e-Babel projet					
Contexte:	Projet d'harmonisation virtuelle des formats et des codages des supports d'informations.				
Référence:	EBPROJ-01	**Date:**	2008-09-02 2008-09-10	**Version:**	0.1
Auteur(s):	Sievering Johann	**Réviseur(s):**			
Contenu:	Description du projet e-Babel.				
Mots clés:	e-Babel, projet, format, donnée, protocole, base de connaissances, composant, sémantique				
Type:	Projet	**Destinataire(s):**	Andreas Schweizer Morel Raymond		
Persistance:	\\Cmyle-dev\cmyle_work\Projets\EnCours\eBabel\eBabel projet ver 0.1 FR (2008-09-02).doc				
Sécurité:	**Confidentiel** : document en cours d'élaboration. Ce document est un draft.				

Table des matières

Projet e-Babel

description du projet e-Babel

Johann Sievering

Résumé.
Le projet e-Babel est un projet qui a pour objectif de pérenniser les équipements, les logiciels et les données indépendamment des technologies les sous-tendant. L'objectif est de permettre en tout temps d'accéder à des ressources du passé et dus utiliser avec les moyens présents. Un autre point clé est la possibilité de mettre en relation des données de formats différents et de supports hétérogènes.
L'architecture e-Babel est par nature collaborative et permet d'impliquer une communauté d'acteurs pour la mise à disposition des ressources. Ce qui permet de démarrer avec quelques composants et réaliser une montée en charge pour développer à large échelle le réseau. Les utilisateurs auront un accès homogène à tous les équipements et à toutes les données.
Le paradigme de base permettant de réaliser une telle architecture est la délégation des compétences sur les composants eux-mêmes en utilisant leur propre technologie.

1. Introduction

Ce document est destiné à présenter l'architecture générale d'e-Babel, les étapes principales et les ressources nécessaires à l'élaboration de la première phase du projet. Il concerne un public intéressé par les aspects projets plus que par les aspects techniques. Pour les compléments ou les détails techniques, se référer aux documents annexes.

> *L'objectif d'e-Babel est de pérenniser les informations indépendamment des technologies les sous-tendant. C'est-à-dire conserver et rendre utilisable en tout temps la valeur des contenus et des processus de calculs[1] du passé et de ceux à venir.*

Dans le monde de l'information, il n'est pas possible de conserver une donnée sans un *support*[2]. La donnée est également fondamentalement liée à son *codage*[3]. Ce sont précisément sur ces deux notions que le principe e-Babel est construit (principe que nous les détaillerons plus bas) :

- Comment des supports hétérogènes peuvent-ils être harmonisés ?
- Comment rendre compatible des codages de différentes natures ?

La première question adresse la problématique des équipements de différentes natures, de différents constructeurs et de différentes époques. La seconde question adresse la problématique de la représentation de la donnée produite dans différents formats avec différents codages par des logiciels de sources variées dans des équipements de diverses architectures.

[1] Par processus de calculs, il est sous-entendu « programmes ». Une même donnée peut être produite par plusieurs programmes ou versions et être lue soit par les mêmes programmes, soit par des programmes compatibles ou des versions supérieures.
[2] Le support de la donnée peut être quelconque : écrit, gravé, électronique, mécanique, etc. Il doit permettre d'inscrire une donnée et de la relire sans perte. Dans notre contexte, nous parlons de supports compatibles avec les moyens informatiques. Par exemples : bande magnétique, disquette, disque dur, etc.
[3] Le codage est l'expression d'une donnée dans un format adapté au support de transmission ou de stockage. La fonction de codage doit être réversible de telle manière que la donnée puisse être restituée exactement.

Des réponses immédiates peuvent être mises en place et existent :

- Transposer toutes les données sur un support commun (par exemple des DVD) avec un format standard (par exemple PDF) ;
- Développer un protocole « universel » qui mettrait en commun les logiciels au moyen d'un ensemble de connecteurs de traduction interprétant d'un côté les logiciels hétérogènes et de l'autre le protocole (par exemple utilisation d'un canal de communication et XML) ;
- Ecrire des composants distribués qui peuvent être interrogés (par exemple les web services) ;
- Il existe des dizaines d'autres solutions architecturales et techniques dont certaines répondants très bien au besoin actuel.

Mais toutes ces solutions sont élaborées de manière ad hoc et résolvent à chaque fois une problématique locale. Ces solutions, bien que parfois très élégantes et efficaces, ne permettent pas d'adresser de manière générique le domaine de l'information et sont pour la grande majorité « présent-centré[4] ».

De plus pour des raisons économiques ou stratégiques, seules les domaines ayant une rentabilité potentielle captent l'attention des directions et des équipes de développement. Tous les autres équipements, logiciels et données restent absents des projets actuels. Or il existe un grand nombre d'équipements, de logiciels et de données qui ensemble participent à la connaissance générale et proposent des solutions ainsi que des informations pertinentes aujourd'hui ou qui ont des valeurs historiques.

Nous constatons qu'il existe une communauté d'acteurs qui sont intéressés par la mise à disposition des ces informations du passés et qui sont individuellement prêt à participer à mettre à disposition leurs ressources. Le projet e-Babel va exactement dans ce sens : il propose une architecture où chacun peut participer à la construction d'e-Babel par la mise en commun raisonnée de ses des contributions.

Le projet e-Babel est une architecture constructiviste dans laquelle une multitude d'acteurs délocalisés peuvent participer à la construction d'une banque d'équipements actifs, de logiciels fonctionnels et de données accessibles.

Ce que nous recherchons dans ce projet, ce n'est pas l'élaboration de *LA* solution, mais de faire une proposition d'architecture organique qui possède une cohérence intrinsèque, qui laisse la latitude à chaque objet d'information de gérer ses contenus et qui puisse être pérenne. Ainsi dans l'approche e-Babel que nous présentons, il n'est ni besoin de trouver un dénominateur commun forcément restrictif, ni besoin de modifier ou convertir les supports. Ce paradigme permet en outre de mettre en place une communauté d'acteurs actifs qui développement le réseau de ressources.

[4] *Présent-centré* : j'entends par là que les solutions sont conçues pour adapter les données du passé à nos technologies présentes. Qu'elles édictent des normes pour rendre compatibles les futurs développements selon des vues technologiques actuelles. Et qu'elles formalisent les applications présentent afin qu'elles soient aujourd'hui compatibles.

2. Principes de base

Considérant que l'évolution des systèmes d'information et de l'informatique en générale nous a légué un ensemble hétérogène et souvent non compatible entre eux d'équipements, de logiciels et de données. Et considérant que d'autres équipements, logiciels et données seront produits dans le futur ; l'équation de ce projet paraît inextricable. En effet, si nous pouvons envisager des solutions avec ce qui est connu et ce que nous avons à disposition, nous ne savons rien des évolutions et des futures nouveautés. Il est donc illusoire de rechercher une solution absolue générique englobant toutes les technologies.

Pour les technologies passées et présentes, il serait possible d'envisager un ensemble de « compilateurs[5] » rendant lisible les anciens formats dans notre contexte actuel. Mais cette stratégie n'est pas transposable pour les futures évolutions (que nous ne connaissons pas encore). Pour adresser les technologies à venir, un des moyens à disposition est d'édicter des normes fortes qui imposent une compatibilité à la source obligeant tous nouveaux développements à intégrer cette norme.
Ces deux approches fonctionnent bien et sont mêmes mises en œuvre actuellement. Cependant, la première approche demande un travail considérable qui aboutira à une compatibilité avec notre présent, mais pas forcément avec les futurs formats. La seconde approche restreint fortement l'amplitude des développements futurs en imposant une norme qui sera peut-être mal adaptée dans le contexte des nouvelles technologies.

Une solution organique

Les problèmes soulevés en introduction ont pour origine une *vue centralisée*. C'est-à-dire que l'hypothèse implicite est de décréter que l'architecture doit être organisée autour d'un seul concept général ou d'un seul système qui aurait tout prévu d'avance pour son fonctionnement correct.

e-Babel propose duver cette hypothèse restrictive implicite en déléguant la responsabilité des traitements à chaque composant du système. Le seul point commun permettant de rendre cohérente l'architecture est le *langage de communication* qui est par nature indépendant de la technologie sous-jacente.

> *La responsabilité des traitements est délocalisée dans les composants locaux. Les composants communiquent au moyen d'un langage, ce qui assure la cohérence de l'architecture indépendamment des technologies en jeu.*

Cette vue organique pose par hypothèse que chaque composant, lié à un seul équipement, un seul logiciel ou une seule données, peut être développé dans la technologie de l'équipement, du logiciel ou de la donnée auquel il est attaché. Ainsi les composants sont dans la position la plus adéquate pour prendre en charge tous les détails de la mise à disposition de la ressource. Ce paradigme est valable pour tous les éléments du passé, mais est également applicable pour les éléments futurs. Nous avons donc un modèle pérenne quelque soit l'objet d'information envisagé.

[5] *Compilateur* : programme traduisant un texte source (original) en un autre texte cible (résultat de la transformation selon des règles). Sans précision, un compilateur désigne souvent un programme permettant de traduire un code informatique (par exemples : C++, Pascal, Java, etc.) en un code utilisable par un ordinateur. Dans notre contexte, il s'agit d'un programme permettant d'adapter un ancien format de données à un format compatible avec nos machines et logiciels modernes.

Le modèle doit permettre d'adresser tous les équipements, les logiciels et les données de la même manière qu'ils soient du passé, du présent ou à venir.

Un corolaire de ce modèle est l'aspect constructiviste de son développement. Si chaque objet d'information[6] est pris en charge séparément, alors il est possible de développer la structure e-Babel par projets indépendants et ouvrir une communauté de développement. La collaboration entre tous les objets d'information est assurée par un langage et donc il est possible mettre en place des groupes spécialisés.
Par exemple, dans le cas de la lecture d'un fichier produit par WordPerfect sous MS-DOS dans un M24 (Olivetti) avec une conversion en PDF, quatre objets d'informations peuvent collaborer :

M24 (OS : **MS-DOS** (exécute : **WordPerfect** (ouvre (<*mon fichier*>))))
⇒
Résultat (**PDF** (<*mon fichier*>))

Ces quatre objets peuvent être produits par quatre équipes travaillant sur le hardware (M24), sur les systèmes d'exploitation (MS-DOS), sur les logiciels (WordPerfect) et sur les formats de données (PDF).

La communication entre l'ensemble de ces composants doit être réalisée au moyen d'un langage symbolique. Il est essentiel que la communication ne transporte aucune notion technique. Toute notion technique restreint la compatibilité et impose une connaissance a priori du composant source sur le composant cible. La contrainte de cette approche est que chaque composant recevant un message l'interprète conformément à l'attente de l'émetteur. Il faut donc qu'une représentation commune existe entre les émetteurs et les destinataires.

L'utilisation d'un langage symbolique permet de rendre cohérent et de structurer l'architecture décentralisée d'e-Babel en s'affranchissant de l'approche omnisciente des composants. C'est-à-dire que dans les approche actuelle, chaque composant voulant communiquer et/ou collaborer avec un autre doit avoir une connaissance a priori des aspects techniques de l'utilisation de la ressource. L'utilisation d'un langage symbolique repose sur la notion de performatif qui sera interprété dans les composants selon leur propre technologie et logique.

L'utilisation d'un langage symbolique et de performatifs permet à chaque composant d'interpréter les messages selon sa propre technologie et ses propres logiques. Ce qui les libère de la contrainte d'omniscience technique.

Cette représentation commune reposant sur un langage est le seul point fixe liant les composants. Il faut donc choisir attentivement ce langage afin de ne pas poser un nouveau point fixe restrictif. Dans le projet e-Babel, je propose d'utiliser la notion d'ontologie[7] et du langage ACL[8]. Cette structure permet de modéliser les domaines en leur donnant un sens commun interprétable par tous les composants. Ainsi il est possible de modéliser les objets d'information passés et intégrer au fur et à mesure les nouveaux objets d'information. A nouveau cette structure a une nature constructiviste.

[6] *Objet d'information* : j'utilise ce terme pour indiquer de manière générique un équipement, un logiciel ou une donnée. Ceci afin d'alléger le texte et de considérer l'équipement, le logiciel ou la donnée comme une entité ayant des propriétés et sur laquelle il est possible de réaliser des actions. (Je faits référence à la notion d'objet en programmation ou d'composant informatique s'il y a proactivité).
[7] *Ontologie* : ontologie au sens informatique du terme. Une ontologie est une description du domaine constituée par un vocabulaire et un ensemble de théories donnant un sens aux mots du vocabulaire par une interprétation.
[8] *ACL* (Composant Communication Language) : langage formel basé sur les logiques du premier ordre pour l'échange et l'interprétation de messages entre composants. La théorie sous jacente sont les « actes de langage ».

La représentation commune des objets d'information est le point fixe de l'architecture e-Babel. L'utilisation d'une ontologie construite sur OWL permet de conserver les propriétés souhaitées et est compatible avec le futur Web Sémantique (W3C[9]).

En résumé, les avantages l'approche e-Babel sont en substance :

- Chaque composant peut être développé dans la technologie de l'objet à compiler ;
- Possibilité d'impliquer une communauté de développeurs (travail distribué) ;
- Les objets du passé peuvent être adaptés et ceux du futur peuvent intégrer nativement le composant spécialisé ;
- De nouveaux objets plus adaptés peuvent prendre le relais de ceux devenus obsolètes sans remise en cause de l'ensemble ;
- Le langage peut être étendu dans le futur tout en gardant une compatibilité avec le passé. Il pourrait même être modifié si le vocabulaire est conservé (car indépendant de la technologie) ;
- Les composants peuvent communiquer entre eux et réaliser des tâches collaboratives complexes sans nécessité de recréer l'ensemble à chaque fois ;
- Les charges de travail et les coûts peuvent être distribués.

Cette solution peut donc être qualifiée d'*organique*, car elle est constituée de composants ayant chacun un rôle à jouer et pouvant être remplacés ou ajoutés dynamiquement. Ce qui donne à cette structure ses propriétés *dynamiques*, *évolutives* et *collaboratives*.

Architecture générale

Cette section présente l'architecture générale du projet e-Babel.
L'architecture e-Babel est composée d'un ensemble de composants autonomes et communicants. Chaque composant est un micro-environnement encapsulant un objet d'information. L'*autonomie* est requise pour la délégation de la compilation, l'accès au composant et pour les échanges de messages.

Un composant est constitué de :

- l'équipement, le logiciel ou la donnée ;
- l'interpréteur dans la technologie cible ;
- l'ensemble d'interfaces conçu spécifiquement pour l'équipement, le logiciel ou la donnée ;
- l'interface de communication inter composants

e-Babel est structuré :

- par un ensemble de composants présentant des interfaces de communication compatibles ;
- par un réseau de communication ;
- par des points d'accès utilisateurs.

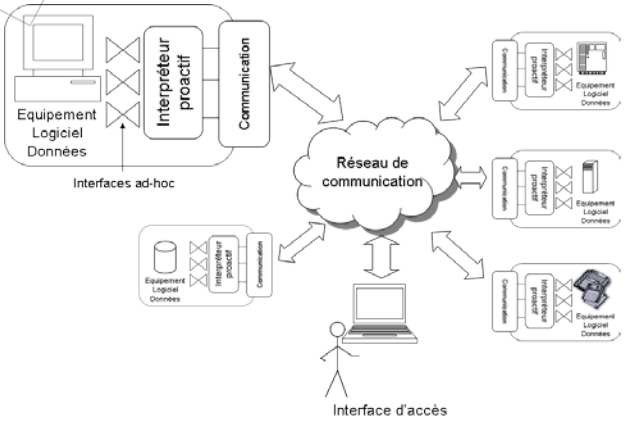

Figure 1 : architecture e-Babel.

[9] *W3C* (World Wide Web Consortium) : organisme international de normalisation des technologies du web.

Cette architecture est compatible avec la description du projet faite en introduction. En effet, les composants peuvent être de diverses natures, ils peuvent être développés individuellement et indépendamment, ils peuvent être insérés ou retirés sans remise en cause du système et l'accès est possible de n'importe quel point.
L'indépendance de la technologie est obtenue par le fait que les messages sont des textes qui peuvent être transportés dans n'importe quel support.

Etat de l'art

L'architecture e-Babel décrite doit être implémentée dans une ou plusieurs technologies pour pourvoir être concrètement mise en œuvre. Il existe divers candidats et solutions pertinentes. Cette section passe succinctement en revue les plus populaires du moment et en dégage les points forts et les faiblesses *par rapport au projet e-Babel*.
Remarque : ce n'est pas une analyse concernant les SI[10] en général.

> Le projet e-Babel doit reposer le plus possible, sinon complètement, sur des normes et des standards internationaux et reconnus. Si possible ces standards ne doivent pas être propriétaires mais libre.
> Les standards du W3C sont largement intégrés dans l'architecture e-Babel.

SOA (Service-Oriented Architecture)

Cette technologie à fait faire un grand saut dans le monde des SI modernes en proposant une vue *orientée service* plutôt qu'*application*. C'est-à-dire que les composants d'un SI diffusent leurs compétences dans un annuaire (UDDI[11]) et permet ainsi aux autres composants dus utiliser sans avoir à reprogrammer les fonctions déjà disponibles. L'utilisation des services est indépendante des technologies de chaque acteur, car les flux d'informations sont basés sur des messages XML[12].
Le principe repose sur un échange de type producteur – consommateur (ou fournisseur – client).

e-Babel peut s'identifier à une architecture SOA dans le sens que c'est une architecture de type constructiviste qui se développe par l'ajout successif de services indépendamment duur technologie et des équipements sur lesquels ils fonctionnent.

Le principe de base de cette indépendance est l'inscription des services (publication) dans un annuaire (UDDI). Le client peut accéder à ce service par une requête. Dès que le service est trouvé et accepté, une mise en relation permet son utilisation (SOAP[13]), voir figure 2.

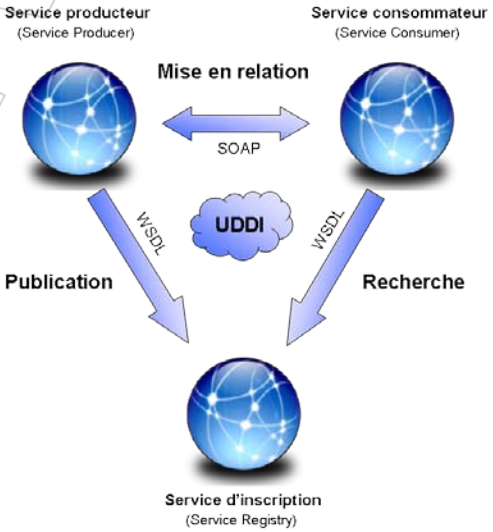

Figure 2 : Exemple d'une architecture SOA.

[10] *SI* : système d'information.
[11] *UDDI* (Universal Description Discovery and Integration) : annuaire permettant d'inscrire et de retrouver des services web.
[12] *XML* (eXtensible Markup Language) : langage de balises permettant de définir librement des structures de données et de transmettre des données sous forme de texte. L'intérêt principal est la séparation faite entre la présentation (par exemple HTML) et le contenu (XML). Ce langage est actuellement très répandu et ne cesse d'accroître ses domaines d'utilisation.
[13] SOAP (Simple Object Access Protocol) : protocole technique permettant à un composant d'utiliser les fonctionnalités d'un autre (pour autant que l'autre composant en ait permis l'accès par une description et une autorisation).

Les messages sont des textes et ne dépendent ainsi pas des technologies des acteurs intervenant dans l'utilisation des services.
Cette technologie est également indépendante du lieu et des ressources locales. Le service peut se trouver sur un serveur possédant la configuration requise à la mise en œuvre des fonctions proposées. Le client peut résider dans des systèmes légers et accéder à des fonctionnalités qui seraient autrement hors de sa portée. Les services peuvent également se greffer dans des systèmes préexistant et proposer leurs fonctions sans modifier la structure existante.

ESB (Entreprise Service Bus)

Un bus de données est une architecture permettant à des applications hétérogènes d'interagir entre-elles par l'entremise d'une structure technique que l'on nomme le « bus ». Cette structure est basée sur des services qui lui sont reliés au moyen de connecteurs. Les connecteurs ont un rôle de traducteur et de compilateur. Le *traducteur* permet de passer d'une normalisation à une autre, le *compilateur* permet de passer d'une structure de donnée à une autre. Ainsi toute application, nouvelle ou héritée, devient compatible avec le bus et peut être utilisée par les autres applications : c'est précisément ce que l'on désire obtenir dans le projet e-Babel.

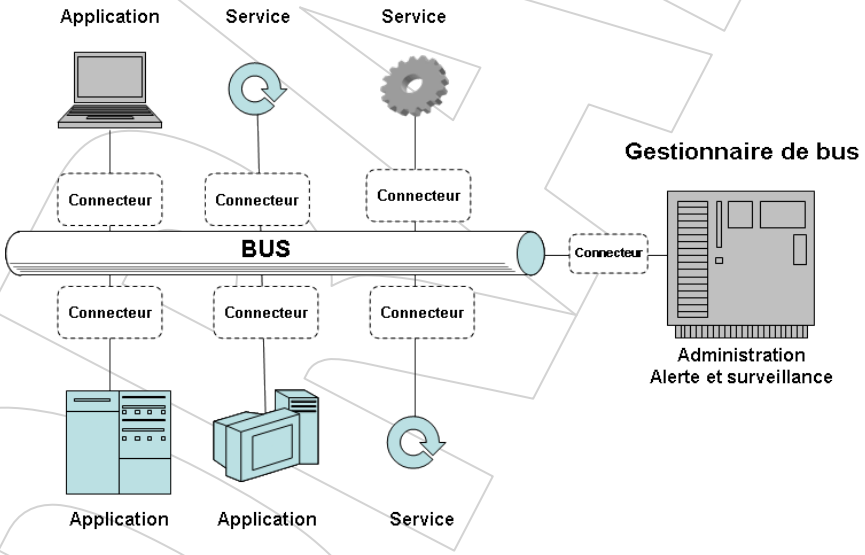

Figure 3 : architecture orientée bus.

Le bus a pour rôle l'acheminement et le contrôle des messages normalisés dans un langage commun entre des applications qui peuvent être hétérogènes et technologiquement différentes. La seule contrainte, mais parfois de taille, est que chaque application et chaque service puisse traduire ses flux de communication dans le langage commun du bus et vis versa (connecteur). Cela impose deux activités : conversions au bon format (compilation) et interprétions du contenu des messages (traduction).

SOA et ESB complémentaires

Les deux approches, SOA et ESB répondent aux contraintes que nous avons développées en introduction : *indépendance technologique*, *interactions de composants hétérogènes* et *intégration des interfaces*. Mais elles ne répondent pas complètement aux problèmes des développements ad hoc qu'il faut encore réaliser pour chaque environnement et à la centralisation des processus qui restent situés au niveau du gestionnaire du bus. L'utilisation des services conserve son mode client-serveur. C'est donc au gestionnaire de bus et aux clients que revient la responsabilité de diriger les opérations, il n'existe pas de *« motivation interne »* ou de notion d'*« autonomie »*. C'est-à-dire qu'une application ou un service ne peut pas naturellement *« décider »* d'engager une action en fonction de son contexte ou de son état.

> *Dans le contexte du projet e-Babel, ce manque de proactivité ne permet pas l'utilisation directe des architectures SOA et ESB. C'est pourquoi, je propose d'étendre cette notion à celle que j'appellerais :* ***« système orienté besoin »****.*

Système orienté besoin

Un *système orienté besoin* est un système ayant la propriété de s'« auto-organiser » pour aider, guider, l'acteur à la réalisation de ses buts. Par rapport aux *systèmes orientés service*, qui exige des utilisateurs une connaissance précise des fonctions à mettre en place pour atteindre leurs objectifs, le système orienté besoin propose à l'utilisateur un sous-ensemble de fonctionnalités pré-connectées répondant à un critère contextuel.

> *Le système orienté besoin permet aux acteurs de retrouver l'ensemble des composants qui lui sont utiles pour atteindre son objectif et du mettre en place leurs interactions sans en connaître les détails techniques.*

L'acteur (humain ou composant) ne voit que les services dont il pourrait avoir besoin en fonction : de son *contexte*, de son *profil* et du *but* qu'il poursuit. Ainsi le système global peut se développer par l'ajout d'autant de fonctionnalités que les développements peuvent proposer sans que les acteurs se trouvent confrontés au phénomène de saturation fonctionnelle. C'est-à-dire de se retrouver avec des listes de choix tellement grandes que leur utilisation en devient difficile, voir impossible.

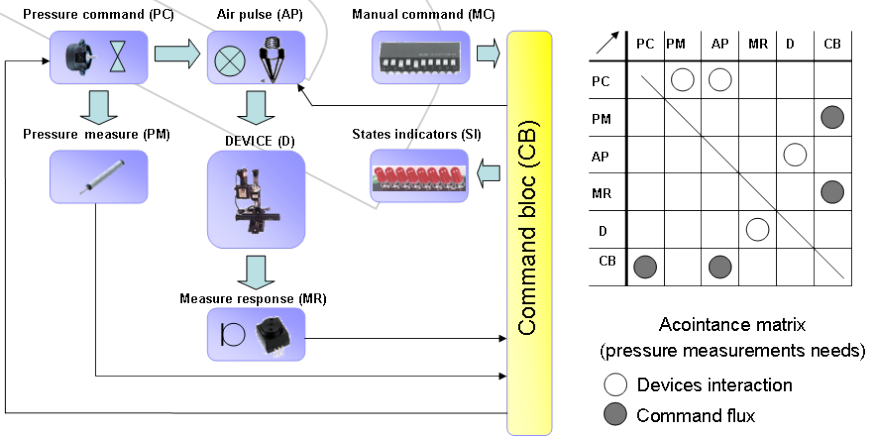

Figure 4 : exemple de relations d'accointances.

Le système orienté besoin tisse des relations d'accointances entre les composants, ce qui permet de créer des structures de *coopérations/collaborations* spécialisées entre les composants pour la résolution de tâches communes.

Par exemple un composant qui serait spécialisé dans la lecture des disquettes 5'1/4 peut structurer ses interactions de telle manière à répondre à un composant pouvant lire des fichiers WordPerfect. Le seul fait d'évoquer ce besoin, ou être dans une configuration prédéterminée, active l'ensemble des fonctionnalités requises à la réalisation du ou des buts assignés de ce besoin : lire de fichier WordPerfect sur la disquette 5'1/4.

3. Planification du projet

Cette planification décrit les étapes clés pour la réalisation de la première partie du projet e-Babel:

> *Etude d'une architecture multi-composants pour la pérennisation des équipements, des logiciels et des données avec module web de présentation temporel.*

Work Package	Jalons
WP-1 : modélisation de l'architecture / choix technologiques	
	Etat de l'art ; Analyse de la problématique et définition du domaine d'action ; Modélisation générale d'e-Babel (schéma bloc) ; Choix des technologies à utiliser dans le cadre du projet e-Babel ; Intégration de la technologie choisie dans l'architecture ; Liste des équipements, des logiciels et des données à traiter dans cette première phase.
WP-2 : conception et réalisation de l'interface web temporel	
	Conception de l'interface d'accès aux services ; Design et mise en place des éléments graphiques ; Gestionnaire des interactions ; Réalisation de l'interface web ; Saisie des données de base.
WP-3 : architecture générique du SMA et des objets d'information	
	Mise en place des scénarios de fonctionnement ; Spécification des protocoles internes ; Conception des gestionnaires d'objets d'informations ; Spécification de l'architecture des composants e-Babel ; Schéma détaillé du fonctionnement e-Babel.
WP-4 : formalisation du langage de communication	
	Spécification du langage ACL dans le contexte e-Babel ; Définition des messages ; Définition des modules d'interprétation des performatifs ; Définition des protocoles de communication.
WP-5 : création de l'ontologie	
	Utilisation d'une ontologie de base normalisée ; Définition des termes du vocabulaire ; Définition des concepts et des rôles dans le contexte d'e-Babel ; Conception OWL de l'ontologie.
WP-6 : développement d'une preuve de concept	
	Choix d'un sujet représentatif; Développement du sujet ; Implémentation et tests.
WP-7 : Conclusion	
	Rédaction d'un ou plusieurs articles; Démonstration / présentation du projet.

4. Ressources nécessaires

Temporellement, il est possible de démarrer simultanément :

- WP-1 : modélisation de l'architecture / choix technologiques
- WP-2 : conception et réalisation de l'interface web temporel

En effet, le développement de l'interface web peut être indépendant de la conception de l'architecture, car il existe un découplage fort entre la présentation, les informations et les processus (principe MVC).

La seule condition est que le choix technologique pour la partie web soit fixé.
Je propose d'utiliser des outils rependus et libres afin de permettre la réalisation de contribution à e-Babel par le plus large public possible.

> *Le développement du projet utilisera, dans la mesure du possible, des technologies et des outils libres afin ; d'une part de permettre l'accès au développement du réseau e-Babel à tous et d'autre part de garantir une pérennité des environnements.*

Description	Compétences requises	Ressources
WP-1 : modélisation de l'architecture / choix technologiques		
Architecture théorique basée sur mes travaux antérieurs : les systèmes orientés composants et les base de connaissance.	• Logiques formelles • SMA • Base de connaissances • OWL • Architectures • Etat de l'art des technologies actuelles et des systèmes d'informations	• Johann Sievering
WP-2 : conception et réalisation de l'interface web temporel		
Conception d'une interface web ergonomique pour la présentation des technologies de l'informatique dans sur des axes temporels et fonctionnels. Création de requête SQL	• Conception multimédia • HTML / CSS • JavaScript • AJAX • Php / MySql (PostgreSql) • SQL • Notion de Java • Eventuellement Flash / ActionScript	•
WP-3 : architecture générique du SMA et des objets d'information		
WP-4 : formalisation du langage de communication		
WP-5 : création de l'ontologie		
WP-6 : développement d'une preuve de concept		
WP-7 : Conclusion		

5. Détails d'architecture

Cette partie décrit les principes fondamentaux de l'architecture e-Babel.

Principes de base de l'approche orientée besoin

L'approche orientée besoin est structurée autour de la délégation des compétences. C'est aux composants de prendre en charge les processus fonctionnels en relation directe avec l'environnement et le contexte : ce sont les « composants métiers » qui résident au niveau des équipements, des logiciels ou des données qui ont cette responsabilité. L'architecture interne des composants est proche de celle du BDI[14]. Le **B**(elief) désire représente l'ensemble des informations issues de l'environnement ou des résultats de calculs (base de prédicats). Le **D**(esire) désire est l'ensemble des compétences (fonctions) que le composant peut utiliser pour réaliser ses buts. Le **I**(ntention) intention est le sous-ensemble des désires sélectionnés pour la réalisation de l'objectif courant. Dans les systèmes PRS est ajouté le plan qui est la séquence des intentions à réaliser dans cet ordre pour atteindre rationnellement le but ou le sous-but.

Le but est fourni à le composant sous la forme d'un « contrat » qui peut être accepté ou refusé en fonction des conditions locales, voir figure 5. Une fois le but accepté, le composant prend la responsabilité du mener à terme. Les buts et sous-buts sont définis au niveau de l'interface utilisateur vu ci-dessus. L'interface utilisateur ne peut pas présenter le composant lui-même qui ne se trouve pas nécessairement dans le même environnement et peut être, dans certain cas, en sommeil ou pas encore démarré. C'est donc un représentant, le composant « proxy », qui est chargé présenter à l'utilisateur les compétences du composant métier et de générer le contrat.

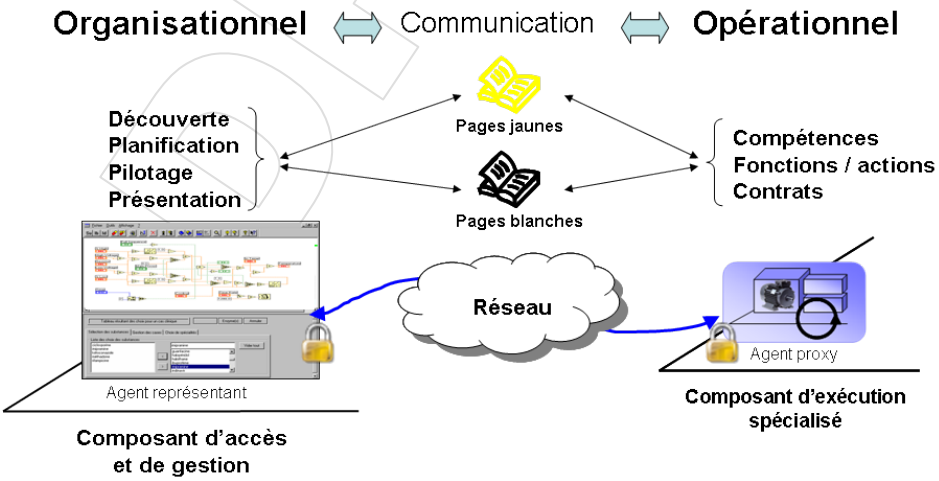

Figure 5 : exemple de relations entre les composants.

L'utilisateur peut donc créer les fonctions qu'il souhaite en organisant et connectant les composants proxy à sa disposition dans la palette. Une fois les schémas établis, chaque composant proxy fait un « appel d'offre » aux composants métiers et lui soumet le contrat qui, s'il est accepté, liera le composant proxy et le composant métier sélectionné. Le composant proxy est ensuite chargé d'assurer la gestion des commandes et la présentation des données du composant métier. Ces informations dépendent du domaine du composant métier.

[14] BDI : Belief, Desire, Intention. Rao and Georgeff, 1995.

Planification globale des comportements

Le problème qui se pose dans la gestion d'un système d'informations est la réalisation d'un processus global pris en charge par un ensemble d'composants collaboratifs distribués. Dans notre projet, les notions d'apprentissage et d'auto-optimisation ne sont pas pertinentes.
Il est alors tout à fait acceptable d'envisager que le processus globale puisse être décrit en entier dans l'interface utilisateur et qu'ensuite il soit réparti sous forme de multiple contrats vers chacun des composants métiers qui les réaliseront de manière autonome.
Subsiste deux questions : quelle doit être la forme de description des processus et comment synchroniser l'ensemble des opérations inter-composants tout en conservant la maîtrise des actions et la possibilité de connaître à tout moment l'état du système.
J'ai choisi les réseaux de Pétri. D'une part ils répondent exactement aux spécifications évoquées ci-dessus et d'autre part ils possèdent une base formelle qui permet de prouver certaines propriétés et de simuler les processus. Et finalement ils peuvent être implémentés concrètement dans les composants.
Il existe une norme de représentation des réseaux de Pétri en format XML : PNML[15].

Processus et réseaux de Pétri

Le terme de « gestion de processus » concerne la logistique des composants métiers. L'objectif est d'assurer que les activités soient correctement exécutées au bon moment et par les bons acteurs. La WFMC[16] défini la gestion de processus comme : « un système complètement défini, gérant et exécutant un processus par la mise en œuvre d'un moyen informatique qui réalise la logique de la représentation des workflow en conservant leur sémantique ».

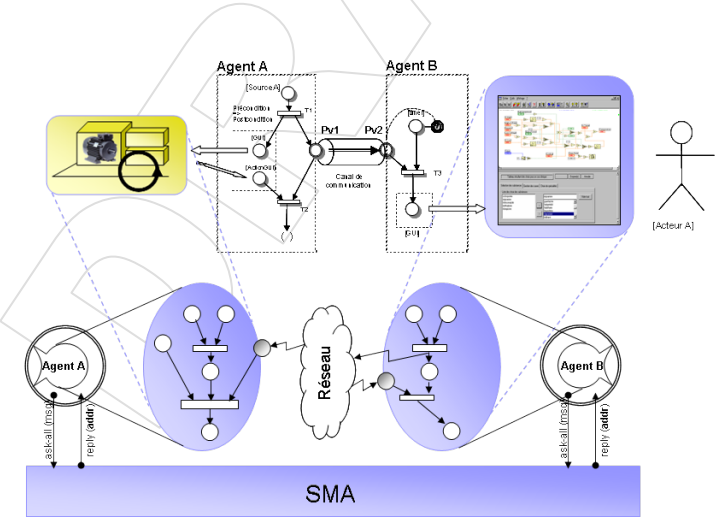

Fig. 6. Exemple illustratif de collaboration de deux composants avec un réseau de Pétri.

Les tâches sont exécutées selon le réseau de Pétri décrivant le processus. Leur déclenchement est gouverné par une précondition. Les postcondtions représentent le résultat de l'exécution de la tâche (la modification de l'environnement). Les synchronisations entre composants ou comportements sont simulées par l'envoi d'un jeton dans un message.
Dans la théorie des réseaux de Pétri, il est supposé que toutes les transitions soient mises à feu simultanément, ce qui n'est pas le cas dans un système distribué et autonome. Dans le cadre de notre projet, il est acceptable de décider de considérer la partition du réseau de Pétri comme étant un ensemble de sous-réseaux autonomes (pris en charge individuellement par des composants) et qu'ils possèdent des places particulières qui sont autant de canaux de synchronisations entre composants.

[15] PNML : Petri Net Markup Language.
[16] WFMC : workflow management coalition.

Technologies et réalisation

Les équipements, les logiciels et les données cibles sont hétérogènes et doivent pouvoir accueillir les nouvelles technologies moyen-long terme. Il faut donc jouer sur quatre plans :

1. *indépendance du support* (matériel et système d'exploitation) ;
2. *indépendance du lieu et du temps* (communications réseaux asynchrones) ;
3. *pérennité* (un changement technologique doit toujours permettre une compatibilité avec l'existant) ;
4. *interprétation homogène* (tous les paramètres et toutes les commandes doivent être interprétables de la même manière quelque soit le composant ou le module impliqué).

Il est actuellement possible d'atteindre ces quatre objectifs en choisissant d'une part un système basé sur une machine virtuelle, d'autre part en organisant les communications au niveau langage et finalement en définissant la sémantique de tous les termes. La figure 6 ci-dessous résume cette situation.

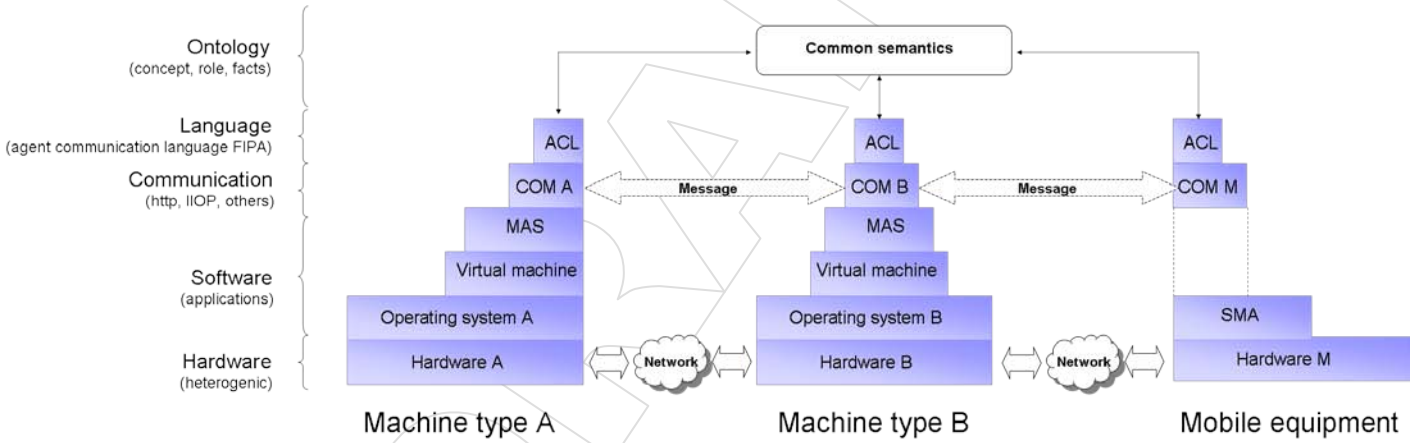

Fig. 7. Architecture permettant d'atteindre les quatre contraintes citées plus haut.

Il ne faut ni avoir d'a priori sur le hardware, ni sur le système d'exploitation. La meilleure approche actuelle est l'utilisation d'une machine virtuelle. La couche fédérant l'ensemble des composants est la couche des messages. Elle doit être compatible entre tous les systèmes. La couche message est mise en œuvre physiquement par la couche physique. Il est possible d'envisager l'utilisation de JXTA. Finalement, au niveau message chaque MAS (chaque composant) doit pouvoir communiquer sans préoccupation technique. Le niveau langage symbolique permet l'accès aux fonctions, requêtes et données indépendamment de toute technologie. L'ontologie assure l'interprétation correcte et unique de chaque concept et de chaque performatif.

6. Conclusion

Le projet e-Babel est un projet ayant pour objectifs la pérennité des équipements, des logiciels et des données. Cette pérennité est assurée par une architecture délocalisée dont la responsabilité des traitements et des stockages est données aux composants eux-mêmes.

Le principe organique du projet permet de créer une communauté d'acteurs qui pourront contribuer chacun dans leurs spécialités respectives. La mise en commun de l'ensemble des composants est assurée par des canaux de communication transportant des messages en langage symbolique.

La première phase de ce projet est la création de l'architecture de base et du choix des technologies permettant de développer une preuve de concepts.
Les résultats permettront de rédiger les spécifications qui seront mises à disposition des acteurs pour le développement des composants d'e-Babel.

Les choix technologiques d'implémentation seront privilégiés dans le domaine du libre afin que chaque acteur puisse collaborer sans barrière économique et afin de garantir une pérennité des outils qui sont maintenus eux-mêmes par une communauté de développeurs.

7. Bibliographie

[A venir …]