framework .NET a apporté le code géré sur les plateformes Windows. Le code est dit géré car une nouvelle couche, nommée CLR (*Common Language Runtime*), a été ajoutée pour gérer la mémoire à la place du développeur, pour fournir une solution homogène aux interactions avec les diverses bibliothèques et ressources, et pour obtenir un code plus sûr. CLR apporte également une couche de code indépendant du langage, appelé MSIL (*Microsoft Intermediate Language*), ou plus simplement IL,

dans lequel laquelle tous les programmes sont compilés. Le moteur d'exécution interprète ce code et le convertit en instructions natives adaptées à la machine cible.

Le concept de code géré date d'une époque antérieure au framework .NET. Dans les années 1960 et 1970, lorsque les mainframes étaient répandus, il n'était pas rare d'écrire des émulateurs afin de porter du code d'un système vers un autre. L'une des premières sociétés de développement de jeux, créée en 1979, avait produit une "fiction interactive", plus connue sous son nom, *Zork*, et plusieurs titres compatibles avec différentes plateformes, notamment les ordinateurs Commodore 64 et Apple IIe, grâce à un interpréteur Z-machine. Les technologies Java sont apparues en 1991, avec une machine virtuelle Java (JVM, *Java Virtual Machine*) pour gérer le code écrit dans ce langage de programmation.

Ce modèle a pour avantage d'ouvrir la plateforme à divers langages et d'élargir la compatibilité des logiciels avec différentes plateformes et différents systèmes d'exploitation. Les programmes développés avec du code non géré ciblent souvent une version précise du système d'exploitation ou disposent de bibliothèques conçues pour assurer leur compatibilité avec plusieurs variantes (comme Windows XP, Windows Vista et Windows 7). Le code géré correspond en grande partie à une version précise du framework .NET, qui se charge ensuite de tenir compte des différences entre les systèmes d'exploitation sur lesquels il est installé.

La version initiale du framework .NET proposait une API nommée WinForms (*Windows forms*) pour le développement d'interfaces utilisateur graphiques. Elle se fondait sur GDI (*Graphics Device Interface*), qui offrait des méthodes directes d'accès au matériel graphique sous-jacent. La version 3.0 du framework .NET est arrivée en 2006 et a proposé WPF (*Windows Presentation Foundation*). Cette nouvelle technologie a représenté une évolution importante, car elle se fonde sur un langage de balisage extensible, XAML (*Extensible Application Markup Language*), pour concevoir des interfaces utilisateur de manière déclarative. XAML permet d'obtenir des graphiques vectoriels et des mises en page fluides, qui s'adaptent aux différentes tailles d'écran, et apporte le concept de liaison de données que nous présenterons plus loin.

L'architecture d'une application écrite avec le framework .NET est illustrée à la Figure 1.4. Cet exemple utilise le framework WPF, qui se place au-dessus du moteur d'exécution principal. Le développeur a la possibilité d'employer divers langages et technologies, comme XAML, pour générer du code qui exploite le framework et la bibliothèque de classes de base (BCL, *Base Class Library*) sous-jacente afin de bénéficier de services communs, comme l'accès au système de fichiers et au réseau. Tous les langages produisent du code MSIL, que la couche CLR compile en instructions natives pendant l'exécution.

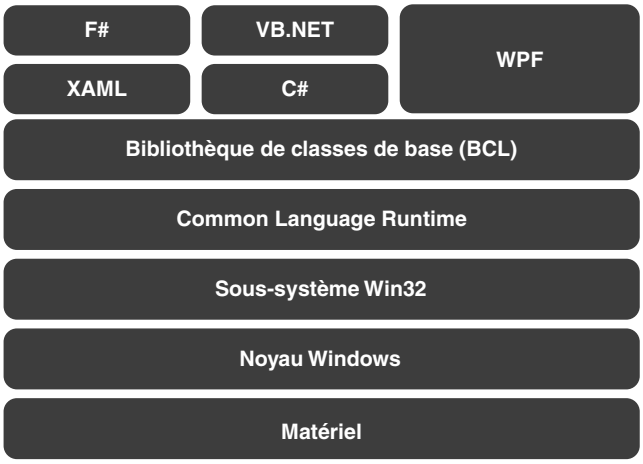

Figure 1.4
Architecture d'une application fondée sur le framework .NET.

Ces dix dernières années, les technologies fondées sur le framework .NET et l'API Win32 ont dominé le monde des logiciels pour Windows. Bien que certains frameworks, comme WPF, aient révolutionné l'interface utilisateur des applications Windows et que Silverlight ait augmenté la richesse de la plateforme, Windows a souffert d'un concurrent inattendu, les tablettes tactiles. En avril 2010, quinze ans après la soirée de lancement de Windows 95, Apple s'est vengé en vendant environ 15 millions d'iPad lors de la seule première année de sa commercialisation. Cet audacieux produit a conduit à une révolution, qui avait commencé quelques années plus tôt, en apportant aux consommateurs ce que l'on appelle une interface utilisateur naturelle (NUI, *Natural User Interface*).

Vision d'avenir : avènement des NUI

La machine à écrire a été inventée au début des années 1800. Le modèle d'origine disposait d'un jeu de touches organisées par ordre alphabétique et d'un mécanisme qui se bloquait si l'opérateur saisissait trop rapidement. Christopher Sholes, un éditeur de presse écrite, a résolu ce problème au début des années 1870 en organisant le clavier de manière que les touches qui correspondent aux lettres les plus fréquentes ne soient pas trop proches l'une de l'autre. Contrairement à une idée répandue, l'idée était non pas de ralentir l'opérateur mais d'éviter les blocages et donc de permettre des saisies plus rapides (Weller, 1918).

Le clavier était purement mécanique : il reliait une lettre à une barre qui était utilisée pour frapper le papier et appliquer l'encre de façon à imprimer la lettre. Les machines à écrire sont désormais des curiosités et la majorité des claviers sont électriques. Il en existe même qui peuvent même être enroulés dans de petits cylindres pour faciliter leur transport et être connectés sans fil à une machine. En dépit des

avancées technologiques spectaculaires, l'aspect du clavier n'a pratiquement pas évolué au cours du siècle dernier.

À la fin des années 1960, le clavier a été complété par la souris. Bill English et Douglas Engelbart travaillaient pour Xerox au PARC (*Palo Alto Research Center*), où ils ont développé les premiers prototypes de la souris (Edwards, 2008). L'appareil réalisé a ainsi été nommé car il était relié par un cordon qui ressemblait à une queue. Aujourd'hui, vous pouvez acheter une souris sans fil qui, à la place d'une boule mécanique, utilise un rayon infrarouge ou laser pour suivre ses déplacements.

L'ensemble clavier-souris représente probablement la manière la plus répandue d'interagir avec les ordinateurs, mais elle est peu intuitive. Si vous lisez ces lignes, nous pouvons supposer que vous êtes un développeur et qu'il est fort probable que vous serviez de "support technique" à vos amis et aux membres de votre famille. Vous avez certainement déjà dû prendre votre mal en patience pendant qu'un parent pianotait laborieusement sur son clavier afin de saisir du texte, et lui avoir également expliqué les différences entre un clic, un double-clic et un clic du bouton droit.

Les recherches sur les différentes manières d'interagir avec les ordinateurs ont débuté quasiment en même temps que la souris se perfectionnait au début des années 1970. L'idée d'interface "naturelle" s'est focalisée sur une méthode d'interaction à la fois intuitive et facile à apprendre. Si vous avez vu le film *Minority Report* avec Tom Cruise, vous avez probablement été émerveillé par les images holographiques qu'il pouvait manipuler par de simples déplacements et rotations de ses mains. La manipulation d'un objet à l'aide des mains est quelque chose de naturel que nous maîtrisons très jeunes. Il est donc plus facile d'employer de tels gestes pour déplacer un document sur un écran d'ordinateur que d'apprendre à pointer, à cliquer ou à faire glisser à l'aide de la souris.

Les NUI ont commencé à entrer dans l'histoire peu après l'année 2000, lorsque la majorité des smartphones ont été équipés d'écrans tactiles. Il était ainsi plus facile de pointer, de taper, de pincer et de balayer pour effectuer différentes actions sur le téléphone. Apple a sorti l'iPhone exclusivement tactile en 2007, avec une interface extrêmement facile à maîtriser. Il est sans doute juste de supposer que cette interface plus naturelle est l'une des raisons essentielles du succès de l'iPhone : il a été le premier téléphone que les utilisateurs peu intéressés par la technique pouvaient s'approprier.

Pendant que la popularité de l'iPhone ne faisait qu'augmenter, Nintendo a mis sur le marché la console Wii. Elle était fournie avec une télécommande particulière dont les capteurs permettaient aux utilisateurs d'interagir avec la console en déplaçant leurs bras et en faisant pivoter leurs mains. Grâce à cette interface, il était plus facile d'employer dans les jeux vidéo des mouvements que vous aviez déjà appris, comme lancer une boule de bowling ou frapper une balle de golf. Le succès de la console a

été immédiat et ses ventes ont battu tous les records (**http://fr.wikipedia.org/wiki/Wii**).

En 2009, on a commencé à entendre des rumeurs sur une initiative Microsoft au nom de code *Project Natal*. À l'automne 2010, Project Natal est officiellement né sous la forme du périphérique Kinect pour Xbox 360 et a poussé plus loin le concept de la Wii en supprimant la télécommande. Les capteurs du Kinect emploient des caméras spécifiques pour combiner des images et une perception de profondeur de manière à analyser les objets présents dans la pièce, notamment les joueurs. Grâce à ces caméras, ils peuvent interagir avec la console par des mouvements du corps, sans avoir besoin d'un quelconque contrôleur. Les différents microphones intégrés sont capables de repérer avec une grande précision l'origine d'un son et permettent aux joueurs d'émettre des commandes vocales de manière naturelle.

La popularité du Kinect a conduit à plusieurs projets open-source dont l'objectif était de créer des pilotes et des logiciels pour que ce périphérique puisse être utilisé sur des ordinateurs. Après que plusieurs produits concurrents (non officiels) sont arrivés sur le marché, Microsoft a répondu en juin 2012 en publiant le kit de développement du Kinect pour Windows. La version 1.5 dispose de pilotes compatibles avec Windows 8.

La révolution des NUI s'est faite au cours de ces dernières années. Les ventes des tablettes et autres périphériques tactiles ont explosé. L'adoption rapide de ces appareils a créé un phénomène appelé "consumérisation des IT"[2]. Les employés refusent de transporter leur volumineux ordinateur portable professionnel et lui préfèrent une tablette tactile plus légère et plus facile à utiliser, qu'ils ont achetée avec leurs propres deniers et qu'ils apportent de leur domicile. Les services informatiques répondent en incorporant ces types d'appareils dans les environnements professionnels.

Le système d'exploitation Windows 7 (voir Figure 1.5) comprend une API grâce à laquelle les développeurs peuvent écrire des logiciels qui répondent aux interactions tactiles et aux gestes. Malheureusement, la majorité des applications écrites pour cette plateforme se fonde encore sur l'ancien couple clavier-souris. Nombre d'entre elles réagissent aux événements tactiles quand on remplace simplement un tapotement du doigt par un clic de souris. Les interactions sont difficiles car ces programmes ne proposent pas des zones adaptées à la manipulation du contenu. Les utilisateurs aux doigts épais, en comparaison de la taille d'un stylet, auront beaucoup de difficultés à sélectionner du texte ou à cocher des cases empilées dans un espace réduit sur l'écran.

2. Le résumé du rapport publié en avril 2007 par Gartner sur ce phénomène peut être consulté à l'adresse **http://www.gartner.com/DisplayDocument?id=503272**.

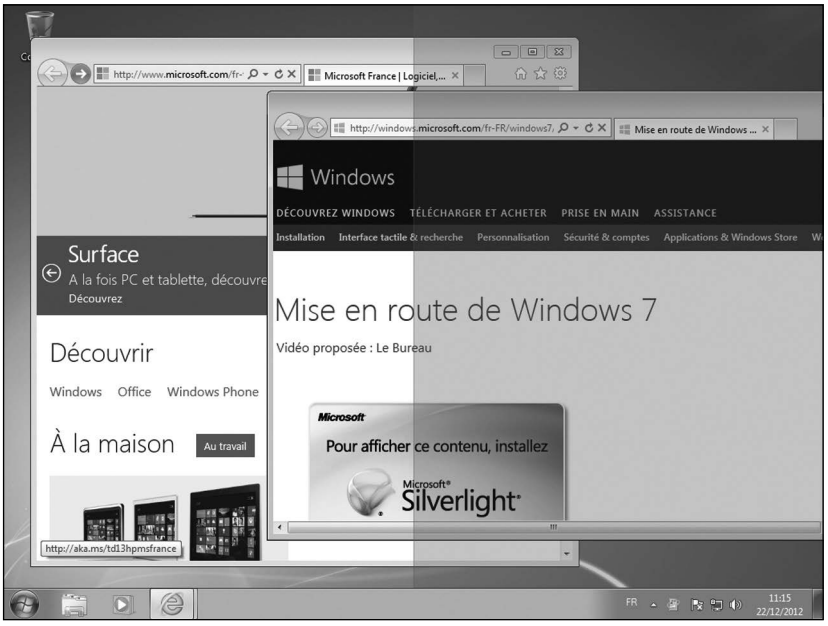

Figure 1.5
Windows 7.

Lorsque les consommateurs auront réalisé la facilité d'interaction qu'offre une interface naturelle, ils vont peu à peu délaisser le clavier et la souris. Dans le monde du jeu, Microsoft a répondu par le Kinect et, dans le monde des téléphones mobiles, par le nouveau Windows Phone. Pour les ordinateurs Windows qui ne sont pas équipés d'une interface tactile, la solution se devait d'être meilleure. Il fallait un système d'exploitation optimisé pour les interactions tactiles afin que les utilisateurs puissent prendre une tablette et être immédiatement productifs. Contrairement à Apple et à Android, Microsoft devait également travailler sans pouvoir exploiter une nouvelle plateforme. Environ 1,25 milliard de machines Windows dans le monde (500 millions équipées de Windows 7) font tourner des logiciels développés au cours des dernières décennies. Il est impossible de les mettre de côté et de les oublier lors de la sortie d'une nouvelle version de Windows (**http://articles.businessinsider.com/2011-12-06/tech/30481049_1_android-apps-ios**).

Microsoft a répondu à ce défi en proposant Windows 8. Ce système d'exploitation révolutionnaire s'appuie sur l'architecture Windows classique, qui permet d'exécuter cet indispensable logiciel que vous utilisez sur vos machines Windows 7, Windows Vista et Windows XP. Cependant, la plateforme a été repensée de manière à offrir une interface utilisateur naturelle tactile de premier ordre au travers d'une nouvelle forme d'application, les applications Windows Store ajustées.

Introduction aux applications Windows Store

Pour Windows 8, le défi est de conserver une rétrocompatibilité avec une base d'installations clientes importante tout en adoptant les interfaces utilisateur naturelles et les plateformes de type tablettes tactiles. La composante essentielle de Windows 8 réside dans un type particulier d'applications qui sont écrites pour le nouveau Windows Runtime (WinRT). Ces applications sont généralement désignées sous l'expression "applications Windows Store", et "applications Windows 8", dans cet ouvrage.

Les applications Windows 8 sont spécifiquement adaptées à l'utilisateur qui les exécute. Elles exploitent les caractéristiques matérielles particulières et s'ajustent au contexte dans lequel elles s'exécutent. Elles changent de taille en fonction de la résolution et de l'orientation de l'écran et s'accommodent facilement d'une souris et d'un clavier en cas d'absence d'un dispositif tactile. Elles peuvent consulter différents capteurs pour déterminer l'emplacement de l'utilisateur et répondre au déplacement de l'appareil sur lequel elles s'exécutent.

Ces applications doivent rester actives, rapides et fluides. Les logiciels développés pour la plateforme doivent s'interfacer aisément avec d'autres applications, les réseaux sociaux et le cloud. Ils doivent être d'un apprentissage simple et d'une utilisation intuitive, car ils tirent parti d'interactions et de gestes naturels. Ces caractéristiques d'une application Windows 8 concernent également le cycle de développement. Les outils fournis doivent permettre de développer rapidement et facilement des applications de qualité, dans le langage de notre choix.

La plateforme de prise en charge des applications Windows 8 se fonde sur une couche spéciale appelée Windows Runtime ou WinRT. À l'aide d'une technique de *projection*, WinRT associe les API du système d'exploitation à des objets du langage de programmation retenu. En C#, les projections prennent la forme d'interactions entre des classes. Cette solution ne compromet pas les performances car le code compilé invoque directement les API, comme si nous écrivions du code C/C++ non géré. Aucune conversion ou correspondance intermédiaire n'est requise et les API compilées ont un accès direct au système d'exploitation de Windows 8. Contrairement à l'API Win32, Windows Runtime est orienté objet.

On expose les API de WinRT en employant la technique déjà mise en place pour le framework .NET. La CLI (*Common Language Infrastructure*) est utilisée pour fournir les métadonnées qui concernent les API et le compilateur s'en sert pour effectuer la projection des méthodes vers le langage choisi et pour réaliser une compilation en code natif. Les métadonnées respectent la norme ECMA-335 et sont enregistrées dans des fichiers ayant pour extension *.winmd* (pour *Windows Metadata*). Pour de plus amples informations sur cette norme, consultez le site web de l'ECMA à l'adresse **http://www.ecma-international.org/publications/standards/Ecma-335.htm**.

La Figure 1.6 illustre l'architecture employée lors du développement d'applications Windows 8. Un grand nombre d'API sont directement fournies par Windows 8, notamment celles relatives à l'affichage, aux appareils, à la sécurité, au réseau, à l'interaction avec le système d'exploitation et à la communication avec d'autres applications. À ce jour, quatre langages sont reconnus par la plateforme de développement Windows Store, avec deux moteurs de balisage graphique, l'un fondé sur XAML (à la fois *via* du code non géré et le CLR), l'autre fondé sur HTML5 (avec l'utilisation du moteur interne "Trident" pour le rendu et du moteur "Chakra" d'Internet Explorer 10 pour l'interprétation du code JavaScript).

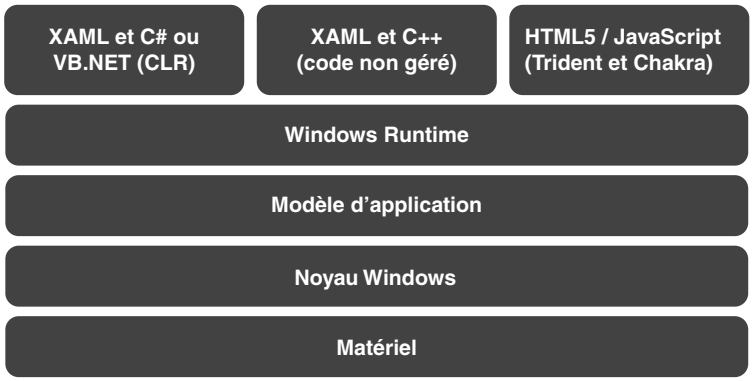

Figure 1.6
Architecture d'une application Windows 8.

Il est essentiel que vous vous familiarisiez avec les caractéristiques spécifiques des applications Windows 8. Leur interface graphique principale passe par DirectX et il n'existe aucune possibilité d'accès direct à l'ancien GDI (*Graphics Device Interface*). Les applications Windows 8 ne prennent pas en charge les fenêtres superposées. Elles s'exécutent dans un conteneur applicatif spécial qui peut avoir plusieurs états. Elles peuvent être suspendues lorsqu'elles ne sont pas actives et peuvent être arrêtées lorsque les ressources système, comme la mémoire, viennent à manquer.

Les API de WinRT existent en deux variantes : appels directs au noyau sous-jacent et appels d'API négociés. Les appels d'API négociés sont des appels particuliers qui peuvent avoir un impact sur l'intégrité des données, sur l'intégrité de l'utilisateur ou sur la sécurité. Pour les employer, l'application Windows 8 doit déclarer ses intentions en indiquant les appels concernés dans un manifeste applicatif. L'utilisateur est souvent invité à autoriser (*opt-in*) ces appels avant que l'application ne puisse les effectuer.

Jensen Harris, directeur du programme sur l'expérience utilisateur pour Windows 8 chez Microsoft, a détaillé, lors d'une conférence Microsoft Build, les huit caractéristiques particulières des applications Windows 8 bien écrites[3]. Microsoft est resté

3. Cette excellente présentation peut être visionnée en ligne à l'adresse **http://channel9.msdn.com/Events/BUILD/BUILD2011/BPS-1004**.

très discret sur les détails de Windows 8 jusqu'à cette conférence pendant laquelle le système exploitation a été officiellement annoncé et sa préversion développeur proposée au grand public. C'est au cours d'une séance spéciale que Jensen a expliqué que les caractéristiques décrites dans les sections suivantes devaient servir de lignes directrices lors de l'écriture d'une application pour la plateforme Windows 8.

Conception Windows 8

Les applications Windows 8 ont une apparence homogène. Pour que l'utilisateur puisse apprendre à les employer de manière intuitive, il est essentiel de respecter autant que possible les recommandations de conception des applications Windows 8. Pour faciliter le travail, l'environnement de développement fournit plusieurs modèles. Vous découvrirez les différentes pratiques et recommandations tout au long de cet ouvrage. N'hésitez pas à les consulter à l'adresse **http://msdn.microsoft.com/fr-fr/library/windows/apps/hh465427**.

Rapidité et fluidité

Toutes les applications Windows 8 doivent être rapides et fluides. Le framework aide à assurer la réactivité de l'application en autorisant uniquement des accès asynchrones aux API dont l'exécution risque d'être lente. Dans le cas d'une application Windows 8, une opération "lente" est une opération qui peut durer plus de 50 ms. Il s'agit notamment des accès au système de fichiers et au réseau. Un fonctionnement asynchrone évite un blocage de l'interface utilisateur pendant le déroulement de l'opération en tâche de fond. Grâce à diverses améliorations apportées au langage C# et au framework sous-jacent, la gestion, l'attente et la réponse aux appels asynchrones sont facilitées.

Les modèles fournis avec Visual Studio 2012 et l'IDE proposent des animations intégrées qui permettent d'offrir une interface utilisateur fluide. La plupart des affichages s'étendent d'un côté à l'autre de l'écran et l'utilisateur a la possibilité de zoomer sur des détails. Au Chapitre 2, vous découvrirez comment les différents modèles apportent des transitions fluides et, au Chapitre 3, comment appliquer des transitions encore plus élaborées.

Ancrage et mise à l'échelle

Si la définition de l'écran est suffisante, c'est-à-dire si sa largeur est d'au moins 1 344 pixels, une application Windows 8 peut facilement être ancrée sur une région de l'écran afin qu'elle s'exécute à côté d'une autre application. Cela se fait à l'aide d'un geste prédéfini, que les utilisateurs peuvent effectuer de façon à gérer les applications Windows 8 en cours d'exécution. Les applications peuvent également

facilement changer de taille en fonction de l'espace disponible, en modifiant leur configuration lorsqu'il devient plus grand ou plus petit, ou lorsque l'utilisateur pivote sa tablette et bascule entre les orientations portrait et paysage. À nouveau, les modèles disponibles dans Visual Studio 2012 apportent les bases de la prise en charge de ces caractéristiques.

Utilisation des bons contrats

Les applications Windows 8 et WinRT viennent avec un nouveau concept nommé *contrats*. Les contrats peuvent être vus comme un mécanisme indépendant du langage pour exprimer des hypothèses sur les possibilités du code. Par exemple, le contrat de partage équivaut à un presse-papiers sous stéroïdes car il est capable de gérer plusieurs types de données, notamment le contenu HTML et les images bitmap. Les contrats permettent aux développeurs d'exposer des services qui interagissent directement avec le système d'exploitation ou qui peuvent être invoqués par l'utilisateur *via* des icônes[4]. Ces icônes sont une fonctionnalité de l'interface utilisateur et de la plateforme Windows 8 qui permet à l'utilisateur d'invoquer des contrats au travers d'un élément d'interface homogène. Il existe notamment un contrat de recherche et son icône correspondante.

Lorsque l'utilisateur active l'icône RECHERCHER à partir d'une application Windows 8, une fenêtre de dialogue du système d'exploitation lui est présentée. Elle comprend un champ de texte dans lequel il peut saisir les termes à rechercher. Sous la boîte de recherche, le système d'exploitation affiche tous les programmes qui prennent en charge le contrat de recherche. Il passera les termes saisis au programme sélectionné afin qu'ils soient traités dans le contexte approprié. Ainsi, une application vidéo peut comparer les termes recherchés aux titres des films, et une application Windows 8 qui agrège des flux RSS peut les comparer au contenu des derniers éléments d'information reçus. Ce fonctionnement est illustré à la Figure 1.7.

Parmi les autres icônes proposées, vous trouverez PARAMÈTRES, PÉRIPHÉRIQUES et PARTAGER (pour échanger des données entre des applications). Nous reviendrons en détail sur les icônes et les contrats au Chapitre 8.

4. N.d.T. : la terminologie anglaise emploie le mot "charm", qui est parfois traduit par "charme", comme dans "barre des charmes". Les termes français que nous avons retenus sont ceux donnés par le portail linguistique de Microsoft (**http://www.microsoft.com/Language/fr-fr/Default.aspx**).

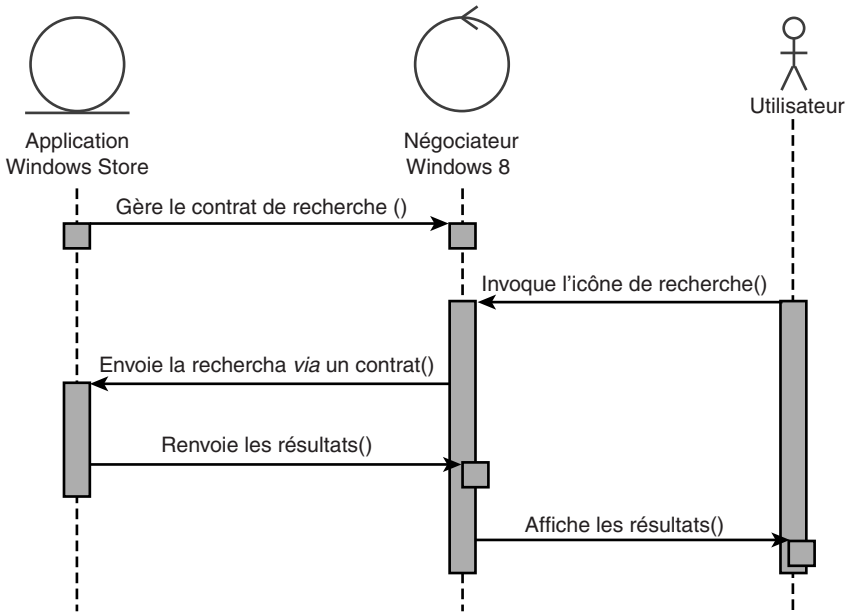

Figure 1.7
Icônes et contrats.

Vignettes accrocheuses

Le concept de vignette est arrivé avec Windows Phone 7. Contrairement aux icônes d'applications qui prennent de la place sur l'écran d'accueil uniquement pour permettre le lancement des applications associées, les vignettes sont des espaces dynamiques interactifs qui fournissent des informations et un contexte à l'utilisateur. Par exemple, une vignette météo affichera la température actuelle et les prévisions. Une vignette Twitter déroulera les tweets les plus récents qui vous mentionnent. Une vignette de courrier électronique affichera le nombre de messages non lus.

L'utilisation des vignettes dynamiques transforme le menu d'accueil en un tableau de bord qui propose à la vue des informations riches. Très souvent, il est possible d'obtenir les informations nécessaires sans lancer l'application à laquelle la vignette est reliée. Ce fonctionnement est comparable à celui de l'Active Desktop que l'on trouvait dans certaines versions précédentes de Windows (**http://fr.wikipedia.org/wiki/Active_Desktop**) et des gadgets de bureau (**http://msdn.microsoft.com/en-us/library/windows/desktop/dd834142.aspx**). Toutefois, les vignettes font partie intégrante de la plateforme Windows 8 et sont reliées directement aux applications.

La Figure 1.8 illustre un ancien bureau statique. Les seules informations "dynamiques" sont la date et l'heure affichées dans l'angle inférieur gauche. Tout le reste est statique et les icônes servent uniquement à lancer les applications. Vous remarquerez la grande quantité d'espace vide inutilisé sur le côté droit du bureau.