

LE PORTAGE APPLICATIF : UN PROJET BIEN SPECIFIQUE



Siège social :
14, rue Gaillon – 75002 Paris – France
Tél. : +33 (0)1 42 682 800 – Fax : +33 (0)1 40 075 486
www.os4i.com – info@os4i.com
S.A. au capital de 169 990 euros – RCS 437 664 394 Paris
SIRET 437 664 394 00040 – Code APE 722 C

SOMMAIRE

1.	LE PORTAGE APPLICATIF : UN PROJET BIEN SPECIFIQUE.....	3
1.1.	Les différents types de projets	3
1.2.	L'organisation des développements	4
1.2.1.	Les points à valider avant de démarrer un projet de portage sous Linux.....	4
1.2.2.	Méthodologie : les bienfaits du cycle en V.....	5
1.3.	Conclusion : 5 pièges à éviter.....	6
2.	RETOUR D'EXPERIENCE SUR UN PORTAGE DE WINDOWS VERS LINUX.....	8
2.1.	La libwinapi	8
2.2.	Portages annexes : les COTS.....	9

1. LE PORTAGE APPLICATIF : UN PROJET BIEN SPECIFIQUE

Spécialiste du monde Linux, OPEN WIDE/OS4I suit de très près l'adoption des technologies du libre par les industriels.

Le portage est une étape clé de cette démarche. Considéré à la fois comme démonstrateur de faisabilité et point de départ pour de nouvelles opportunités applicatives, ce « grand saut » exige une rigueur méthodologique sans faille et une maîtrise technologique pointue et diversifiée.

Retour d'expérience sur un type de projet pour lequel OPEN WIDE/OS4I a développé une expertise reconnue...

1.1. Les différents types de projets

Le terme portage englobe des projets aux enjeux techniques très différents. Nous pouvons toutefois distinguer deux grandes catégories de projets :

1. Le portage applicatif

On retrouve ici le très classique portage Windows vers Linux. Le remplacement de l'API Win32 par des fonctions POSIX est au cœur des projets, soit directement dans le code applicatif, soit par le biais de couches d'abstraction existantes. Mais c'est aussi un changement complet des outils de développement à gérer, en particulier les bibliothèques et le compilateur. Le cas particulier du portage MFC vers une bibliothèque graphique C++ devient alors un projet de développement applicatif à part entière.

Autre type de projet, le choix d'une solution Linux Temps Réel (cf. notre article sur Xenomai) mise en place d'un RTOS propriétaire, voire le changement de solution TR sous Linux (par ex. RTAI vers Xenomai). Dans ce cas, la validation ne concernera pas seulement les fonctionnalités, mais aussi le respect des contraintes de performances spécifiées

2. L'upgrade du hardware et des composants bas niveau

La démarche est ici « inverse » au portage applicatif. Nous sommes dans un contexte classique d'obsolescence matérielle qui amène à réaliser un upgrade global : nouvelle plateforme matérielle, nouvel environnement logiciel pour accueillir des applications dont le code source ne va pas évoluer.

Ce type de portage est aussi souvent une étape préliminaire aux évolutions fonctionnelles. Amélioration des performances, ajout de composants matériels : autant d'objectif qui nécessitent la mise à niveau de l'OS et la validation préliminaire des drivers.

1.2. L'organisation des développements

1.2.1. Les points à valider avant de démarrer un projet de portage sous Linux

☞ L'état de l'application

Cette analyse doit concerner au moins deux aspects :

1. La validation fonctionnelle : le périmètre attendu après portage est-il déjà correctement couvert par l'application originale ?
2. L'architecture de l'application, dont dépendent directement les risques du projet : existence d'une couche d'abstraction, mécanismes de communication, gestion des erreurs, organisation des données... L'analyse fine de ces points permettra en outre de connaître la robustesse de l'application à des modifications sémantiques légères.

☞ L'existence de tests unitaires et de tests d'intégration

Eviter si possible d'être dans une logique selon laquelle les premiers tests seront effectués lorsque tout sera porté. L'investigation de chaque dysfonctionnement sera alors extrêmement laborieuse. Dans ce cas il sera intéressant, quitte à réaliser des développements complémentaires, de se doter d'outils de tests d'intégration (simulateur, espions de bus, etc.).

☞ Les composants externes utilisés, les bibliothèques

De nombreux projets s'appuient sur des COSTs qui devront être remplacés par leur équivalent libre ou bien leur version portée sous Linux par le fournisseur, et donc validés de manière exhaustive au préalable.

☞ La disponibilité et l'état des drivers sous Linux

C'est un point sensible concernant les projets de portage sous Linux, car de nombreux fabricants de cartes ne fournissent pas de drivers ou – pire – fournissent une version aux possibilités limitées voire non aboutie...

Il faut de même vérifier la compatibilité des drivers entre eux pour des configurations plus complexes. Nous avons déjà eu le triste cas de cartes disposant de drivers ne pouvant pas fonctionner sur la même version de noyau Linux...

☞ La disponibilité des compétences sur l'application

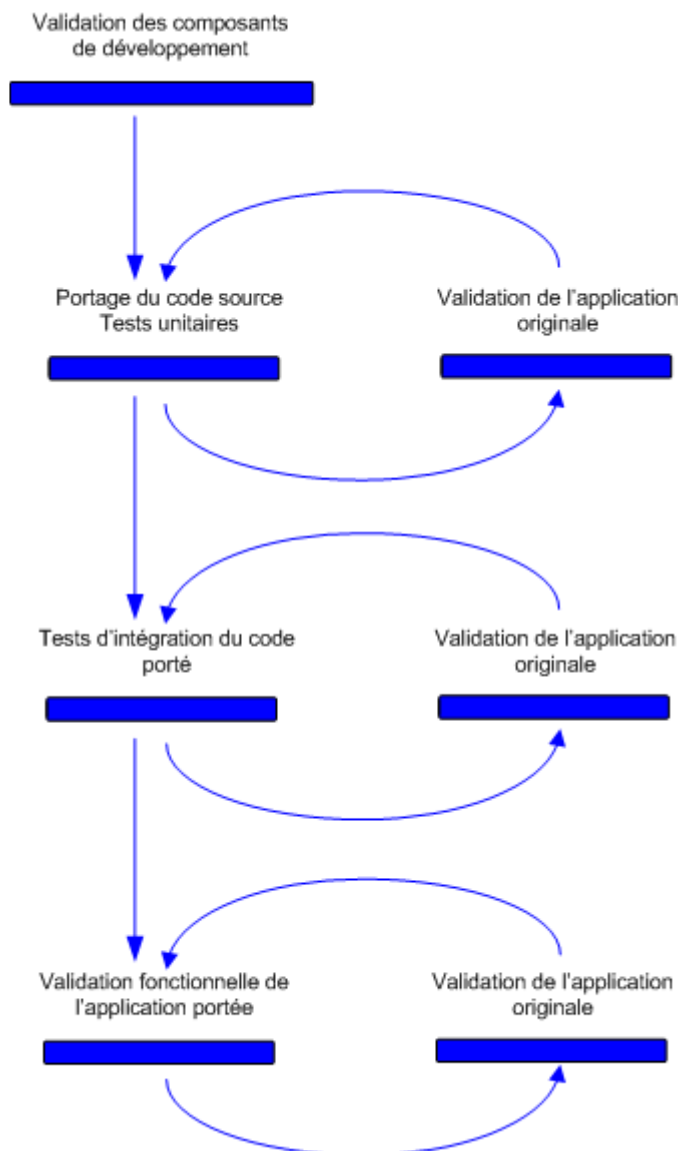
Un projet de portage ne peut pas être réalisé efficacement si les compétences ne sont plus là, autant pour donner des indications techniques que pour mettre en œuvre les tests de validation. Un projet de portage est toujours un travail d'équipe qui réunit sur une même problématique des compétences techniques et fonctionnelles.

1.2.2. Méthodologie : les bienfaits du cycle en V

Dans de nombreux cas, il faudrait d'ailleurs parler de « double-cycle en V ». Il s'agit en effet souvent de porter le code source d'une application tout en garantissant son fonctionnement dans l'environnement d'origine. Double version, double cycle de validation donc...

En matière de portage applicatif, le fameux cycle en V reste une référence absolue, surtout pour sa partie montante : les étapes de tests et validation. Car en terme de développement logiciel, de quoi s'agit-il ? D'isoler toute partie de code spécifique à l'API d'origine, et de développer le code équivalent dans le nouvel environnement. Aussi simple que de développer des fonctions systèmes basiques...Mais aussi structurant que de développer les briques de base constituant l'application !

Le schéma ci-dessous présente le principe d'organisation d'un projet de portage :



On retrouve dans ce schéma :

- ▶ La nécessité de tester l'application originale à chaque étape du projet si le maintien d'une double version et d'un seul code source est imposé. La validation post-portage initial du code source joue un rôle essentiel. Pendant le portage lui-même, il est par ailleurs conseillé de vérifier régulièrement que l'application compile toujours dans l'environnement d'origine. Ce n'est pas une garantie de bon fonctionnement, mais les erreurs grossières pourront être détectées au plus tôt ! Bien entendu, les applications s'appuyant initialement sur des couches d'abstraction limitent grandement les risques exposés ici...
- ▶ Les tests d'intégration, qui sont en général le point faible de ce type de portage. Chaque développeur réalise ses propres tests unitaires et la validation fonctionnelle permet de terminer le projet : ces deux étapes sont rarement négligées. Il est cependant bien tentant de passer de l'une à l'autre en oubliant les tests d'intégration. Grave erreur, qui se paie d'autant plus cher que le projet initial est de volume important ! Chaque investigation de bug devient alors un parcours du combattant dans les méandres d'un code qui a souvent déjà connu plusieurs évolutions successives et pas toujours cohérentes...
- ▶ Les tests fonctionnels, pour lesquels les utilisateurs de l'application initiale (nos clients donc...) ont un rôle fondamental. Jusqu'alors, le développeur pouvait travailler « en aveugle » (tests unitaires de fonctions de base) ou juste avec des œillères (tests d'intégration au périmètre limité). Il s'agit ici de solliciter le nouveau code dans tous les cas de figure et – surtout – d'aider à l'investigation de dysfonctionnements fonctionnels qu'il est difficile de relier aux différents modules logiciels pour un développeur qui connaît peu l'architecture globale.

On l'aura compris : un projet de portage est souvent déroutant pour les développeurs. Ils ont en effet l'impression de commencer par la fin : il va falloir assimiler une masse de code existante, importante et totalement inconnue. On ne connaîtra les détails de l'application que sur la fin... Difficile de trouver des repères, d'autant que revient souvent le fameux chant des sirènes « ça marchait très bien avant » qui encourage à négliger les étapes intermédiaires de validation (il n'y a pas de raison que cela ne fonctionne plus maintenant...).

Heureusement, aussi spécifique soit-il, ce type de projet peut s'appuyer sur une méthodologie très classique et ayant déjà largement fait ses preuves.

1.3. Conclusion : 5 pièges à éviter

En guise de conclusion, retenons qu'un projet de portage n'est jamais à prendre à la légère quelque soit la qualité de l'application initiale. Il exige une grande rigueur méthodologique dont nous avons voulu donner dans cet article les principales clés.

Citons pour terminer **5 pièges très classiques à éviter** à tout prix pour mener à bien un projet de portage :

1. Ne pas mélanger les types de portage : si le projet nécessite à terme un update de hardware et un portage applicatif, il doit être découpé en phases séquentielles.
2. Ne pas négliger les moyens de tests : si nécessaire, réaliser les outils adéquats
3. Ne pas utiliser de modules logiciels externes sans avoir effectué une étude de faisabilité ou – mieux – un POC (*Proof Of Concept*)

4. Ne jamais mélanger portage et évolution fonctionnelle : là encore, les travaux doivent être séquentiels. De même, il est toujours très difficile d'intégrer après portage des évolutions fonctionnelles qui auraient été réalisées en parallèle sur la version originale.
5. Ne pas sous-estimer la phase de mise au point fonctionnelle : c'est en général la plus longue, car chaque dysfonctionnement constaté nécessitera une investigation longue du code source original...

2. RETOUR D'EXPERIENCE SUR UN PORTAGE DE WINDOWS VERS LINUX.

Quel est le contexte? Thales transportation systems travaille notamment dans deux domaines différents : parking et transport. Ces deux activités, jusqu'alors complètement indépendantes, vont, petit à petit, être amenées à se regrouper au sein d'une même borne qui distribuerait des tickets de parking et de transport.

Chacune des deux applications tourne sous un OS différent. La problématique est donc de faire fonctionner une application Visual C++ Windows sous un Linux standard.

Open Wide est intervenu sur cette partie en tant qu'expert dans le monde Linux présentant des compétences reconnues dans le portage d'application.

Le portage représente environ 80000 lignes de code réparties dans 20 modules pour une moyenne de 7 fichiers par module.

Le code utilise largement l'API Visual C++ de Windows; le principal choix technique fut de développer une couche d'abstraction : la libwinapi, une librairie d'interface entre l'API Windows et l'API Linux. Pour les fonctionnalités additionnelles (les COTS) telles que le client FTP, la communication CORBA et la base de Données, nous n'avons eut qu'à chercher dans le monde du libre pour trouver les équivalents Linux des DLL Windows, et réécrire les interfaces entre l'application principale et les librairies Open Source.

2.1. La libwinapi

Ce choix technique présente énormément d'avantages tant du point de vue technique, que gestion du projet. En effet, 135 fonctions de l'API Windows portées sont facilement testées unitairement et assurent ainsi un risque minimum lors de l'étape suivante qui est l'exécution des tests fonctionnels.

Par la suite, l'intégration de modifications dans le code Windows original ou l'ajout de nouveaux modules représente un travail moindre : seules les quelques nouvelles fonctions spécifiques Windows devront être portées.

Ainsi, la libwinapi permet actuellement d'utiliser les timers, les threads, les sockets, les sémaphores, les mutexes, les sections critiques, les pipes, les événements, une base de registre (en lecture seule), et toutes les fonctions standards de manipulation de fichiers, de dossiers, de fichiers de configuration (.ini), de chaînes de caractères, de temps, les constantes et les types.

Du point de vue technique, la libwinapi n'est pas intrusive, elle est complètement indépendante du code de l'application principale. Cela permet de ne pas surcharger le code de directives préprocesseur qui perturbe la lisibilité du code et d'assurer au maximum la compatibilité Linux et Windows.

Il reste tout de même quelques modifications nécessaires au niveau du code du client. Tous les fichiers utilisant l'API Windows (contenant `#include windows.h`) doivent maintenant inclure la libwinapi (avec `#include linux.h`). Un simple `#ifdef` sur une variable WIN32 / LINUX définit pour le préprocesseur va assurer le bon fonctionnement du code quel que soit le système d'exploitation utilisé.

2.2. Portages annexes : les COTS

Il n'est pas rare qu'une application fasse appel à des DLL pour réutiliser des fonctionnalités complexes telles que, dans notre cas, un client FTP, Base de données, bus CORBA. Il existe sous Linux, les équivalents de beaucoup des DLL Windows. Dans le cas présent, nous pouvons citer libCURL pour le FTP, SQLITE pour la base de données, et omniORB pour la communication CORBA. Ces bibliothèques présentent des interfaces différentes de celles utilisées par Windows, il faut donc : soit enrichir la libwinapi, soit écrire du code parallèle, le préprocesseur choisira alors le morceau de code en fonction de flags de compilation WIN32 ou LINUX. Lorsque l'application gère déjà une couche d'interface avec ces DLL, ce deuxième choix est pertinent afin d'éviter de cumuler les couches d'abstraction et donc de compliquer/ralentir l'application finale.