SARL MEZZONOMY

mezzonomy@orange.fr – (33|0)6.32.97.90.27

Programmation informatique (6201Z)

34 Place de Catalogne 31700 BLAGNAC

Toulouse B 504 641 473

De Mirza à Cargo

Pierre Gradit, Avril 2015

Introduction

L'innovation résulte souvent de deux contraintes contradictoires : un effort préparatoire de fond et une urgence qui oblige à faire des compromis productifs. Dans le cas de **Cargo**, l'effort de recherche et de développement a été réalisé dans le cadre du projet **Mirza**, financé par nos actionnaires et avec l'aide d'OSEO.

Mirza avait pour but de valoriser une découverte théorique majeure par une des possibilités qu'elle offrait : celle de pouvoir réaliser un système d'« *information liquide* » **sans duplication de donnée**. Proche dans les attendus de la plateforme Xanadu de Ted Nelson, ce système était dotée d'un fondement algébrique solide et d'une solide expérience ergonomique. Sa réalisation concrète est déclarée faisable. Mais une fois le projet terminé, l'ambition du projet Mirza était incompatible avec la stratégie d'innovation par petits pas promue par les investisseurs institutionnels. Réaliser une **plate-forme de type Office garantissant la propriété intellectuelle** sur le réseau est une ambition démesurée dans la logique actuelle de l'innovation. Ce projet suit son cours sous la forme d'une plate-forme de valorisation de la recherche en lien avec une entreprise industrielle.

L'urgence est venu de la nécessité de trouver des débouchés de court-terme pour assurer la survie de la société. Nous avons pris contact avec plusieurs

sociétés qui développaient des **plate-formes applicatives autour de la simulation numérique**. L'une s'entre elle, la société **SCILAB**, nous a demandé de réaliser un module programmable en Java pour la gestion des préférences : **Cargo** est né. Ensuite, pour nous dégager des contraintes de propriétés intellectuelles, nous avons réécrit cette œuvre en Python, puis l'autre société que nous avions contactée, **SILKAN**, a choisi **Cargo** comme plate-forme de développement métier pour son **SealDesigner** puis nous a demandé de développer la plate-forme **BUILDER** pour la « *simulation robuste* » avec le même système.

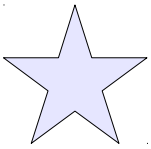
L'application i3L_QAO est la dernière née sur CARGO, elle contient l'ensemble des innovations développée depuis plusieurs années. Son développement a ainsi été rendu possible pour des coûts très inférieurs au marché. Ce document détaille les liens entre le projet **Mirza** et le programme **Cargo**.

Formulation modale

La formulation modale est issue d'un besoin formalisé en 2005 dans le cadre d'un projet ambitieux de coopération informatique dans le cadre aéronautique.

Ce contexte initial trouve une illustration dans l'application i3L_QAO qui vous servir d'exemple illustratif : **le lien avec le tableur d'une suite Office**.

Lorsque vous sélectionnez une catégories de ressource un onglet « *Inventaire* » apparaît, que ce soit une classe d'objets ou l'ensemble des processus. Dans cet onglet « *Inventaire* », une table peut être **copié-collée** dans un tableur, modifiée dans ce tableur et réinjectée par copier-coller dans l'onglet inventaire.



La formulation modale a été crée pour répondre à une situation où ces opérations interviennent partout dans le système : **extraction d'un fragment en un point du système et réinjection au même point d'une version modifiée de ce fragment**.

Lorsque l'utilisateur sélectionne toute la table et la copie, il transfère l'apparence de l'inventaire *choisi* dans la « *mémoire de la souris* ». Puis quand il « *colle* » ce contenu dans le tableur, le modèle ainsi transmis est interprété

par le logiciel cible et intégré à son propre modèle. L'opération de retour procède de manière homologue, mais *dans l'autre sens*.

Pour aller plus loin

Comment cela fonctionne ? *CARGO travaille sur modèle de donnée XML qui permet de construire l'interface graphique par un HTML étendu, comme celui affiché dans l'onglet d'inventaire. Par exemple, le calcul réalisé pour l'inventaire des processus s'écrit :*

HTML = repository(XML/processes)

Cet HTML est copié dans le clipboard et interprété par le tableur. L'utilisateur le modifie à loisir.

♦

Au retour, Cargo récupère un HTML proche de l'initial, il lui applique une transformation inverse à la première :

XML' = $\overline{repository}$(HTML')

Une fois obtenu ce nouvel XML, il doit être incorporé au modèle initial avec le minimum d'opérations possibles :

Différences entre XML' et XML/Processes = opérations à réaliser

Quand ce type d'échange peut intervenir n'importe où dans le système, **comment reconstruire sans erreur le modèle à partir du point de lecture** (e.g. XML/processes) ? Voila la difficulté initiale. La difficulté pour un modèle de description des données qui serait indépendant du point de lecture réside dans le sens de l'écriture, qui impose à tout texte de commencer et de finir en deux points bien déterminés.

L'idée centrale de la formulation modale consiste à considérer ces deux points (début et fin, ouverture et fermeture d'une parenthèse, etc.) comme étant partie d'un même objet – un « *lien* ».

Chaque texte se referme sur lui-même à la manière d'une bulle, et chaque hyperlien est comme une parenthèse entre les bulles : nécessairement à double-sens et en deux exemplaires. A la manière de **l'opération d'extraction-réinsertion** décrite dans le cas du copier-coller.

Pour aller plus loin

Comment cela fonctionne ? *Un diagramme de formulation modale est un arbre de décision binaire (BDD) dont les fonctions de construction (*left, right*) sont des bijections (*) – au lieu de fonction partielles (*). Par convention,* right *est une convolution (*right *o* right = Identité*) et l'étiquetage du réseau se fait sur ses orbites qui sont de taille inférieure à deux. Cet étiquetage se fait à la manière d'XML par un dictionnaire avec une entrée obligatoire – la « balise ».[Gradit-2014]*

(*) sans chaînes infinies

Un diagramme de formulation modale à une **représentation graphique pouvant être produite et lue par une machine**. Par exemple la situation d'échange entre deux logiciels décrite dans notre exemple s'écrit avec des conventions graphiques automatisables :

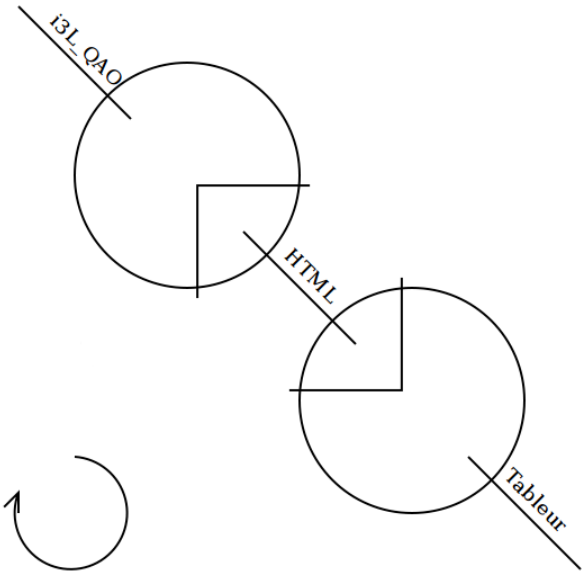


Figure 1 : La passerelle

Pour aller plus loin

Comment cela fonctionne ? *Les éléments en bijection sont les* intersections *entre les traits sans courbure (liens) et les traits avec courbure (bulles). La passerelle donnée en Figure 1 contient 8 intersections. Les bijections sont donnée sous forme de listes (left[3]=4) et les balises sont associées aux orbites de* right *:*

- *left→[2,3,4,1,6,7,8,5],*
- *right→[1,4,7,2,5,8,3,6] et {1}→i3L_QAO, {2,4}→Vue, {3,7}→HTML, {6,8}→Vue , {5}→Tableur*

Sous forme de « textes cycliques *» nous avons deux textes cycliques où chaque balise est citée au maximum deux fois :*

- *i3L_QAO* Vue_1 $HTML_1$ Vue_1
- *Tableur* Vue_2 $HTML_1$ Vue_2

Nous avons trouvé un façon d'écrire les graphes compatible avec la théorie de l'information : les résultats pleuvent... Faire le tri est la principale difficulté.

Un monde sans duplication

La « *passerelle* », bien que motivation initiale, n'est pas le diagramme le plus simple qui soit, et son sens n'est fixé que depuis 2013, jusque là le copier-collé

Ò

était sans formulation simple. En 2009, la passerelle n'existait pas, le copier-coller était interprété comme une duplication de donnée, analogue de la duplication de fichier, **une opération très compliquée en formulation modale**.

Pour aller plus loin

Pourquoi cela coince ? *La duplication de textes cycliques est interdite : cela violerait la condition que chaque balise est citée au maximum deux fois. Si nous dupliquons la ligne « Tableur »,* Tableur *est cité deux fois, ce qui fait sens, mais* Vue_2 *est cité quatre fois et surtout HTML est cité trois fois, et là... C'est un impair.*

- *i3L_QAO* Vue_1 ***HTML₁*** Vue_1
- ***Tableur*** ***Vue₂*** ***HTML₁*** ***Vue₂***
- ***Tableur*** ***Vue₂*** ***HTML₁*** ***Vue₂***

Pour pouvoir disposer de modalité d'échange entre les textes cycliques, l'idée retenue est analogue à celle exprimée par Ted Nelson en 1965. **Le système ne produit aucune donnée lors d'un copier-coller** d'une partie digne d'intérêt mais crée un lien vers la source de cette partie, et cette source mémorise de son coté un lien vers la cible.

Pour aller plus loin

Comment cela fonctionne ? *Soit deux textes cycliques sans formatage, tous les formatages sont à des fins d'éclairage. Dans la situation étudiée, l'auteur du deuxième texte veut recopier une partie digne d'intérêt dans le premier :*

- *Premier « il était une fois un texte qui avait une partie digne d'intérêt »*
- *Deuxième « Et un autre texte qui voudrait citer »*

Le copier-coller standard fonctionne sur tous les éditeurs de texte, vous sélectionner « une partie digne d'intérêt », vous la copier, et la coller à la fin du deuxième texte :

- *Premier « il était une fois un texte qui avait une partie digne d'intérêt »*
- *Deuxième « Et une autre texte qui voudrait citer* ***une partie digne d'intérêt*** *»*

Le résultat est qu'aucun des textes ne conserve l'information que la partie digne d'intérêt provient du premier texte, privant l'auteur de la partie digne d'intérêt de tout droit sur le deuxième texte.

⬧

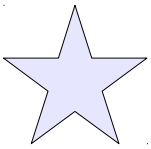
Pour réaliser le fonctionnement « liquide *», la première opération-clé intervient au moment de la « copie », qui devient l'identification par deux balises identiques de la section digne d'intérêt :*

- *Premier « il était une fois un texte qui avait »* ***Clip*** *« une partie digne d'intérêt »* ***Clip***
- *Deuxième « Et un autre texte qui voudrait citer »*

La deuxième opération-clé intervient au moment du « collage », qui devient l'ajout de deux balises permettant de signifier le lien à double-sens :

- *Premier « il était une fois un texte qui avait »* ***Clip View*** *« une partie digne d'intérêt »* ***Clip***
- *Deuxième « Et un autre texte qui voudrait citer »* ***View***

Ainsi seule l'image de la partie digne d'intérêt est échangée entre la source et la cible mais la donnée est conservée par son propriétaire. Nous avons substitué au « *copier-coller* » défectueux une opération déposée appelée « *Clip&View* ». Mais nous avons aussi déposé un brevet sur cette opération qui concerne un problème consécutif : **que se passe-t-il si le deuxième auteur souhaite contribuer à la partie digne d'intérêt ?**



Le brevet stipule que **toute action sur la partie dupliquée a le même effet que s'il avait lieu directement sur l'original, sauf que l'original reste intègre.**

Pour aller plus loin

Comment cela fonctionne ? *Repartons de la situation finale de l'encart précédent :*

- *Premier « il était une fois un texte qui avait » **Clip View** « une partie digne d'intérêt » **Clip***
- *Deuxième « Et un autre texte qui voudrait citer » **View***

Le « partage global » fonctionne comme si l'auteur « voyant » le « clip » avait les mêmes droits que le détenteur initial, et s'il désire modifier la partie digne d'intérêt, il le peut – jusqu'à pouvoir lui ôter son intérêt :

- *Premier « il était une fois un texte qui avait » **Clip View** « une partie digne d'intérêt » **« rendue caduque » Clip***
- *Deuxième « Et un autre texte qui voudrait citer » **View***

C'est ballot. Remarquez que la situation s'est arrangée par rapport au copier-coller car au moins le premier auteur peut détecter la destruction involontaire de son œuvre et prendre des mesures adéquates. Mais comme rien ne lui permet de détecter automatiquement la modification, et à moins de lire ou faire lire ses œuvres complètes à date régulière, la perte d'intérêt de son œuvre est irrémédiable. Pour remédier à cette situation il faut marquer la modification et faire du « partage partiel » [Gradit-2010] :

- *Premier « il était une fois un texte qui avait » **Clip View Append** « rendue caduque » **Append** « une partie digne d'intérêt » **Clip***
- *Deuxième « Et un autre texte qui voudrait citer » **View***

Ce qui signifie que d'une façon ou d'une autre, **l'original conserve la trace de toutes les modifications réalisée sur l'œuvre partagée**.

Ceci n'est pas possible de façon naïve en conservant toutes les versions possibles et imaginables, ne différant que de quelques caractères à des fins d'insertion ou de mise en forme. L'original doit conserver ses modifications **sous la forme de programmes** (Cf. « *Append* » dans l'encart).

Nous allons voir que c'est cette conséquence de la réflexion qui va motiver la réalisation de **Cargo**.

Construire des machines interactives

Pour qu'un tel système d'information « *liquide* » et sans duplication puisse fonctionner, il faut le structurer dès le début sur une logique où l'original n'existe plus et **où seul compte l'ensemble des modifications** :

apparences + interactions possibles = calcul(historique)

Pour que l'utilisateur fasse son œuvre, l'apparence doit permettre d'interagir avec le modèle en lui proposant des interactions possibles, **car le choix demeure l'apanage de l'humain** :

interaction = choix(interactions possibles)

Cette séparation est essentielle, car elle détermine ce qui est du ressort de l'humain et ce qui relève de la responsabilité de la machine : **présenter à l'opérateur humain une apparence cohérente de l'ensemble de ses décisions passées** et lui énoncer de manière ergonomique l'ensemble des décisions possibles.

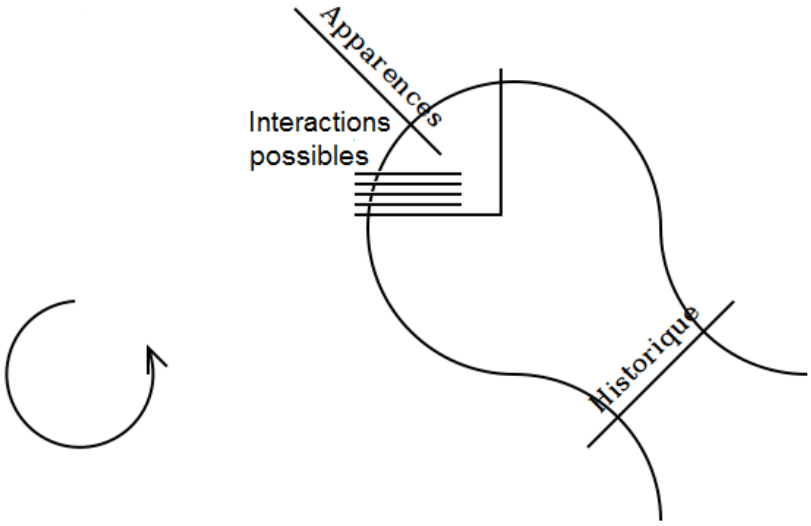


Figure 2 : L'usage

L'apparence cohérente est réalisée en deux phases, par le calcul sur l'historique d'une part, mais aussi **en donnant aux interactions la qualité de programme** qui leur permette d'agir de manière ciblée sur l'historique :

interaction : Historique → Historique

Ce faisant, nous posons que l'interaction est un programme qui transforme l'historique. Nous résumons ceci par la formule : « *Interagir avec une machine, c'est toujours programmer* ».

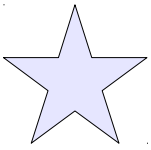
CARGO

Arrive l'urgence, les fonds pour le programme de R&D sont consommés, nous avons développé une maquette sur l'utilisation du brevet dans le cadre d'un tableur, mais il ne suscite par l'enthousiasme escompté. Il faut exploiter tous ces contenus pour en faire **un produit utilisable et exploitable dans le marché**. L'opportunité viendra d'une société commercialisant une version libre de la plate-forme MATLAB appelé SCILAB. Ce client a besoin d'un moteur programmable pour définir son assistant (ang. « *wizard* ») de gestion des préférences.

CARGO implémente alors notre conception d'une application interactive en travaillant sur modèle de donnée XML qui permet de **construire l'interface graphique par un HTML étendu**, la transformation étant codée par une XSL :

HTML = XSL(XML)

La difficulté consiste à ne pas reconstruire toute l'interface à chaque fois, ce qui produirait un effet de clignotement désagréable et pourrait nuire aux performances. Pour éviter cela, nous construisons en parallèle à l'HTML une VUE.



Chaque fois qu'un nouveau HTML est calculé, il est comparé à la VUE courante et **seules les modifications nécessaires sont appliquées**. La notion de « nécessité » est affinée au fil de l'eau.

modifications nécessaires = Différences entre HTML et VUE

L'HTML étendu contient l'ensemble des interactions possibles, sous forme de programmes modifiant l'XML. Lorsqu'une interaction est terminée, le modèle d'historique est mis à jour, l'HTML est généré, la vue est impacté et la boucle est bouclée.

Retour sur la passerelle

Dans l'application développée en CARGO pour le compte d'i3L, il existe une capacité d'import-export vers le tableur d'une suite Office. Nous avons déjà vu que cette capacité faisait écho **aux préoccupations initiant la formulation modale**.

Si l'importation est réalisée de façon naïve par construction d'un modèle équivalent et remplacement du modèle existant par le modèle généré, nous perdons toute capacité de déterminer les apports des différents acteurs. En revanche avec un algorithme de différentiation, nous pourrions construire un « ***programme raisonnable*** » **qui simulerait que les éditions réalisés** dans le tableur ont été réalisées dans l'interface et attribuer leur mérite à celui qui fait l'opération de copier-coller.

L'algorithme de différentiation est le cœur de CARGO, l'importation en bloc depuis le tableur d'un fragment modifié oblige aujourd'hui a coder le même algorithme dans un contexte différent. Les deux instances sont des images de l'archétype d'un algorithme central d'une plate-forme de formulation modale.

Mais **cet algorithme central se doit d'intégrer un transformateur XSLT** qui fonctionnerait avec les BDD bijectifs et non plus avec des BDD à fonction partielle, dont le différentiateur actuel, et celui de l'import Excel, seraient des utilisations. En outre, un tel code – réécrit – pourrait intégrer la notion d'impact et ne faire que les calculs nécessaires suite à une interaction et ouvrir la voie à l'utilisation de ces techniques sur de gros volumes de données et en particulier les textes et les documents où les conséquences en terme de propriété intellectuelle sont compris de tous.

Pour conclure, CARGO est un pur produit de la formulation modale avec les moyens du bord. A charge à lui de **trouver les clients capables de solliciter l'envie de financer** l'obtention du moteur final dont les plans sont achevés et dont toutes les applications CARGO existantes forment autant de cas de test de non-régression.

Bibliographie

[Gradit-2014] Travailler sur les réseaux

[Gradit-2010] Brevet de partage partiel d'application