

# Informatique appliquée à la Géomatique

---

Cours destiné à former les

Ingénieurs Informaticiens Géomaticiens  
de l'ENSG

et les Master 2 "IASIG" de l'UPMC

**François BOUILLÉ**

Prof. UPMC / ENSG



**Chapitre 11.7**



*Cryptomeria japonica*

## Objectif du chapitre 11.7 :

L'objectif de ce court chapitre est de présenter quelques principes pour assurer la portabilité des données et celle des applications.

## Mots-clés :

portabilité, algorithmes commentés, chaîne DSS, code intermédiaire, A-code, compilateur de compilateur, précompilateur, *boot-strapping*.

# PLAN

- 1 - Définition
- 2 - Ce qui n'est pas intégralement portable
- 3 - Quelques recommandations
- 4 - La portabilité d'un modèle à TAD persistants
- 5 - Portage du systeme complet

# 1 - Définition

La portabilité est la propriété permettant d'assurer une indépendance vis à vis de la machine sur laquelle des données où des programmes sont installés.

Elle permet de changer de matériel, de site d'implantation, d'être indépendant d'un constructeur informatique et de différents éléments de l'environnement informatique.

Cette propriété contrarie évidemment les souhaits d'hégémonie d'un certain nombre de sociétés commercialisant des systèmes et applications, qui cherchent à ce qu'un client reste à vie dans la liste de ses acheteurs, ou continue à payer ( *très cher !* ) les licences annuelles.

Pendant longtemps, l'on a distingué les données et les programmes. Il est toujours possible de faire cette distinction dans des domaines très classiques, comme les bases de données relationnelles, mais avec la manipulation des TAD, des morceaux de code sont présents au sein des données.

L'on peut formuler un certain nombre de recommandations permettant d'assurer une portabilité aisée, ou dans certains cas, de limiter les conséquences lors d'un changement de contexte.

## 2 - Ce qui n'est pas intégralement portable

- au niveau des supports : près de 20 % des bandes magnétiques stockées par la NASA sont indécodables. Là, ne parlons même pas de portabilité...
- les fichiers élaborés sur un site sont très fréquemment non reconnus sur un autre, sauf si les deux sites ont exactement les mêmes équipements et systèmes.
- les bases de données relationnelles : essayez donc de transférer directement une base ORACLE vers un système INGRES ! Les bases de données sont liées aux SGBD et les passerelles qui devraient exister et être pleinement opérationnelles font défaut.
- déjà au sein d'un même SGBD, les performances sont très variables et les temps de résolution de requêtes varient dans un rapport de 1 à 700 (\*),
- les traitements de textes courants et les tableurs ne fournissent pas toujours sur deux sites strictement les mêmes textes.

### Pourquoi ???

- parce que le souci de portabilité n'a pas été promu dès le départ d'un projet,
- parce que l'impossibilité de portabilité a parfois été érigée en principe.

(\*) thèse d'Emmanuelle MADER ("Une plateforme pour le réglage des optimiseurs de requêtes").



### Au niveau des programmes :

-très peu nombreux sont les langages de programmation ayant permis le portage d'un site à un autre sans changement ; signalons en deux :

.SIMULA, parfaitement défini par Ohle-Johann DAHL, et dont la standardisation était assurée par le SSG (Simula Standard Group), même s'il était parfois frustrant de voir des propositions d'améliorations soumise au SDG (Simula Development Group) refusées par le SSG (\*),

-malheureusement, SIMULA, toujours aussi clair et efficace, est très peu utilisé

.ADA, mais qui a raté le train de l'orienté-objet (manque de vision des auteurs) et dont la lenteur était difficilement compatible avec des applications dites « temps réel » (\*\*); il y a même eu la possibilité d'avoir des cartes dédiées à ADA (circuit iAPX432) qui aurait pu assurer une portabilité sans problème...

Remarquons au passage que ces deux langages ont été parfaitement définis par une BNF.

(\*) mais c'est justement grâce à cette standardisation que je pouvais passer mes programmes entre six machines différentes sans opérer le moindre changement !

(\*\*) la lenteur par contre ne semble pas inquiéter certains adoreurs de Java...

### 3 - Quelques recommandations

- gardez les algorithmes : eux sont portables !
- ayez des algorithmes propres, truffés de commentaires : en cas de problème, le « retour aux sources » du code sera aisé,
- lorsque vous écrivez des programmes (qui sont en principe la traduction de vos algorithmes...), choisissez un langage de programmation clair et bien structuré, et conservez les commentaires abondants,
- si vous n'avez pas le choix du langage ni de sa version, arrangez-vous pour ne jamais utiliser les instructions très spécifiques qui sont vantées dans la version X, et qui sont supposées faire gagner quelques nanosecondes ; ils vous feront perdre ultérieurement des heures, voire des jours, lors des changements de machine ou de centre de calcul,
- évitez tous les logiciels non stabilisés, en cours de développement, mais « promis à un grand avenir », les cimetières informatiques en sont pleins,
- ne croyez pas que l'utilisation de logiciels déjà existants sur le marché et mis bout à bout est toujours plus rentable que de développer l'interface manquante,
- ne courez pas après la dernière version de tel système, il n'est pas encore « *debuggé* » ...



-inutile de vous lancer dans un logiciel qui ne présente pas directement les interfaces souhaitées, vous risquez de commettre ce qu'a réalisé une certaine société :

-on pense à utiliser X , un nouveau produit plein d'avenir, mais il n'a pas d'interface avec Y ; cependant il en a une avec Z ; oui, mais Z n'a pas d'interface avec Y, mais ce n'est pas grave, car il a une interface avec T, qui doit pouvoir s'interfacer, car il y a eu des applications assez proches qui y sont parvenues (tant bien que mal...), etc... ; plus on a investi de temps dans ce projet, et moins le chef de projet a envie de renoncer ...

mais....

au bout de six mois, il faut bien se rendre à l'évidence...

-bilan pour la société : six mois-ingénieur(s) perdus, démotivation de l'équipe travaillant sur le projet, chef de projet suspecté d'incompétence... (*il n'a été qu'imprudent*).

## 4 - La portabilité d'un modèle à TAD persistants

-les données (au sens strict) :

- .sont implantées sous forme de chaînes DSS, intégralement portables,

- ne cherchez pas à transporter directement les fichiers qui les contiennent, seules les chaînes peuvent être envoyées sur une ligne directement d'un site à un autre,

-l'environnement des données :

- .les métadonnées sont incorporées dans les chaînes DSS, et ne posent pas de problème,

- .les séquences de code sont incorporées elles aussi dans les chaînes sous forme de code intermédiaire propre au langage ADT'81, le A-code portable (\*)

- .les méthodes sont aussi implantées sous forme de A-code.

Il reste à voir le noyau permettant d'utiliser les TAD.

(\*) voir le cours de compilation (cours 15)  
et l'implantation du compilateur ADT.

## 5 - Portage du systeme complet

- le système est écrit en ADT'81, qui nécessite un noyau,
- les tables du compilateur sont générées automatiquement via un compilateur de compilateurs (voir cours n°15.1)
- le noyau est écrit en ADT<sup>MIN</sup> (voir cours n°14.1)
- ADT<sup>MIN</sup> est écrit via un précompilateur (voir cours n°15.7), qui nécessite 1200 lignes de code d'un langage quelconque, aussi bestial soit-il (Fortran, C, Pascal, ...).
- la méthodologie de montage sera détaillée dans le cours consacré à ce sujet et intitulé « Technique de boot-strapping » (cours n°15.10).

Au lieu de porter l'intégralité du système sur une autre machine, on peut se contenter d'installer un noyau pour recevoir les données demandées voir le cours sur la répartition (cours n°11.10).

## Simple utilisation des données (création, mise à jour, consultation...) :

- il est inutile de disposer du système complet,
- il suffit de disposer de primitives sous forme d'une bibliothèque de sous-programmes manipulant les divers TAD :

Exemple : `create "PONT" . "hauteur" ;`

peut être remplacé par :

`CREAT_ATTCL ( "PONT" , "hauteur" , mess)`

Exemple : `specify "PONT" . "hauteur" float ;`

peut être remplacé par :

`SPEC_ATTCL ( "PONT" , "hauteur" , type, valtype, mess)`

C'est évidemment plus lourd, moins pratique, mais fonctionne.

L'on peut aussi implanter directement un noyau recevant les couches nécessaires à une utilisation élémentaire (voir cours de *boot-strapping*).

## Pour compléter votre information :

- chaque chapitre faisant l'objet d'une vidéo est accompagné d'un **diaporama**. Regardez-le !
- reportez-vous aux **supports de cours**. Les divers fascicules (plus de 1000 pages) sont beaucoup plus détaillés.
- reprenez les **projets** des Travaux Pratiques faits à l' **ENSG**.



## **Sur ce cours, pour plus d'information :**

- regarder la vidéo du cours,
- se reporter au support de cours intitulé :

### **"L'architecture du système HBDS"**

- revoir les algorithmes des mini-projets successifs des couches du système (SFSI, ZE, GTAD, MFCT, ...) utilisant trames, automates et moteurs.

## **Travaux pratiques à l'ENSG :**

Mise en pratique dans le cadre du projet de Géomatique et du projet de Génie Logiciel.





*Cunninghamia lanceolata*

**Bon courage !!!**