LibreOffice : le portage WebAssembly reprend des couleurs

LibreOffice dans un navigateur web ? En soi, ce n'est pas nouveau. La Document Foundation propose une version de sa suite bureautique fonctionnant sur ce modèle. Elle utilise l'élément canvas de HTML5 pour afficher les documents. Mais l'essentiel des opérations (rendu, édition, gestion des utilisateurs…) se fait côté serveur.

Cette architecture allège le code côté client. Tout en favorisant la sécurité et l'édition collaborative, avec une seule version de document à tenir à jour. En revanche, elle ne permet pas le hors-ligne, l'édition P2P ou encore le chiffrement de bout en bout. Il faut y ajouter les coûts d'hébergement (la Document Foundation les estime entre 50 et 100 $/an pour chaque utilisateur). Et tenir compte de la difficulté de passer à l'échelle.

Et si tout pouvait se passer dans le navigateur ? La Document Foundation avait exploré cette piste il y a quelques années en s'appuyant sur WebAssembly. Ce langage de représentation intermédiaire (*bytecode*) sert de cible de compilation pour permettre l'exécution de langages de haut niveau dans des applications web à une vitesse proche du natif. Il fonctionne de concert avec JavaScript.

LibreOffice – WebAssembly : on prend les mêmes et on recommence

L'expérience WebAssembly n'avait pas été probante. Mais on l'a finalement réenclenchée au vu de la maturité du *bytecode*, promu standard W3C fin 2019. Le coup d'envoi officiel de cette initiative de relance avait été donné en octobre dernier, à l'occasion de l'openSUSE + LibreOffice Conference. On nous avait alors promis une première démo au plus tard pour la FOSDEM 2021.

Promesse tenue, mais de façon minimale. En l'occurrence, sous la forme d'une application Qt5 qui affiche l'ensemble de Mandelbrot. Le reste du code est compilé, avec Emscripten, mais ne fonctionne pas encore.
La Document Foundation maintient, dans l'ensemble, son calendrier initial. Elle vise, pour cet été, un rendu interactif de Writer en HTML5. Et, d'ici à la fin de l'année, la mise en œuvre d'une session d'édition chiffrée de bout en bout, toujours sur la partie traitement de texte.

L'un des grands enjeux sera de réduire le poids de Writer (objectif : 25 à 30 Mo). Cela impliquera de s'appuyer sur des API du navigateur pour remplir les fonctions qui dépendent actuellement de bibliothèques logicielles tierces (NSS pour le réseau, ICU pour la localisation…). En toile de fond, les limites que WebAssembly impose en matière de consommation de mémoire : communément, 2 Go maximum par application.

Les modules dynamiques, le partage de mémoire, la gestion du système de fichiers et le débogage sont d'autres obstacles. L'absence de *multithreading* sur WebAssembly l'est aussi, mais dans une

moindre mesure concernant Writer.

Illustration principale © WavebreakMediaMicro – stock.adobe.com