Automated Intelligent Doc-Ops

Table of Contents

Contexte et généralités

Dans la plupart des petites et moyennes structures, la conduite quotidienne des opérations se traduit, au niveau informatique, par des actions récurrentes bien définies. Les suspects habituels sont :

- Injection de données par les utilisateurs
- Interaction des différents systèmes de données
- Production de documents marquant les étapes officielles du cycle de vie d'un projet (Bon de commande, Devis, Facture…)
- Cycles de validation (en interne, la compta valide une note de frais ; en externe, le client accepte un Devis),
- Suivi en temps réel et consultation à posteriori des archives. Cela inclut les alertes sur le temps, la consultation d'un projet en cours, et la consultation des projets terminés.

Dans beaucoup de cas, l'injection de données se fait au travers des applications métier (incluant les ERP, CRM et autres initiales), ainsi que d'interfaces web / desktop d'appoint développées spécifiquement.

Les interactions entre systèmes de données se font par transfert de fichiers, synchronisation manuelle de bases de données, des web-services, etc. La pénétration d'outils de type ETL qui centralise les communications entre systemes est extrêmement faible dans ce marché.

La production de documents, quand elle n'est pas faite depuis Word, est souvent embarquée par les applications métier au travers de modules couteux, propriétaires et peu personalisables. Dans l'idéal, cette fonction est assurée par une solution d'éditique, mais la pénétration dans ce marché reste inégale.

Les cycles de validation sont souvent des évènements un peu magiques, combinaison d'échanges de fichiers PDF mal versionnés par mail, puis de coups de fil et d'accords verbaux.

Cette réalité sur le terrain a l'avantage de fonctionner, mais quand on vient au suivi, les données sont tellement éparpillées qu'il est difficile d'avoir une image en temps réel de son activité.

La couche d'archivage fournie par une solution de GED adresse ce problème en partie, mais elle se limite aux documents « morts » : elle ne peut fournir que l'image finale des projets, et manque d'outils pour décrire l'activité courante.

A ce jour, la seule réponse existante est organisationelle : trier méticuleusement ses mails, tenir ses fichiers Excel à jour, ranger au cordeau les dossiers partagés et les Dropbox. Encore une fois, une solution qui fonctionne et ne coûte apparemment rien à mettre en place, mais qui vient avec tous les inconvénients des solutions manuelles : coût en temps, erreurs de saisie… La réalité fait qu'on ne peut pas toujours garantir 100 % de respect des process de ses clients, de ses employés, et même parfois de soi même. On a

donc, mécaniquement, des données opérationelles (et parfois des documents importants) qui disparaissent régulièrement dans la nature.

Ce qu'on y peut

AIDO a pour mission d'adresser ce problème en proposant un framework de développement structuré pour la production d'applications professionelles.

Ce framework inclut les couches suivants :

- Une couche de données capable de centraliser les actions et synchronisations sur différents systèmes de données
- Une couche d'ordonnancement capable de traiter les différentes étapes du cycle de vie d'un dossier, dans toutes leurs incarnations possibles (documents, interfaces de saisie, mail, courrier…)
- Une couche de composition capable de produire des documents simples dans des formats standards
- Un portail utilisateur dynamique permettant, pour chaque utilisateur selon ses droits :
  - de saisir ou modifier des dossiers
  - de suivre ses dossiers en cours (alertes de délais etc.)
  - de consulter ses dossiers archivés
- Des préconisations d'architecture, ainsi que les API et standards permettant de cabler les modules ensemble.

Pour chaque déploiement d'AIDO, il faudra donc développer spécifiquement :

- Les interfaces de saisie. Ces interfaces peuvent être web, desktop ou mobile, elles interagissent toutes avec l'API standard AIDO.
- Les modèles de documents et de courrier produits
- Les cycles de vie des dossiers
- La configuration des différents modules AIDO
- Eventuellement des plugins et connecteurs spécifiques pour accéder au système informatique du client

Philosophie

Les principes directeur d'AIDO sont :

- AIDO s'intègre souplement dans les systèmes existants, avec un impact minimal
- AIDO respecte les standards
- AIDO permet de configurer un nouveau client plus rapidement...
- … et plus fiablement
- AIDO fait fièrement partie de l'éco-système open-source

Dans la pratique, ces principes sont accompagnés des conventions et standards suivants :

Workflow non intrusif

Personne n'aime utiliser un progiciel. Personne n'aime utiliser sa solution d'archivage. Personne n'aime trier ses documents et faire de la saisie. Le fonctionnement d'AIDO vise à automatiser tout ce qui peut l'être, et limiter les interactions avec l'utilisateur final. Là où c'est possible, on préfèrera toujours des interactions non intrusives (échanges de mails, micro-interfaces ad-hoc) à la présence de l'utilisateur sur l'interface. *Vous ne saurez même pas qu'il est là.*

Modularité

Chaque brique du framework est indépendente, et peut être remplacée si besoin par une brique pré-existante du système d'information du client, ou une solution du marché.

« Convention over configuration »

Les paramètres par défaut doivent refléter les cas d'utilisation les plus fréquents. Chaque brique est configurable, mais on ne cherche pas à couvrir tous les cas marginaux : ceux-ci seront implémentés sous forme de plugins ou connecteurs. Ainsi, on isole la complexité des cas particuliers du coeur de la base de code. Keep It Simple, Stupid.

Spécialisation

Conformément à la philosophie Linux, chaque brique du système ne sait faire qu'une chose, mais la fait mieux que tout le monde. Corrolaire : si un outil standard du monde Linux (ou une fonction du kernel) existe, on l'utilise.

Testabilité

Chaque module, et chaque configuration qu'on lui concocte, doivent être testables individuellement et automatiquement. Pour chaque client déployé, on aura un jeu de données de test avec un résultat connu et un moyen automatique de produire un environnement de test identique à sa production. Ceci permettra d'effectuer des tests de non-régression en situations de maintenance, de reprise de code, et de mise à jour

du système d'exploitation ou de la base de code AIDO.

« Free as in freedom »

Dans la mesure de la raison, les composants qui peuvent l'être seront publiés sous des licences libres afin de participer à la création de standards de gestion documentaire.

De même, on aura à coeur de s'impliquer dans le développement et la maintenance des outils clefs utilisés par AIDO.

Sous le capot

Couche de données

La couche de données est en cours de développement sous le nom de RestQL : https://github.com/Dam-Buty/RestQL.

RestQL se connecte à une ou plusieurs bases de données et fournit une API REST standard permettant d'éditer et consulter ces bases de données de manière transparente. RestQL s'occupe en coulisse de toutes les problématiques de connections aux différents serveurs, de création de requêtes dans les divers dialectes SQL, et du maintien en synchronisation des bases dédoublées. Des fonctionnalités plus avancées sont également disponibles, utiles pour les environnements multi-bases et multi-systèmes.

API REST

L'API REST offre un point d'entrée unique pour l'édition et la consultation des bases de données. Elle permet d'adresser de manière standard la plupart des cas d'usage de base de données, comme par exemple :

Méthode	Adresse	Equivalent SQL
GET	http://server/**user**	SELECT * FROM user
GET	http://server/**user**/**12**	SELECT * FROM user where user_id = 12
PUT	http://server/**user**/	INSERT INTO user – *les champs de données sont envoyés dans la requête PUT*
POST	http://server/**user**/12	UPDATE user SET – *les champs de données sont envoyés dans la requête POST*

Sélection des sources de données

Dans un contexte multi-bases, RestQL permet de donner des « notes de vitesse » aux différentes sources. Les requêtes de consultation types SELECT peuvent ainsi être exécutés sur le serveur le plus proche, le plus rapide, le moins coûteux…

Certaines des tables peuvent également être conservées en cache dans la RAM du serveur. Ceci permet d'éviter les appels répétitifs aux serveurs pour des données qui changent peu, comme les paramétrages ou listes de catégories.

Le gain de performance est évident, et on pourrait même imaginer pour certaines volumétries de fonctionner 100 % en RAM. *Ca ne coûte pas si cher qu'on le croit…*

Options de synchronisation

Dans un contexte multi-bases, les opérations destructives (type INSERT, UPDATE, DELETE) sont effectuées sur toutes les bases contenant la table affectée. Le comportement de RestQL est configurable en cas d'échec d'une, de plusieurs ou de toutes les bases.

Les bases de données externes (susceptibles d'être modifiées hors AIDO) peuvent être surveillées périodiquement ou à la demande, et faire autorité sur des tables spécifiques. De même, un module de correspondance pourra gérer les cas où des colonnes ont des noms disparates dans les différents systèmes.

Connection Pooling & budgeting

Selon les particularités de la configuration locale, RestQL peut moduler le trafic vers les serveurs de base de données : connection pooling pour limiter les connections concurrentes, connection persistente ou à la demande, etc.

Conception modulaire

Les particularités des différents systèmes de bases de données (MySQL, SQL Server, SQLite…) sont gérées par un système de plugin, permettant de coder de nouveaux transports très rapidement.

Dans l'absolu, RestQL est conçu avec les systèmes *SQL en tête, mais on peut réaliser des plugins pour d'autres types de sources de données, des webservices aux fichiers CSV et plein texte.

Couche d'ordonnancement

C'est là que la magie opère. La couche d'ordonnancement est alimentée par des injecteurs, qui peuvent être des boites mails surveillées, des dossiers, des interfaces, des évènements sur base de données (via RestQL) etc.

Un injecteur amène de nouvelles données dans le système. Ces données peuvent inclure un fichier PDF ou autre, mais pas nécessairement. Trois parcours sont possibles :

- Les données sont nouvelles et entrainent la création d'une Entité (exemple : saisie d'un nouveau projet dans une interface)
- Les données concernent une Entité existante et l'Entité est mise à jour (exemple : le client renvoie un contrat signé)
- Ce n'est pas clair… il y a un doute… les données seront présentées aux utilisateurs les plus pertinents pour être identifiées (exemple : le client renvoie un contrat scanné de travers, l'OCR n'arrive pas à lire les informations…)

A tout moment, chaque Entité est dans un bac, en attente de données ou d'un document. Quand les données ou interactions nécessaires arrivent, AIDO se charge de faire avancer l'Entité dans le ou les bac(s) suivants de son cycle de vie.

L'entrée dans un bac peut entrainer des actions automatiques du serveur, telles que des actions bases de données, la création d'alertes de délais, l'émission de documents, de mails etc...

Equivalence des canaux

Un principe à garder en tête est que tous les canaux de communication sont équivalents pour AIDO. Le contrat peut être imprimé, signé sur papier et scanné, mais il peut aussi être signé électroniquement sur une interface web. Un devis peut être généré depuis le portail de la compagnie, mais aussi offline dans un formulaire PDF qu'on renvoie par mail dans le système. Ou même pourquoi pas par une conversation SMS avec le serveur ?

Les stratégies pour les sorties de système sont simples :

- Tous les fichiers PDF émis par le système contiendront une pièce jointe de tracking permettant de les réidentifier quand ils reviennent dans le système par mail ou upload. Les ruses de sioux habituelles permettront également de tracker avec une relative fidélité l'ouverture du document.
- Tous les fichiers PDF destinés à être imprimés et remplis sur papier incorporeront un identifiant unique sous forme de datamatrix à forte correction

Tout est dans le contexte

En accord avec le principe « Personne n'aime utiliser sa solution d'archivage », AIDO exploite au maximum le contexte afin de reconnaître et classer les documents de manière automatique.

Grace à cela, archiver un document dans AIDO peut devenir aussi simple que de mettre son serveur AIDO en copie d'une conversation mail, ou copier un document sur un dossier qu'il surveille. Sans autre instruction, AIDO peut identifier l'Entité concernée, savoir ce qu'elle attend, et lui fournir le document.

On peut imaginer un workflow sans les mains tel que :

- Le commercial saisit un devis sur une interface AIDO
- AIDO génère le PDF et l'envoie au client de la part du commercial, et se met en copie
    - Le mail contient le devis en pièce jointe, et un bouton « accepter » dans le corps du mail
- Le client accepte le devis
- AIDO génère un bon de commande et l'envoie dans la même conversation
- Le client renvoie le bon de commande signé et scanné
- Le commercial est notifié et la comptabilité reçoit une copie

Il n'y a pas besoin d'intervention manuelle car à tout moment, AIDO sait quelles entités sont en attentes de documents entrants ou sortants. L'adresse de l'expéditeur et du destinataire lui suffisent pour déterminer le reste.

Ainsi, même si l'on n'arrive pas à identifier à 100 % le document, on va souvent pouvoir réduire le nombre de possibilités et estimer avec précision quel utilisateur est le plus susceptible d'identifier le document.

Couche de composition

La couche de composition s'appuie sur deux standards :

Le format PDF, omniprésent sur nos postes et doté de fonctionnalités modernes.

La spécification « CSS Paged Media Module Level 3 » du W3C. Cette norme est la base des formats utilisés par les liseuses électroniques types Kindle (Mobi, ePub...). Dans les faits, elle permet une conversion fiable et robuste du HTML vers des formats « à pages » comme le PDF et le papier.

Le projet open source le plus mature respectant cette norme est wkhtmltopdf. Basé sur le moteur de rendu de Chrome, il est standard, à jour, et performant sur un serveur sans interface graphique.

Le CSS3 paginé permet de gérer la plupart des cas d'école de la composition de documents, comme les tableaux complexes sur plusieurs pages, les répétitions d'entêtes, les notes de bas de page etc...

La problématique de production de documents PDF devient donc une problématique de production de documents HTML, qui peut être résolue avec un framework de templating côté serveur qui reste à choisir.

Bien sûr, la composition de mails et de SMS est de la responsabilité de cette couche.

Portail utilisateur

Le portail utilisateur donne accès, selon les droits de chaque utilisateur, à trois mondes différents :

- Les interfaces métier qui auraient pu éventuellement être développés pour le client
- Le tableau de bord de l'utilisateur, qui lui présente ses dossiers en cours
- La salle des archives qui contient les dossiers terminés

Interfaces métier

Afin d'assurer une correcte interception des évènements sur les bases de données, ces interfaces doivent s'appuyer sur le backend RestQL. Il n'y a pas de préconisation particulière sur les frameworks de frontend à utiliser pour ces interfaces. Ces interfaces peuvent par ailleurs être des applications desktop ou mobile, tout système capable d'envoyer des requêtes HTTP est capable d'interagir avec AIDO.

Tableau de bord

Le tableau de bord va rassembler toutes les Entités suivies par l'utilisateur, et les lui présenter sous forme d'une mosaïque de widgets, les plus urgentes et les plus facilement résolvables étant groupées vers le haut.

Pour les Entités qui attendent peu de données, ou seulement un document, on essaiera de présenter des widgets actionables (micro-formulaire, upload) directement plutôt que des liens vers d'autres parties de l'application.

Salle des archives

La salle des archives permet de rechercher et consulter les dossiers terminés.

AIDO ne jette rien. Chaque entité contient tous les documents qui ont composé son cycle de vie, ainsi que l'historique des actions. Mails, IP, contexte, tout est historisé !