La persistance des données

Lorsque l'on manipule des variables, que cela soit des variables simples ou complexes tel des tableaux ou des objets, celle-ci sont généralement supprimées, et la mémoire qui leur est liée libérée, dès que le programme ne les utilise plus. C'est le cas, par exemple, à l'arrêt d'un programme. Les données sont alors « perdues », puisque la mémoire utilisée par les variables est libérée en fin d'exécution et ne persiste donc pas entre les différentes exécutions. Il incombe au programmeur d'effectuer les étapes nécessaires à une bonne persistance des données dont le programme se doit de conserver. Il serait par exemple dommage de perdre une liste de contacts qu'un utilisateur aurait renseignés en utilisant un logiciel incapable d'effectuer la persistance de ces données. L'utilisateur serait ainsi obligé de ressaisir les informations des contacts à chaque nouvelle exécution du logiciel !

Le seul moyen de faire persister une donnée à un « reboot » de l'ordinateur est de la sauvegarder sur une mémoire « morte » (disque dur, ssd, ...). Cette persistance peut se faire grâce à des systèmes de gestion de données tels que des bases de données ou des Framework (ensemble de fonctionnalités) implémentant ce mécanisme (comme Hibernate pour Java). Mais ce n'est que déléguer le problème, car ces systèmes devront eux aussi assurer la persistance des informations. Quelque soit le système il faudra assurer la sauvegarde des données depuis la mémoire vive sur la mémoire morte et vice et versa. Cette étape passe par une méthode de sérialisation (ou d'encodage) qui a pour but de transformer les données en format facilement enregistrable et une méthode de dé-sérialisation lors de l'étape de récupération. La performance temporelle de cette fonction doit être un critère important de sélection si la quantité de données à sauvegarder est importante.

Les formats utilisés par la sérialisation peuvent être diverses, il faut « juste » qu'ils assurent la bonne sérialisation des types de données de base, puisque tous les types complexes sont basés sur ceux-ci, ainsi que la représentation des tableaux et des objets. On peut utiliser des formats standards (JSON, XML), des formats natifs aux langages de programmation (la sérialisation Java par exemple) ou encore des formats défini par le programmeur en utilisant ou non des standards. L'utilisation de format standard tel que JSON à l'avantage d'offrir une persistance de l'information « cross language », puisque ce format de description des données est supporté par un grand nombre de langage de programmation dont les plus usités font partis (vous pouvez voir la liste des langages qui supporte ce format sur le site officiel de JSON). L'utilisation de la sérialisation native aux langages de programmation empêche le plus souvent la persistance « cross language » mais reste souvent plus simple à implémenter. La dernière solution « maison » apporte une particularité qui n'a peut être pas lieu d'être dans la plupart des logiciels, mais peut être intéressante pour certain cas particulier.

Quelque-soit la solution choisie, il faut ensuite se poser la question du moment où cette sérialisation et dé-sérialisation seront faites. Le plus simple serait de faire la dé-sérialisation au démarrage du logiciel et de faire la sérialisation lors de son arrêt, puisque qu'entre les deux les données peuvent vivre en mémoire vive sans problème. Bien que cette solution simple semble efficace, il faut quand même offrir une étape de sauvegarde supplémentaire afin d'éviter une perte de données trop importante lors d'un crash système par exemple. Une action de sauvegarde déclenchée par l'utilisateur (un bouton de sauvegarde ou un ctrl-s par exemple) ou une sauvegarde automatique par le système semble deux bonnes solutions. Une autre solution serait de faire la sauvegarde d'une structure de données chaque fois que celle-ci change.