**République Algérienne Démocratique et Populaire**
**Ministère de l'Enseignement Supérieure et de la Recherche Scientifique**

Université Abderrahmane Mira, Bejaïa
Faculté des Sciences Exactes
Département d'Informatique


# Mémoire de fin d'études

*En vue de l'obtention du diplôme de master recherche en informatique*

*Option : Réseaux et Systèmes Distribués*

# Thème

## Orchestration de services web dans les applications mashups

**Réalisé par:**

- Mr KHABER Farid
- Mr KHELOUFI Rabah

**Encadré par :**

- Mr ATROUCHE Abdelghani

**Membre du jury :**

- Pr. AISSANI Sofiane
- Ex. DJEBARI Nabil
- Ex. AIT ABDLOUAHAB Karima

**Promotion : 2012 - 2013**

# DÉDICACES

*À mon père **Rachid** et ma mère **Yasmina** à qui je dois ce que je suis, qu'ils trouvent dans ce travail, le fruit de leurs sacrifices consentis pour mon éducation, l'expression de mon amour et ma gratitude pour la bienveillance avec laquelle ils m'ont toujours entouré, que Dieu leur préserve bonne santé et longue vie.*

*À mes trois sœurs **Meriem**, **Saliha** et **Asma**.*

*À mon petit frère **Hamza**.*

*À mon amie **Lydia** qui m'a tant aidé et soutenue pour effectuer ce travail.*

*À tous les gens qui me connaissent.*

**KHABER Farid**

*Je dédie ce modeste mémoire :*

*À ma très chère et douce mère, à mon très cher père a qui j'adresse au ciel les vœux les plus ardents pour la conservation de leur santé et de leurs vies.*

*À mes chères sœurs.*

*À mon cher frère.*

*Et à tous mes amis (es).*

**KHELOUFI Rabah**

# REMERCIEMENTS

*Nos vifs remerciements vont d'emblée à Dieu tout puissant qui nous a doté d'une grande volonté et d'un savoir adéquat pour mener à bien ce modeste travail.*

*Il nous est spécialement agréable, d'exprimer toute notre reconnaissance envers les personnes qui de près ou de loin nous ont apporté leurs soutiens dans la réalisation de ce projet.*

*Au premier rang notre promoteur MONSIEUR* ***ATROUCHE ABDELGHANI****, son aide, ses conseils précieux, ses critiques constructives, ses explications et suggestions pertinentes qui nous ont permis de réaliser notre travail convenablement.*

*Aux membres de la commission qui jugeront notre travail, à tous nos enseignants et les membres du département informatique de l'université* ***ABDERAHMENE MIRA****.*

*Nous remercions de même nos familles pour leurs grandes attentions, leurs grands soutiens et encouragements tout au long de l'évolution de ce travail, et de l'énorme intérêt qu'ils ont montré envers ce sujet.*

# Table des matières

## Chapitre 3 : Les approches de développementdes mashups

## Chapitre 4 : Orchestration et composition des IUs dans les mashups

## Chapitre 5 : Mashup web à base d'orchestration de services web

# Table des figures

# Liste des tableaux

# Table des listes

# Liste des abréviations

| | |
|---|---|
| SOA | Service oriented architecture |
| IHM | Interface homme machine |
| API | Application Programming Interface |
| HTML | Hypertext Markup Language |
| CSS | Cascading Style Sheets |
| Ajax | Asynchronous JavaScript and XML |
| SOAP | Simple Object Access Protocol |
| REST | REpresentational State Transfer |
| XML | eXtended Markup Language |
| JSON | JavaScript Object Notation |
| IDE | integrated development environment |
| IT | Information technology |
| HTTP | HyperText Transfer Protocol |
| HTTPS | Hypertext Transfer Protocol Secure |
| RDF | Resource Description Framework |
| RSS | Really Simple Syndication |
| XHTML | Extensible HyperText Markup Language |
| DaaS | Data as a Service |
| DOM | Document Object Model |
| XSLT | eXtensible Stylesheet Language Transformations |
| PHP | Hypertext Preprocessor |
| JEE | Java Enterprise Edition |
| RPC | Remote procedure call |
| SaaS | Software-as-a-Service |
| SAML | Security assertion markup language |
| JSONP | JSON with padding |
| CSV | Comma-separated values |
| URL | Uniform Resource Locator |
| BPEL | Business Process Execution Language |
| WSDL | Web Services Description Language |
| XAML | eXtensible Application Markup Language |
| UI | User interface |
| RCP | Rich Client Platform |
| WSRP | Web Services for Remote Portlets |
| WebML | Web Modeling Language |
| OOHDM | Object Oriented Hypermedia Design Method |
| UWE | UML-based Web Engineering |
| JSR | Java Specification Requests |
| IIOP | Internet Inter-ORB Protocol |
| DTD | Document Type Definition |
| XSD | XML Schema Definition |
| EMML | Enterprise Mashup Markup Language |
| OMA | Open Mashup Alliance |
| WS-BPEL | Web Services Business Process Execution Language |
| BPELJ | BPEL for Java |
| WSFL | Web Services Flow Language |

| | |
|---|---|
| YAWL | Yet Another Workflow Language |
| XLANG | XML LANGuage |
| XPDL | XML Process Definition Language |
| AOP | Aspect-oriented programming |
| BPMN | Business Process Model and Notation |
| CORBA | Common Object Request Broker Architecture |
| DCOM | Distributed Component Object Model |
| URI | Uniform Resource Identifier |
| SGBD | Système de gestion de base de données |
| JSP | JavaServer Pages |

# INTRODUCTION GÉNÉRALE

L'idée derrière le terme mashup n'est pas nouvelle. L'intégration des différentes ressources est une question généralement rencontrée au cours du développement de logiciels. En fait, la plupart des méthodes d'ingénierie prennent en compte le fait que certaines données et les fonctionnalités sont fournies par des systèmes externes et fournissent des mécanismes pour les spécifier correctement.

La principale raison pour laquelle les mashups gagnent en popularité énorme, c'est que même les non-techniciens sont en mesure de créer de nouveaux contenus et des représentations de ressources sans trop d'effort ou de connaissances des langages de programmation. Un autre avantage est que l'exécution des mashups n'est pas effectuée par les systèmes de boîte noire (black-box) (comme dans l'exécution orientée services), mais plutôt axée sur l'utilisateur, ce qui rend le processus d'intégration des ressources beaucoup plus transparent par rapport aux plates-formes d'intégration d'applications classiques.

Au moment d'entreprendre une revue des documentations sur les mashups, les résultats de la recherche sur les méthodes ou les infrastructures pour la création des mashups sont décevant. L'apport des approches compréhensibles proposées jusqu'ici est minime et aucune ligne claire n'existe entre les mashups et les technologies telles que l'architecture orientée services (SOA).

Le but de notre mémoire est de proposer un framework pour le développement des mashups. La définition du terme mashup affecte également une distinction entre le terme mashup et l'architecture SOA qui est actuellement fréquemment utilisée (comment ces technologies se complètent mutuellement et comment font-elles la différence). Par ailleurs, notre mémoire présente un extrait d'outils et langages supportant la création de mashups. Enfin, nous cherchons à discuter des approches de développement des mashups et de présenter les préoccupations méthodologiques qui devraient être considérées lors de l'élaboration des mashups.

Le présent mémoire s'articule autour de 5 chapitres. Le chapitre1 présente une définition claire des mashups et comporte aussi une définition des types et architectures des mashups. Les technologies mashups sont présentées dans le chapitre 2. Le chapitre 3 décrit les différentes approches de développement des mashups. Dans le chapitre 4 on aborde le principe d'orchestration et d'intégration des **IHMs** dans les mashups. Ensuite nous passerons

à l'implémentation et à la réalisation de notre mashup web. Enfin nous terminerons le mémoire par une conclusion générale et quelques perspectives.

# Chapitre 1 : Description du thème

## 1.1. Introduction

Les mashups sont à la hausse comme génération novatrice de la technologie pour les non programmeurs et les développeurs Web qui souhaitent faire usage des différents produits et services offerts sur le Web. Comme son nom l'indique, les mashups mélangent et mixent les interfaces de programmation de différents produits d'entreprises et services pour créer de nouveaux produits et services. Yahoo, Google, Amazon, eBay et Microsoft ont publié des interfaces de programmation d'applications (API) basées sur les standards du web qui vous permettent d'utiliser leur fonctionnalités complexes même sans être un expert en programmation **[1].**

## 1.2. Définition des mashups

Les mashups à l'origine font référence à la pratique de la musique pop (notamment le hip-hop), c'est le fait de produire une nouvelle chanson en mélangeant deux ou plusieurs morceaux existants **[2]**.

Dans la technologie, un mashup est une application Web qui combine des données provenant de plusieurs sources en un seul outil intégré, un exemple est l'utilisation de données cartographiques de Google Maps pour ajouter des informations de localisation de données immobilières, créant ainsi un nouveau et distinct service Web qui n'a pas été fourni à l'origine par d'autres sources.

Ainsi, une application web ou un site web qui combine le contenu de différents sites est nommé comme mashup comme illustré à la **figure 1.1**.

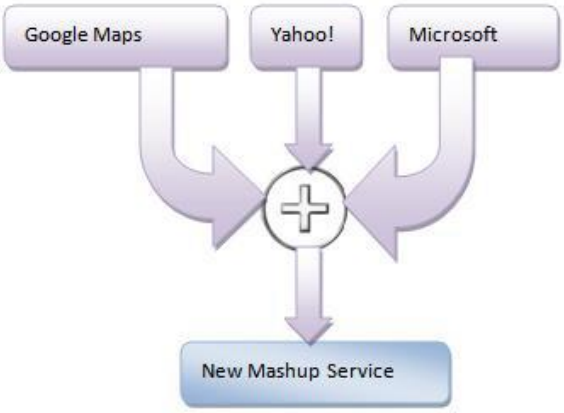



**Figure 1.1 :** Définition d'un mashup.

Les mashups ont explosé récemment sur le web, pour deux raisons principales **[2]**.

Premièrement, la plupart des sociétés Internet majeures, tels que Yahoo!, Google et Amazon ont ouvert leurs données pour être utilisées avec les autres sources de données sans une négociation de licences de longue durée.

Deuxièmement, l'avènement de nouveaux outils qui rendent la création de mashups facile pour n'importe qui, indépendamment de leur savoir-faire technique.

Selon **[3]**, les raisons d'utiliser les Mashups :

✓ Le Mashup est une technologie qui permet de créer avec un poids léger des applications centrées sur les données pour un usage quotidien.

✓ Les Mashups permettent la réutilisation des applications existantes.

✓ Ils permettent également le développement rapide d'applications.

✓ Le développement d'un mashup n'implique pas nécessairement de vastes compétences en programmation.

✓ Le coût associé du développement d'applications est considérablement réduit.

✓ Les demandes sont mieux adaptées aux besoins des utilisateurs puisque les utilisateurs peuvent désormais intégrer des contenus qu'ils étaient incapables de développer en raison de contraintes de temps ou de ressources.

✓ Ils comportent des domaines variés d'applications telles que la cartographie, la photo, la recherche, la vidéo, la comparaison prix/produits, ainsi que le commerce électronique (marketing).

## 1.3. Types de mashups

Il existe différents types de mashups en fonction du but qu'ils servent: les mashups cartographiques, les mashups photos, vidéos, les mashups de recherche et d'e-commerce. La **figure 1.2** montre les différentes applications composites prenant la source d'entrée à partir de différentes API comme Yahoo, Google, Microsoft, etc **[4].**

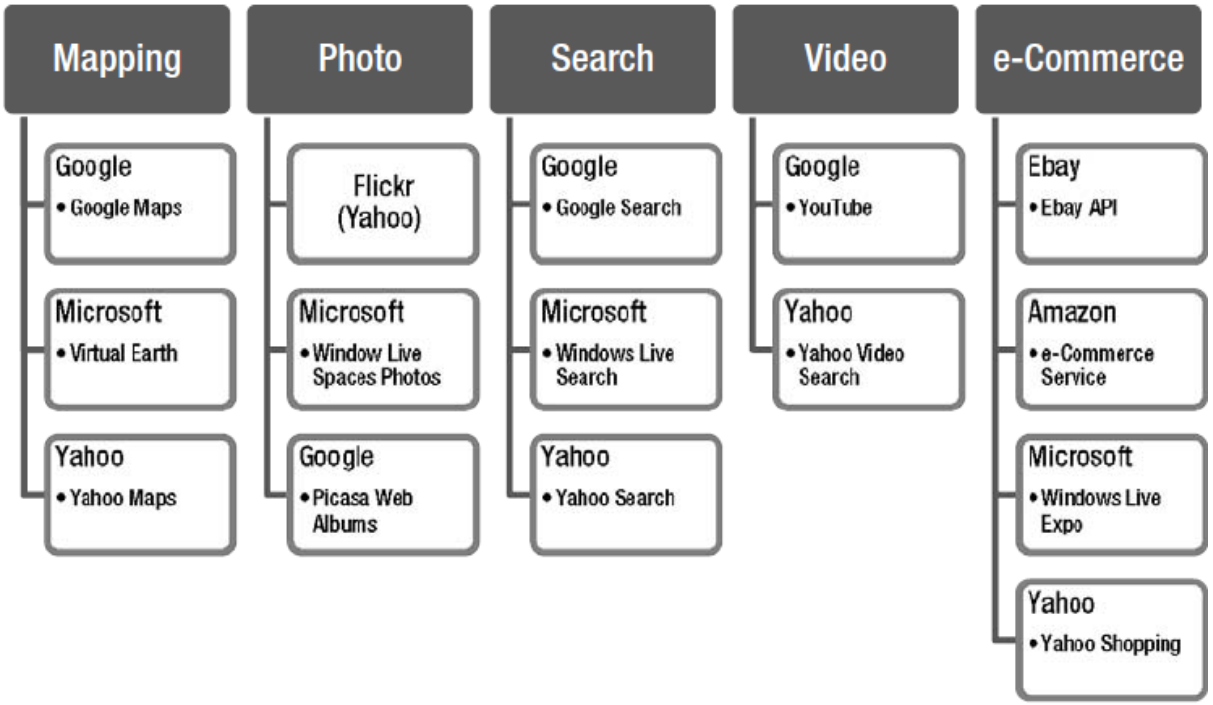



**Figure 1.2 :** Types de mashups **[5]**.

Outre cela, les mashups peuvent être classés comme suit:

### 1.3.1. Les Mashups de données

Il rassemble des données transversales et références provenant de sources web diverses. Un mashup de données mélange des données de types similaires de différentes sources, comme par exemple, il pourrait créer un rapport des parts de marché en combinant une liste externe de toutes les maisons vendues la semaine dernière avec les données internes d'une maison vendue d'une agence **[5].**

### 1.3.2. Les Mashups utilisateur

Le meilleur type connu est le mashup du consommateur, comme en témoignaient les nombreuses applications de Google Maps. Il traite les différentes visualisations et éléments de données pour la consommation attrayant plus d'informations **[5].**

### 1.3.3. Les Mashups d'entreprise

Il a des combinaisons internes des ressources de l'entreprise souvent renforcé avec les services Web externes. Un mashup d'entreprise est une combinaison de tout ce qui précède, en se concentrant à la fois sur l'agrégation et la présentation des données et en outre l'ajout de fonctionnalités de collaboration, ce qui rend le résultat final approprié pour une utilisation

comme une application d'entreprise. Maintenant, les entreprises optent de plus en plus pour les mashups pour augmenter leurs profits **[5].**

Par exemple, le site *Viral Video Chart* **[123]** peut être considéré comme un mashup utilisateur qui s'appuit sur *YouTube* **[124]**, *MySpace* **[125]**, et *Google Video* **[126]** afin d'identifier globalement les nouveaux clips en vogue et par catégorie.

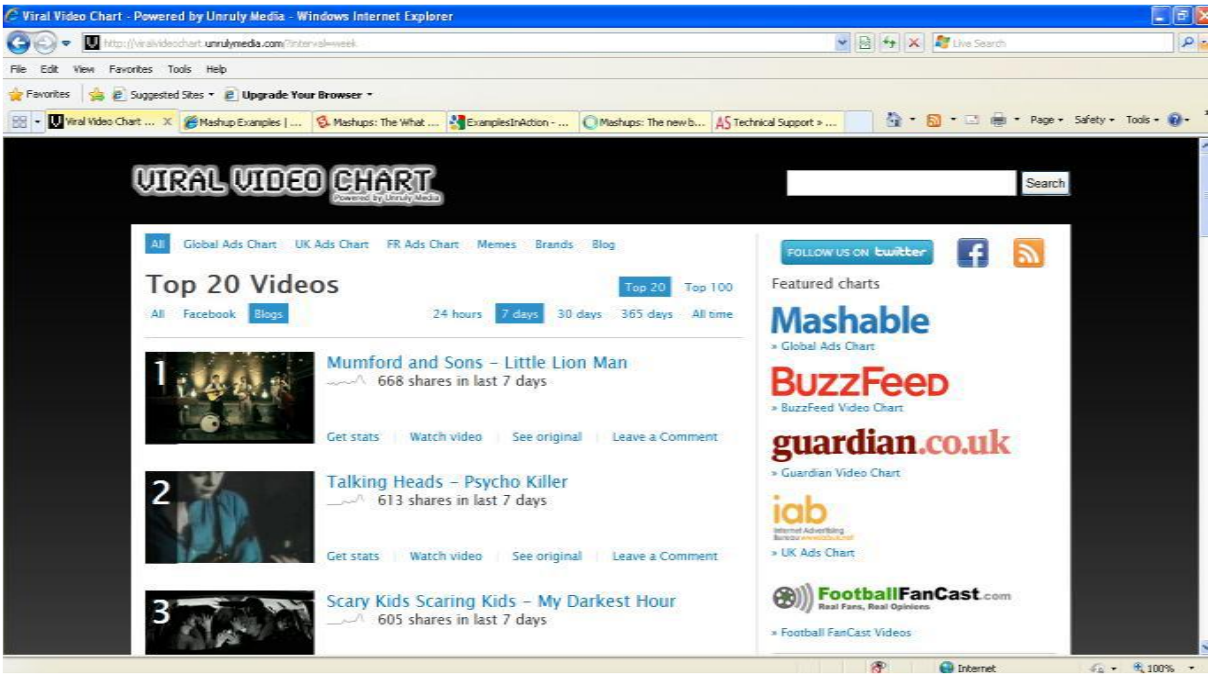



**Figure 1.3 :** Exemple de Viral Video Chart **[5]**.

## 1.4. Architecture des mashups

L'architecture des mashups est essentiellement une architecture en couches avec trois couches, la couche de présentation ou d'interactivité, la couche services Web ou la couche API et la couche de manipulation de données. La **figure 1.4** suivante représente l'architecture des mashups. La couche de présentation/interactivité est l'interface utilisateur de l'application composite. HTML combiné ou stylé avec CSS affiche l'interface du mashup pour l'utilisateur. JavaScript, avec les techniques Ajax améliore l'expérience de l'utilisateur en rendant le mashup plus réactif et comme une application de bureau. Javascript reçoit également une entrée de l'utilisateur et si nécessaire, initie les demandes aux APIs de services Web utilisés par le mashup **[5].**

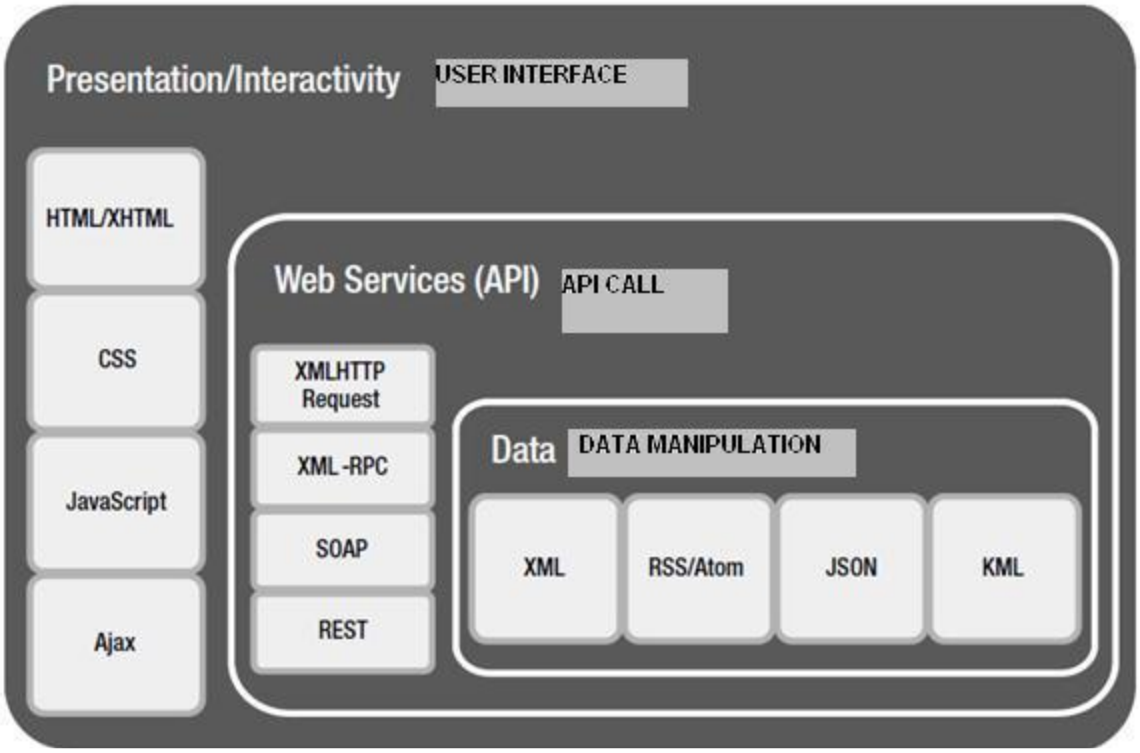



**Figure 1.4 :** Couches de l'architecture d'un mashup. **[5]**

Par exemple, dans une application web de recherche de livres, les utilisateurs peuvent rechercher le livre qu'ils veulent acheter en entrant l'ISBN de l'ouvrage dans l'interface. Peu importe si nous parlons de XML-RPC, SOAP ou services web REST, XML HTTPRequest est utilisée pour envoyer l'ISBN de l'ouvrage en utilisant XML ou JSON pour le serveur. Les données de l'annuaire (par exemple, le prix, l'auteur, le résumé, etc) stockés sur le serveur sont extraites d'une base de données de livres et mise généralement au format XML et JSON avant de retourner à la présentation et à la couche d'interactivité. Une fois de retour, JavaScript est utilisé pour afficher le titre du livre, l'auteur et l'image de couverture à l'aide d'HTML avec CSS.

Pour créer un mashup, il faut connaître les réponses aux questions suivantes:

- Quels sont les appels d'APIs nécessaires? (REST, XML-RPC, SOAP, Javascript).
- Quelle est la manipulation des données faite?
- Comment se fait la présentation?

Ainsi le mashup peut être formulé comme suit:

Mashup = appels d'APIs + manipulations de données + UI **[6]**

La **figure 1.5** montre comment les appels d'API sont transmis aux sites visés et comment les données sont intégrées au site web du mashup et enfin les affichés pour l'utilisateur.

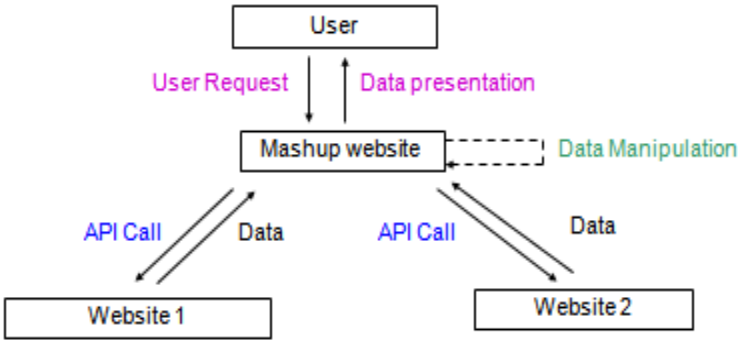



**Figure 1.5 :** Fonctionnement d'un mashup **[5].**

## 1.5. Outils et éditeurs pour la création de mashups

Il existe différents outils et éditeurs pour développer les mashups. Voici certains des outils et éditeurs célèbres pour la création de mashups :

- Google Mashup Editor (éditeur de mashup de Google) **[7].**
- Yahoo Pipes (un mashup RSS innovant par le moteur Yahoo!) [**8].**
- Microsoft Popfly (mashup IDE de Microsoft) **[120]**.
- GeoMonkey (mashup visuel crée par Google Maps).
- Schmapplets (semblable dans le concept à GeoMonkey mais a d'intéressantes fonctionnalités hors ligne) **[9].**
- Wayfaring (mashup visuel crée par Google Maps). **[10]**.

## 1.6. Problématique et objectifs

La composition des services web se réfère au processus de création d'un service composite offrant une nouvelle fonctionnalité, à partir de services web existants plus simples, par le processus de découverte dynamique, d'intégration et d'exécution de ces services dans un ordre bien défini afin de satisfaire un besoin bien déterminé **[11].** A titre d'exemple, nous pouvons modéliser le service de planification de voyage d'une agence de voyages comme la composition de plusieurs Services web atomiques (basiques) appartenant à des organisations différentes : réservation de vol, réservation d'hôtel et paiement.

Dans la réalité de tous les jours, la complexité des développements impose très souvent l'utilisation de plusieurs services web pour parvenir au résultat souhaité. Comment, alors, composer ces services web pour atteindre le résultat qu'on souhaite?

Quelques spécifications ont été définies pour aider les développeurs. Par exemple: **BPEL4WS** (Business Process Execution Language For Web Services), ou encore **WSFL** (Web Services Flow Language). Grâce à ces outils et à la grammaire formelle qu'ils apportent, on peut décrire les processus d'affaires, les messages échangés, les opérations de composition de base, etc. Mais est-ce suffisant de composer des services web pour parvenir au résultat ou au service souhaité?

L'objectif du présent mémoire est d'explorer, de proposer un *framework* et une approche de composition de services performante pour l'orchestration des services web de notre mashup dans les architectures orientées services.

## 1.7. Motivations

L'information qui est essentielle pour réaliser des tâches et prendre des décisions est souvent répartie sur plusieurs systèmes logiciels et transférée inefficacement entre ceux qui peuvent fournir et ceux qui ont besoin de consommer des données. Les utilisateurs ont souvent recours à des moyens inefficaces, telles que le **copier-coller** des informations, l'emballement dans des feuilles de calcul ou des documents et l'envoi par e-mail. Ces processus sont manuels, source d'erreurs et non évolutifs. Des solutions qui répondent à ces besoins entraînent généralement le développement personnalisé. En raison de la nécessité immédiate et du besoin de courte durée, ceci est très souvent coûteux et non rentable.

La capacité des mashups à fournir des informations à partir de leur source vers leurs consommateurs, reformater et agréger des données pour créer un aperçu en connectant des données liées ou proches promet de grandes valeurs dans les opérations des organisations.

Ainsi, les mashups sont prévus pour avoir des répercussions importantes sur les futurs environnements IT des entreprises, en s'appuyant sur les systèmes existants pour répondre aux besoins immédiats des utilisateurs finaux par le biais de la création simple et rapide de demandes ad-hoc. Au moment de la rédaction de ce mémoire, un large consensus, la définition nette des mashups est encore inexistante. Bien que cela présente des inconvénients considérables en termes de création de solutions fondées sur les normes établies ou les meilleures pratiques, il ouvre aussi le champ à l'innovation.

## 1.8. Conclusion

Tout au long de ce chapitre, nous avons essayé de mettre au point le cadre général de notre travail. Pour ce faire nous avons tout d'abord défini le terme **mashup**, ses types, ses architectures et les outils utilisés pour sa création, puis s'ensuit une explication de la problématique qu'on va traiter et l'objectif de notre travail.

# Chapitre 2 : Les technologies des mashups

## 2.1. Introduction

Les technologies utilisées aujourd'hui pour produire une application mashup comprennent des portions de HTML dynamique, des widgets, JavaScript, AJAX et les formats du web sémantique. Le contenu d'un mashup est récupéré à partir des systèmes internes ainsi que des sites Web tiers. Les protocoles et formats de données comprennent des protocoles HTTP, HTTPS, XML, SOAP, RDF, JSON et d'autres.

Les mashups sont créés la plus part du temps en ad hoc. Toutefois, dans le domaine de l'entreprise, les applications mashups doivent prendre en considération des éléments tels que la confidentialité, l'authentification, la gouvernance, la conformité et d'autres activités liées à des contraintes.

## 2.2. Domaines technologiques des mashups d'entreprises

Les domaines des Mashups dépendent de ce qui devra être mélangé ensemble. En règle générale, trois catégories de haut niveau d'éléments peuvent être mélangées ensemble, des objets de l'interface utilisateur (présentation), des données et/ou des fonctionnalités d'applications (processus). Cela pourrait inclure des portions de HTML, **on-demand JavaScript** pour accéder à une **API** externe, des **APIs** de services Web à partir de l'un des serveurs de l'entreprise, des flux **RSS** et/ou d'autres données destinées à être mélangées dans l'application ou pages web. Le style de mise en œuvre, les techniques et technologies utilisées pour un mashup donné dépendent de cette détermination. Une fois que les éléments sont déterminés, une équipe de développement peut procéder à l'application des langues, des processus et des méthodologies pour l'application à portée de main.

Les technologies utilisées pour créer un mashup dépendent également des sources à partir desquelles les éléments mashup seront accessibles, on a besoin des talents d'une équipe de développement pour construire le mashup, ainsi que les services qui doivent être créés ou consultés pour récupérer les objets nécessaires pour le mashup **[12].**

## 2.3. Les styles, les techniques et les technologies mashups :

Pour commencer le travail de conception sur un mashup, on doit déterminer ce qui doit être mélangés ensemble. Trois catégories de haut niveau d'éléments peuvent être mélangées ensemble des objets de l'interface utilisateur (présentation), des données et/ou des fonctionnalités d'applications (processus). Cela pourrait inclure des portions de **HTML**, **on-demande JavaScript** pour accéder à une **API** externe, des **APIs** de services Web à partir de l'un des serveurs de l'entreprise, des flux **RSS** et/ou d'autres données destinées à être mélangées et écrasées dans l'application ou pages. Le style de mise en œuvre, les techniques et les technologies utilisées pour un mashup donné dépendant de cette détermination. Une fois que les éléments sont déterminés, votre équipe de développement peut procéder à l'application de langues, des processus et des méthodologies pour l'application à portée de main.

Dans ce chapitre, on souligne quelques-uns des styles les plus largement utilisés, les techniques et les technologies pour construire des mashups pour chacune des trois catégories principales ou des articles.

## 2.4. Détermination du domaine technologique pour un Mashup

Avec la détermination de ce qui doit être mélangé, on doit également déterminer les sources à partir desquelles les éléments mashup seront accessibles, le style, les technologies et les techniques dont un développeur a besoin pour construire le mashup et quels sont les services dont il a besoin pour construire ou d'accéder à la récupération des objets nécessaires à votre mashup.

Les mashups comptent sur la capacité de mélanger des objets faiblement couplés associés au sein d'un domaine technologique donné. Les exigences de l'application de déterminer quels éléments (interface utilisateur, de données et/ou une fonctionnalité) seront nécessaires pour construire le mashup.

Du point de vue du haut niveau, le domaine technologique appliqué aux mashups peut être considéré comme la présentation orientée, axée sur les données et les processus. Les langues, les méthodes différentes et les techniques de programmation applicables à chaque domaine technologique.

Comme le montre la **figure 2.1**, les catégories des mashups peuvent être divisées selon les objets de présentation, des données, des fonctionnalités et des applications/processus.

Chacune de ces catégories exige des compétences et des technologies spécifiques de programmation, comme nous le verrons par la suite.

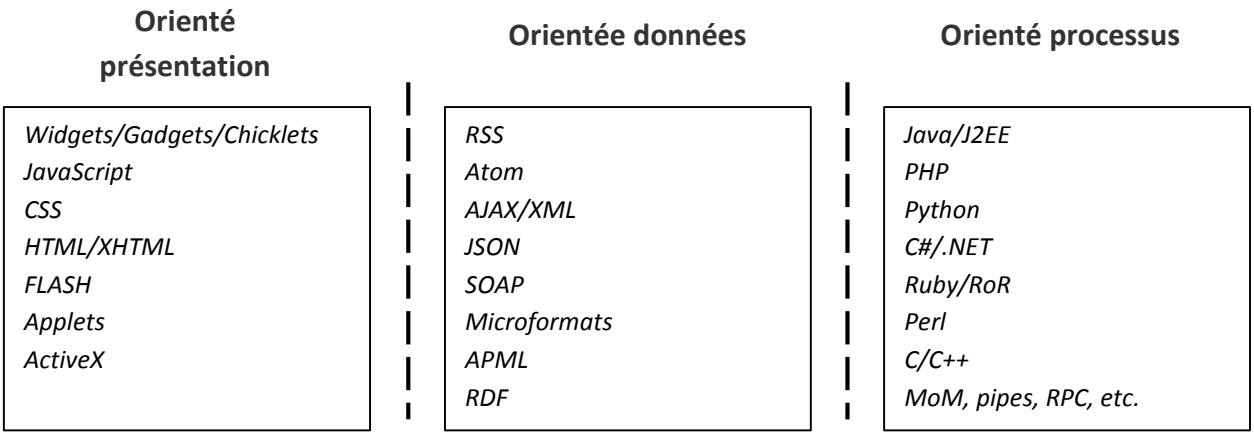



**Figure 2.1 :** Les Trois principaux domaines technologiques des mashups.

### 2.4.1. Technologie orientée présentation

Un mashup orienté présentation mélange les différentes éléments d'interfaces utilisateurs (UI) pour créer une nouvelle application ou une nouvelle page. En règle générale, ce type de mashup a pour objectif de créer une application ou une page affichant les éléments des interfaces utilisateurs d'une manière similaire à un portail classique. Autrement dit, chaque élément '**artefact**' doit être affiché dans une petite zone séparée sur un panneau ou d'une page d'application d'une manière séparée par rapport aux autres éléments. Il y a très peu ou pas d'interaction entre les objets des interfaces utilisateurs dans un mashup orienté présentation.

Les technologies utilisées dans un mashup orienté présentation peuvent inclure :

- **Gadgets et widgets :** Un *widget* ou, selon les éditeurs, gadget, *block* ou *flake*, est un petit composant applicatif ou un contenu dynamique qui peut être facilement placé dans une page HTML. Il peut être écrit en n'importe quel langage (Java, .NET, PHP), voire en HTML, et il encapsule, en général, une API pour accéder à une source ou à un flux de données. La définition a été élargie par rapport à des pages Web et des mashups pour inclure des composants constitués de plusieurs fonctionnalités complexes comme une horloge, une calculatrice, des informations météorologiques et ainsi de suite. Les Gadgets et widgets peuvent ou ne peuvent pas récupérer les données d'un site externe **[13].**

- **On-demand JavaScript, extraits de code JavaScript et badges :** Des Petites sections de code JavaScript qui peuvent être insérées dans un volet application ou une page pour créer des composants de l'interface utilisateur. Typiquement, le JavaScript repose sur l'interaction avec une API de service Web qui renvoie des données et les fonctionnalités utilisées pour construire l'élément de l'interface utilisateur.

- **CSS / HTML / XHTML :** Des *Snippets* qui peuvent être insérés pour créer différents composants de l'interface utilisateur qui peuvent être réutilisés sans tenir compte du domaine d'application.

- **Composants Flash / Java applets / commandes ActiveX :** Composants autonomes de l'interface utilisateur qui s'appuient sur des technologies propriétaires conteneurs comme une machine virtuelle ou d'exécution qui est incorporée dans le volet application ou une page du navigateur **[12].**

Un mashup orienté présentation est généralement le type de mashup le plus simple et le plus rapide à construire. Comme il y a peu d'interaction ou pas entre les éléments mélangés, le développeur de mashup peut simplement s'inquiéter de placer les articles dans le volet application ou dans une page à l'endroit désiré avec le thème de l'interface utilisateur désiré.

### 2.4.2. Technologie orientée données

Les mashups orientés données (*in-process* ou *out-of-process*) impliquent la combinaison des données provenant d'un ou plusieurs sites hébergés à l'extérieur ainsi que dans une page web ou un volet d'application, généralement en utilisant des techniques de script et des langages tels que JavaScript, JScript et d'autres. Dans ce type de mashup, les données sont renvoyées à partir d'un site ou d'un serveur sous la forme de fichiers XML, RSS, JSON, Atom et ainsi de suite. Une grande partie des données retournées proviennent de services orientés données parfois appelé **Data-as-a-Service** (DaaS). DaaS décrit une API orientée vers les services de données qui peut être appelée sans compter sur les processus de tierces parties ou de composants entre le prestataire et le consommateur du service.

#### 2.4.2.1. Mashups orientés données axés sur les processus

Les mashups orientés données axés sur les processus s'appuient sur des données mélangés ensemble, en utilisant les technologies d'application ou un navigateur tel que JavaScript, AJAX, ou le Document Object Model (DOM). Dans ce style de mashup de données, les données sont récupérées à partir d'un ou de plusieurs sites/serveurs et utilisées

comme des valeurs ou des paramètres de configuration pour construire des objets de l'interface utilisateur au sein d'un processus de demande ou de la page du navigateur **[14]**.

#### 2.4.2.2. Mashups orientés données hors processus

Les mashups orientés données hors processus s'appuient sur des données mélangés ensemble, en utilisant des technologies telles que Java, Python, Perl, C + +, XML et XSLT, pour n'en nommer que quelques-uns. Dans ce type de mashup, les données sont récupérées à partir d'un ou de plusieurs sites/serveurs et utilisées en tant que valeurs des paramètres de configuration pour créer de nouveaux modèles de données au sein d'un processus côté serveur ou d'un processus différent côté client.

### 2.4.3. Technologie orientée processus

Les mashups orientés processus (out-of-process) impliquent des fonctionnalité combinant ensemble dans un ou plusieurs processus externes à l'aide des langages de programmation tels que **Java, PHP, Python et C**. les mashups construits pour des applications d'entreprises ou des applications web peuvent comporter des cadres tels que **JEE, .NET, Ruby on Rails**.

Dans un mashup axé sur les processus, la fonctionnalité est combinée en utilisant des techniques de communication interprocessus / *interthread* telles que la mémoire partagée, les fils de messages/bus, les appels de procédure distante (RPC) et ainsi de suite. Même si les données sont échangées entre processus et threads dans un mashup axé sur les processus, le résultat final diffère d'un mashup orienté données en ce que le mashup orienté données cherche à tirer de nouveaux modèles de données, tandis qu'un mashup orienté processus cherche à tirer de nouveaux procédés et/ou services.

Plus souvent, les entreprises ont besoin d'une approche hybride pour construire un mashup. Cela est dû au nombre de sources disparates dont les fonctionnalités et les données sont récupérées comme les sites de géocodage, les flux RSS et les magasins de données d'entreprise.

Ainsi, lors de la préparation pour la construction d'un mashup, on doit d'abord déterminer ce qui doit être mélangé, le domaine technologique dans lequel créer le mashup et les sources à partir desquelles les éléments du mashup seront accessibles. Avec cette information en main, on peut alors déterminer si une équipe de développement possède les compétences nécessaires pour utiliser les techniques et les technologies nécessaires à la

construction du mashup. Les compétences du développeur à un grand impact sur la décision du style de mashup choisi. La section suivante présente les raisons du choix des différents styles et/ou domaines de mashups **[12].**

## 2.5. Avantages et inconvénients des technologies de Mashups

Il y a des raisons claires pour choisir un style ou un domaine de mashup. Selon l'objectif du mashup, on doit peser les avantages et les inconvénients de chaque style de mashup avant le début des travaux.

### 2.5.1. Avantages et inconvénients des mashups orientés présentation

Les mashups orientés présentation sont populaires parce qu'ils sont rapides et faciles à construire. Ils s'appuient principalement sur des données et des objets de l'interface utilisateur extraites de sites à l'aide des APIs de service, des flux de données et ainsi de suite. Ce modèle est souvent considéré comme un **Software-as-a-Service (SaaS)**, bien que de nombreux objets utilisés dans ce modèle ne soient pas techniquement des services. Les mashups orientés présentation ont souvent besoin d'aucune autorisation préalable, aucune étape d'installation et pas d'autres technologies que celles trouvées dans n'importe quel navigateur Web standard **[15].**

Il est facile à mettre en œuvre des mashups orientés présentation parce qu'on peut généralement utiliser les services, les composants et bibliothèques de scripts qui sont accessibles au public sans avoir à installer de plates-formes applicatives ou des outils. Dans ce modèle, on peut simplement intégrer ou inclure le code de script ou l'appel de service correctement dans une page HTML en fonction des besoins. De nombreux composants de script publiquement disponibles aujourd'hui peuvent nous permettre de personnaliser l'aspect et la convivialité de l'interface utilisateur qu'ils génèrent.

Les mashups orientés Présentation n'ont généralement pas besoin de service de codage ou de déploiement, tout service de codage est assuré par des processus externes/sites susceptibles d'être utilisés par les consommateurs de services à volonté.

Les performances sont généralement très sensibles aux mashups orientés présentation, puisque toutes les demandes sont faites directement à un fournisseur ou prestataire de script. Ce modèle d'accès direct élimine toute interaction avec un processus intermédiaire par lequel les données sont récupérées ou la fonctionnalité est dérivée. Cependant, cela crée également

un couplage direct au service ou des services qui peuvent éventuellement se transformer en liens rompus, des pages partiellement fermées et/ou des pages fermées lentement si l'un ou plusieurs des services échouent ou deviennent improductifs (non performants).

S'appuyant sur les techniques et les technologies orientées présentation, les mashups d'entreprise peuvent réduire la charge qui pourrait autrement être épaulé ou assumé par un ou plusieurs serveurs de l'entreprise. Comme les demandes sont faites directement entre un consommateur de service ou de données et le fournisseur, l'utilisation des techniques et des technologies orientées présentation dans au moins une partie d'un mashup place la charge de traitement en dehors du domaine de développement de l'entreprise.

### 2.5.2. Avantages et inconvénients des mashups orientés données

Les mashups orientés données présentent leur propre ensemble de défis et d'avantages.

Principalement, un mashup orienté données doit faire face à l'exigence intrinsèque à agir en tant que médiateur et/ou courtier de données. Le mashup dans ce modèle doit prendre des données à partir de plusieurs sites et les mélangez ensemble dans une forme qui sera utile à l'application ou la page mashup. En tant que courtier et/ou intégrateur de données, le mashup doit composer avec de multiples formats de données et éventuellement les protocoles de communication tels que les formats de documents bureautiques (de bureau), des protocoles d'échange de messages de file d'attente et HTTP.

#### 2.5.2.1. In Process (axés sur processus)

Le mélange des données dans un navigateur Web peut être une tâche redoutable. Selon le type de navigateur basé sur les scripts et les technologies qui ne sont pas particulièrement adaptés pour l'intégration des données. La plupart des techniques de mélange des données réalisées dans un navigateur impliquent la manipulation basée sur **XML** ou sur **JSON** de données en utilisant des techniques différentes JavaScript et l'application des résultats à la page Web en utilisant la manipulation **DOM [16].**

La **figure 2.2** illustre le flux de données dans un mashup axé processus. Comme illustré, les données sont reçues à partir de plusieurs sites/hosts et traitées en utilisant les technologies de script dans une page du navigateur.

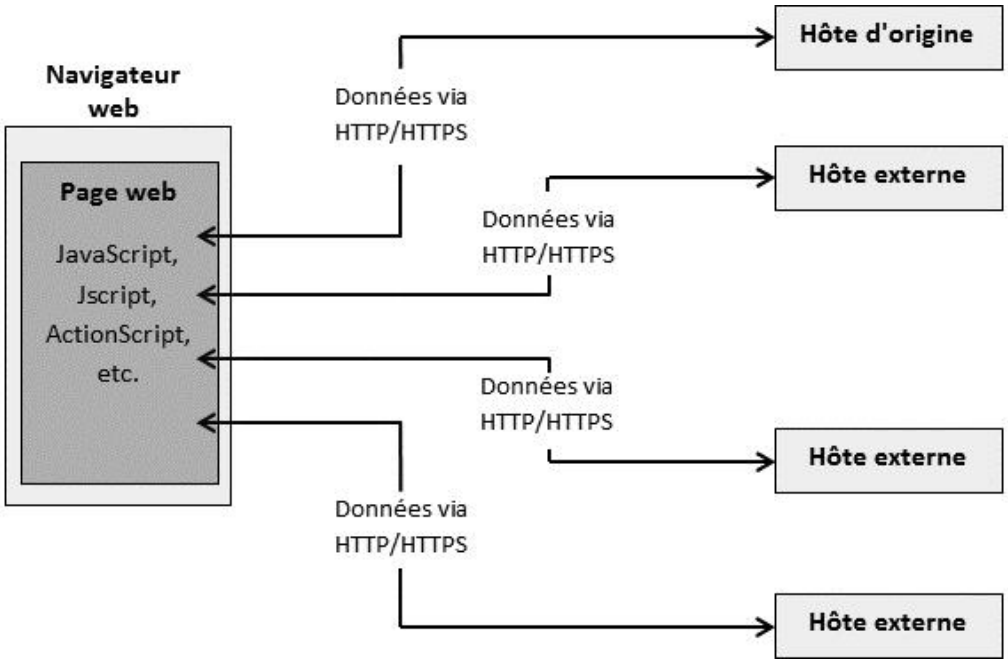



**Figure 2.2 :** Flux de données dans un mashup axé processus.

### 2.5.2.2. Out-of-process (Hors processus)

Le mélange des données en dehors d'un navigateur Web n'est pas nouveau. Toute application web ou cadre d'une application de bureau fournira une certaine forme de temps de test des technologies d'intégration des données. Les Cadres (*framework*) pour le traitement de texte, le texte séparé par une virgule, XML, les données relationnelles et ainsi de suite ont été présentes depuis des décennies et optimisés pour traiter des données hétérogènes de manière très efficace. Les nouvelles technologies ont été construites spécifiquement pour l'intégration des données disparates, tels que les bus de services d'entreprise sont également disponibles dans ce modèle de mashup. **[12].**

La **figure 2.3** illustre le flux de données dans un mashup hors processus. Comme illustré sur la figure, un serveur mashup reçoit des données de plusieurs sites/hosts et intègre les données en utilisant des langages, des bibliothèques et des cadres bien adaptés pour l'analyse des données, l'assemblage, la médiation et ainsi de suite.

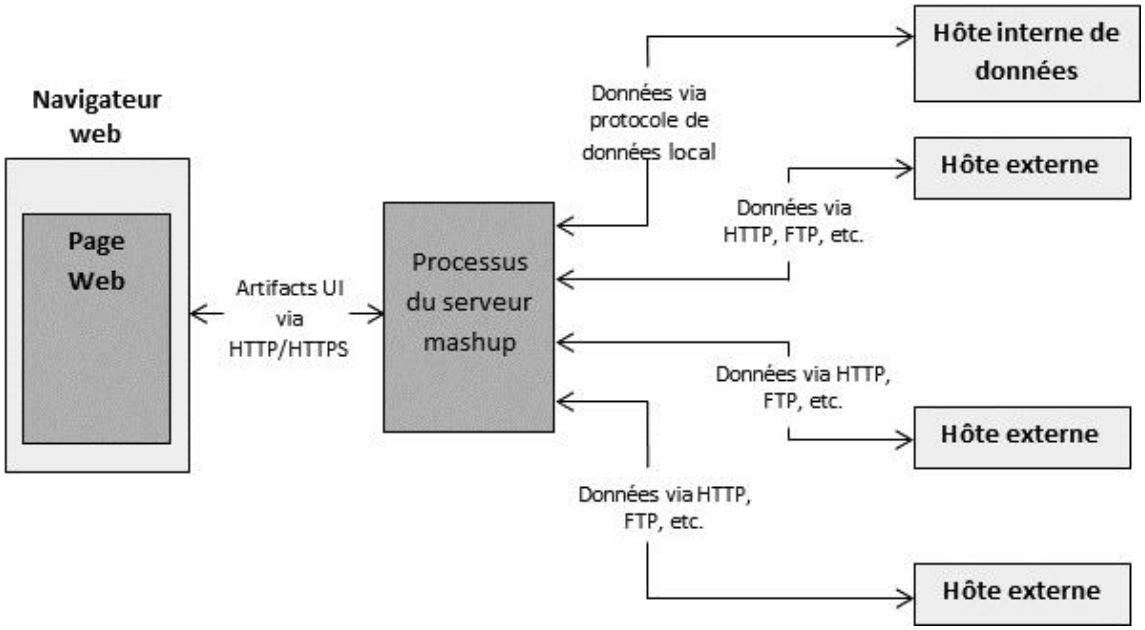



**Figure 2.3 :** Flux de données dans un mashup hors processus.

Le mélange ou le mixage des données hors processus peut généralement traiter les formats de données et appliquer des transformations plus robustes et des techniques d'augmentation de données plus que dans un modèle axé processus.

Le mélange des données hors processus offre l'avantage de l'évaluation de la charge utile de données avant de retourner le résultat au client mashup. Dans ce modèle, un serveur mashup peut surveiller un flux de données uniquement pour les mises à jour de données et renvoyer les mises à jour du client plutôt que la charge utile entière.

Les données de mise en cache sur le serveur mashup et le retour des résultats directement au client, plutôt que d'invoquer la demande de données sur un hôte distant est également un avantage du mixage des données hors processus.

### 2.5.3. Avantages et inconvénients des Mashups orientés processus

Dans un mashup axé sur les processus, les demandes de service sont transmises du client mashup vers le serveur mashup de la même manière comme dans un mashup orienté données hots processus. Donc, du point de vue du client mashup, le processus est le même.

Cependant, du point de vue du serveur mashup, les choses changent.

Un Serveur Mashup orienté processus traite de l'intégration des données ainsi que l'intégration des services et des processus. Tout comme les données peuvent être récupérées à partir de plusieurs sources internes et externes différentes dans un mashup orienté données, les services et les processus peuvent être invoqués sur plusieurs hôtes de services internes et externes et/ou processus différents.

Comme le montre la **figure 2.4**, un Serveur Mashup orienté processus traite de l'intégration des processus et des services de nombreux processus internes et externes et des hôtes différents. Étant donné que cette situation existe dans la plupart des applications Web standard ou des serveurs orientés environnements, il est inévitablement rencontré dans la plupart des environnements mashup où toute forme de traitement a lieu en dehors du client mashup (**navigateur**). Les avantages et les inconvénients de ce modèle sont également partagés avec les environnements orientés services standard du serveur, y compris les subtilités de la IPC, la gestion des transactions et la disponibilité des services.

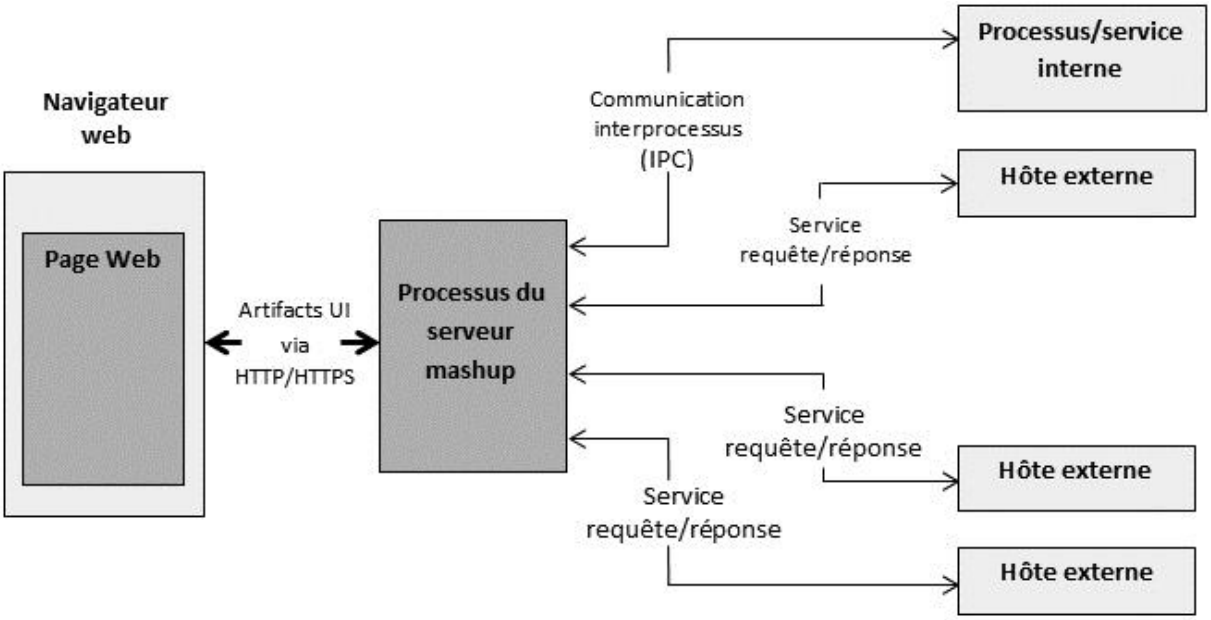



**Figure 2.4 :** Flux de services et de processus dans un mashup orienté processus.

Les mashups orientés processus et les mashups orientés données hors processus permettent de faire face à des clés de sécurité communes, des jetons, des certificats et ainsi de suite en utilisant de nombreuses technologies actuellement disponibles et des *frameworks* tels que **SAML, OpenID, OAuth**. Les mécanismes de sécurité partagés sont désormais disponibles de plus en plus à partir d'un dans le domaine mashup axé processus, mais les technologies serveur dominent encore. La manipulation de la sécurité des données au niveau du serveur on peut également d'intégrer la récupération des données et de la sécurité dans la même étape qui entraîne moins de sauts du client vers le serveur.

Hors processus, les mashups orientés données et les mashups orientés processus permettent aux données et aux services d'être traité de manière asynchrone, ce qui entraîne souvent une utilisation plus efficace de la puissance de traitement et de temps. La concurrence basée sur un navigateur est habituellement beaucoup moins limitée aux appels de concurrence basée sur un serveur.

## 2.6. Techniques des Mashups orientés présentation

Lorsqu'on travaille avec un mashup dans le domaine de la présentation, on est contraint à un environnement qui a évolué d'une manière méli-mélo de traitement de textes simple et les pages graphiques à celui qui est sur le point d'offrir autant de fonctionnalités ou plus comme un *framework* d'application de bureau complexe. Cette évolution désordonnée a souffert en raison de l'inadéquation entre les besoins naturels de l'environnement web et de la nature restreinte des rares fournisseurs de navigateurs primaires. Néanmoins, les normes et les meilleures pratiques voient le jour en offrant un sens au désordre. Les sections suivantes examinent certaines des techniques et technologies populaires utilisées pour produire des mashups dans le domaine de la présentation.

### 2.6.1. Mélange des objets de Présentation

La forme la plus simple du mashup orienté présentation implique l'agrégation d'objets de l'interface utilisateur dans une page Web d'une manière type portail qui est complètement séparé les uns des autres en utilisant des zones distinctes dans une seule page HTML. Dans ce modèle, les objets de l'interface utilisateur tels que des **gadgets, des widgets, des extraits de code HTML**, **JavaScript** et **on-demand JavaScript** sont intégrés dans un document HTML en utilisant des éléments de mise en page et des techniques telles que les tableaux HTML et CSS de positionnement **[12].**

Lors du mélange ou mixage des artefacts de l'interface utilisateur dans une page Web en utilisant des techniques de mise en page du navigateur, chaque *artefact* se trouve généralement dans sa propre zone distincte, comme l'illustre la **figure 2.5**.

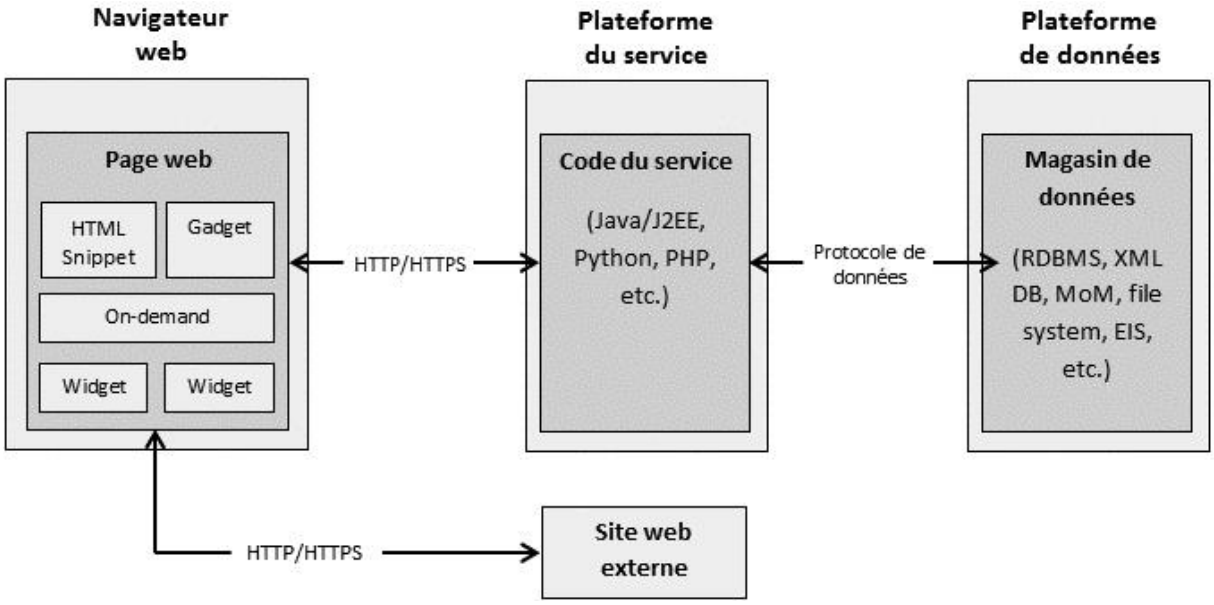



**Figure 2.5 :** Mixage des objets de présentation.

Comme l'illustre la **figure 2.5**, un mashup utilisant l'interface utilisateur des objets agrégés fait référence à un ou plusieurs sites Web à partir d'une page Web et récupère le code de l'interface utilisateur qui construit chaque *artefact* comme un élément distinct dans une zone séparée sur la page du navigateur.

### 2.6.2. Mélange des données de présentation

Une page du navigateur peut également créer et modifier des objets de l'interface utilisateur à l'aide des données extraites de sources multiples en utilisant des formats de données au format **XML, RSS, Atom et JSON**. Dans ce modèle, les données sont récupérées à partir d'un ou plusieurs sites et analysées par le navigateur et les objets de l'interface utilisateur sont créés ou mis à jour à l'aide des techniques de script tels que la manipulation du **DOM** pour modifier le document HTML résultant.

La **figure 2.6** illustre le flux de données à partir d'hôtes de services et de sites externes à un mashup créé par agrégation de données dans le domaine de présentation (page du navigateur).

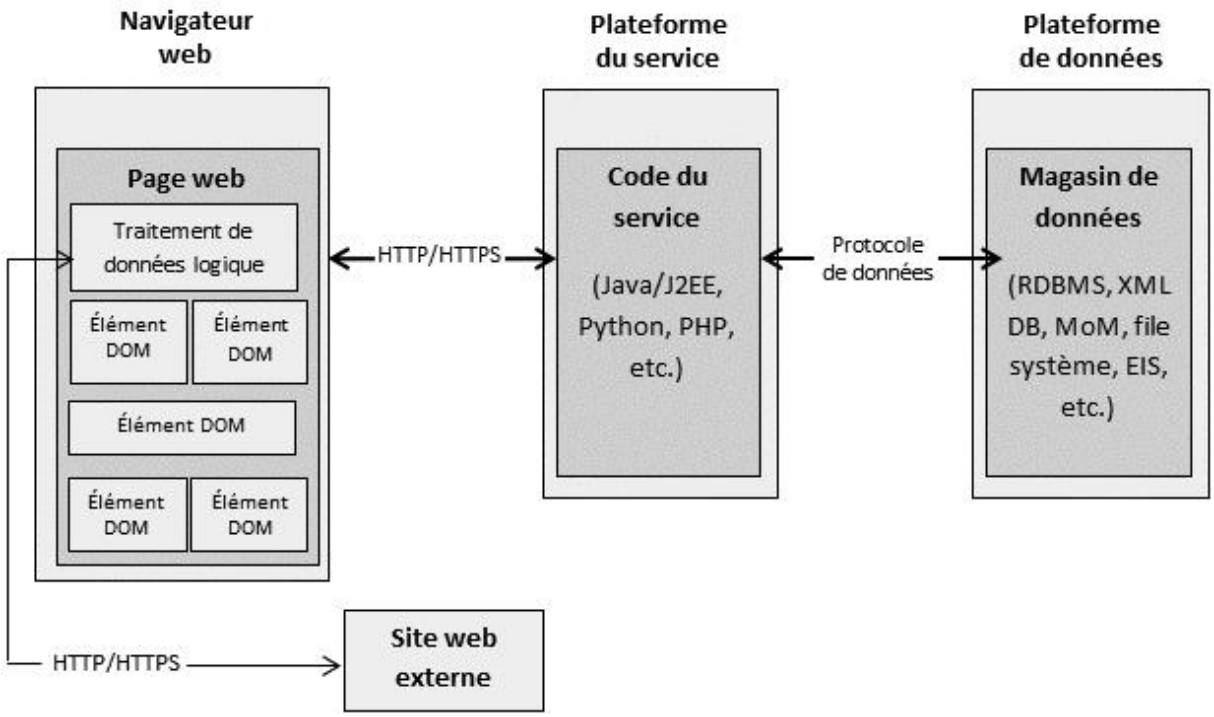



**Figure 2.6 :** Mélange des données de présentation.

Ce modèle est plus complexe que le mélange des objets de l'interface utilisateur. Dans ce modèle, le code de script qui traite les données doit être suffisamment robuste pour gérer plusieurs formats de données ou de restreindre la page d'accès aux services qui ne supportent pas les formats pris en charge par la page. Toutefois, étant donné que le code de script sera l'analyse des données à un niveau élevé, l'interface utilisateur peut être créé et mise à jour afin de créer une expérience utilisateur plus sophistiquée. Ce modèle offre une plus grande flexibilité au prix d'une complexité supplémentaire.

### 2.6.3. Utilisation d'AJAX et de l'objet XMLHttpRequest

*Asynchronous* **JavaScript** et **XML (AJAX)** est un ensemble de technologies et de techniques utilisées pour le traitement des flux de données en utilisant JavaScript et pour créer des pages web dynamiques avec un look sophistiqué. Une caractéristique d'AJAX est l'utilisation des techniques JavaScript et CSS pour mettre à jour un artefact ou objet de l'interface utilisateur sur une page web sans rafraîchir la page entière. AJAX dispose également d'un framework JavaScript basé sur **HTTP requête/réponse** qui peut être utilisé dans une page Web **[17]**.

Le *framework* **HTTP requête/réponse** fourni par AJAX est activé par un composant connu sous le nom de l'objet **XMLHttpRequest**. Cet objectif est soutenu par la plupart des navigateurs et offre la possibilité de passer des données à un hôte externe et recevoir des

données à partir d'un hôte externe en utilisant le protocole **HTTP**. Les demandes peuvent être faites de manière synchrone ou asynchrone. Les requêtes AJAX Asynchrones sont fabriquées en arrière-plan sans affecter la page Web à partir de laquelle la demande est faite.

Le framework AJAX peut envoyer et recevoir pratiquement n'importe quel format de données. Toutefois, les données basées sur **XML** et les données **JSON** sont les charges utiles les plus fréquemment utilisées pour diverses raisons discutées ailleurs dans le présent chapitre. Une fois les données reçues, elles sont analysées et appliquées à la page, généralement en manipulant le **DOM**.

La **figure 2.7** illustre le processus par lequel les flux de données dans une page Web utilisent le framework AJAX. Comme illustré, lors de l'utilisation d'AJAX, les données sont reçues par l'objet **XMLHttpRequest**. Les objets de l'interface utilisateur peuvent alors être créés et modifiés en utilisant les données.

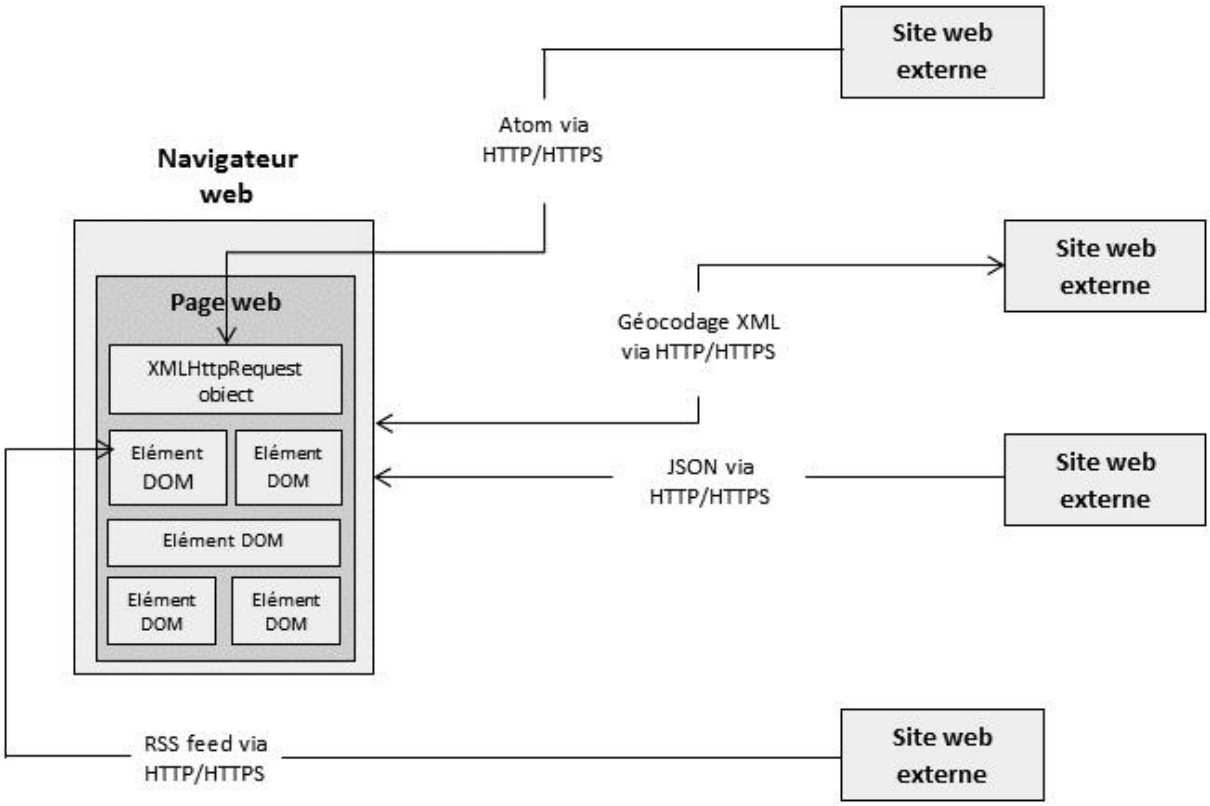



**Figure 2.7 :** Présentation des données d'un mashup en utilisant AJAX.

### 2.6.3.1. Document Object Model (DOM)

Chaque page Web compatible avec JavaScript est représentée en interne à un navigateur comme une instance de **Document Object Model W3C DOM**. DOM est un modèle d'objet indépendant de la plateforme et du langage neutre pour représenter des documents XML qui permet aux programmes et aux scripts d'accéder dynamiquement et de mettre à jour le contenu, la structure et le style d'un document **[18]**.

Le **DOM HTML** fait partie de la spécification DOM officielle qui définit un modèle standard pour les documents HTML. Le DOM HTML facilite l'accès et la manipulation des éléments HTML dans une page Web donnée. Le DOM HTML présente une page Web comme une structure arborescente à base de nœuds contenant des éléments, des attributs et du texte.

Chaque élément d'une page Web (par exemple, table, image, ou paragraphe) est accessible comme un nœud DOM. JavaScript permet la manipulation d'un élément DOM sur une page dynamique. Cela nous permet d'effectuer des opérations telles que le masquage des éléments, en ajoutant ou supprimant des éléments et en modifiant leurs attributs (couleur, taille, position, et ainsi de suite).

La **liste 2.1** présente un document HTML simple.

```
<html>
<head>
<title>A simple HTML doc</title>
</head>
<body>
   <p>This is a simple HTML document.</p>
   <img id="image1" src="http://example.com/image1.png" width="250" height="350"/>
</body>
</html>
```

**Liste 2.1 :** Document HTML simple.

```
function changeImageSize()
{
   var anIMGElement = document.getElementById("image1");
   anIMGElement.width = "400";
   anIMGElement.height = "300";
}
```

**Liste 2.2 :** Manipulation JavaScript du DOM.

Comme le montre la **liste 2.2**, le DOM peut être consulté par le variable globale "document" Avec une référence à la variable document, on peut parcourir les nœuds du DOM pour trouver un élément par son nom ou id.

### 2.6.3.2. Extensible Markup Language (XML)

XML (eXtensible Markup Language) est une spécification et un standard pour créer des langages de balisage auto-descriptifs. Il est extensible en ce sens qu'il permet de définir nos propres éléments. Il est largement utilisé dans les frameworks de transformation de données et d'intégration pour faciliter le transfert et l'intégration des données structurées entre systèmes et applications disparates. XML est utilisé dans de nombreux environnements Web compatibles comme une norme d'annotation de documents et comme format de sérialisation des données **[19].**

La **liste 2.3** est un exemple de document XML simple.

```
<?xml version="1.0" encoding="UTF-8"?>
<contacts>
   <contact>
      <name>John Doe</name>
      <address1>123 anywhere st.</address1>
      <address2>Apt 456</address2>
      <city>Yourtown</city>
      <state>CA</state>
      <zip>12345</zip>
      <country>USA</country>
   </contact>
   <contact>
      <name>Jane Doe</name>
      <address1>456 S 789 W</address1>
      <address2>Suite 987</address2>
      <city>Mytown</city>
      <state>NY</state>
      <zip>54321</zip>
      <country>USA</country>
   </contact>
</contacts>
```

**Liste 2.3 :** Document XML simple.

Les mashups orientés Présentation consomme XML et les dérivés XML et retournées par les hôtes de services et les sites Web. Une fois que XML est reçu, il est analysé et appliqué à une page Web donnée. Comme XML est analysé, les techniques de manipulation du DOM sont généralement appliquées pour mettre à jour des objets de l'interface utilisateur.

### 2.6.3.3. JavaScript Object Notation (JSON)

JSON (JavaScript Object Notation) est simple, basé sur une chaîne, échange le format de données provenant de littéraux d'objet JavaScript. JSON est très facile pour les utilisateurs pour lire, écrire et analyser les moteurs JavaScript. Les chaînes, les nombres, les booléens, les tableaux et les objets peuvent tous être représentés par des littéraux de chaîne dans un objet JSON **[16].**

La **liste 2.4** est un exemple d'un simple objet JSON:

```
{
  'contacts':
  [{
    'name':'John Doe',
    'address1':'123 anywhere st.',
    'address2':'Apt 456',
    'city':'Yourtown',
    'state':'CA',
    'zip':'12345',
    'country':'USA'
  },
  {
    'name':'Jane Doe',
    'address1':'456 S 789 W',
    'address2':'Suite 987',
    'city':'Mytown',
    'state':'NY',
    'zip':'54321',
    'country':'USA'
  }]
`
```

**Liste 2.4 :** JavaScript Object Notation Objet (JSON)

Les mashups orientés Présentation consomment également des objets JSON renvoyées par les hôtes de services et les sites Web. Une fois un objet JSON est reçu, il doit être analysé afin d'être appliqué à une page Web donnée. Depuis que JSON est pur JavaScript, il est facilement analysé à l'aide des utilitaires standards JavaScript. Comme avec XML, les techniques de manipulation du DOM sont généralement appliquées pour mettre à jour les artefacts ou éléments de l'interface utilisateur une fois l'objet JSON analysé.

## 2.7. Techniques des Mashups orientés données

Les données peuvent être mélangées ensemble in-process ou out-of-process (hors processus). Ces deux domaines généralement synonymes d'un navigateur Web et d'un serveur d'applications distant, respectivement. Beaucoup de données horaires seront mélangées dans une approche hybride utilisant à la fois les techniques in-process et out-of-process. Cette section présente en détail quelques-unes des techniques utilisées pour les mashups de données in-process et out-of-process.

### 2.7.1. Mélange des données in-process

Le mélange des données in-process consiste à appliquer des techniques d'intégration de données dans le même processus que la page du mashup. Cela est généralement fait avec du code de script comme **JavaScript** et **JScript**. Cependant, les technologies des composants propriétaires tels que les **applets Java** et **ActionScript** peuvent être utilisés.

Pendant le processus de mixage ou mélange des données dans ce modèle, une demande est faite à un service et les données sont renvoyées dans la réponse du service. Les données sont ensuite analysées, traitées et appliquées aux objets de l'interface utilisateur dans la page. La manipulation du DOM est généralement utilisée pour appliquer les données traitées.

#### 2.7.1.1. Mélange des données XML In-Process

Tous les navigateurs Web standard exposent un objet analyseur XML en JavaScript qui peut être utilisé pour charger et analyser les données XML. Chaque analyseur lit les données XML à partir d'une chaîne, par conséquent, une réponse chaîne renvoyée par un appel de service peut être transmise à l'analyseur XML et traitées en cas de besoin.

```
<script>
       function parseXMLData(xmlString){
         var xmlDoc;
         if (document.implementation.createDocument){
           // Create the Mozilla DOM parser
           var domParser = new DOMParser();
           // Create the XML document object
           xmlDoc = domParser.parseFromString(xmlString, "text/xml");
         }
         else if (window.ActiveXObject){
           // Create the Microsoft DOM parser
           xmlDoc = new ActiveXObject("Microsoft.XMLDOM");
           xmlDoc.async = "false";
           // Create the XML document object
           xmlDoc.loadXML(xmlString);
         }
         // get root node
         var contactsNode = xmlDoc.getElementsByTagName('contacts')[0];
         // traverse the tree
         for (var i = 0; i < contactsNode.childNodes.length; i++){
           var contactNode = contactsNode.childNodes.item(i);
           for (j = 0; j < contactNode.childNodes.length; j++){
             var itemNode = contactNode.childNodes.item(j);
             if (itemNode.childNodes.length > 0){
               var itemTextNode = itemNode.childNodes.item(0);
               var paraEl = document.createElement("p");
               var textEl =
                 document.createTextNode(itemNode.nodeName + ":"
                                          + itemTextNode.data);
               paraEl.appendChild(textEl);
               var bodyEl =
                 document.getElementsByTagName("body").item(0);
               bodyEl.appendChild(paraEl);
             }
           }
         }
       }
</script>
```

**Liste 2.5 :** Analyseur XML à l'aide de JavaScript.

Dans la liste qui précède un analyseur DOM est instancié et est accessible pour obtenir le nœud racine.

Chaque nœud est alors traversé par l'intermédiaire de ses nœuds fils jusqu'à ce que le nœud du texte de données désiré se trouve. Notez qu'il existe un analyseur différent utilisé pour Microsoft Internet Explorer et Mozilla.

#### 2.7.1.2. Mélange des données JSON In-Process :

Depuis que JSON est pur JavaScript, une chaîne contenant les données JSON peut être évaluée simplement pour créer un objet JavaScript. Ensuite, l'objet JavaScript peut être consulté en utilisant JavaScript normalement. Par exemple, supposons qu'on travaille avec une chaîne contenant les données JSON indiquées précédemment dans le **Liste 2.4**. On peut passer cette chaîne à la fonction JavaScript **eval** comme suit:

```
var jsonObj = eval ("(" + jsonString + ")");
```

Cela crée un objet JavaScript sur lequel nous pouvons opérer en utilisant des techniques standard de JavaScript. Par exemple, on peut accéder au champ 'nom' pour le premier contact avec l'extrait suivant:

```
jsonObj.contacts[0].nom
```

En règle générale, on connaît la structure des données au préalable. Cependant, on pourrait ne pas connaître la longueur de données du tableau dans la structure. Dans ce cas, l'objet JSON doit être transversal et analysé les données du tableau dynamique.

### 2.7.2. Mélange des données Out-of-Process

Le mélange des données hors processus consiste à appliquer des techniques d'intégration de données dans un processus séparé, généralement sur un hôte/serveur distant. Dans ce modèle, mashup, un module logiciel à distance reçoit des requêtes d'un client et prend les mesures nécessaires pour recueillir et transformer les données nécessaires pour formuler une réponse.

Les technologies et techniques dans ce chevauchement d'approche avec des technologies et des techniques d'intégration de données d'entreprise, dont la portée est au-delà de cette discussion. Toutefois, cette section présente une vue de haut niveau de quelques-unes des approches les plus courantes de transformation et d'intégration des données d'entreprise en cours d'utilisation:

- **La conversion des données par Force-Brute :** Cette technique consiste à convertir un format de données vers un autre utilisant des outils de conversion de propriétés ou d'un programme de conversion personnalisé octet par octet. Les applications propriétaires offrent souvent un cadre ou *framework* étendu permettant à des tiers de créer des composants ou plug-ins qui permettent de convertir un format de données vers un autre.

- **La mise en correspondance des données cartographiques :** Implique la création et l'application d'une carte d'éléments de données entre deux modèles de données disparates de source et de destination. La carte est utilisé par les programmes de conversion pour déterminer comment un élément de l'ensemble de données source s'applique à l'ensemble de données de destination. **Extensible Stylesheet Language Transformations (XSLT)** est souvent utilisé dans cette approche pour convertir les données XML d'une forme à une autre **[20]**.

- **Cartographie sémantique :** Cette approche utilise un registre de métadonnées contenant des synonymes pour les éléments de données qui peuvent être interrogés par des outils de conversion qui utilisent les synonymes de guide pour la conversion d'un élément de données à l'autre.

## 2.8. Techniques des Mashups orientés processus

Un mashup orienté processus implique mixage ensemble des services et/ou processus. Les techniques utilisées pour cette gamme de modèles est la méthode combinant simplement les appels à un objet à un système complexe de *workflow* (flux de travail) structuré.

Dans la **figure 2.8**, object2 interagit avec un certain nombre de services et de processus différents. Il est en interaction avec object1 à l'aide d'un appel de méthode standard et d'un site externe à l'aide d'un appel de service via un protocole Internet. Object2 interagit également avec un système de flux interne en utilisant une messagerie asynchrone. Les résultats de ces appels sont alors mélangés ensemble pour formuler la réponse qui sera finalement renvoyé au client mashup (page web).

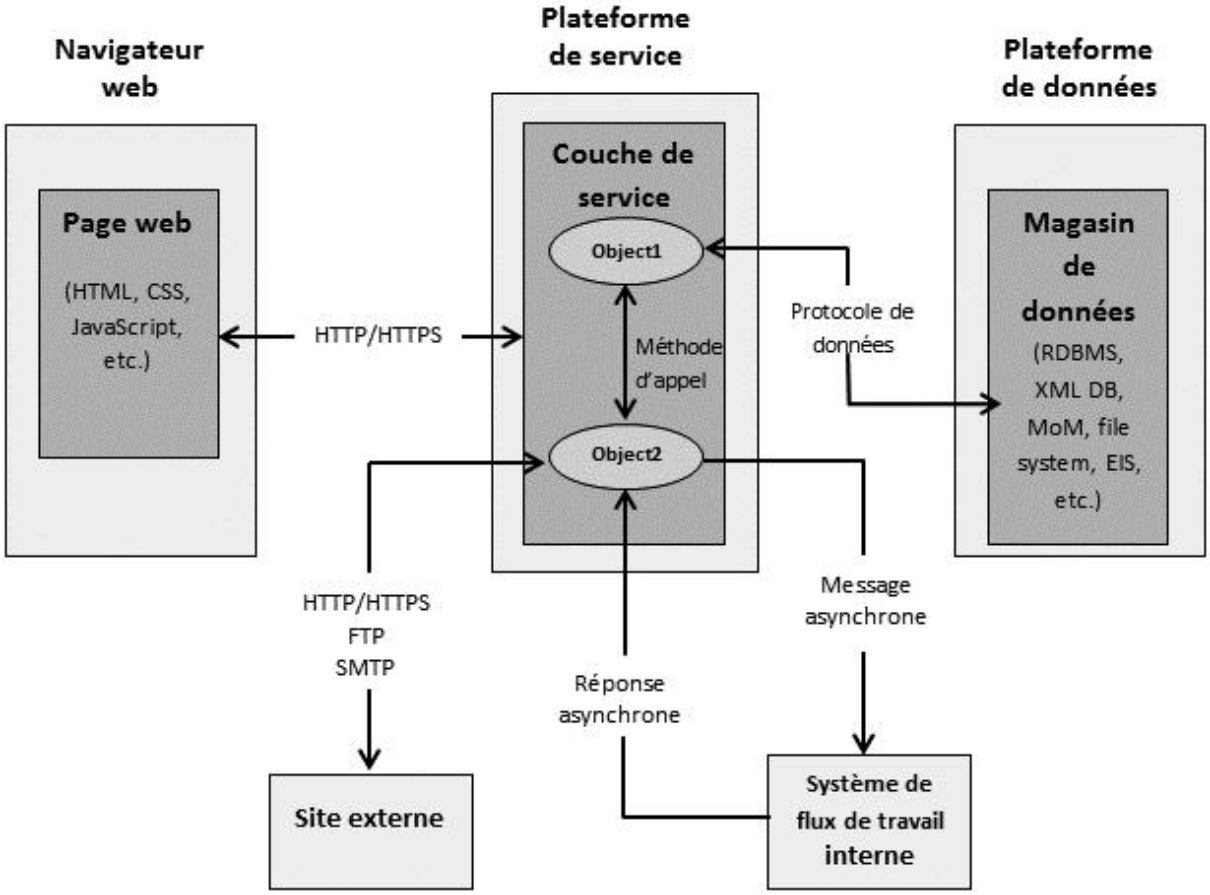



**Figure 2.8 :** Architecture des mashups orientés processus.

## 2.9. Mashups hybrides

En réalité, les mashups d'entreprise impliquent généralement des techniques et technologies pour chacun des trois domaines mashup. Les Widgets, les Gadgets, ou les scripts dynamiques seront récupérées en utilisant les techniques orientés présentation. Les données seront récupérées à l'aide des techniques orientées données in-process. Plus les données seront récupérées sur le serveur en utilisant les techniques hors processus. Les Services et processus seront regroupés à formuler des réponses sur le serveur qui va retourner au client mashup les résultats des requêtes APIs de services.

La **figure 2.9** illustre un environnement d'un mashup d'entreprise typique où les données et les services sont accessibles à partir d'une page web (in-process) et de la plateforme de service (sur le serveur, out-of-process). Les techniques de présentation, les techniques orientées données et orientées processus doivent tous être utilisées pour répondre aux besoins de cet environnement.

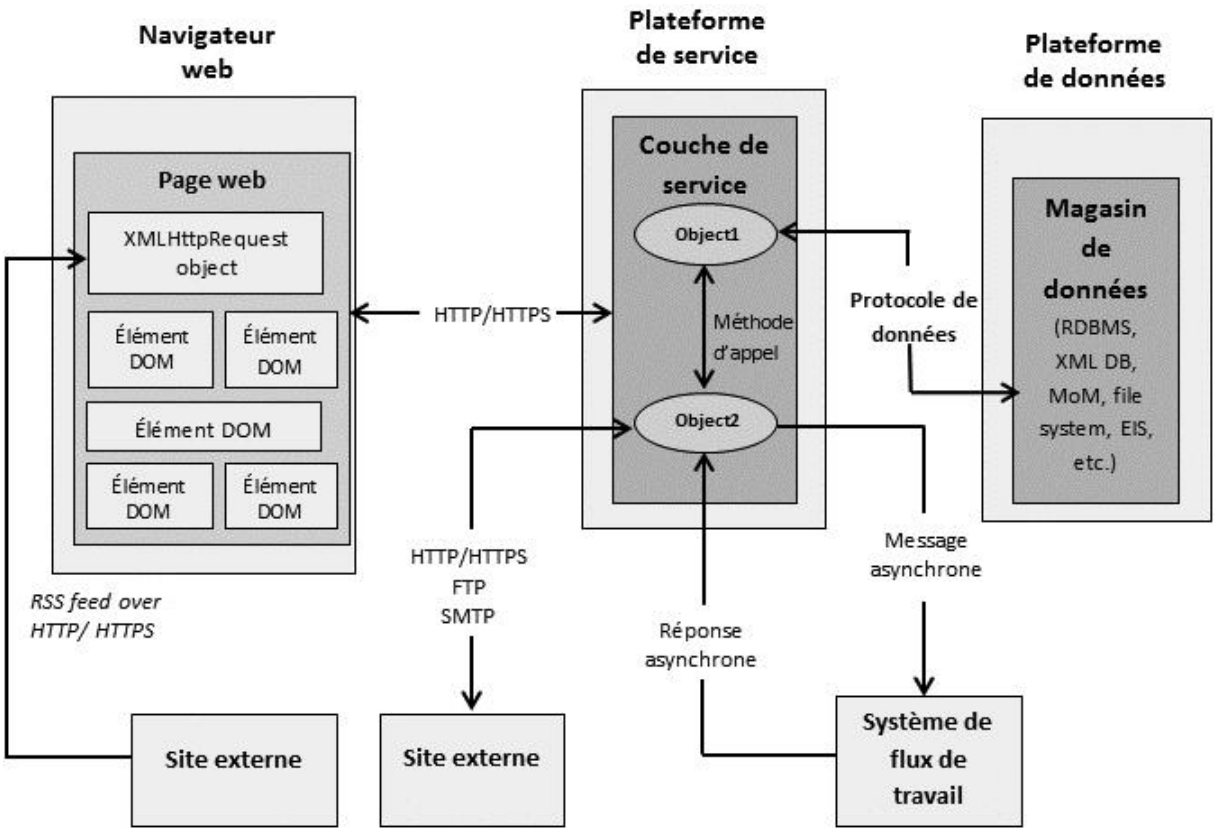



**Figure 2.9 :** Architecture des mashups hybrides.

Les techniques et les domaines mashup présentées dans ce chapitre sont abordés dans les chapitres suivants. Pour l'instant, je démontre certaines des techniques les plus simples d'un mashup orienté présentation dans un petit exemple dans la section suivante.

## 2.10. Implémentation d'un Mashup simple

Cette section montre l'application des concepts présentés dans ce chapitre pour créer un mashup simple orienté présentation. En dehors d'une plate-forme de services locaux, les sources utilisées pour le mashup sont accessibles au public et fournit soit une API de service, un flux de données **RSS**, ou un flux du script dynamique.

Pour des raisons de couverture complète des domaines des mashups abordés, le mashup fera des appels de service et récupérera des données à partir des sites externes ainsi que d'une plate-forme de service local qui fonctionne sur le modèle illustré à la **figure 2.10**.

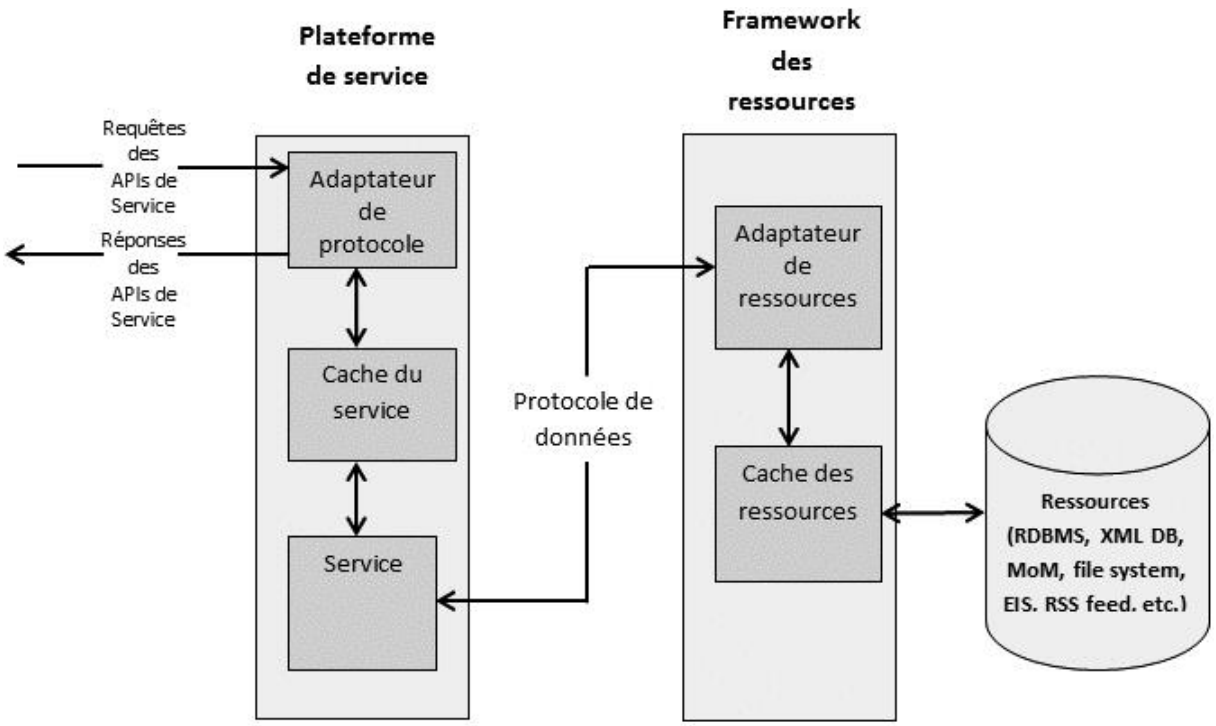



**Figure 2.10 :** Architecture de la plateforme de services.

La **figure 2.10** illustre une plate-forme de service qui utilise seulement un petit nombre de composants primaires pour le processus de service et les demandes de ressources. La plate-forme utilise l'approche **Representational State Transfer (REST)** pour le service et les requêtes/réponses des ressources. La plate-forme de service fournit un accès à des services et utilise un framework de ressources simple pour créer, récupérer, mettre à jour et supprimer des ressources **[21]**.

L'application (dans le **liste 2.6** et à la **figure 2.11**) permet aux utilisateurs de visualiser les données disparates qui pourraient être disponibles dans une entreprise typique. Les données sont présentées de manière type portail pour simplifier la configuration et l'interface de gestion du code. L'application intègre une liste des documents de l'entreprise, un *feed* de carte (map feed), un flux RSS, un gadget calendrier, un chicklet *Twitter* compte et une liste d'archives *Twitter* livré sous forme de flux.

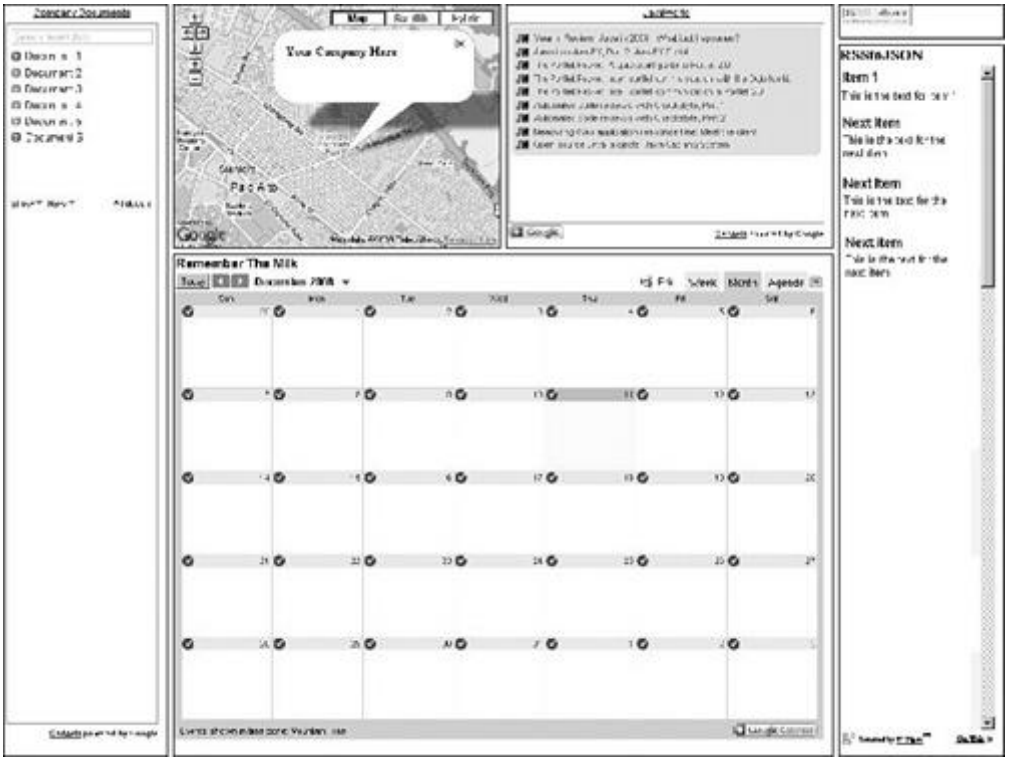



**Figure 2.11 :** Exemple d'un mashup orienté présentation.

```
<html>
<head>
<!-- Include Google Maps Javascript Library -->
<script type="text/javascript" src="http://maps.google.com/maps?file=api&v=1& key=
ABQIAAAA01HpWF7mf2aW91RNaGDc7xTfGML3OZxtDDthfq-
aZ1uFtrk9MRS_VWEizymnfki_h89lqU7A0ts2PA">
</script>

<script type="text/javascript">
```

**Liste 2.6 :** Mashup orienté présentation avec les éléments agrégés de l'interface utilisateur.

La **figure 2.11** illustre le résultat final de l'exemple du mashup orienté présentation discuté précédemment.

La fonction de charge dans la **liste 2.7** récupère une carte *Google* codée en dur et il l'applique à l'élément **div** de la carte dans le DOM HTML.

```
function load()
  {
    if (GBrowserIsCompatible())
    {
      // create map component in div with the id = "map"
      var map = new GMap2(document.getElementById("map"));
      // create map components components
      map.addControl(new GSmallMapControl());
      map.addControl(new GMapTypeControl());
      // create center point when map is displayed
      map.setCenter(new GLatLng(37.4419, -122.1419), 13);
      map.openInfoWindow(map.getCenter(),
                         "<b>Your Company Here</b>");
      // re-open the info balloon if they close it
      var point = new GLatLng(37.395746, -121.952234);
      map.addOverlay(createMarker(point, 1));
    }
  }

  function createMarker(point, number)
  {
    var marker = new GMarker(point);
    // create clickable point with title for address
    GEvent.addListener(marker, "click", function()
    {
       marker.openInfoWindowHtml("<b>Your Company Here</b>");
    });
    return marker;
  }

  function retrieveExternalData()
  {
      var script = document.createElement("script");
      script.src =
        'http://www.example.com/mashups/someservice';
      script.type = 'text/javascript';
      document.body.appendChild(script);
  }

</script>
</head>
<body onload="load()">
```

**Liste 2.7 :** L'application d'une carte Google à une page Web.

Dans l'exemple montré dans la **liste 2.8** l'élément **div** est ajouté pour contenir une liste de documents d'entreprises récupérées à partir d'un compte *Google Docs* en utilisant un script dynamique.

```
    <!-- div to hold documents -->
    <div id="docs"
         style="border-style:ridge; position: absolute;
            left: 10px; top: 10px; width:200px; height:930px">
    <script src="http://gmodules.com/ig/ifr?
            url=http://www.google.com/ig/modules/docs.xml
            &amp;up_numDocuments=9&amp;up_showLastEdit=1
            &amp;synd=open&amp;w=180&amp;h=860
            &amp;title=Company+Documents
            &amp;lang=en&amp;country=ALL
            &amp;border=%23ffffff%7C3px%2C1px+solid+%23999999
            &amp;output=js"></script>
    </div>
```

**Liste 2.8 :** Script dynamique pour afficher une liste de documents.

La **liste 2.9** illustre un élément **div** qui contiendra les résultats de la récupération de *Google map*.

```
<div id="map"
        style="border-style:ridge; position: absolute;
        left: 220px; top: 10px; width:400px; height:300px">
</div>
```

**Liste 2.9 :** Élément pour contenir une carte *Google Map*.

Dans la **liste 2.10** un élément **div** est ajouté pour contenir un flux RSS en utilisant un script dynamique.

```
    <!-- div to hold RSS feed -->
    <div id="feed"
         style="border-style:ridge; position: absolute;
         left: 630px; top: 10px; width:400px; height:300px">
    <script src="http://gmodules.com/ig/ifr?
      url=http://customrss.googlepages.com/customrss.xml
      &amp;up_rssurl=http%3A%2F%2Fwww.javaworld.com%2Findex.xml
      &amp;up_title=CustomRSS
      &amp;up_titleurl=http%3A%2F%2Fcustomrss.googlepages.com
      &amp;up_num_entries=10&amp;up_linkaction=showdescription
      &amp;up_background=E1E9C3&amp;up_border=CFC58E
      &amp;up_round=1&amp;up_fontfamily=Arial
      &amp;up_fontsize=8pt&amp;up_openfontsize=9pt
      &amp;up_itempadding=3px&amp;up_bullet=icon
      &amp;up_custicon=Overrides%20favicon.ico
      &amp;up_boxicon=1&amp;up_opacity=20
      &amp;up_itemlinkcolor=596F3E&amp;up_itemlinkweight=Normal
      &amp;up_itemlinkdecoration=None&amp;up_vlinkcolor=C7CFA8
      &amp;up_vlinkweight=Normal&amp;up_vlinkdecoration=None
      &amp;up_showdate=1&amp;up_datecolor=9F9F9F
      &amp;up_tcolor=1C57A9&amp;up_thighlight=FFF19D
      &amp;up_desclinkcolor=1B5790&amp;up_color=000000
      &amp;up_dback=FFFFFF&amp;up_dborder=DFCE6F
      &amp;up_desclinkweight=Bold&amp;up_desclinkdecoration=None
      &amp;synd=open&amp;w=380&amp;h=240&amp;title=JavaWorld
      &amp;border=%23ffffff%7C3px%2C1px+solid+%23999999
      &amp;output=js"></script>
    </div>
```

**Liste 2.10 :** Élément pour contenir un flux RSS.

La **liste 2.11** montre un élément **div** pour contenir un agenda *Google* récupéré à l'aide d'un badge JavaScript.

```
    <!-- div to hold calendar -->
    <div id="calendar"
         style="border-style:ridge; position: absolute;
         left: 220px; top: 320px; width:810px; height:620px">
    <iframe src="http://www.google.com/calendar/embed?
                 src=o78s3eqe3ov403cpuav2bje5ja9j1tp2%40
                 import.calendar.google.com&ctz=America/Denver"
      style="border: 0"
      width="800" height="600" frameborder="0" scrolling="no">
      </iframe>
    </div>
```

**Liste 2.11 :** Élément *Google Agenda*.

La **liste 2.12** est un élément **div** pour contenir la chicklet compte Twitter.

```
    <!-- div to hold the Twitter counter chicklet -->
    <div id="chicklet" style="border-style:ridge; position:absolute; left:1040px;
top: 10px; width:200px; height:40px">
        <a href="http://twittercounter.com/?username=jhanson583"
            title="TwitterCounter for @jhanson583">
             <img src="http://twittercounter.com/counter/?username=jhanson583"
width="88"
height="26" style="border:none;" alt="TwitterCounter for @jhanson583" />
        </a>
    </div>
```

**Liste 2.12 :** Élément div pour contenir un *Chicklet Twitter*.

La **liste 2.13** illustre l'élément div pour contenir le flux RSS de Twitter en utilisant un script dynamique.

```
    <!-- div to hold twitter feed -->
    <div id="ufbadge"
         style="border-style:ridge; position: absolute;
         left: 1040px; top: 60px; width:200px; height:880px">
      <script src="http://pipes.yahoo.com/js/listbadge.js">
              {"pipe_id":"dq0Qhuqp3BG1psChjtzu1g",
               "_btype":"list",
               "pipe_params":{
                  "urlRSS":"http:\/\/twitter.com
                  \/statuses\/user_timeline\/10852552.rss"},
               "width":"190",
               "height":"870"}
      </script>
    </div>
```

**Liste 2.13 :** Élément div pour le flux RSS de *twitter*.

Dans la **liste 2.14**, on ajoute l'élément div pour contenir le flux RSS en utilisant le script locale dynamique.

```
<!-- div to hold local RSS feed -->
<div id="localfeed"
     style="border-style:ridge; position: absolute;
     left: 10px; top: 950px; width:1210px; height:200px">
  <script src="http://localhost:8080/mashups/js/rssbadge.js">
          {"urlRSS":"http:\/\/localhost:8080
          \/mashups\/services\/feeds\/zurn.rss"}
  </script>
</div>
```

**Liste 2.14 :** Élément du Flux RSS.

Dans ce qui suit, un tableau comparatif des différents domaines technologiques, ainsi que les avantages et inconvénients de chacun d'eux.

| | **Définition** | **Avantages** | **Inconvénients** |
|---|---|---|---|
| **Orienté présentation** | • Mélange les différents éléments d'interfaces utilisateurs pour créer une nouvelle application ou une nouvelle page. | • rapide et facile à construire.<br>• s'appuie principalement sur des données et des objets de l'interface utilisateur. | • Les performances sont généralement très sensibles. |
| **Orienté données** | • Implique la combinaison des données provenant d'un ou plusieurs sites hébergés à l'extérieur ainsi que dans une page web ou un volet d'application. | • Le mélange des données peut traiter tous les formats de données.<br>• Applique des transformations plus robustes. | • Le rôle de courtier joué par le Mashup est une tache redoutable. |
| **Orienté processus** | • Implique des fonctionnalités combinant ensemble dans un ou plusieurs processus externes à l'aide des langages de programmation. | • Disponibilité des services. | • Manque de flexibilité. |

**Tableau 2.1 :** récapitulatif des avantages et inconvénients des différents domaines.

## 2.11. Conclusion

Dans ce chapitre, on a discuté de certains des styles, des techniques et des technologies qui sont utilisées pour construire des mashups pour chacun des trois domaines primaires des mashup à savoir présentation, données et processus.

On verra comment déterminer le domaine pour notre mashup en analysant les objets et les données qui doivent être mixés. Le domaine est déterminé par l'analyse des artefacts de l'interface utilisateur (présentation), de données et/ou des fonctionnalités d'applications (processus).

Le style de mise en œuvre, les techniques et les technologies utilisées pour un mashup donné dépendent du domaine du mashup déterminé par l'analyse des artefacts et des données. Les techniques et les technologies dépendent aussi de l'endroit où le traitement se fera in-process ou out-of-process.

Une fois que le domaine est déterminé et les sources de ces objets et des données sont mis en place, on peut procéder à l'application des langues, des processus et des méthodologies pour concevoir et construire le mashup.

Le mashup dans l'exemple précédent illustre les différentes techniques et technologies appliquées à un mashup simple orienté présentation. Dans le prochain chapitre nous allons discuter sur les approches de développement des mashups.

# Chapitre 3 : Les approches de développement des mashups

## 3.1. Introduction

Un mashup peut être défini comme une application Web de la situation qui extrait et combine les données et fonctionnalités provenant de différentes sources pour répondre aux besoins des utilisateurs **[22]**. Au cours des dernières années, un certain nombre de frameworks ont été proposés pour simplifier le processus de création des mashups de sorte que les utilisateurs sans connaissances en programmation soient en mesure de créer des mashups. Dans ce qui suit, nous comparons ces frameworks par rapport aux besoins en compétences de l'utilisateur. L'analyse est basée sur un aperçu représentatif des frameworks mashup qui ont été proposés par les entreprises et la recherche.

Le reste est organisé comme suit. Dans la section 2, nous décrivons la catégorisation de l'ensemble des frameworks et de fournir un aperçu sur ça. Dans la section 3 nous identifions les limites des approches existantes. La section 4 conclut ce papier avec un aperçu sur les sujets qui devraient être abordés par la recherche future sur les frameworks mashups.

## 3.2. Analyse

Dans cette section, nous donnons un aperçu sur les différents frameworks de mashups et leurs paradigmes pour le développement de ces derniers. Le **tableau 3.1** illustre l'environnement des frameworks mashup en relation avec le paradigme du développement ainsi que les compétences requises de l'utilisateur.

### 3.2.1. Modèle de catégorisation des framewoorks de mashups

Les compétences des utilisateurs peuvent être divisées en catégories comme développeur, utilisateur expert et utilisateur occasionnel. Un développeur doit être familier avec la programmation, les technologies web, les différentes APIs, ainsi que l'utilisation d'outils de développement.

Un utilisateur expert n'a pas de compétences en programmation, par définition, mais il possède une connaissance fonctionnelle détaillée d'un outil spécifique ou d'un ensemble d'outils.

Les utilisateurs occasionnels ont uniquement les compétences nécessaires pour utiliser les fonctionnalités d'un navigateur et sont capables de naviguer à travers le Web. Les compétences requises pour les utilisateurs dépendent de l'approche de développement qui est utilisée pour créer le mashup. L'approche de développement des mashups peut être basée sur la **création manuelle, semi-automatique ou automatique.**

La **création manuelle** signifie que les sources de données et les fonctionnalités doivent être intégrées par **programmation** ou par **scripting**.

La **création Semi-automatique** couvre la création de mashup avec des tableurs ou des feuilles de calculs, les outils orientés câbles et de la programmation par démonstration.

La **création automatique** signifie que le mashup est généré sans intervention de l'utilisateur. Cela signifie que les ressources (par exemple, les services Web connus) sont choisies et appelées automatiquement. En outre, la création de mashup peut être appelée automatique et adaptative, si ce processus génère des mashups adaptés aux intérêts changeants des utilisateurs, des tâches et de l'expérience d'un utilisateur, ce qui signifie qu'il est basé sur un modèle d'utilisateur spécifique **[23]**.

| | | |
|---|---|---|
| **Création automatique** | -Mashup automatique des applications composites d'IBM Lotus Expeditor.<br>-Génération automatique de mashups sémantiques dans IBM WebSphere Portal.<br>-Mashups personnalisés: Opportunités et défis pour la modélisation de l'utilisateur. | **Création Automatique** |
| | MashMaker, Piggy Bank | |
| **Programmation par démonstration** | Internet Scrapbook, Karma, Dapper, Potluck. | **Création Semi-automatique** |
| **Paradigme de câblage** | Apatar, IBM Damia, IBM Lotus Mashups, JackBe Presto Wires, Microsoft Popfly, Yahoo Pipes, Openkapow, Proto Financial, Antracithe, Marmite, SABRE. | |
| **Tableur** | Strikelron SOA Express, Extensio Extender | |
| **Langage de script** | RSS Bus, Mashup Feeds, GME, JackBe Presto Mashup Studio, Dynamic Fusion of web Data, WSO2 Mashup Server. | **Création Manuelle** |
| **Environnement de programmation** | Bungee Connect, Eclipse, Mashup Feeds, IBM WebSphere sMash. | |

**Tableau 3.1 :** Tableau récapitulatif sur les frameworks de Mashups **[24].**

### 3.2.2. Frameworks basés sur le paradigme de programmation

Un certain nombre d'outils pour la création de mashups sont basés sur un environnement de développement intégré. **IBM WebSphere sMash [25]** est un environnement de développement et d'exécution pour les applications web dynamiques. Il permet de réutiliser facilement des services Web et permet une intégration rapide des différents services web.

**BungeeConnect [26]** est une autre plate-forme qui est offerte comme service en ligne permettant aux utilisateurs de créer des applications web complètes. Bungee automatise l'importation de services Web accessibles au public ainsi que les bases de données traditionnelles et les entrepôts de données. Ce développement manuel des mashups est supporté par une variété d'autres environnements de développement et ne peut être fait que par les développeurs expérimentés.

### 3.2.3. Frameworks basés sur les langages de script

Le développement de mashups est pris en charge par divers outils qui sont basés sur certains langages de script tels que **Google Mashup Editor [27] (GME)**, **Mashup Web Scripting Language (WMSL) [28]**, **Dynamic Fusion of Web Data [29] [30]**, **WSO2 Mashup Server [31]**. En général, il semble être trop compliqué pour un non-développeur de créer des scripts de ce type dans un moment opportun, car les mashups plus complexes auront besoin d'une quantité considérable de code de script assez complexe.

### 3.2.4. Frameworks basés sur les tableurs (feuilles de calcul)

Des outils basés sur des feuilles de calcul tels que **StrikeIron SOA Express** pour Excel **[32]**, **Extensio Excel Extender [33]** mettent l'accent sur le mixage de données. Contrairement aux outils orientés câbles, les données sont directement insérées dans une feuille de calcul. Cela signifie que les signaux de sortie d'une source de données sont écrits dans les cellules qui ont été sélectionnés par l'utilisateur. Les valeurs des cellules servent alors les entrées suivantes des requêtes de source de données. **StrikeIron SOA Express** et **Excel Extensio Extender** utilisent les services Web **SOAPful** pour créer des mashups. En outre, Extensio Excel Extender peut donner accès à **SAP**, à plusieurs bases de données ainsi qu'aux fichiers plats.

### 3.2.5. Frameworks basés sur le paradigme de câblage

Les outils basés sur les fils tels que **Apatar [34], IBM Damia [35], Marmite [36], SABRE [37], JackBe Presto Wires, Microsoft Popfly, Yahoo Pipes, Openkapow, Proto Financial, Anthracite, Lotus Mashups [38]**, remixe et fusionne des données, la fonctionnalité, ou la présentation par le biais d'un câblage graphique des blocs de construction de base. Cette connexion manuelle est parfois appelée le câblage ou les tuyaux de différents modules, connecteurs, composants ou blocs. Les composants disponibles fournissent des fonctionnalités différentes (par exemple, la récupération des données, la transformation de données, la présentation des données, etc) et doivent être connectés pour assurer la coordination voulue des mashups. Les outils prennent souvent en charge différents types de sources de données telles que **RESTful** et/ou **SOAPful** (par exemple **Openkapow, Proto Financial, Anthracite**) des services web, des bases de données, des tableurs et des fichiers CSV.

### 3.2.6. Frameworks basés sur la programmation par démonstration

La programmation par démonstration permet aux utilisateurs d'apprendre un système par la mise à disposition d'exemples. Le système **Internet Scrapbook [39]** permet aux utilisateurs ayant des compétences en programmation d'automatiser les petites tâches récurrentes de navigation. L'utilisateur est en mesure d'indiquer les fragments de différentes pages Web qui sont intéressantes pour lui et de les agréger en une page remixée personnalisée. L'extraction des données est basée sur la structure HTML de la page Web spécifique. **Dapper [40]** est un service en ligne qui est en mesure de créer une API pour n'importe quel site web. Le site source doit être spécifié initialement, puis l'utilisateur peut sélectionner graphiquement à partir de quelques exemples de sorties les champs qui doivent être extraits. **Karma [41]** utilise la programmation par démonstration pour en extraire des listes de données à partir de pages web par un simple **glisser-déposer** des éléments d'une page Web. Le système exploite les informations de l'arborescence DOM du navigateur et crée une table des données. Les données peuvent être automatiquement jointes à d'autres tables dérivées à partir d'autres pages, par un couple de noms d'attributs et les paires de valeurs. **Huynh et al** proposent l'outil de mashup **Potluck [42]**. **Potluck** prend en entrée un ensemble d'URL qui contiennent les données qui doivent être remixées. Les ensembles de données à partir des pages Web sont indiquées dans un environnement tabulaire. Les champs qui représentent le même attribut sémantique peuvent être définis comme égales à travers un

simple **glisser-déposer** des noms de champs. Dans ce cas, la présentation est automatiquement réorganisée. Ainsi, la médiation entre les différentes hétérogénéités sémantiques se fait implicitement par l'utilisateur bien qu'il agit sur les données récupérées. En outre, le système permet à l'utilisateur de nettoyer et d'homogénéiser les données dans un environnement de navigation à facettes.

### 3.2.7. Frameworks basés sur la création automatique des Mashups

La génération automatique d'applications mashup n'a que récemment commencé à susciter l'intérêt dans la communauté scientifique. **Carlson et al [43]** proposent un *framework* mashup pour les applications mashup composites automatiques à base de **Lotus Expeditor**. Le *framework* met l'accent sur les applications composites constituées de composants non services web (par exemple, **les widgets**). Dans leurs travaux **[44]**, ils ont proposé une approche adaptative et automatique pour la création de mashups. Dans ce *framework* mashup, **SOAPful** ou les services Web de connaissances **RESTful** sont composés automatiquement par composition de services Web sémantique basés sur les intérêts, les tâches et l'expérience d'un utilisateur. Les données récupérées sont fusionnées et présentées comme un mashup. **MaxMash** est un framework qui est dirigé vers la génération automatique. Il promet de composer et de sélectionner les fonctions d'applications en réseau et génère automatiquement l'application mashup **[45]**. Au lieu de réutiliser l'API existante, il s'appuie sur la **messagerie extensible et Presence Protocol (XMPP)**, qui est utilisée pour les applications de messagerie instantanée et utilise le trafic sur le réseau sous-jacent pour créer l'application mashup automatiquement.

Un certain nombre d'initiatives de recherche sont orientées vers les mashups d'adaptation. **MashMaker** est un plugin pour navigateur qui étend le processus normal de navigation avec les fonctionnalités mashup. **MashMaker** ne nécessite pas la spécification du mashup à l'avance par les utilisateurs, comme il est requis par le **workflow** comme outil **[46]**. Les logiciels devinent un mashup que l'utilisateur trouve utile, basé sur le contenu du navigateur. **PiggyBank [47]** est capable de traiter des données RDF qui sont intégrées dans des pages web. Le développement manuel des pages Web de grattage d'écrans spécifiques (par exemple, pour **les pages Flickr**) permet l'extraction non-sémantique des données et leur transformation en RDF. Un avantage de **PiggyBank** est que l'utilisateur n'a pas à se préoccuper sur la récupération et le remixage de données, ce qui est fait automatiquement par la fusion de triplets RDF.

## 3.3. Les approches traditionnelles de composition et de développement

Plusieurs domaines de recherche sont liés à la composition et aux mashups sur le Web. Dans cette section, nous présentons brièvement en revue les domaines de la composition de services, la composition de l'interface utilisateur, les outils d'ingénierie web assistés par ordinateur, les portails Web et portlets, tous les domaines sont particulièrement liés à la composition universelle pour le Web. Dans la section suivante nous mettrons un peu plus l'accent sur les mashups.

### 3.3.1. Approches de composition de services

**BPEL** est un exemple des approches d'orchestration de services **[48]**, c'est un langage de composition standard par **OASIS**. **BPEL** est basée sur les services Web **WSDL-SOAP** et les processus BPEL sont eux-mêmes exposés en tant que services Web. Les flux de contrôle sont exprimés par le biais d'activités structurées et peuvent inclure des exceptions plutôt complexes et le support des transactions. Les données sont transmises entre les services via des variables (style Java). Jusqu'à présent, BPEL est le langage de composition de services le plus largement utilisé. Bien que BPEL ait donné des résultats prometteurs qui sont certes utiles, il est principalement destiné aux programmeurs professionnels comme aux développeurs de processus d'entreprises. Sa complexité **[48]** le rend difficilement applicable pour les mashups Web.

De nombreuses variantes de BPEL ont été développées, par exemple, visant à l'invocation de services **REST [49]** et à exposer les processus BPEL en tant que services REST **[50]**. Dans **[47]**, les auteurs décrivent **Bite**, un langage BPEL de composition léger spécialement conçu pour les environnements **RESTful**. La Plate-forme de code IBM Partageable **[51]** suit une stratégie différente pour la composition de services **REST** ou **SOAP**, un langage de programmation spécifique au domaine à partir duquel le code de l'application Ruby on Rails est généré, comprenant également des interfaces utilisateur pour le Web. Dans **[52]**, les auteurs combinent les techniques de langages de requêtes déclaratifs et la composition des services pour appuyer les requêtes multi-domaines sur plusieurs services, alors que dans **[53]**, les auteurs suivent une approche centrée sur le document de composition de services et propose l'utilisation de **AXML** pour les mashups de services. Toutes ces approches se concentrent sur la couche application et données, les interfaces utilisateur **UI** peuvent alors être programmées au-dessus de la logique d'intégration des services. **mashArt** propose plutôt une intégration universelle comme paradigme de la composition simple et

transparente de l'interface utilisateur, des données et des composants d'application. Nous soutenons que l'intégration universelle offrira des avantages qui sont semblables à ceux que la **SOA** et l'intégration centrée processus prévu pour simplifier le développement des processus de l'entreprise.

### 3.3.2. Approches de Composition de l'interface utilisateur UI :

Dans **[54]**, nous avons discuté du problème de l'intégration sur la couche de présentation et on a conclu qu'il n'y a pas de réelle approche de composition d'interface utilisateur facilement disponible, Les technologies des composants de bureau de l'interface utilisateur tels que **.NET CAB [55]** ou **Eclipse RCP [56]** sont fortement tributaires de la technologie et pas prêt pour le Web. Les **Plug-ins** tels que les **applets Java, Microsoft Silverlight, ou Macromedia Flash** peuvent facilement être intégrés dans des pages HTML, les communications entre les différentes technologies restent cependant encombrantes (par exemple, via JavaScript personnalisé). Les **Portlets Java [57]** ou **WSRP [58]** représentent une solution mature et conviviale pour le développement des applications de **portails**; les **portlets** sont toutefois généralement exécutés de manière isolée et la communication ou la synchronisation avec d'autres portlets ou services Web reste difficile. Les Portails ne fournissent pas de support pour la logique d'orchestration des services.

### 3.3.3. Outils d'ingénierie Web avec assistance par Ordinateur

Afin de faciliter le développement d'applications web complexes, la communauté des ingénieurs web a jusqu'ici généralement axée les approches sur le modèle de conception. Parmi les avancées les plus notables et les outils d'ingénierie de modèles Web, nous trouvons par exemple, **WebRatio [59]** et **VisualWade [60]**. Le premier est basé sur un langage spécifique à la bande de modélisation visuelle (**WebML**), le second sur une notation de modélisation orientée objet (**OO-H**). Similaire, mais moins avancé, les outils de modélisation sont également disponibles pour les langages de modélisation web/méthodes comme **Hera**, **OOHDM**, et **UWE**. Tous ces outils permettent aux développeurs web experts des abstractions de modélisation et la génération de code automatique des capacités, qui sont cependant bien au-delà des capacités de notre public cible, à savoir des internautes avancés et non des programmeurs web.

### 3.3.4. Portails et Portlets

Toujours dans le contexte des applications Web, les portails et les portlets représentent une approche différente du problème d'intégration de l'interface utilisateur sur le Web. Leur approche distingue explicitement entre les composants de l'interface utilisateur (les portlets) et les applications composites (portails) et elle est sans doute l'approche la plus avancée pour la composition de l'interface utilisateur au jour d'aujourd'hui (Nous utilisons le terme «portlets» tiré à partir de la spécification portlet **JSR-168 [57]**, mais nos considérations sont également valables pour les composants **ASP.NET** WebPart). Les *portlets* sont à part entière, des composants enfichables d'applications Web qui génèrent des fragments de balisage de document (par exemple, **(X) HTML**) et de faciliter l'agrégation de contenu à un serveur de portails. Les portlets sont conceptuellement très similaire aux **servlets**. La principale différence entre les deux réside dans le fait qu'un servlet génère une page web complète alors que les portlets génèrent simplement un morceau de page (communément appelé fragment) qui est conçu pour être inclus dans une page de portail. Par conséquent, si un servlet peut être atteint par le biais d'une URL spécifique, un portlet ne peut être atteint via l'URL de l'ensemble de la page de portail. Un portlet n'a pas de communication directe avec le navigateur, mais ces communications sont gérées par le portail et le conteneur de portlet qui permettent les flux **Request/Response** et la communication entre portlets. Un serveur de portail permet généralement aux utilisateurs de personnaliser les pages composites (par exemple, de réorganiser ou de montrer/cacher les portlets) et fournir l'authentification unique (single sign-on) et la personnalisation basée sur les rôles.

Aujourd'hui, il existe plusieurs normes pour les portlets **JSR-168**, étant la spécification originale. **JSR-286** introduit la communication entre portlets via un conteneur de portlets qui gère une infrastructure **publier/s'abonner** qui peut être utilisée par les portlets. Enfin, **WSRP [58]** a également ajouté un support pour accéder aux portlets distants comme les services Web sur le Web. Le modèle de **portlet** est puissant pour ce qui concerne la partie intégration de la présentation, mais les portails ne sont pas naturellement faits pour soutenir les interactions avec les services Web génériques ou la description de la logique d'orchestration.

## 3.4. Les mashups Web

Les mashups Web en quelque sorte comblent les lacunes ci-dessus. Ce sont des applications Web qui sont développées en combinant le contenu, la présentation et les fonctionnalités des applications Web à partir de sources disparates **[61]**. Le terme mashup

implique généralement une intégration facile et rapide basée sur les APIs ouvertes et les sources de données, ce qui donne des applications à valeur ajoutée pour les différents composants de l'application et donc souvent utiliser des composants d'une manière qui diffère de la vraie raison qui a conduit à la production originale de sources brutes.

Les mashups sont étroitement liés avec le Web. Le Web est le milieu naturel pour la publication de contenus et de services aujourd'hui et il est donc le milieu naturel pour l'accès et leur réutilisation. Le contenu et les services sont publiés dans une variété de formes différentes et en utilisant une multiplicité de différentes technologies, nous pouvons classer les moyens des sources de contenu et des services sur le Web en trois groupes principaux:

- **Les données des services :** Comme **RSS** (Really Simple Syndication) ou **Atom**, **JSON** (Object Notation Java-Script) ou des ressources **XML** ou des fichiers texte simples. Un exemple typique est les journaux et les magazines qui publient leurs nouvelles via **RSS** ou **Atom** qui permettent aux utilisateurs de passer facilement les articles respectifs. Ces technologies simples sont utilisées pour publier des données sur le Web qui sont destinées à la consommation par des machines, non pas par les humains. En fait, elles se concentrent sur la distribution efficace des contenus, plutôt que sur la présentation efficace de ces contenus à des utilisateurs humains. L'acquisition des données via l'une de ces technologies est généralement très simple, il faut la plupart du temps l'accès à une ressource en ligne et le traitement de la réponse.
- **les services Web ou les APIs accessibles au public sur le Web :** Tels que les web services avec le protocole **SOAP** (Simple Object Access Protocol) ou **RESTful** (Representational State Transfer) ou, à un moindre degré, des **classes Java** (accessible via le protocole **IIOP**). Ces technologies sont utilisées pour publier une logique d'application sur le Web. Leur but n'est donc pas seulement de fournir un accès à des contenus ou des données, mais aussi à la logique de calcul (par exemple, le traitement d'une commande pour un magasin de livres). En règle générale, l'interaction avec les services web ou APIs est soi-disant régi par des protocoles d'interaction, qui indiquent quelles opérations peuvent être invoquées, dans quel ordre, par quels partenaires, etc. Ne pas suivre les règles établies par le protocole peut entraver le bon fonctionnement du service ou de l'API.
- **Les éléments de l'interface utilisateur :** Tels que des clips HTML ou des APIs JavaScript avec sa propre interface utilisateur (par exemple, Google Maps), mais aussi des bannières ou des annonces. Le contenu peut également être représenté par des données déjà formatées et graphiquement rendu (généralement au format HTML). Dans de nombreux cas,

l'accès à ce genre de contenu signifie les extraire à partir d'une page Web, car il n'y a pas de service de données équivalent disponible qui peut être utilisé à la même source de données. En général, cela se produit sans que le prestataire du contenu sache qu'il y a quelqu'un qui soutire de données à partir de ses pages Web. Dans d'autres cas, par exemple, Google Maps, le fournisseur du contenu publie explicitement ses données uniquement au niveau de l'interface utilisateur.

L'aspect très novateur de mashups web, c'est qu'ils intègrent également des sources à la couche de l'interface utilisateur, non seulement au niveau des données et des couches logiques d'application. L'intégration dans les données et les couches logiques d'application a été étudiée dans le passé, tandis que l'intégration aux trois couches est encore un objectif qui a mis les architectes et programmeurs devant d'importants problèmes conceptuels et techniques.

Le développement des mashups est toujours un processus ad hoc de longue haleine, qui nécessite des compétences de programmation avancées (par exemple, l'emballage ou la couverture des services Web, l'extraction de contenu à partir de sites Web, l'interprétation du code tiers JavaScript, etc.) Il existe une variété d'outils mashups disponibles en ligne, mais, comme nous le verrons, que quelques-uns d'entre eux répondent au problème de l'intégration à l'ensemble de ses couches. Dans cette section, nous donnons un aperçu de l'état de l'art dans le monde mashup, allant du développement manuel au développement semi-assisté et entièrement assisté pour les approches de développement.

### 3.4.1. Développement manuel

Le développement d'applications qui regroupe des données, la logique d'application et les UIs en provenance de diverses sources nécessite une connaissance approfondie sur les technologies telles que: **(X)HTML**, le **HTML dynamique**, **AJAX** (Asynchronous JavaScript and XML), **RSS**, **Atom**, les **spécifications XML** comme **DTD**, **XSD**, **XSLT**, les protocoles comme **SOAP** ou **HTTP** pour les web services **SOAP** et **RESTful**, les langages de programmation comme **JavaScript**, **PHP**, **Ruby**, **Java**, **C**# et ainsi de suite; les bases de données relationnelles ou orientées objet, etc. En outre, il pourrait être nécessaire de maîtriser les protocoles d'entreprises des services utilisés et d'avoir des connaissances sur la façon de composer des services dans des orchestrations de services. Cette longue liste non exhaustive des technologies montre comment mixer ou mélanger jusqu'à même une application simple, comme celui de notre scénario de référence qui est une tâche difficile et de longue haleine qui ne peut être complété que par des programmeurs qualifiés.

### 3.4.2. Développement Semi-assisté

Afin d'accélérer et de simplifier le développement, en particulier des composants destinés à être mélangés, des outils Web utiles et des frameworks ont été récemment mis en place. En règle générale, ils abordent le problème de l'extraction de données à partir de sites Web et l'approvisionnement de ces données sous forme de services de données ou des éléments réutilisables de l'interface utilisateur. Dans ce qui suit, nous analysons deux outils de représentation, c'est-à-dire **Dapper [121]** et **Openkapow [122]**, qui sont très conviviaux.

**Dapper** est un outil en ligne gratuit pour la production d'emballages de données qui extraient des données à partir de pages Web bien structurées. **Dapper** est basé sur un point et une technique de clic dans la mesure d'aider l'utilisateur dans le choix du contenu à extraire et d'en déduire des règles d'extraction appropriées (par exemple, les expressions régulières). Plus précisément, l'extraction de données s'appuie sur la structure de la mise en forme HTML pour comprendre quels sont les éléments à extraire (par exemple, les premières cellules de tous les enregistrements d'une table). Une fois bien identifiée, les champs de données extraites peuvent être nommés et structurés, puis publié, par exemple, sous forme de flux XML ou des services de données. Les services publiés sont facilement accessibles via une URL unique et sont traités chaque fois que l'URL correspondante est accessible.

**Openkapow** est une plate-forme de service semblable ouverte basée sur le concept de robot d'extraction, qui est, créé par l'utilisateur d'emballages ou de couverture. Les utilisateurs de Openkapow peuvent construire leurs propres robots, d'exposer leurs résultats via des services web et de les exécuter à partir d'openkapow.com gratuitement. Les robots sont en mesure d'accéder à des sites Web et l'extraction et la réutilisation des données, la fonctionnalité et même des morceaux d'interfaces utilisateur. Les robots sont construits grâce à un environnement de développement visuel appelé **RoboMaker**. RoboMaker permet à l'utilisateur de naviguer à l'intérieur du site cible et à définir une série d'étapes simples, chacune représentant un événement dans la page, jusqu'à ce que les données cibles soit atteintes. Les résultats de l'extraction peuvent être exposés de deux façons: comme un service **RESTful** ou comme un flux **RSS** en fonction du contenu extrait et sur l'utilisation attendue de celui-ci. Après leur publication sur les serveurs **Openkapow**, les robots sont accessibles via une URL publique, qui identifie le robot spécifique pour fonctionner. Les services ainsi exposés peuvent également avoir besoin des valeurs d'entrée (par exemple, user-id et mot de passe) qui peuvent être utilisés pour paramétrer les services. Les entrées peuvent facilement

être transmises en les ajoutant à l'URL du service en tant que paires nom-valeur, suivant le modèle d'URL standard.

### 3.4.3. Le développement entièrement assisté

Les analyses précédentes montrent que le développement des mashups est typiquement un travail à forte intensité cognitive, impliquant une variété de technologies et de composants. En plus de simplifier la création d'outils d'extraction de données pour les pages Web, qui abordent le problème du développement des composants pour les mashups, il est également important de faciliter la composition réelle des composants dans les applications, ce qui est aussi difficile et de longue haleine en tant que composants de développement, s'il n'est pas correctement pris en charge. Les outils de mashups ou les plates-formes de mashups répondent exactement à ce problème, chacun d'entre eux en se concentrant sur les différents aspects de composition et suivant les différentes approches mashups. Dans ce qui suit, nous analysons cinq de ces outils, qui nous semblent les plus représentatifs de ce type de développement mashup assisté: **Yahoo! Pipes**, **JackBe Presto [119]**, **Microsoft Popfly [62]**, **Intel Mash Maker [63]** et **Taverna [64]**. Il y a aussi d'autres outils comme **Google App Engine [65]** ou **IBM Lotus Mashups [84]** et ainsi de suite, mais leur discussion dépasse le cadre de ce chapitre.

#### 3.4.3.1. Yahoo! Pipes

Il fournit un éditeur simple et visuellement intuitif qui permet de concevoir des compositions centrées sur les données. Il prend des données en entrée et fournit en sortie des données aussi, les formats les plus importants supportés sont les flux **RSS/Atom**, **XML** et **JSON**. Un tube est un pipeline de traitement de données dans lequel des données d'entrée (provenant de différentes sources de données) sont traitées, manipulées et utilisées comme entrée pour d'autres étapes de traitement, jusqu'à ce que la transformation cible soit terminée. Ce processus de style pipeline est mis en œuvre par un nombre arbitraire d'opérateurs intermédiaires, qui manipulent les éléments de données à l'intérieur du flux de données ou de fournir des fonctionnalités telles que les boucles, les expressions régulières ou des fonctionnalités plus avancées telles que l'extraction automatique de localisation ou la connexion à des services externes. L'ensemble des opérateurs sont prédéfinis et fixes, de nouvelles fonctionnalités peuvent être incluses sous forme de services Web. En outre, les tuyaux ou tubes stockés peuvent être réutilisés en tant que sources d'un autre tuyau.

L'environnement de développement **Yahoo! Pipes** se caractérise par un modèle de développement simple et intuitif qui est cependant destiné aux utilisateurs avancés et aux programmeurs web. En fait, le niveau d'abstraction de ses activités (par exemple, la composante expression régulière) et les caractéristiques de la logique du flux de données est difficilement compréhensible pour les non-programmeurs. Le tuyau de sortie n'est pas destiné à la consommation humaine (**RSS**, **Atom**, **JSON**, etc), mais plutôt pour l'intégration dans d'autres applications. Cette limite et la variété des sources d'entrée qui peuvent être utilisés et l'accessibilité de ses sorties. En fait, l'absence de tout soutien pour les interfaces empêche l'utilisation directe des tuyaux de sortie par les internautes ordinaires. Cependant, le tube est un outil de développement très populaire des mashups de données, très probablement en raison de la mise en place efficace et intuitive des composants et d'un mécanisme de connexion.

L'outil de développement ne nécessite aucune installation ou de plug-ins, il fonctionne dans n'importe quel navigateur Web AJAX. L'environnement de développement est livré avec un très efficace outil de débogage intégré qui aide le développeur lors de la phase de conception. Les tuyaux sont stockés en ligne et accessible via une URL propre. Lors de l'invocation d'un tuyau, un processus d'exécution est démarré sur le serveur pour soulager le client de la surcharge d'exécution. Cette caractéristique pourrait représenter un problème dans une perspective d'évolutivité, si un grand nombre d'accès simultanés à un tuyau sont faits, les performances et la stabilité pourrait en souffrir.

#### 3.4.3.2. JackBe Presto

Est une plateforme de mashup robuste et complete qui offre des solutions au niveau des entreprises. **Presto** donne la possibilité de produire facilement (conception, test et déploiement) des mashups de fusion de données provenant de sources disparates. En particulier, il peut également être connecté à des sources de données très courantes dans le monde de l'entreprise (comme les feuilles de calcul Excel, le logiciel de données Oracle, etc), que la plupart des solutions concurrentes de mashups ne peuvent y accéder. La composition simple de mashup peut être faite par des non-utilisateurs des TI, grâce à l'outil de fils **Presto**. La composition plus avancé ne peut être obtenue que par des développeurs professionnels qui les appliquent dans la langue **EMML** avec le soutien du plug-in du Mashup Studio Presto pour Eclipse. Ce langage est le principal acteur de l'**OMA** (Open Mashup Alliance), qui vise à

définir un langage ouvert permettant l'interopérabilité et la portabilité des mashup d'entreprise.

L'environnement de développement est constitué de plusieurs outils indépendants. Les fils est un éditeur visuel basé sur une approche de composition de données du pipeline simple et intuitive. Il permet de fusionner des données disparates provenant de sources internes et externes qui produisent un résultat final qui peut être représenté graphiquement par une **mashlet**. Les Mashlets peuvent être branchés à un tableau de bord comme l'interface utilisateur ou un portail, ou ils peuvent être intégrés dans une page web classique. Le développement du Mashlet est assisté par l'outil **Presto Mashlet**, alors que le Mashup Studio est un plug-in Eclipse fournissant les programmeurs Java avec un contrôle complet sur le processus de développement mashup. Les connecteurs permettent de raccorder Presto à divers logiciels, telles que **Microsoft Excel**, les **portails web**, toute **la technologie Oracle**. Les services Presto sont accessibles via des APIs, disponibles pour les principaux langages de programmation (**Java**, **JavaScript**, **C#**, **Python**, etc).

Le serveur d'exécution prévoit des mécanismes sécurisés pour virtualiser et normaliser (mettre la production de services dans des formats standards: JSON ou XML) tout type de service ou de données (SOAP, REST, RSS, DB, Excel) et les exposer d'une manière régie et sûre. **Presto** n'est pas un service hébergé, comme Yahoo! Pipes, il doit être installé et configuré sur chaque entreprise individuellement.


**Figure 3.1 :** Page d'accueil de JackBe Presto, montrant une vue d'ensemble des composants disponibles **[66].**

Par rapport à **Yahoo! Pipes**, la plate-forme JackBe a un riche ensemble d'outils pour développer des applications Web composites et des composants Mashup. La plate-forme identifie les « Mashables », les « Mashlets », et les Mashups. Les Mashables sont essentiellement des sources de données et des services élémentaires dans la couche de ressource, tandis que les composants Mashlets sont positionnés dans la couche widget. La plate-forme permet un stockage des composants disponibles dans les différentes couches, comme représenté sur la **figure 3.1**.

### 3.4.3.3. Microsoft Popfly

Il a Gagné un large consensus dans la communauté mashup et a atteint un bon niveau de popularité et d'utilisation. Bien que le projet **Popfly** ait été abandonné, nous analysons cet outil de mashup parce que nous considérons qu'il est un exemple intéressant pour la composition de l''interface utilisateur avec des particularités qui ne peuvent être trouvés dans d'autres outils.

**Popfly** fournit un environnement de développement visuel pour la réalisation de mashups basés sur le concept de composants ou blocs comme on les appelle dans **Popfly**. Une

composition est créée en faisant **glisser/déposer** les blocs d'intérêt sur une toile de conception graphique et en les reliant pour créer la logique de l'application souhaitée. Un bloc peut prendre le rôle de connecteur à des services externes ou il peut représenter certaines fonctionnalités internes (mis en œuvre par le biais d'une fonction JavaScript). Chaque bloc fournit des ports d'entrée et de sortie qui permettent son raccordement à d'autres blocs. Les blocs peuvent également être utilisés pour fournir une interface utilisateur qui permet d'afficher le résultat d'une transformation. Placer des blocs de visualisation multiples dans une même page permet de définir la mise en page globale de la page. L'aménagement intérieur des blocs peut être personnalisé en insérant du **HTML ad hoc**, **CSS** ou **du code JavaScript**. **Popfly** a une vaste collection de blocs disponibles, offrant des fonctionnalités telles que les **lecteurs RSS**, les **connecteurs**, les **composants de services** basés sur la carte **Virtual Earth**, etc. Les nouveaux blocs et compositions peuvent être définies (en JavaScript), enregistrées, partagées et gérées dans une section dédiée de la plate-forme.

Lors de l'exécution, le flux de communication est piloté par les événements, qui sont, l'activation d'un certain composant qui dépend de l'élévation d'un événement par un autre composant. Il n'y a pas de support pour la gestion des transactions et des exceptions, mais **Popfly** fournit une section consacrée à l'examen et l'aperçu de la composition. Les compositions Prêtes sont stockées sur le serveur Popfly, mais l'exécution se fait sur le client comme la plupart des blocs intégrés sont basés sur la plate-forme Silverlight. L'exécution côté client du mashups allège le serveur de charges lourdes et les limites d'évolutivité et de performance.

#### 3.4.3.4. Intel Mash Maker

Il fournit une approche mashup complètement différente, un environnement pour l'intégration des données provenant de sources annotées pages Web basé sur un puissant navigateur plug-in dédié pour le navigateur Firefox. Plutôt que de prendre d'entrée des sources de données structurées telles que des flux **RSS/Atom** ou **des services web**, Mash Maker permet aux utilisateurs de réutiliser des pages Web entières si elles sont convenablement annoté, pour extraire des données à partir des pages. Autrement dit, les «composants» qui peuvent être utilisés dans Mash Maker sont des pages Web standards. Si une page a été annotée par le passé, il est possible d'extraire les données annotées de la page et de les partager avec d'autres composants dans le navigateur. Si la page n'a pas été annotée, il est

possible de réutiliser la page telle quelle, sans toutefois soutenir toute communication inter-page.

Afin d'annoter une page, **Mash Maker** permet aux développeurs et aux utilisateurs d'annoter la structure des pages Web pendant la navigation et à utiliser les annotations pour supprimer le contenu des pages annotées. Les utilisateurs avancés peuvent tirer parti de l'éditeur de structure intégré de saisie des expressions **XPath** à l'aide de **Firebug DOM Inspector** (autre plug-in pour le navigateur Firefox). Les annotations sont liées à des pages cibles et stockées sur le serveur **Mash Maker** afin de les partager avec d'autres utilisateurs.

La composition des mashups avec **Mash Maker** se fait par un paradigme copier/coller, qui repose sur deux modes de fusion de contenus, **la fusion de page entière** où le contenu d'une page est inséré dans l'en-tête d'une autre page et **la fusion en point avisé** où le contenu de deux pages sont combinés au niveau de la ligne, basée sur les annotations supplémentaires des utilisateurs. Les deux techniques peuvent être utilisées pour fusionner aussi plus de deux pages. L'échange de données entre les composants est réalisé au moyen d'une approche de type **«blackboard »**, où les données de composants intégrés dans une application sont immédiatement disponibles pour tous les autres composants. Non seulement le développement, mais aussi l'exécution des mashups est entièrement réalisée à l'aide du plug-in du côté du client, côté serveur il n'y a que les annotations pour l'extraction des données et les définitions stockées de mashups.

#### 3.4.3.5. Taverna

Taverna est un outil logiciel open source pour la conception et l'exécution des workflows (flux de travail) scientifiques. Il s'agit d'un programme autonome côté client, généralement exécuté sur la machine de l'utilisateur, qui utilise des données locales et à distance (internet) ainsi que la fonctionnalité de traitement des données. Comme toutes les plates-formes de Mashup d'entreprise, Taverna est un éditeur visuel qui permet de générer des flux de travail. Les flux de travail peuvent utiliser plusieurs sources de données en entrée et effectuer diverses opérations sur ces données pour générer une sortie, ce qui est très similaire à ce que les mashups font. La base d'utilisateurs Taverna comprend plus de 350 organisations. Un exemple d'une session d'édition à l'intérieur du client Taverna est illustré à la **figure 3.2** suivante.

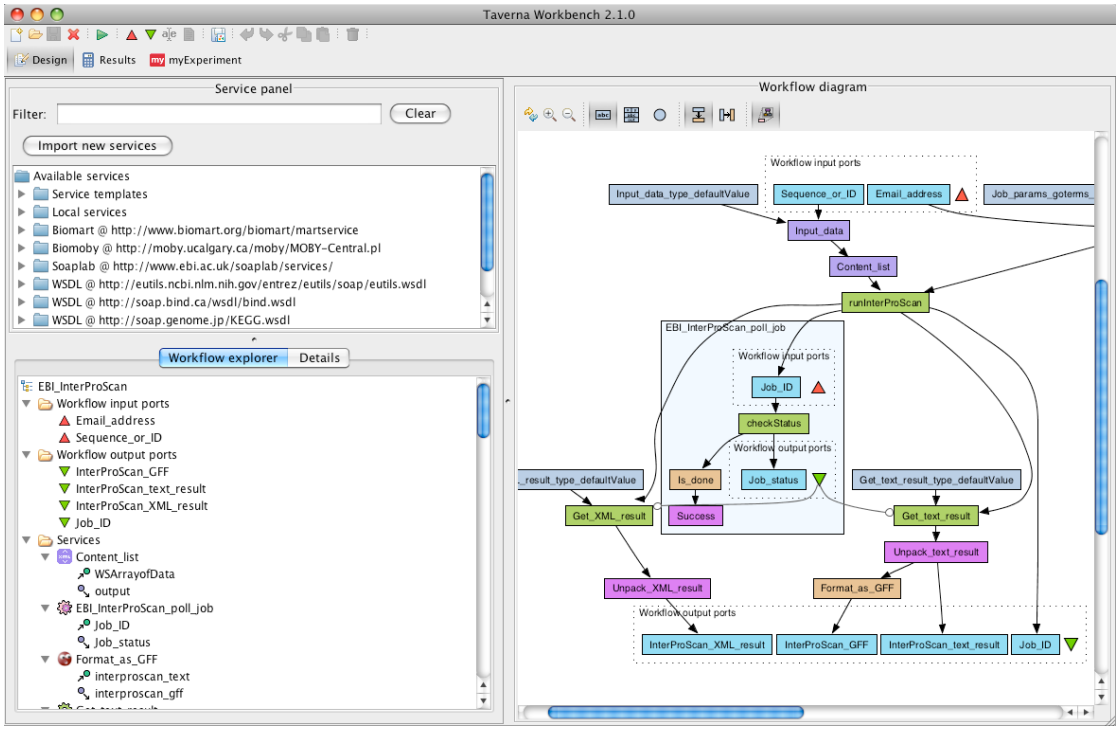



**Figure 3.2 :** Capture d'écran d'une session d'édition de workflow dans le Workbench Taverna **[67]**.

Taverna peut utiliser différents types de sources de données dans un flux de travail, dont certaines ont des formats et des sémantiques spécifiques à un domaine. Les utilisateurs de Taverna proviennent de domaines de recherche où de grandes quantités de données expérimentales et l'analyse des données structurelles jouent un rôle important, tels que la bioinformatique, la chimie, l'astronomie, l'exploration de données, etc. Par conséquent, les sources de données qui sont préconfigurés dans cet environnement proviennent de ces domaines. Un exemple de l'ajout d'une source de données spécifique à un flux de production scientifique au sein de Taverna.

Taverna prend en charge un large éventail de composants de traitement qui sont communs dans les milieux scientifiques (dont certains sont dans un domaine spécifique). Un exemple est l'intégration de l'environnement **"R"** calcul statistique avec Taverna, un outil qui est utilisé dans un large éventail de domaines scientifiques.

**Taverna** s'intègre à l'environnement de recherche virtuel **myExperiment [68]** pour trouver, utiliser et partager des flux de travail scientifiques. Le site web myExperiment est comparable à celui des dépôts de composants et de Mashups et des catalogues disponibles pour les plates-formes Mashup. En outre, Taverna dispose d'une fonctionnalité de traitement pour décharger des étapes du flux de travail vers d'autres sites et d'autres machines. **Taverna**

soutient l'installation de plugins et son code source est disponible sous une licence de logiciel libre, ce qui signifie qu'il est entièrement extensible ou renouvelable.

## 3.5. La composition universelle (les principes directeurs)

Comme souligné plus haut, bien que les approches existantes des mashups aient donné des résultats prometteurs, les techniques qui répondent à une intégration simple et universelle de composants Web au niveau des trois couches de la pile d'application sont toujours inexistantes. Nous pensons que ces techniques sont nécessaires pour la transition à la programmation Web 2.0 à partir d'élites types d'environnements informatiques à des environnements où les utilisateurs tirent parti des abstractions simples pour créer des applications web composites sur les composants Web potentiellement riches développés et maintenus par des programmeurs professionnels.

Nous visons l'intégration universelle et cela a des différences fondamentales en ce qui concerne la composition traditionnelle. En particulier, le fait que nous cherchons à intégrer également l'interface utilisateur implique que:

✓ La synchronisation, et pas (seulement) une orchestration BPEL, devrait être adoptée comme paradigme d'interaction.

✓ Les composants doivent être en mesure de réagir à la fois l'entrée utilisateur humain et l'interaction programmatique.

✓ Nous devons être en mesure de concevoir l'interface utilisateur de l'application composite, pas simplement le comportement et l'interaction entre les composants.

Cela montre la nécessité d'un modèle fondé sur l'état, les événements et la synchronisation plus sur les appels de méthode et l'orchestration. Nous reconnaissons en particulier que les événements, les opérations, la notion d'état et les propriétés de configuration sont tout ce qu'il faut pour modéliser un composant universel.

Sur le plan des données, nous nous rendons compte que l'intégration de données sur le Web peut également exiger différents modèles: par exemple, **les flux RSS** sont naturellement gérés par l'intermédiaire d'un tuyau de données orienté débit ou flux de données/modèle de streaming ( **Yahoo! Pipes**) plutôt qu'une approche basée sur les variables comme cela se fait dans la composition des services conventionnels.

Une autre dimension de l'universalité réside dans les protocoles d'interaction. Comme il peut y avoir une variété de composants et des implémentations de composants, nous devons

être en mesure de faire face avec plusieurs protocoles de communication en même temps. Par exemple, on citera les Protocoles les plus utilisé sur le Web qui sont **REST/HTTP**, **SOAP**, **RSS**, **Atom** et **JSON**.

Ces exigences sont souvent en contradiction avec l'objectif clé de la conception que nous avons, c'est-à-dire la simplicité. Nous voulons permettre aux internautes de créer des applications de pointe (un vieux rêve des langages de composition de services qui est toujours un peu un objectif de grande envergure). Cela signifie que le paradigme de la composition universelle doit être fondamentalement plus simple que les langages de programmation et les langages de composition actuels. A titre d'exemple, nous ciblons la complexité de la création de pages Web avec un éditeur de page web, ou la complexité de la construction d'un pipe ou d'un tube avec Yahoo Pipes (quelque chose qui peut être appris en quelques heures plutôt qu'en semaines).

## 3.6. La plate-forme mashArt

Pour atteindre la simplicité dans mashArt, il faut trois décisions de conception : d'abord, **mashArt** vise à cacher la complexité du protocole spécifique ou d'un modèle de données pris en charge par chaque composant. Autrement dit, l'objectif est que du point de vue du compositeur toutes ces spécificités sont cachés à l'exception des aspects qui ont une incidence sur la composition (par exemple, si un composant est un flux, donc nous sommes conscients qu'il fonctionne, conceptuellement, en forçant le contenu périodiquement ou à l'occurrence de certains événements).

Comme deuxième décision, il faut garder le modèle de composition léger, par exemple, il n'y a pas d'exception complexe ou des mécanismes de transaction, aucune activité BPEL de type structuré complexe. Cela permet encore un modèle qui rend plus simple à définir des applications très complexes et assez sophistiquées. Des besoins complexes peuvent encore être mis en œuvre, mais cela doit être fait d'une manière "ad hoc" (par exemple, par des combinaisons appropriées d'écouteurs d'événements et de la logique des composants), mais il n'existe pas de concepts spécialisés pour cela. De tels concepts peuvent être ajoutés au fil du temps, si on se rend compte que la majorité des applications en ont besoin.

La troisième décision est de se concentrer sur la simplicité du point de vue de l'utilisateur des composants, c'est-à-dire le concepteur des applications composites. Dans les applications complexes, la complexité doit résider quelque part, et nous croyons que, autant

que possible, elle doit être à l'intérieur des composants. Les composants fournissent habituellement des fonctionnalités de base et sont réutilisés encore et encore (c'est l'un des objectifs principaux des composants). Par conséquent, il est logique d'avoir des programmeurs professionnels pour développer et entretenir les composants. Nous croyons que cela est nécessaire pour le paradigme de mashup pour vraiment avancer. Par exemple, des questions telles que les protocoles d'interaction (par exemple, **SOAP** vs **REST** ou autres) ou d'initialisation des interactions avec les composants (par exemple, les échanges de messages pour l'authentification du client) doit être intégrée dans les composants.

## 3.7. Conclusion

Dans ce chapitre, nous avons inspecté une nouvelle approche pour l'interface utilisateur et la composition de services sur le Web, à savoir, la composition universelle. Cette approche de composition est le fondement du projet **mashArt** qui vise à permettre, même aux programmeurs non-professionnels (ou les utilisateurs du Web) de réaliser une interface utilisateur complexe, l'application et des tâches d'intégration de données en ligne et dans un mode hébergé (intégration en tant que service). L'accessibilité et la facilité d'utilisation des outils de la composition sont facilitées par la logique de composition simple et la mise en œuvre par l'éditeur graphique intuitif et l'environnement d'exécution en mode hébergé.

La plate-forme est livrée avec un registre en ligne pour les composants et les compositions et fournira des outils pour le suivi et l'analyse des compositions hébergées.

Tout au long de ce chapitre, nous avons constamment gardé un œil sur le lien entre la composition universelle et l'informatique de recherche. Les outils mis en œuvre à l'aide des outils et des langages **mashArt** montre qu'il est effectivement possible de développer une application à base de composants qui fournit des réponses au problème de la recherche de conférence, à condition que les éléments de base nécessaires soient facilement disponibles. La logique d'intégration de l'application est réalisée au moyen d'un paradigme de composition **glisser-déposer** impératif qui permet aux utilisateurs de la plate-forme **mashArt** de composer des applications en fonction de leurs propres connaissances sur les composants qui sont nécessaires et sur la façon de les coller ensemble. Il existe de nombreuses solutions alternatives à la mise en œuvre de la même application, contrairement à **[53]**, où un plan de requête optimal est automatiquement identifié, dans mashArt c'est au développeur de choisir quelle solution répond le mieux à ses besoins individuels.

En termes de production de la composition, il est intéressant de noter que tandis que dans le scénario de recherche traditionnel, la sortie est un ensemble de résultats en tuples, la sortie en **mashArt** est plutôt représentée par l'ensemble de l'application, c'est à dire, les différents composants et leur interconnexion.

# Chapitre 4 : Orchestration et composition des IUs dans les mashups

## 4.1. Introduction

Dans ce chapitre, nous allons faire une caractérisation des modèles et canevas d'orchestration de services. Puis, nous détaillons certaines approches proposées par des industriels comme le standard WS-BPEL et d'autres issues du monde académique. Ensuite nous décrirons l'architecture orientée service sur la couche présentation et l'intégration des services, on terminera ce chapitre par des perspectives d'avenir.

## 4.2. Caractérisation des modèles d'orchestration de services

Afin de caractériser les modèles d'orchestration de services, dans **[69]** les auteurs ont proposé six dimensions, à savoir : la technologie des éléments composés, le formalisme de description de l'orchestration, le modèle de données et d'accès aux données, la sélection des services participants, le modèle de gestion de transactions utilisé et finalement le modèle de gestion des exceptions.

Nous allons reprendre ces dimensions, En analysant entre autres, le support des aspects non-fonctionnels de la part des approches d'orchestration et finalement le critère d'extensibilité des modèles d'orchestration.

### 4.2.1. Technologie des éléments composés

Cette dimension détermine les hypothèses faites par l'approche d'orchestration par rapport au type des éléments qui seront composés. Plusieurs approches utilisent les services Web comme unités de composition. Par conséquent, la composition suppose que les éléments sont décrits avec le langage WSDL, qu'ils communiquent en utilisant le protocole SOAP, et que les données échangées sont représentées par des documents XML. Parmi ces approches nous trouvons **WS-BPEL [70]** et **SELF-SERV [71]**. Pour sa part, **eFlow [72]** utilise les e-Services (précurseur des services Web) comme unité de composition.

D'autres approches d'orchestration supportent un ensemble prédéfini de technologies capables de composer. Par exemple, le langage **BPELJ [73]** (extension du WS-BPEL)

supporte la technologie de services web et du code Java (Java Snippets). Ces approches sont plus générales mais ont l'inconvénient d'être techniquement plus compliquées à cause de l'hétérogénéité des composants utilisés.

Une troisième alternative, consiste à fournir un support prédéfini pour un ensemble de technologies et offrir des mécanismes pour supporter l'ajout d'autres. Par exemple, **JOpera [74]** supporte la composition de services Web, de code Java (Java Snippets) et d'un langage de script. L'avantage principal est la généralité et la capacité d'extension, mais l'hétérogénéité des composants et la complexité associée aux mécanismes d'extension rend problématique ce type d'approche.

### 4.2.2. Formalisme de description de l'orchestration

Cette dimension analyse le langage utilisé pour décrire l'ordre des invocations des services ainsi que les conditions nécessaires pour réaliser ou non l'invocation. Les approches d'orchestration de services généralement reprennent des formalismes issus de la technologie des workflow.

Un type de formalisme bien connu de la technologie de workflow est le diagramme d'activité. C'est un graphe dirigé où les nœuds correspondent aux activités (tâches) et les arcs représentent des liens exprimant des contraintes de flux de contrôle et/ou de flux de données entre les tâches. Les activités peuvent être spécialisées pour réaliser des tâches spécifiques comme par exemple l'invocation d'une opération d'un service Web. En plus, dans certains formalismes, il est possible d'exprimer des conditions sur les liens, pour indiquer par exemple, si un flot de contrôle est suivi ou non. Le langage d'orchestration **WSFL [75]** et le canevas JOpera **[74]** utilisent ce type de formalisme, le langage **APEL [76]** peut être classé dans cette catégorie.

Bien que le diagramme d'activités soit le type de formalisme le plus utilisé par les approches d'orchestration, d'autres formalismes proposent des propriétés intéressantes et ils sont surtout utilisés par des approches académiques. Ainsi, les diagrammes d'états permettent de décrire un ensemble d'états et les transitions entre eux. Les transitions sont associées à des règles ECA (Evénement, Condition, Action), de cette façon lorsqu'un événement arrive, la condition est évaluée, ensuite l'action est réalisée et la composition change son état. Le canevas SELF-SERV **[71]** et le système **Mentor [77]** utilisent des formalismes de ce type.

Il existe une catégorie de formalismes plus formelle, permettant l'analyse des propriétés structurelles des spécifications afin de détecter des problèmes comme par exemple les **deadlocks** ou les **livelocks**. Dans ce groupe nous trouvons **les réseaux de Petri [78]** et le **π-calculus [79]**. Cependant, ces formalismes sont de bas niveau d'abstraction et peu intuitifs, plus adaptés à l'usage des mathématiciens. Néanmoins, certaines approches d'orchestration les utilisent comme base de leurs propres formalismes, les réseaux de **Petri** ont inspiré **Orchestration Nets [80]**, **YAWL [81]** et **[82]**, tandis que le **π-calculus** a inspiré le langage **XLANG [83]**.

Finalement, les hiérarchies d'activités permettant une vision structurée de la composition. Une spécification dans ce type de formalisme, dispose d'une activité de premier niveau qui se décline en un arbre des sous-activités. Chaque sous-activité peut être une feuille (activité simple) ou un autre arbre (activité composée). Les activités composées servent à définir des contraintes de contrôle du flux appliquées à leurs sous-activités, pour indiquer par exemple, une exécution en séquence ou en parallèle. Les activités simples sont spécialisées pour la réalisation d'une tâche spécifique, par exemple l'invocation d'un service ou l'assignation d'une variable. WS-BPEL **[70]** utilise un formalisme basé sur une hiérarchie d'activités.

Ce type de formalisme est moins intuitif et plus restrictif que le diagramme d'activités, puisqu'il a besoin d'utiliser des activités afin d'exprimer le flux de contrôle de la composition.

### 4.2.3. Modèle de données et d'accès aux données

Ce critère détermine la façon de manipuler les données pour communiquer avec les services, ainsi que les données associées à l'état de l'orchestration. Le modèle de typage de données utilisé par les approches d'orchestration sera aussi analysé.

L'approche d'orchestration utilise deux façons de manipuler les données afin d'interagir avec les services. La première approche est de considérer les données comme des boîtes noires (black boxes). Dans ce cas, l'orchestration n'a pas à connaître ni la structure ni le contenu des données qui sont échangées avec les services. Seulement des pointeurs sont passés d'une activité à l'autre. Par contre, les développeurs sont responsables de produire la logique de récupération de données, ainsi que la transformation des données en paramètres attendus par les services. L'avantage de cette approche est qu'elle évite des échanges de

données complexes entre les activités, et dispense ainsi le moteur d'exécution de la manipulation de grands volumes de données.

Nous allons maintenant considérer comment les données servant à maintenir l'état de l'orchestration sont traitées. Deux approches sont utilisées, la première consiste à avoir un espace de mémoire dédié (blackboard) pour garder les valeurs des variables utilisées. La deuxième consiste à définir de façon explicite des flots de données entre les activités.

Dans l'approche de blackboard, chaque instance de l'orchestration possède son espace en mémoire où elle garde une copie des variables. Lors de l'invocation d'un service, les variables sont accédées et passées comme paramètres de l'invocation, ensuite la réponse de l'opération affecte les valeurs des variables du blackboard. L'affectation des variables écrase leurs valeurs antérieures, donc le canevas est responsable de fournir une politique à appliquer dans le cas de modifications concurrentes de données. **WS-BPEL** et eFlow ont un système de traitement de données de type **blackboard**. La méthode blackboard a l'avantage d'être bien connue puisqu'elle est vastement utilisée par les langages de programmation conventionnels.

Pour sa part, l'utilisation de flots de données implique que l'orchestration doit indiquer explicitement les données qui seront transférées d'une activité à l'autre. Ainsi, les données d'entrée et de sortie d'une activité sont explicitement définies. **APEL [76]** et **JOpera [74]** usent la définition explicite de flots de données. JOpera sépare complètement le flux de données et de contrôle, tandis qu'APEL utilise l'abstraction de flot de données pour exprimer les deux. Une propriété du mécanisme des flots de données est qu'il permet d'avoir différentes versions de la même variable dans la composition, ce qui fournit une flexibilité pour sa définition, mais qui ajoute de la complexité à la composition.

Finalement, par rapport au modèle de typage de données, certains formalismes utilisent leur propre système de typage et des autres font recours à des modèles répandus. Ainsi, JOpera **[74]**, **eFlow [72]** et **YAWL [81]** utilisent un modèle propre. WS-BPEL et **XPDL** utilisent la spécification **XML-Schema**, ce choix est naturel puisque la plupart des approches visent la composition de service Web.

### 4.2.4. Liaison de services

Ce critère vise à déterminer comment est réalisée la liaison entre l'orchestration est les services qu'elle compose. Une approche d'orchestration peut réaliser une liaison statique ou une liaison dynamique. Une liaison statique peut être réalisée au moment de la conception de

l'orchestration ou au déploiement. A la conception, les services à utiliser sont exprimés dans la spécification de l'orchestration, tandis qu'au déploiement, généralement un fichier de configuration est utilisé pour les spécifier.

Une liaison dynamique ou à l'exécution peut être mise en place par le biais de différents mécanismes. En premier lieu, un mécanisme utilisant une variable pour stocker la référence du service à invoquer (**dynamic binding by reference**). Ce mécanisme n'indique pas comment la variable sera affectée. Une option consiste à déterminer les valeurs des références de services lors de la création d'une instance de l'orchestration. Par exemple, ces valeurs peuvent être affectées par un administrateur en utilisant une interface pour fournir ce but. Une autre option est d'utiliser le cadre d'interaction définie par l'approche à services, donc chercher dans un annuaire les références des services à utiliser. L'avantage de ce mécanisme est sa facilité d'utilisation et de mise en place. Par contre, le modèle de l'orchestration est pollué avec la définition des variables gardant les références des services, ainsi que pour les activités chargées d'affecter ces variables. Ce mécanisme est proposé par la spécification de WS-BPEL **[70]**.

Un autre mécanisme consiste à utiliser une requête exprimée dans un langage déterminé et basée sur les propriétés du service (prix, temps de réponse, etc.). Cette requête est associée aux références des services. Lors de l'exécution de l'activité d'invocation la requête est analysée par le middleware supportant l'exécution de l'orchestration afin de trouver les services pertinents (**dynamic binding by lookup**). Si lors de la requête, plusieurs services sont retournés par le middleware, une phase de sélection doit être accomplie afin d'en choisir un. **SELF-SERV [71]**, **WSFL [75]** et **eFlow [72]** utilisent ce mécanisme de liaison. **SCENE [85]**, propose une extension de WS-BPEL incluant un langage de requêtes et un runtime afin de supporter ce type de liaison à l'exécution. En plus, ce canevas supporte la reconfiguration dynamique de la composition pour gérer des situations comme la disparition d'un service ou la détérioration de ses propriétés.

Il existe des travaux essayant de pourvoir des mécanismes de liaison et sélection de services pour les applications orientées services. Dans **[86]**, la sélection de composants fournissant la fonctionnalité requise est une préoccupation gérée tout au long du cycle de vie d'une application à services, dès la conception jusqu'à l'exécution.

### 4.2.5. Gestion des exceptions

Ce critère détermine comment l'orchestration gère les situations d'exception lors de son exécution. Ces situations peuvent être la conséquence des problèmes des fournisseurs des services, par exemple un serveur qui tombe en panne ou bien par une réponse indiquant une faille d'un service invoqué. Des actions associées à la logique de l'orchestration peuvent aussi produire des situations d'exception, par exemple l'annulation d'une commande dans une orchestration supportant un processus de vente de produits.

Afin de gérer ces situations d'exception, nous identifions trois moyens différents. D'abord, pour les approches qui n'ajoutent pas de constructions dans le formalisme de spécification pour gérer des exceptions. Une méthode basée sur le contrôle de flux (flow-based) peut être utilisée, elle consiste à évaluer l'état de l'orchestration après l'invocation d'une opération et à ajouter des actions afin de gérer les situations d'exception. Dans APEL **[76]**, ce type de gestion est mis en place.

Une deuxième technique consiste à ajouter des constructions dans le langage afin de capturer, gérer et propager des exceptions (**try-catch-throw**). La logique de gestion des exceptions est ajoutée à une activité (ou à un groupe d'activités) et une condition booléenne est associée à un ensemble des variables, généralement le résultat de l'invocation de l'activité. Si la condition est évaluée à vrai, la logique de gestion est exécutée. Ensuite, l'orchestration peut soit continuer avec l'exécution de l'activité suivante, soit réessayer d'exécuter l'activité qui a échoué, soit terminer l'exécution du procédé. Cette technique a été adoptée par le langage WS-BPEL. Une technique try-catch-throw permet de séparer la logique des exceptions de la logique normale de l'orchestration, cette structuration facilite la conception et la maintenance de la spécification.

La troisième technique, consiste à gérer les exceptions en utilisant des règles ECA. Dans ce cas, des événements sont associés aux situations d'exception et la condition sert à vérifier si la situation doit être traitée, finalement l'action définit comment l'exception sera traitée. Les règles sont définies dans un langage autre que le langage de définition de l'orchestration. Cette technique permet aussi de séparer la logique normale de la composition de la logique de gestion des exceptions, par contre il n'existe de structuration de la spécification comme dans la technique **try-catch-thow**. Il existe aussi un problème associé au fait que le développeur doit maîtriser deux langages différents. Finalement, le passage à l'échelle est limité car une grande quantité de règles rend difficile de comprendre et traiter la

spécification. Une variante intéressante est présentée par le système de workflow **YAWL [87]**. En plus d'utiliser les règles pour détecter les situations d'exception, ce système sélectionne dynamiquement la logique (l'action) qui va gérer l'exception.

### 4.2.6. Orchestration de services et les aspects non-fonctionnels

Les modèles de composition fournissent des abstractions afin de faciliter la composition de services tout en cachant un certain nombre de détails techniques de bas niveau comme par exemple la communication entre la composition et les composants fournissant les services composés. Cependant, une composition de services comme toute autre application doit faire face à plus d'une préoccupation, notamment des aspects non-fonctionnels doivent être considérés comme par exemple, la sécurité, la gestion de transactions, etc.

L'alternative proposée à ce type de problématique dans les systèmes de workflows développés dans les années 90 était de fournir toutes ces propriétés dans un seul système monolithique. Cette solution produisait des formalismes de définition de procédés complexes avec un grand nombre de concepts, et par conséquent difficiles à manipuler. En plus, ça donnait des implémentations de middlewares lourdes, difficiles à mettre en place, configurer, exploiter et maintenir. Ces middlewares sont très coûteux tant au niveau des licences, que du développement et de la maintenance. Un défi auquel se confrontaient les fournisseurs de ce type de solutions, était de déterminer les préoccupations qui seront adressées par leur produit. Chaque fournisseur proposait une solution généralement adaptée à un métier spécifique.

L'approche à services a permis de bien définir les interfaces entre les applications (les services) et le système contrôlant la composition (le workflow). Cependant, cette approche ne spécifie pas comment les aspects non-fonctionnels de services seront traités. Par exemple, dans le cas des services Web, prolifèrent un nombre important de standards cherchant à définir comment les services supportent les propriétés non-fonctionnelles. Parmi eux, nous pouvons citer **WS-Security [88]** qui spécifie comment sécuriser le protocole SOAP et **WS-AtomicTransaction [89]** qui spécifie comment le 2PC doit être utilisé par un groupe de services web participants dans une transaction. Pour sa part, les technologies de composition se focalisent dans la définition de la logique de composition laissant de côté le support des aspects non-fonctionnels.

Il existe deux façons alors de s'attaquer aux problèmes des aspects non-fonctionnels dans une composition de services. La première approche, consiste à inclure dans le

formalisme de spécification de la composition des concepts servant à exprimer les aspects non-fonctionnels.

L'autre alternative est l'utilisation d'un middleware supportant l'aspect non-fonctionnel. Dans le premier groupe, nous trouvons des canevas comme **WebTransact [90]** qui ajoute les concepts de traitement de transactions dans le langage de composition. Dans le second cas, nous pouvons citer par exemple des middlewares implémentant la spécification **WS-AtomicTransaction**. Par contre, ces middlewares exposent leurs services non-fonctionnels en tant que services Web, et la logique de la composition est modifiée pour ajouter l'invocation de ces services. Par conséquent, la logique de composition est mélangée avec la logique de traitement des aspects non-fonctionnels.

Une autre solution, consiste à utiliser des mécanismes similaires à ceux proposés par l'**AOP [91]**. Les différentes préoccupations sont alors spécifiées séparément de la logique de composition et ensuite, en utilisant une définition de **pointcuts**, ces aspects sont tissés pour produire une nouvelle composition. Dans ce groupe, nous trouvons des approches comme **AOBPEL [92]** et **[93]**. Bien que résolvant les problèmes de modularité et de séparation des préoccupations, cette solution est soumise aux mêmes contraintes que la première, c'est-à-dire, soit le formalisme de composition fournit les concepts pour exprimer les aspects non-fonctionnels ou soit un middleware fournit le support en utilisant la même technologie à services que la composition.

Nous considérons qu'une troisième alternative consiste à permettre l'utilisation de différents formalismes pour exprimer les différentes préoccupations non-fonctionnelles. Une intégration des différentes préoccupations doit être réalisée afin de produire la spécification finale de la composition à exécuter. En conséquence, les machines d'exécution supportant chaque préoccupation doivent aussi être composées afin de créer une machine supportant la composition de services avec les aspects non-fonctionnels ciblés. Cette vision, permet d'abord de séparer les préoccupations et de modulariser la composition, mais à la différence d'une approche du style AOP, il n'est pas obligatoire d'utiliser le même formalisme de composition pour exprimer tous les aspects non-fonctionnels. Une autre propriété importante de cette approche est son niveau de réutilisation. Etant donné que les préoccupations sont exprimées séparément, la spécification de la composition peut être réutilisée.

### 4.2.7. Extensibilité des modèles d'orchestration

Le support de différents types d'extension permet à un canevas ou modèle d'orchestration d'attaquer des préoccupations pour lesquelles il n'a pas été conçu au départ. Un nombre important de ces préoccupations est d'ordre non-fonctionnel, mais il existe aussi des préoccupations fonctionnelles pour lesquelles un canevas de composition peut être étendu. Par exemple, une propriété importante de systèmes de *workflows* est qu'ils peuvent utiliser différent types de ressources afin d'accomplir les tâches d'un procédé, notamment des humains. Les modèles d'orchestration font l'hypothèse simplificatrice que les tâches sont toujours exécutées par des services. Afin de supporter ce type d'assignation de ressources, des mécanismes d'extension doivent être mis en place dans le modèle de composition.

Les modèles d'orchestration étudiés offrent des mécanismes d'extension au niveau du langage de spécification. Ces extensions permettent l'introduction de nouveaux types d'activités spécialisées pour exécuter des tâches spécifiques. Dans ce type d'extension, nous trouvons **BPEL4People [94]** qui ajoute un type d'activité réalisée par des humains dans WS-BPEL. L'inconvénient avec ce type d'extensions est que le moteur d'exécution de la composition doit être capable de comprendre les nouveaux types d'activités, ce que implique la modification de l'exécution (runtime). Dans **[93],** une approche orientée aspects pour étendre un moteur d'exécution de WS-BPEL est proposée. D'autres extensions non invasives permettent d'ajouter des fonctionnalités sans modifier le langage de définition de la composition. Par exemple, SCENE **[85]** supporte la liaison à l'exécution et la reconfiguration dynamique d'une composition de services Web, sans ajouter de concepts dans le langage de composition.

Nous considérons qu'une alternative d'extension consiste à permettre l'utilisation de différents formalismes pour exprimer les différentes préoccupations (fonctionnelles et non-fonctionnelles), et ensuite fournir des mécanismes permettant l'intégration des différentes préoccupations afin de produire la spécification finale de l'orchestration à exécuter.

Conséquemment, les machines d'exécution supportant chaque préoccupation doivent aussi être composées afin de créer une machine supportant l'orchestration de services avec les nouvelles préoccupations ciblés.

## 4.3. WS-BPEL

WS-BPEL **[70]** est un standard visant la composition de services Web en utilisant un formalisme de définition de procédés issu du monde industriel. WS-BPEL est le résultat de la fusion de deux langages d'orchestration précurseurs, WSFL **[75]** d'IBM et XLANG **[83]** de Microsoft. Maintenant, le consortium OASIS est chargé de standardiser le modèle de composition WS-BPEL.

Afin de mieux comprendre le modèle WS-BPEL, dans cette section nous allons présenter d'abord un exemple de composition en utilisant WS-BPEL sans rentrer dans les détails et ensuite les différentes caractéristiques du formalisme seront décrites en utilisant l'exemple afin de les illustrer. La **figure 4.1** illustre l'exemple de procédé BPEL et décrivant un processus de traitement d'une commande. Le contrôle de la composition est réalisé comme suit : Le client commence l'interaction en utilisant l'activité **Receive Order**, ensuite deux chemins peuvent être suivis en parallèle. Dans le premier chemin Sequence1, l'activité **Send Invoice** est chargée d'envoyer la facture et puis l'activité **Receive Payment** est responsable de recevoir le paiement. Dans le deuxième chemin Sequence2, l'activité **Ship Products** envoi les produits au client. Une fois ces activités exécutées, les deux chemins se synchronisent et l'activité Archive Order est chargé d'archiver le dossier, finalement l'activité **Reply Order** confirme au client le succès de la transaction.

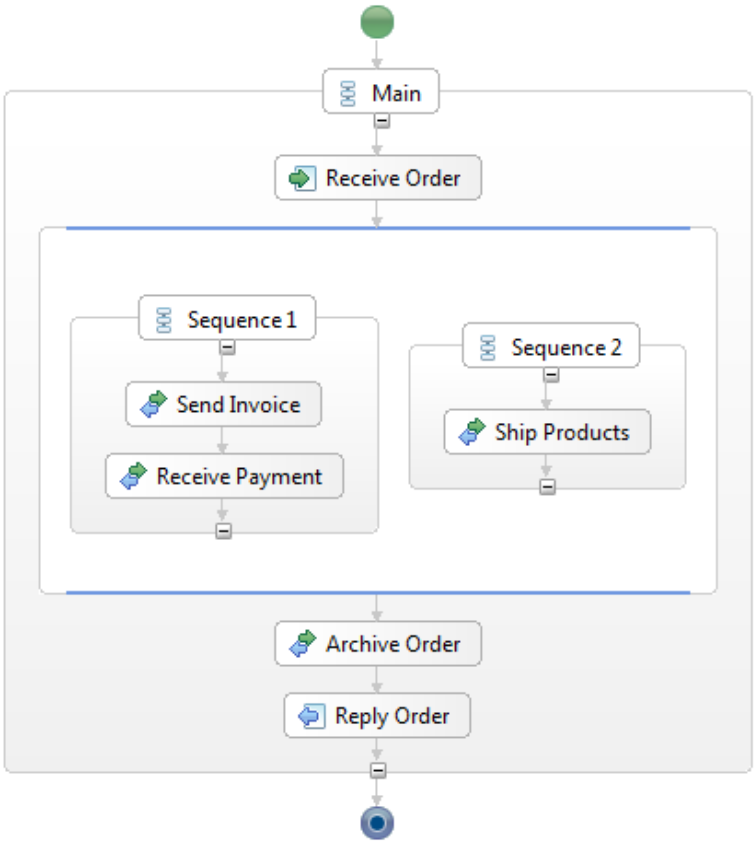



**Figure 4.1 :** Exemple de procédé WS-BPEL.

### 4.3.1. Formalisme de description de l'orchestration

WS-BPEL dispose d'un formalisme héritant des caractéristiques de ses langages précurseurs, donc une vision très structurée proche d'un langage de programmation apportée par XLANG, mais aussi la possibilité de définir de flots de contrôle directement entre des activités comme celui défini par WSFL. Dans la pratique la vision structurée est la plus utilisée.

Le concept central du formalisme WS-BPEL est l'activité. Les activités sont typées et leur type détermine la sémantique d'exécution de l'activité. Les activités peuvent être basiques ou structurées. Les types d'activités basiques permettent l'interaction avec les services, comme **Invoke** (invocation d'une opération d'un service Web.), **Reply** (réponse d'une invocation de la part d'un client), et **Receive** (réception d'un message envoyé par un client). Des autres types d'activités basiques sont **Assign** (assignation de données aux variables) et **Wait** (activité d'attente pour une période ou jusqu'à une date). Les types d'activités structurées permettent l'expression du flot de contrôle du procédé. Dans ce groupe nous trouvons, **Sequence** (une exécution en séquence), **Flow** (exécution parallèle) et **While**,

**ForEach**, **RepeatUntil** (équivalents aux boucles **while**, **for**, **repeat** dans un langage de programmation).

Nous pouvons situer le langage de composition du modèle WS-BPEL comme étant une hiérarchie d'activités. Cette caractéristique peut être constatée par notre exemple, ainsi l'activité structurée Main de type **Sequence** contient quatre sous-activités : **Receive Order**, **Flow1**, **Archive Order** et **Reply Order**. L'activité Flow1 est une activité structurée de type Flow, elle possède deux sous-activités de type Sequence, donc Sequence1 et Sequence2. L'activité Sequence1 contient les sous-activités **Send Invoice** et **Receive Payment**, de sa part l'activité Sequence2 contient la sous-activité **Ship Products**.

### 4.3.2. Liaison de services Web

La liaison entre le procédé **WS-BPEL** et les services Web qui vont offrir la fonctionnalité requise peut être effectuée statiquement (à la conception, au déploiement) ou bien à l'exécution du procédé. Si la liaison est effectuée à la conception, l'adresse (endpoint) du service concret est exprimée dans la définition du procédé. S'il s'agit d'une liaison au déploiement, un fichier de description de déploiement fournit cette information, par contre le format de ce fichier ne fait pas partie de la spécification WS-BPEL.

Lorsqu'il s'agit d'une liaison à l'exécution, le mécanisme d'assignation de références (**dynamic binding by reference**) existe dans le langage. L'utilisation de ce mécanisme entraîne la modification du modèle car il est nécessaire de définir des variables dans le modèle pour garder les références, ainsi que d'utiliser des activités (invocations) pour récupérer les références des services.

### 4.3.3. Modèle de données et d'accès aux données

Le langage WS-BPEL utilise une approche de **blackboard** afin de manipuler l'information de l'orchestration, le concept de variable permet comme dans les langages de programmation de définir des conteneurs pour garder l'information manipulée par l'orchestration. Les variables de WS-BPEL sont typées, mais le formalisme ne fournit pas un moyen de définir leurs types de données, en revanche il s'appuie sur le système de typage de XML-Schema.

Les variables ont une portée déterminée, les variables définies au premier niveau d'une spécification d'un procédé sont considérées comme des variables globales, les autres sont des

variables locales. Les règles lexicales définissant la portée des variables sont les mêmes que celles des langages de programmation.

Il existe deux façons de modifier la valeur d'une variable dans WS-BPEL, tout d'abord les variables sont modifiées par l'invocation des opérations des services Web. Ensuite, la valeur d'une variable peut être modifiée en utilisant une activité de type Assign, ainsi une expression dans un langage de requêtes XML comme XPath est utilisée pour indiquer la modification.

### 4.3.4. Gestion des exceptions

WS-BPEL suit une approche **try-catch-throw** afin de gérer les exceptions. Chaque portée dans le langage peut définir des gestionnaires d'exceptions (**fault handlers**). Un gestionnaire détermine comment l'exception sera traitée, il indique l'exception à gérer ainsi que l'activité à exécuter en cas d'exception. WS-BPEL permet aussi de signaler une exception en utilisant une activité de type **throw**. Le mécanisme de gestion d'exception de WS-BPEL est semblable à celui employé par des langages de programmation comme Java.

### 4.3.5. Extensibilité

Etant donné que la spécification de WS-BPEL spécifie seulement le formalisme d'orchestration de services Web, les mécanismes d'extension proposés sont définis seulement au niveau du langage. Nous allons dans cette section, d'abord présenter le mécanisme d'extension proposé par le langage, et ensuite d'autres mécanismes proposés par certaines des implémentations de WS-BPEL.

Le mécanisme d'extension proposé par WS-BPEL est la création de nouveaux types d'activités. Une activité d'un nouveau type doit fournir les attributs obligatoires pour toutes les activités WS-BPEL, et en plus elle doit ajouter les attributs nécessaires pour l'exécution de l'activité (attributs XML). Si le moteur d'exécution a connaissance du type d'activité il pourra l'exécuter, autrement le moteur ignore l'activité (activité type Empty).

Ce type d'extension offre uniquement un moyen de décrire syntaxiquement un nouveau type d'activité, par contre la sémantique d'exécution est la responsabilité de l'implémentation. Par conséquent, la sémantique d'exécution d'un nouveau type d'activité peut varier entre deux implémentations, ou bien elle n'est simplement pas supportée pour certaines implémentations. Ainsi, le principal inconvénient de ce type d'extension est le besoin de modifier l'infrastructure d'exécution afin d'inclure la sémantique des nouveaux types d'activités.

Quant aux extensions au niveau des implémentations, nous trouvons deux types d'extension: celles qui essayent de combiner le langage WS-BPEL avec d'autres langages, et celles qui ajoutent des propriétés dans l'infrastructure d'exécution afin de supporter des fonctionnalités non fournies dans le formalisme. Dans le premier groupe, sont classifiées des approches comme **BPELJ [73]** qui combine WS-BPEL avec le langage Java afin de supporter l'exécution du code Java à l'intérieur d'un procédé WS-BPEL. Pour sa part, **[92]** propose un langage de règles combiné avec WS-BPEL afin d'avoir des orchestrations de services plus adaptables. Dans les deux cas, les moteurs d'exécution doivent être modifiés manuellement pour inclure la sémantique de l'extension.

Dans le second type d'extension, au niveau de l'implémentation, nous trouvons SCENE **[85]** qui permet d'inclure des caractéristiques comme la liaison des services à l'exécution d'une façon transparente pour l'orchestration. En plus, il est possible de changer les services choisis s'ils ne fournissent pas la qualité de service attendue. Ce type d'extension est techniquement réalisé avec le patron de conception proxy ; de cette façon la modification du moteur d'exécution n'est pas nécessaire, et les composants supportant la nouvelle fonctionnalité sont plus facilement intégrés.

Le langage WS-BPEL n'indique pas comment traiter les aspects non-fonctionnels de la composition, ainsi le support de WS-Transaction **[89]** et WS-Security **[88]** est traité de manière différente pour chaque implémentation, ou non traité. Bien qu'une logique de séparation de préoccupations soit envisagé par WS-BPEL (l'orchestration est spécifie indépendamment de ces aspects comme la sécurité), il est nécessaire d'inclure des mécanismes d'extension permettant d'ajouter ces aspects à l'orchestration et de laisser cette responsabilité comme une tâche de bas niveau réalisée par les développeurs.

Dans l'orchestration de services, deux technologies convergent ; la technologie de **workflow** et l'approche orientée services. La technologie de workflow apportant des formalismes d'expression de procédés ainsi que des outils pour supporter la spécification, l'analyse, la validation et l'exécution de modèles de procédés. Pour sa part, l'approche à services (notamment les services Web) apporte les mécanismes pour réussir un faible couplage entre les services (applications) et leurs clients.

De ce fait, plusieurs modèles, langages, standards et canevas pour l'orchestration de services ont été proposés. Nous avons dans cette section, fait une catégorisation de différentes approches en utilisant un ensemble de critères pour les comparer. Ainsi, au fur et à mesure

que les critères ont été dévoilés, les différents canevas (ou langages) d'orchestration ont été placés selon leurs propriétés vis-à-vis de ces critères.

Nous avons alors insisté sur le langage WS-BPEL **[70]**, car il devient le standard de fait dans cette technologie, même s'il s'intéresse qu'à l'orchestration de services Web.

Dû au manque d'espace, nous n'avons pas détaillé l'approche **JOpera** et **SELF-SERV [95]**, mais nous considérons qu'elle offre une autre vision sur l'orchestration de services. Un tableau comparatif des trois approches est présenté dans le **tableau 4.1**.

De cette analyse, nous concluons que même si l'approche à services a résolu le problème du couplage et de l'interopérabilité (au niveau des protocoles) du système avec les éléments composés, les approches actuelles d'orchestration de services possèdent encore des problèmes hérités de la technologie de *workflow* comme :

- ✓ Des formalismes complexes, de bas niveaux d'abstraction proches des langages de programmation.
- ✓ Support pour un seul type de technologie à services, généralement de services Web.
- ✓ Un support pauvre ou inexistant pour des aspects non-fonctionnels des applications.

Dans le cadre de ce mémoire, nous allons donner un tableau comparatif des différents outils utilisés pour l'orchestration de services.

| | **WS-BPEL** | **JOpera** | **SELF-SERV** |
|---|---|---|---|
| **Technologie des éléments composés** | Services Web | Ensemble prédéfini + possibilité d'ajouter | Services Web |
| **Formalisme de description de l'orchestration** | Hiérarchie d'activités | Diagramme d'activités (Graphe dirigée) | Diagramme d'états et transitions |
| **Liaison de services** | Statique et dynamique (dynamic binding by reference) | Statique et dynamique (combinaison dynamic binding by reference et dynamic binding by lookup) | Dynamique (dynamic binding by lookup) |
| **Modèle de données et d'accès aux données** | Blackboard – Système de typage externe (XML Schema) | Dataflow – Système de typage interne | Blackboard - Système de typage externe |
| **Gestion des exceptions** | Approche try-catch-throw | Approche flow-based | Non spécifiée |
| **Support des aspects non-fonctionnels** | Transactions partiellement supportées avec le concept de Compensation Handler | Aucun aspect non-fonctionnel traité. | Exécution répartie. L'aspect n'est pas Explicitement présenté au développeur |
| **Extensibilité** | Nouveaux types d'activités définies au niveau syntaxiquement | Bibliothèque d'activités prédéfinies. Ajoute de technologies à services | Aucun mécanisme d'extension définie |

**Tableau 4.1 :** Tableau comparatif des approches d'orchestration de services **[96]**.

## 4.4. La SOA (architecture orientée Service)

Le paradigme axé sur le service permet la fourniture de la fonctionnalité logicielle abstraite grâce à des services qui peuvent être composés de manière flexible pour soutenir les processus d'entreprises ou métier. Une architecture orientée services (SOA) permet l'intégration des applications existantes en fournissant leur fonctionnalités comme les services. Ainsi la valeur des actifs logiciels existants est améliorée, la redondance dans l'infrastructure informatique peut être évitée et une réaction rapide aux changements de processus d'entreprises devient possible **[97]**. Pour obtenir ces avantages et soutenir le paradigme axé sur le service, la SOA peut être divisée en plusieurs couches logiques

(voir la **figure 4.2**). La couche inférieure du modèle de référence SOA est l'application et la couche de données qui comprend les systèmes existants (legacy). Ces systèmes peuvent être considérés abstraitement comme des composants fonctionnels. Du point de vue SOA ce n'est pas grave si ces systèmes existants sont des architectures monolithiques internes ou multi-niveaux. SOA intègre et exploite ainsi ces systèmes en exposant leurs fonctionnalités comme des services réutilisables. Lors de la couche processus et intégration de SOA, les services fournis par la couche application sont mappés sur les parties **IT** soutenus par des processus d'entreprises. L'assemblage des services afin d'accomplir la logique métier et les processus d'entreprises est appelé orchestration. **[98]** l'orchestration est définit comme «la mise en place d'un flux de travail lié à l'entreprise appartenant à une entité unique par la combinaison d'activités des services concernés». Les Processus eux-mêmes réalisés par orchestration présentent une interface de service **[99]**.

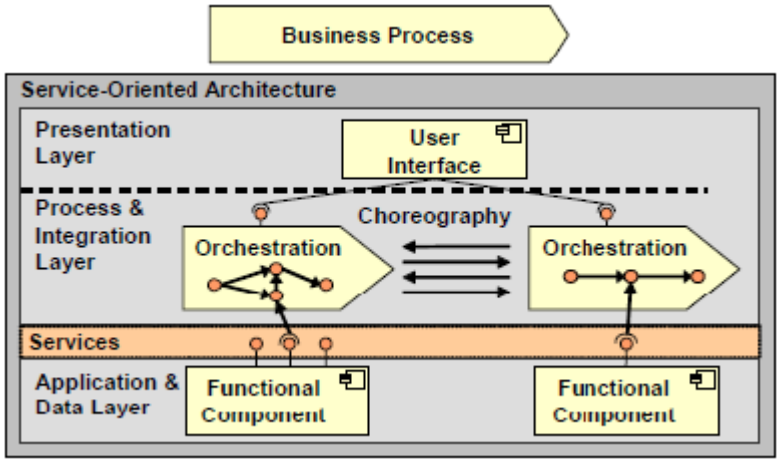



**Figure 4.2 :** Modèle de référence SOA **[100].**

Ces orchestrations d'interfaces de service peuvent être invoquées par la couche de présentation de différentes manières. D'une part, elles peuvent être appelées directement par une interface utilisateur se trouvant sur la couche de présentation, d'autre part, il est possible d'intégrer un système de gestion des tâches entre le processus et la couche d'intégration et la couche de présentation **[101]**. Un système de gestion des tâches assigne des tâches différentes à des utilisateurs ou des rôles responsables et donc permet des processus en cours d'exécution longs dans lesquels plusieurs utilisateurs ou unités d'organisation sont impliqués.

Dans notre mémoire, nous nous concentrons sur l'architecture interne de la couche de présentation de SOA et de l'interface de l'utilisateur final et ne pas examiner avant la connexion de la couche processus. Les questions centrales abordées par notre étude sont :

- Comment appliquer l'approche axée sur le service à la couche de présentation?
- Quelles sont les technologies et les normes possibles utilisées sur la couche de présentation de SOA?
- Comment migrer les interfaces utilisateurs existantes à la couche de présentation de SOA?

Suivant ces questions notre chapitre est organisé comme suit : d'abord l'approche orientée services est introduite et appliquée à la couche de présentation de SOA. Ensuite une classification des services de présentation est donnée. Les *Portlets* et les portails sont les standards de l'état de l'art pour une utilisation sur la couche de présentation de SOA qui sont décrits sur la section 7, la section 8 présente les standards *Web Services for Remote Portlets* (WSRP) comme l'une des normes les plus pertinentes figurant sur la couche de présentation de SOA. La section 9 présente notre approche pour migrer les composants de présentation de la SOA. Comme notre étude donne un aperçu de l'état de l'art sur la couche de présentation de SOA, nous allons souligner les travaux connexes dans la section 10, et enfin dans la section 11 on évoquera des perspectives pour l'avenir et on finira par une conclusion.

## 4.5. L'approche orientée service pour la couche de présentation

Les avantages d'une approche axée sur le service sont comme un couplage dénoué, l'interopérabilité et la réutilisabilité sont basées sur un modèle d'interaction de base impliquant trois parties principales **[102]** : le fournisseur de services, le consommateur de services et le registre de service. L'interaction entre ces trois parties est souvent désigner comme le paradigme « **trouver-lier-exécuter** ». L'approche axée sur le service a été appliquée avec succès à la couche application du SOA. La fonctionnalité qui est mise en œuvre, par exemple une application existante est assurée par une interface de service et ainsi rendue accessible aux consommateurs de services comme indiqué dans la **figure 4.2**. Les avantages de l'approche orientée services mentionnés sont également attrayants pour la couche de présentation, car l'évolution des processus d'entreprises rapides nécessitent des interfaces utilisateur réglables, interopérables et flexibles. Il y a donc la nécessité d'une interface de service bien défini sur la couche de présentation **[103]**. En outre, le paradigme « trouver-lier-exécuter » doit être appliqué à la couche de présentation, ce qui signifie le prestataire de services, le consommateur de services et le registre de service doivent être identifié pour la couche de présentation.

Nous distinguons les services de présentation **[104]** des services d'entreprises tels que décrits plus loin dans la section suivante. Sur la couche de présentation, le paradigme « **trouver-lier-exécuter** » est mis en œuvre par les trois parties suivantes :

- **Fournisseur de Services de Présentation :** Fournit des composants de présentation en tant que services. Comme sur la couche application, il y a généralement plusieurs fournisseurs de services offrant des services de présentation prévu à des fins différentes.
- **Consommateur de services de Présentation:** Invoque un ou plusieurs services de présentation afin de les intégrer dans un contexte spécifique, comme par exemple un panneau de commande. Il peut y avoir plusieurs consommateurs de présentation sur la couche de présentation, chacun d'eux ayant un but commercial.
- **Registre de Services de Présentation:** Tous les services de présentation mis à disposition par les fournisseurs de services sont répertoriés et peuvent être trouvés par les consommateurs de services. Les registres de services peuvent être considérés comme échelle de l'entreprise.

Pour être en mesure d'identifier ces parties sur la couche de présentation, le modèle de référence SOA ci-dessus doit être affiné. La **figure 4.3** se concentre sur la couche de présentation montrant un modèle de référence raffiné où l'interface de service mentionné est présenté en fournissant la base pour réaliser l'interopérabilité et la réutilisabilité sur la couche de présentation. Le conteneur de présentation agit comme un consommateur de service de présentation et invoque plusieurs services de présentation qui sont basés sur les composants de présentation. Ainsi, ces composants de présentation sont fournis par un ou plusieurs fournisseurs de services de présentation et sont intégrés à la SOA par une interface de service.

Afin de trouver le service adéquat, le consommateur de services de présentation peut consulter le registre des services de présentation. Il dispose d'une interface de service aussi bien à travers laquelle les services sont découverts par le consommateur du service de présentation. Pour cette raison, l'interface de service fourni par le registre de service n'est pas axée présentation, mais axée entreprise. Nous avons noté cette interface de service sur la couche de services entreprise.

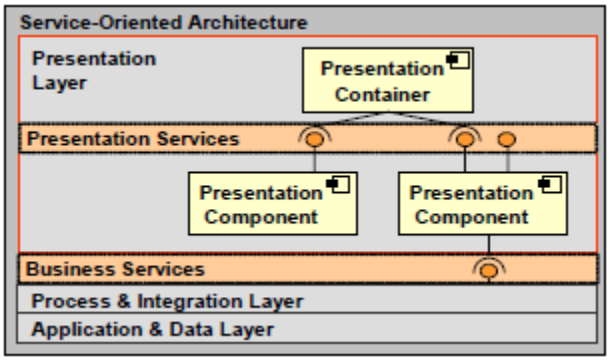



**Figure 4.3 :** Modèle raffiné de la couche de présentation de SOA **[100].**

## 4.6. Les services de présentation

La **figure 4.3** montre qu'il existe différents types de services intégrés dans SOA pour servir à différentes fins. Il faut distinguer entre les services de présentation et les services d'entreprises. Comme souligné **[104]**, les services de présentation fournissent une interface utilisateur et permettent de ce fait une interaction directe entre l'utilisateur et le service. Cette interaction est la principale différence pour les services classiques d'entreprises. Certaines sources rapportent que ces services sont comme des services d'interaction **[105]**. Un service de l'entreprise se concentre sur le traitement des données sans aucune intervention humaine. Ce service suit le modèle de **requête-réponse** par la réception d'une demande, son traitement et ensuite générer une réponse au niveau programmatique. Il génère ainsi une valeur d'entreprise qui est utilisée dans la communication directe de machine à machine.

Un service de présentation ne traite pas les données relatives aux entreprises et ne génère donc pas de valeur commerciale. Il permet aux utilisateurs d'interagir avec les services d'entreprises en leur fournissant par exemple des fragments de balisage qui peuvent être agrégés par conteneurs de présentation comme, par exemple, un portail. Les services de présentation agissent comme une passerelle entre les services d'entreprises et l'utilisateur. Habituellement, un processus métier est soutenu par plus d'un service d'entreprises et a donc peut-être besoin de plus d'un service de présentation. Un portail est une possibilité typique d'impliquer les utilisateurs avec les processus métier **[99]**. Il contient des portlets représentant l'interface de l'utilisateur pour les processus métier.

## 4.7. Les services web

### 4.7.1. Définition de Web Services

Nonobstant les nombreux débats sur la définition du terme Web Services, nous utiliserons deux définitions :

Au sens large, les services web sont des systèmes logiciels, permettant l'interopérabilité entre plusieurs systèmes logiciels (agents) sur un réseau informatique.

Plus spécifiquement, lorsque nous utilisons comme base les normes du W3C, l'interface du système est définie par un langage lisible par un ordinateur (WSDL). D'autres système logiciels vont communiquer avec le service Web selon sa description en utilisant le langage SOAP, généralement en utilisant XML pour sérialiser les messages et HTTP comme protocole réseau.

Les Web Services rendent disponibles à plus grande échelle les intergiciels (middleware) traditionnels.

### 4.7.2. Fonctionnement d'un service web

Pour utiliser un Web Service, il faut premièrement savoir qu'il existe. UDDI (Universal Description, Discovery and Integration Service) est la norme qui définit le mécanisme pour découvrir dynamiquement des services. Un client pointe vers un registraire UDDI, qui lui donnera la définition du service recherché. Le registraire UDDI sert de pages jaunes et liste les services disponibles. Le registraire UDDI est lui-même un Web Service qu'un client peut questionner.

Pour être capable d'utiliser un Web Service et de programmer un client, il est nécessaire d'en connaître la définition. Le langage WSDL (Web Services Definition Language) décrit l'interface au service. En utilisant XML Schema, WSDL défini les paramètres d'entrée et de retour d'un appel au service Web.

Les appels comme tel aux Web Services sont effectués avec le protocol SOAP (Simple Object Access Protocol). SOAP offre le transport d'objets sérialisés et autres données en XML et l'appel de procédures distantes.

UDDI, WSDL et SOAP sont les trois normes principales des Web Services. Les normes UDDI sont proposées par OASIS. WSDL et SOAP font parties des normes W3C.

## 4.8. Portails et Portlets

Un service de présentation peut fournir par exemple des fragments de balisage. Les fragments de balisage doivent être regroupés dans un contexte plus large comme un portail **[106]**. D'un point de vue technique, un portail fournit un conteneur pour agréger du contenu provenant de diverses applications pour la présentation à l'utilisateur final. L'utilisateur ne reconnaît pas les différentes applications et il ne se soucie pas comment le contenu ou la fonctionnalité est fournie. Il veut utiliser une interface unique qui devrait être réglable à ses besoins. Un portail est une solution possible, car il est généralement une application basée sur le Web, il agit en tant que passerelle entre les utilisateurs et une gamme de différents services de l'entreprise. L'utilisateur peut personnaliser le look tout en soutenant davantage l'authenticité unique de l'approche pour la sécurité **[107]**. Du point de vue de l'utilisateur un portail est un espace de travail personnalisable donnant accès à l'intégration de toutes les applications nécessaires avec un seul identifiant.

Au sein d'un portail, plusieurs *portlets* sont agrégés. Ils peuvent chacun être considérés comme une interface utilisateur pour une application et ils sont en cours d'exécution à l'intérieur d'une page portail avec n'importe quel nombre de *portlets* similaires. Les *portlets* sont définis comme des composants de l'interface utilisateur enfichables et autonomes qui sont gérés par un conteneur (portail) **[108]**. Ils traitent les requêtes et génèrent du contenu dynamique. Par exemple, si un utilisateur appuie sur un bouton ou une autre sorte d'élément de déclenchement d'événements de l'interface utilisateur, le *portlet* traite cette requête et génère le contenu adéquat devons être affiché à l'utilisateur.

Un *portlet* peut être mis en œuvre de manière très différente. Certains sont basés sur les standards (par exemple JSR-168), tandis que d'autres sont la propriété exclusive du portail qui les héberge. Chacun de ces *portlets* génère des fragments de balises dont les agrégats du portail créent une page complète qui est présentée à l'utilisateur. Enfin un *portlet* fournit la fonctionnalité d'un service de présentation qui est basée dessus. Si un *portlet* est développé suivant une norme de *portlet*, il peut certainement être appliqué à n'importe quel portail implémenté avec cette norme, mais encore ce *portlet* serait encore limité à un *framework* portail dans le sens qu'il ne supporte pas l'interopérabilité ou le couplage lâche, ceci, n'est pas conforme avec l'approche orientée service. Ainsi, il existe un besoin d'une plate-forme et d'un *framework* couvrant la norme ou le standard *portlet*.

## 4.9. Web Services for Remote Portlets (WSRP)

Les services Web prennent en charge l'approche orientée services SOA en permettant l'interopérabilité entre les différents systèmes basés sur l'utilisation de standards ouverts **[109]**. Par conséquent, l'Organisation pour la promotion des Standards Structurés de l'Information (**OASIS**) a adopté un autre protocole de services Web pour agréger du contenu et des applications Web interactives à partir de sources éloignées nommées Web Services for Remote Portlets (**WSRP**). Avec WSRP il est possible d'intégrer des portlets à partir de différents prestataires de services (à distance) sans se préoccuper de la mise en œuvre et la fourniture du portlet lui-même. Il y a une interface de services Web bien défini où le portlet peut être invoqué. Donc, si par exemple une entreprise doit mettre en œuvre un nouveau processus métier avec l'interaction humaine, elle doit d'abord chercher un service de présentation adéquat dans un registre de service.

En cas de succès, il n'est pas nécessaire de mettre en œuvre l'interface utilisateur. De cette façon l'approche orientée service est appliquée à la couche de présentation de la SOA. Avant de présenter l'architecture proposée par WSRP, les principaux acteurs définis par la norme WSRP doivent être introduits (à comparer avec le paradigme « trouver-lier-exécution ») :

- **Le Producteur WSRP :** Fournit au moins un *portlet* qui peut être personnalisé par un consommateur qui intègre le *portlet* à sa/son portail. Le producteur lui-même est conçu comme un service Web. Si le producteur fournit plus d'un *portlet*, il fournit souvent l'exécution du *portlet* (un conteneur que l'on appelle *portlet*) afin de gérer les *portlets*.
- **Le consommateur WSRP:** Mis en œuvre en tant que client de service Web, comme par exemple, un portail qui est capable d'invoquer des services Web via WSRP. WSRP soutient n:m relations entre les fournisseurs et les consommateurs. Ainsi, il est par exemple possible pour un consommateur de soutenir des flux de processus métier intégrant plusieurs portlets à partir de différents producteurs. Comme les *portlets* sont fournis en tant que services, le consommateur fournit des autorisations pour accéder aux *portlets* nécessaires.
- **Le portlet WSRP :** est très similaire au standard *portlet*. La principale différence se trouve dans l'accès au portlet. Le portlet WSRP est fourni par un producteur WSRP et est accessible à distance via l'interface définie par ce producteur. Contrairement à un portlet, un portlet WSRP ne peut être accessible directement mais doit être accessible par le biais de son producteur WSRP parent.

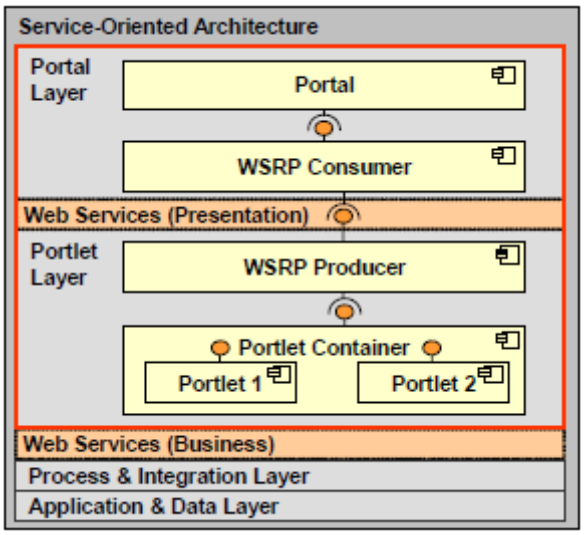



**Figure 4.4 :** Architecture WSRP sur la couche de présentation de SOA **[100].**

La **figure 4.4** étend le modèle de référence d'une SOA avec les spécifications architecturales de WSRP. La norme WSRP définit un ensemble d'interfaces que tous les producteurs WSRP ont à mettre en œuvre. La standardisation de ces interfaces permet aux portails compatibles WSRP d'interagir avec l'exécution à distance des *portlets*. Il existe deux interfaces optionnelles et obligatoires forcées par la spécification WSRP afin de permettre une communication standardisée entre le consommateur WSRP et le producteur WSRP :

- **Interface de description du Service (obligatoire) :** Grâce à l'interface de description du service, un producteur WSRP offre ses *portlets* aux consommateurs WSRP. Le consommateur WSRP utilise cette interface pour découvrir les *portlets* fournis par les producteurs. En outre, cette interface est utilisée si le consommateur a besoin des métadonnées supplémentaires pour le producteur lui-même. Ces métadonnées par exemple contiennent des informations si le producteur requiert un enregistrement ou si un cookie doit être initialisé avant d'interagir avec le consommateur.
- **Interface Markup (obligatoire) :** Cette interface permet à un consommateur d'interagir avec un portlet exécuté à distance. Comme les portlets sont utilisés comme des services, le consommateur utilise cette interface pour effectuer toutes les actions nécessaires concernant l'entrée de l'utilisateur. Par exemple, si un utilisateur soumet un formulaire (markup) à partir de la page du portail, la réponse obtenue est créé en livrant le balisage approprié. Il pourrait également être nécessaire pour le portail de recevoir le dernier balisage du *portlet* (l'utilisateur clique sur un bouton d'actualisation, par exemple). Le *portlet* délivre alors le balisage en fonction de son état actuel.

- **Interface d'enregistrement (facultative) :** Avec cette interface WSRP permet un mécanisme dans la bande d'enregistrement. Cette inscription permet au fournisseur d'arriver avec un comportement et une apparence personnalisée pour le *portlet* qui diffère de l'apparence standard du portlet. Ce mécanisme permet en outre une interaction basée sur les rôles, par exemple, chaque rôle est autorisé à voir les différents *portlets*, le comportement et l'apparence.
- **Interface de gestion du Portlet (facultative) :** Un portlet hébergé par un conteneur de portlet a un cycle de vie. Il est instancié, exécuté et finalement détruit. Avec l'interface de gestion du portlet, le consommateur WSRP obtient l'accès au cycle de vie du portlet exécuté à distance. Le consommateur a alors la possibilité de personnaliser le comportement d'un portlet ou même détruire une instance d'un portlet en cours d'exécution à distance.

Suivant la norme WSRP, l'approche orientée service peut être mise en œuvre sur la couche de présentation de SOA. En outre, WSRP donne les moyens de répondre à une autre tâche relative à l'orientation de services. Comme SOA est largement acceptée comme une architecture logicielle pour l'avenir, il faut examiner les possibilités de migration des applications existantes à toutes les couches de la SOA. La migration des composants de présentation est une tâche qui doit être traitée.

## 4.10.Migration à la couche de présentation de SOA

Avec les progrès des technologies d'intégration, il est possible d'intégrer des applications existantes dans SOA, par exemple, via des services Web. Comme souligné dans l'introduction, il est nécessaire pour une entreprise d'être en mesure de s'adapter à l'évolution rapide des exigences des clients. La mise en place d'une architecture SOA flexible et ajustable pour soutenir ces processus métier en pleine mutation est un objectif stratégique. Alors que la SOA est l'architecture du futur, les systèmes existants doivent être migrés vers SOA. Le processus de migration d'une infrastructure existante de l'application informatique de SOA est un travail très intensif donc il faut réfléchir à la façon dont un processus de migration peut être effectué avec une influence minimale sur les activités au jour le jour **[103]**. Il est raisonnable de viser une migration en douceur qui se définit comme un processus de migration où un composant d'application reste applicable à la fois à l'ancienne et à la nouvelle architecture du logiciel à chaque moment dans le processus de migration. Il y a plusieurs avantages d'une telle migration en douceur **[110][111]** :

- Souvent, des investissements importants ont été faits pour développer des applications héritées ou existantes et de ce fait il est économiquement pas envisageable d'abandonner tout simplement.
- Au cours de l'activité du processus de migration qui doit continuer. Il est impossible pour une entreprise de cesser de vendre ses produits et services pendant plusieurs mois juste pour le but de la réorganisation de ses systèmes informatiques internes.
- comme la documentation sur les logiciels existants ou hérités est généralement pauvre, le logiciel lui-même est le seul endroit où la logique métier est «documentée».

Un autre avantage d'une migration en douceur est que les fournisseurs proposent des modules de tierce partie pour ces applications héritées. Chacun d'entre eux doit être pris en compte si la migration est économiquement avantageuse ou non. Jusqu'à ce que l'application héritée soit arrêté, ses modules restent utilisables.

Une migration en douceur sur la couche de présentation est, selon la définition ci-dessus, fournit des composants de présentation applicables à la fois à l'ancienne et à la nouvelle architecture du logiciel. SOA, avec l'utilisation de WSRP sur la couche de présentation, est notre architecture choisie pour migrer en raison de ses nombreux avantages. Comme des conditions préalables à la notion de migration proposée, nous supposons que l'architecture qui doit être migré n'est pas une application Web, mais une application client-serveur traditionnelle. Ainsi, les principales tâches relatives à la migration sur la couche de présentation sont d'abord d'étendre l'architecture logicielle d'héritage pour être en mesure d'intégrer et d'invoquer les *portlets*, puis de fournir les composants de présentation existants comme *portlets*. Cette disposition se fait par un processus itératif composé de trois étapes de base :

1. Identifier un composant de présentation réutilisable du client hérité.
2. Dupliquer le composant de présentation en utilisant la technologie portlet.
3. Intégrer ce portlet retourné au client hérité.

La première étape consiste en une analyse approfondie du client hérité. Les composants de présentation réutilisables doivent être identifiés et examinés si ces composants peuvent être utilisés pour construire des services Web utiles orientés présentation pour l'architecture SOA. Un autre point important est de vérifier si le composant peut être techniquement réalisé avec la technologie portlet. Comme les portlets sont basés sur les technologies Web standards tels

que HTML, CSS ou JavaScript, et non pas tout ce qui pourrait être fait dans le client hérité fonctionnant sur un système local qui peut être fait dans un portlet. Les Portlets, par exemple, ne peuvent pas utiliser directement les services du système d'exploitation local de l'utilisateur. Il en résulte une liste triée des composants appropriés pour le processus de migration. L'ordre de cette liste est influencé par le coût estimé et le bénéfice attendu de la migration de ces composants.

Ensuite, le composant de l'interface utilisateur doit être implémenté en tant que portlet. Les composants de présentation traditionnels d'applications Web ont tendance à avoir une facilité d'utilisation très inefficace en termes de réaction (retour d'informations) des utilisateurs et les temps de réponse **[112]**. Pour parvenir à l'acceptation par les utilisateurs du client hérité, le nouveau portlet doit atteindre au moins le même niveau de convivialité que le client hérité fourni. Les tendances actuelles dans le développement d'applications Web, AJAX par exemple **[113]** tente de résoudre ces problèmes en transférant une partie du code de l'interface utilisateur comme JavaScript dans le client et de communiquer de façon asynchrone entre le navigateur Web et le serveur.

Le *portlet* peut maintenant être utilisé dans n'importe quel portail WSRP compétent mais le réutiliser dans le client hérité a besoin de quelques travaux initiaux. Le client doit être étendu pour afficher les *portlets* sur l'interface utilisateur actuelle. Cette extension peut être divisée en trois composants, qui doivent être ajoutés au client :

- **Contrôleur du portail :** il agit comme un consommateur WSRP et est chargé d'établir la communication avec le fournisseur WSRP. Il établit en outre la notification aux composants de présentation hérités par l'intermédiaire du dispositif de commande du client.
- **Plugin du portail :** Ce composant est nécessaire pour afficher les fragments de balises agrégées des portlets invoqués pour l'utilisateur.
- **Le contrôleur du client :** Permet au client hérité de communiquer avec les portlets intégrés. Avec cette extension en place le client hérité peut bénéficier de tous les portlets publiés de la SOA.

La **figure 4.5** montre le scénario de migration. Le point central d'intégration dans cette architecture est le conteneur de *portlets*. Chaque composant de l'interface utilisateur qui peut être mis en œuvre en tant que *portlet* peut être immédiatement intégré dans le portail et le client hérité modifié. L'avantage de cette approche est basé sur le fait que, après chaque étape

dans le processus de migration, un système entièrement fonctionnel est disponible. Les composants de présentation des clients hérités sont migrés étape par étape.

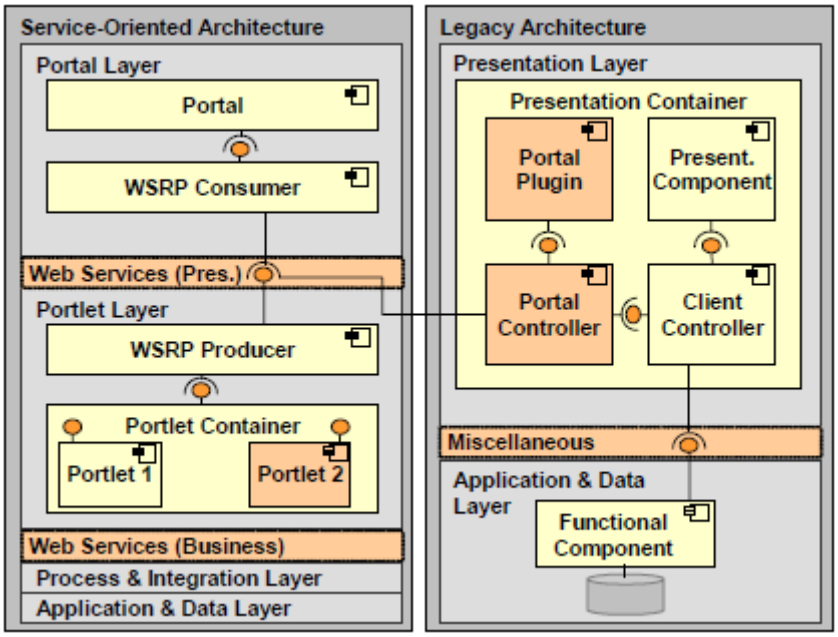



**Figure 4.5 :** Scénario de migration **[100].**

## 4.11. Travaux connexes

Le modèle de référence de l'architecture SOA (Voir **figure 4.2**) qui a été le point de départ de ce chapitre peut être trouvé dans des publications différentes, qui couvrent la couche de présentation comme suit:

- **Erl [98]** est d'accord avec ce point de vue sur SOA, mais ne répond pas à la couche de présentation.
- **Arsanjani [102]** traite la couche de présentation comme une «couche future » qui doit être prise en compte afin de mettre en œuvre des solutions basées sur la convergence croissante des normes telles que WSRP.
- **Caste [104]** attire l'attention sur la couche de présentation en démontrant la conception architecturale donnée par la norme WSRP.

Les deux publications les plus importantes liées aux aspects de la migration d'une SOA sont les suivants :

**Lewis et al. [111]** présente les avantages d'une architecture SOA et souligne que la migration vers une SOA n'est ni facile ni automatique. Ils sont d'accord avec **[110]** que pour

de nombreuses organisations, il est impossible de s'éloigner des investissements réalisés avec les applications existantes et à réaménager la fonctionnalité nécessaire pour les services à partir de zéro. Un processus de migration appelé « La migration et la réutilisation Technique Orientée Service» (SMART) est fourni. SMART rassemble des informations d'entrée sur l'ancien système et après l'exécution de plusieurs activités, il émet une stratégie de migration du service. La stratégie est de ce fait réside à un niveau d'activités élevé et recommande finalement des exemples où les composants de l'application existante peuvent être dérivé de services.

**Hasselbring et Reussner [110]** démarre à partir de la même situation initiale: il y a un système d'héritage qui ne peut être aboli et, par conséquent, il doit être migré vers SOA. Ils présentent un modèle d'architecture appelé **Dublo** (duplication de la logique métier). Nous avons basé notre solution sur l'approche Dublo dont l'idée de base est que la logique métier est formulée dans une nouvelle couche logique métier et un adaptateur existant pour l'accès par la nouvelle logique d'entreprise à la logique métier de l'héritage existant qui est écrit. Cet adaptateur est utilisé pour accéder à la base de données de sorte qu'elle est accessible uniquement via le code hérité existant. Une nouvelle couche de présentation est ajoutée qui gère la nouvelle logique de gestion mise en place pour la nouvelle couche logique métier de l'entreprise.

## 4.12. Conclusion

L'application de l'approche orientée sur le service à la couche de présentation de SOA permet aux entreprises d'ajuster facilement ou reconcevoir des interfaces utilisateur pour l'évolution rapide des processus métiers. Au lieu du réaménagement de l'interface utilisateur à chaque fois que de nouvelles exigences doivent être remplies, l'interface utilisateur est réarrangée par une nouvelle composition de services de présentation. Comme l'indique la norme WSRP qui soutient cette approche orientée sur le service en lui donnant les moyens architecturaux pour fournir des composants de présentation en tant que services. Pourtant, il faut encore composer ces services de présentation manuellement. Comme sur la couche d'application, les langages de modélisation de processus métiers permettent presque de dériver automatiquement un code exécutable à partir d'un modèle de processus métier, ce qui est préférable pour développer des interfaces utilisateur. Plusieurs contraintes comme la convivialité ou la commodité des interfaces utilisateur doivent être prises en compte. Nous comprenons qu'une approche possible consisterait à utiliser un langage de modélisation existant comme **BPMN** et d'ajouter des informations concernant l'interface utilisateur déjà au moment de la conception. Afin d'être en mesure d'ajouter ces informations, il doit examiner si les éléments donnés d'un langage de modélisation sont suffisants ou s'il devient inévitable d'étendre le méta-modèle du langage de modélisation.

L'approche que nous allons présenter permet de migrer les composants de présentation existants aux portlets et de les utiliser dans l'ancienne et la nouvelle architecture orientée services. La migration est et deviendra de plus en plus importante dans les prochaines années, la mise en place d'une architecture orientée services au sein des entreprises devient un objectif stratégique. Considérant les systèmes informatiques hétérogènes qui sont en usage aujourd'hui, il est peu probable qu'un processus de migration commun, général et toujours applicable soit peut être dérivé. Par conséquent, il est d'autant plus raisonnable d'élaborer des stratégies de migration individuelle comme source des principes de base pour de nouveaux projets de migration.

# Chapitre 5 : Mashup web à base d'orchestration de services web

## 5.1. Introduction

Dans ce chapitre, nous allons présenter une réalisation de notre proposition à l'aide de l'exemple des agences de voyage qui permettent d'offrir des réservations de billets, de chambre d'hôtels et d'effectuer le paiement du séjour souhaité. Nous allons décrire l'environnement de travail, les étapes de notre réalisation allant de la conception jusqu'à l'implémentation. Par la suite, nous exposerons quelques interfaces homme machine qui concordent avec les fonctionnalités de notre application. Enfin, nous décrivons le chronogramme.

## 5.2. Proposition

Les formulaires Scénario de tourisme et de vacances sont constitués de différents produits gérés par les différents partenaires (logements, activités, excursions ...) et les canaux de distribution sont nombreux et en constante évolution (agences de voyages, Internet, les comités d'entreprise ...).

Dans le cadre de notre travail, nous allons prendre l'exemple d'une agence de voyage, cette dernière nécessite le développement de certains services web. Ainsi, Nous nous focaliserons sur un service web pour la réservation de billets d'avions, un service web pour la réservation de chambre d'hôtel, un service web pour la localisation géographique de l'hôtel et un service web pour payer les frais du séjour souhaité. Nous savons que quand un client fait un voyage, il commence par la réservation d'un hôtel et d'un vol vers la ville de destination.

Nous savons que pour chaque vol, l'agence sait : la date de départ, la date d'arrivée, la ville de départ, la ville d'arrivée, l'aéroport de départ, l'aéroport d'arrivée, les prix, le temps de départ et le temps d'arrivée. Pour ce qui concerne les hôtels, l'agence recommande dans chaque ville plusieurs hôtels. Les catégories d'hôtels sont variées de 3 à 5 étoiles et chaque hôtel appartient à une de ces trois catégories. Pour tous, l'agence connaît leurs informations : nom, adresse, ville, sa position géographique etc…

Notre *framework* est basé sur le paradigme de programmation, étant donné la facilité de réutilisation des services web et l'intégration rapide de ces derniers.

Sur ce, voici un schéma global du *framework* concernant notre proposition :

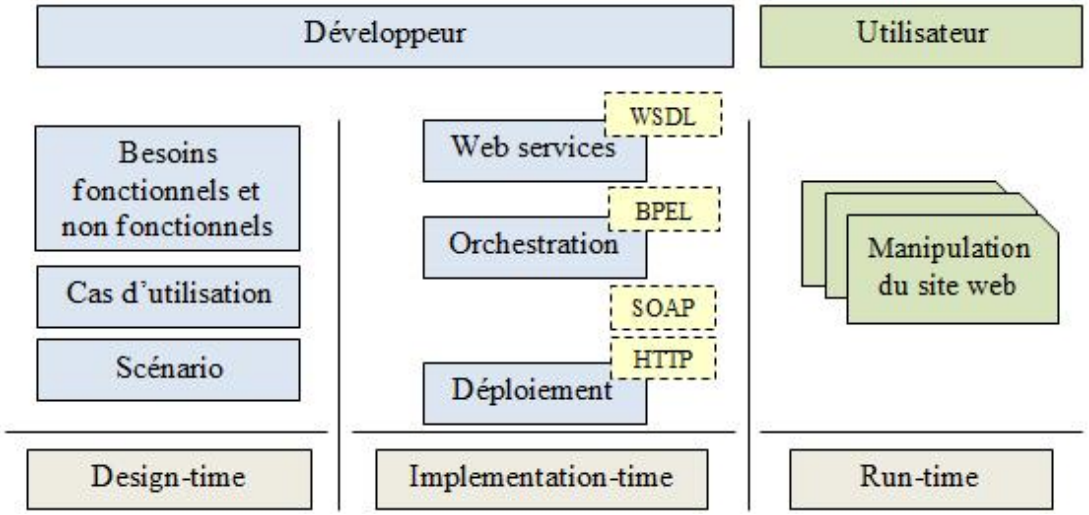



**Figure 5.1** : Schéma global du cadre de travail(framework).

Pour le développement de notre mashup « Agence de voyage » **orienté présentation**, on a opté pour l'approche de **création manuelle** qui consiste au développement d'applications composites qui intègrent des objets intelligents grâce aux technologies du Web telles que HTML, HTTP, Atom et Javascript avec l'utilisation des outils spécifiques des mashups.

Voici une vue globale de notre mashup orienté présentation :

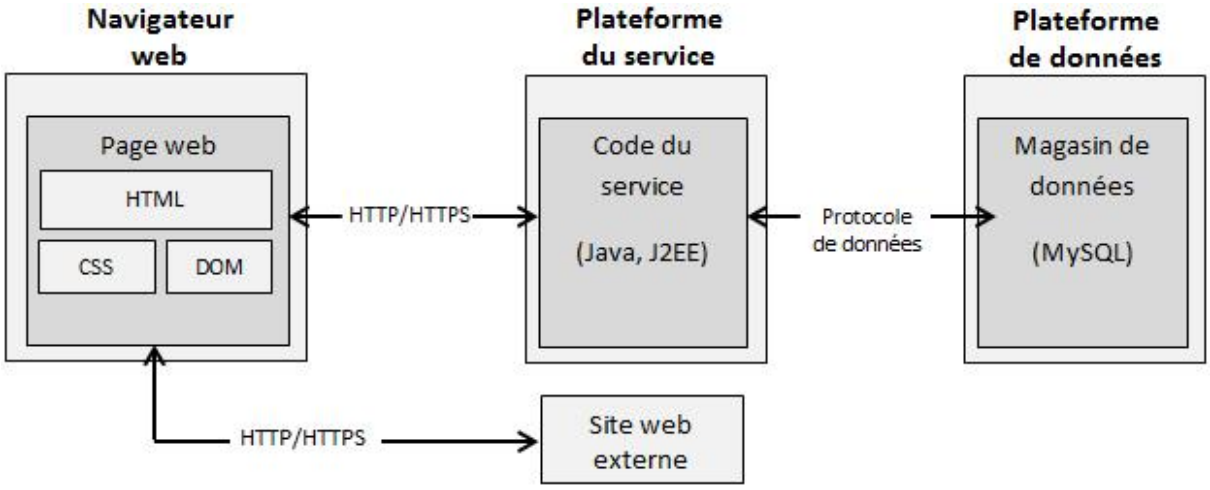



**Figure 5.2** : Vue globale du mixage des objets de présentations de mashup proposé.

Concernant la composition des services web de notre scénario, on a choisi l'orchestration comme approche qui consiste en un mécanisme pour la construction d'un

nouveau service (dit composite) dans une application composée de l'ensemble des services atteignables. Au niveau des orchestrations sont définies des variables qui permettent de partager les informations entre les différentes invocations.

Notre scénario est illustré dans la figure suivante :

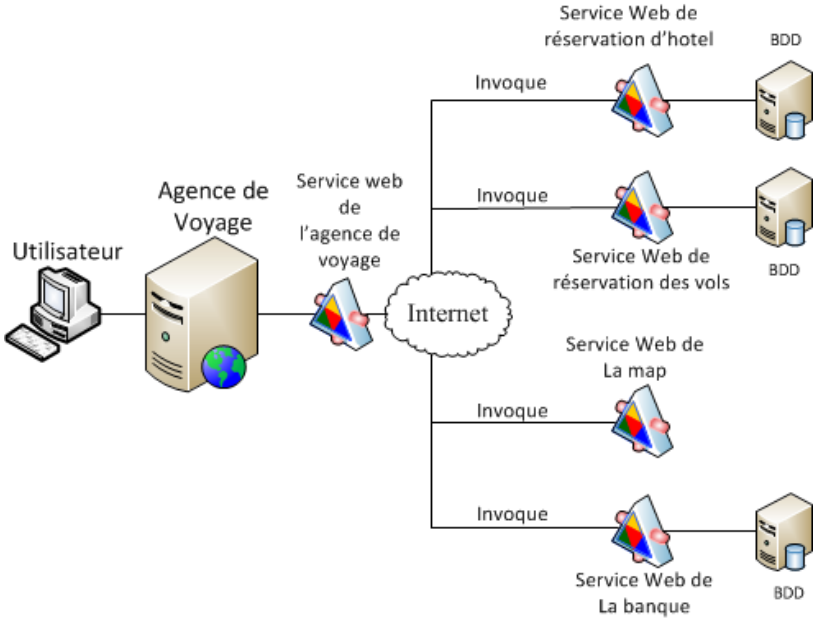



**Figure 5.3 :** Scénario du mashup agence de voyage.

## 5.3. Environnement de travail

### 5.3.1. Environnement matériel

Pour le développement nous avons utilisé deux machines à puissance moyenne : un PC de bureau **Pentium dual-core** avec 2Mo de mémoire cache et 2Go de mémoire vive et un PC portable ainsi qu'une connexion pour tester notre application.

### 5.3.2. Environnement logiciel

Le long de la phase de développement, nous avons utilisé l'environnement logiciel suivant :

- **Système d'exploitation :** Microsoft Windows7.
- **Outils de développement :** Netbeans IDE 6.7.1.
- **Serveur d'application :** GlassFishESB 2.1.
- **Data Base Server :** MYSQL pour la gestion de la base de données.
- **Rédaction du rapport :** Microsoft office Word 2010.

## 5.4. Définition des outils techniques

### 5.4.1. définition de SOAP

Le SOAP est un protocole de communication entre applications par les échanges de messages en XML à travers le Web, il est indépendant des langages de programmation ou des systèmes d'exploitations employés pour l'implémentation des Web services. Il est considéré comme la technologie la plus importante des Web services **[114]**.

Les concepteurs ont préservé la plus grande généralité dans la représentation en XML des principes des protocoles de communication. SOAP vise à servir des protocoles de communication sur les Intranets, dans une optique d'intégration d'applications d'entreprise, et à permettre la communication entre applications et Web services dans un contexte d'échanges interentreprises. Ce protocole ne demande pas de modifications majeures aux serveurs Web déjà installés. Il permet de s'interfacer avec des applications préexistantes grâce au support d'**XML** schéma, en capturant leurs structures de données quelle que soit la complexité. En plus, SOAP propose un mécanisme simple de représentation des différents aspects d'un message entre applications, donc il peut être utilisé dans tous les styles de communication, Intranet ou Internet.

La spécification SOAP est divisée en quatre parties. L'enveloppe SOAP qui définit le contexte d'un message, sa destination, son contenu et différentes options. Les règles de codage SOAP, définissent la représentation des données d'une application dans le corps d'un message SOAP. Le protocole **RPC** (Remote Procedure Call) qui définit la succession des requêtes et des réponses et l'utilisation de **http** comme couche de transport des messages SOAP.

Un message SOAP est un document XML constitué d'une enveloppe contenant un en-tête (header) facultatif et le corps (body) du message **[115]** comme apparu dans le schéma suivant :

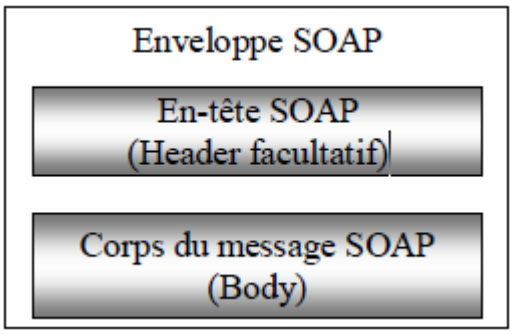



**Figure 5.4 :** Structure du message SOAP **[116].**

- **L'enveloppe SOAP :** représente la racine du document contenant le message SOAP, marquée par la balise (Enveloppe). La spécification impose que tous les attributs de cette balise et de celles imbriquées soient explicitement associées à un en suppriment toute ambiguïté.

Par convention, la spécification SOAP à deux namespaces fréquemment utilisés :

- **SOAP-ENV** associé à l'URI pour définir le message de l'enveloppe.
- **SOAP-ENC** associé à l'URI pour définir les formats de types de données.

- **L'en-tête SOAP (SOAP-ENV : Header) :** la balise (Header) permet de passer dans le message des informations complémentaires sur ce même message. Cet élément est facultatif, mais s'il est présent, doit être le premier dans l'enveloppe SOAP du message. L'en-tête peut contenir par exemple des informations authentifiant l'émetteur ou bien encore le contexte d'une transaction dont le message n'est qu'une des étapes. Pour les couches de transport qui ne fournissent pas, à l'émission, d'adresse de retour, on peut utiliser l'en-tête pour identifier l'émetteur du message.

- **Le corps du message SOAP (SOAP-ENV : Body) :** il contient un ou plusieurs sous éléments qui sont :

- **FAULT :** indique une erreur ou une défaillance en réponse à une requête.
- Des données pour le destinataire du message, à un format défini par les règles de codage SOAP.

### 5.4.2. Définition du langage WSDL

WSDL (Web Services Description Langage) est un langage basé sur XML. Il permet de décrire les composants des Web services nécessaires au protocole SOAP et à l'interaction avec un autre service.

Un document WSDL décrit quelles sont les fonctionnalités qu'offre un Web service, comment communique-t-il et par où est-il accessible. WSDL pourvoi un mécanisme structuré pour décrire les opérations qu'un Web service peut remplir, les formats des messages qu'il peut traiter, les protocoles qu'il supporte et les points d'accès à une instance d'un Web service.

On peut comparer WSDL aux langages de description requis par **CORBA** et **DCOM [117]**. Cependant, WSDL est différent de ceux-ci en ce sens qu'il est tout aussi neutre du point de vue du protocole que du point de vue de l'implantation.

Un document WSDL possède sept sous éléments distincts, divisés en deux parties. Une première partie réutilisable, appelée abstraite, détaille le service. Une seconde partie non réutilisable, appelée concrète, indique la localisation du service. La partie abstraite est composée de cinq éléments : document, import, types, messages et port type. La partie concrète comporte deux éléments: binding et service **[118]**.

- **Document :** cet élément permet simplement d'écrire des commentaires.
- **Import :** cet élément rend possible, si, nécessaire, l'import de documents XML.

Deux attributs doivent être connus. Le premier est l'espace de noms du document à importer qui permet de faire le lien entre ce dernier et le document WSDL. Le second est l'URI de localisation du document à importer.

- **Types :** permet de définir les structures de données contenues dans les messages.
- **Message :** les éléments message définissent l'ensemble des messages qui seront échangé.
- **Port Type :** concerne la mise en place d'opérations, dont chacune est définie par une opération et les messages d'entrée et de sorties associées.
- **Binding :** définit les protocoles de communication utilisés lors des appels du Web service.
- **Service :** permet de décrire les points d'accès du Web service.

### 5.4.3. définition de BPEL4WS

BPEL4WS est un langage de composition proposé par IBM, Microsoft et BEA. Il remplace les précédents langages de workflow WSFL et XLANG. BPEL4WS correspond à une grammaire XML qui décrit des processus métiers. Ceux-ci peuvent être interprétés et exécutés par un moteur d'orchestration (BPWS4J par exemple.).

BPEL4WS est utilisé pour décrire les processus métier exécutables « besiness process » ainsi que les processus abstraits.

Les processus abstraits sont utilisés pour créer les spécifications comportementales des messages mutuellement échangés entre les différentes parties transactionnelles exécutant le processus métier.

#### 5.4.3.1. Fichier BPEL4WS

Structurellement, le fichier BPEL4WS décrit un workflow en déterminant quels sont les participants, que les services doivent implémenter pour être dans le bon endroit, et le flux de control du processus.

Le BPEL4WS process model est construit à base de WSDL 1.1 service model et assume toutes les primitives des actions décrites en WSDL portTypes. La BPEL4WS description décrit la chorégraphie d'un ensemble de messages qui sont tous décrits par leur WSDL définitions. C'est important de faire savoir que WSDL est aussi utilisé pour décrire l'interface externe au workflow. Cela permet au BPEL4WS d'être en matière de composition complet, cela veut dire que la composition des Web services est exposée comme un seul Web service éligible de participer dans d'autres compositions.

#### 5.4.3.2. Structure générale d'un fichier BPEL4WS

La description BPEL4WS se décompose en trois parties :

- Définition des partenaires.
- Définition des types de données et des messages échangés.
- Définition du processus métier.

```
<process>
        <partners>
                <partner/>
        </partners>
        <variables>
                <variable/>
        </variables>
</process>
```

**Liste 5.1 :** Les sections primaires d'un fichier BPEL4WS.

#### 5.4.3.3. Type des activités BPEL4WS

Un processus métier correspond à une séquence d'opérations ou plus exactement à un flux d'activités. Ces activités peuvent faire intervenir de un à plusieurs Web services. On peut distinguer les activités de base des activités structurées.

Les activités de base de ce langage permettent :

- D'invoquer une opération d'un Web service, activité **invoke**.
- De présenter la composition comme un nouveau Web service avec l'activité **receive** pour décrire la réception d'une requête et l'activité **reply** pour générer une réponse.

Les activités structurées utilisent les activités de base pour décrire :

- Des séquences ordonnées **sequence** et des exécutions en parallèle flow.
- Des branchements **switch**, **if** et des boucles **while**.
- Des chemins alternatifs **pick**.

Les activités structurées peuvent à leur tour être combinées pour former une nouvelle activité structurée, celle de plus haut niveau correspondant au processus métier lui-même.

Le langage intègre également des mécanismes supplémentaires, le mécanisme de gestion des exceptions (throw, catch), le mécanisme de compensation (scope) qui permet d'annuler une transaction dans son intégralité lorsque celle-ci échoue.

Il constitue une couche supérieure au langage de description WSDL. Il utilise, en effet, WSDL pour définir les opérations de Web services élémentaires à appeler et pour présenter le processus métier comme un nouveau Web service.

## 5.5. Conception, implémentation et exécution du mashup

### 5.5.1. Conception

Nous entamons maintenant la phase de conception. Cette étape s'avère primordiale pour le déroulement du projet et a pour but de détailler les tâches à entreprendre ainsi que de préparer le terrain pour l'étape d'implémentation.

#### 5.5.1.1. Cahier de charges

Le cahier des charges est un document essentiel à la réalisation d'un projet. Il décrit précisément les besoins auxquels les intervenants doivent répondre.

*a. Les besoins fonctionnels*

Notre mashup doit satisfaire les exigences fonctionnelles suivantes :

- **Chercher un séjour :** Un client pourra chercher un séjour selon les critères suivants: la ville de départ, la destination, la date de départ, la date de retour.
- **Chercher un hôtel :** Un client pourra chercher un hôtel selon la ville où se situe l'hôtel.
- **Chercher un vol :** Un client pourra chercher un vol selon les critères suivants: la ville de départ et la ville d'arrivée, la date de départ et la date de retour.
- **Localiser un hôtel :** Pour chaque hôtel choisi, on affiche sa position géographique.
- **Réserver un séjour :** Un client peut réserver un séjour (vol et hôtel) qui est dans son panier et effectuer le paiement après avoir rempli un formulaire qui comporte les critères suivants : nom, prénom, sexe, date de naissance, carte de crédit, numéro carte de crédit, date d'expiration de la carte de crédit.

*b. Les besoins non fonctionnels*

Il s'agit des besoins qui caractérisent le système. Ce sont des besoins en matière de performance, de type de matériel ou le type de conception. Ces besoins peuvent concerner les contraintes d'implémentation (langage de programmation, type SGBD, de système d'Exploitation...).

Voyons maintenant l'ensemble de nos besoins non fonctionnels :

- **Interface :** L'interface doit être ergonomique et simple à utiliser.
- **Prérequis :** les différents web services sont basés sur une base de données MySQL.
- Notre mashup doit être robuste et maintenable.

#### 5.5.1.2. Spécification détaillée

Afin de détailler les besoins précédemment spécifiés, une bonne réflexion autour du développement de notre mashup par un langage de modélisation comme l'UML (*Unified Modeling Language*) s'avère nécessaire. Nous utilisons alors dans la suite le diagramme de cas d'utilisation et le diagramme de séquences comme moyens de notre spécification.

*a. Le diagramme de cas d'utilisation*

Un cas d'utilisation représente un ensemble de séquences d'actions qui sont réalisés par le système qui produisent un résultat observable intéressant pour un acteur quelconque. On a un seul acteur qui est en l'occurrence le Client.

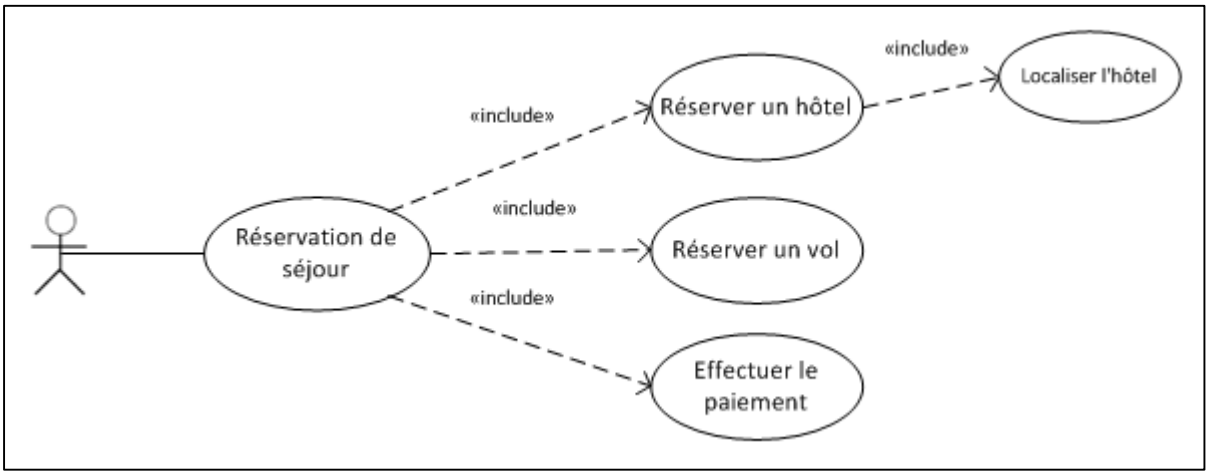



**Figure 5.5 :** diagramme de cas d'utilisation de l'agence de voyage.

| **Description du sommaire** | |
|---|---|
| **Titre** | Réservation de séjour. |
| **But** | Réserver l'hôtel, le vol et effectuer le paiement du séjour souhaité. |
| **Acteurs** | Client. |
| **Description des enchaînements** | |
| **Scénario nominal** | 1. Le client remplit le formulaire du volet accueil concernant le séjour désiré.<br>2. Il sélectionne un hôtel parmi ceux se trouvant dans la ville de destination.<br>3. Il choisit les deux vols aller/retour pour son séjour.<br>4. Il remplit un formulaire pour valider le séjour (réservation de l'hôtel, du vol et le paiement). |
| **Scénario alternatif** | • Si le client omet une action obligatoire, la page web rappelle le client. |

**Tableau 5.1 :** La description textuelle de la réservation de séjour.

*b. Le diagramme de séquence*

Diagramme d'interaction qui se concentre sur l'ordre temporel des messages entre objets, il peut servir à illustrer un cas d'utilisation. La représentation se concentre sur l'expression des interactions. L'ordre d'envoi d'un message est déterminé par sa position sur l'axe vertical du diagramme, le temps s'écoule "de haut en bas" de cet axe. La disposition des objets sur l'axe horizontal n'a pas de conséquence pour la sémantique du diagramme.

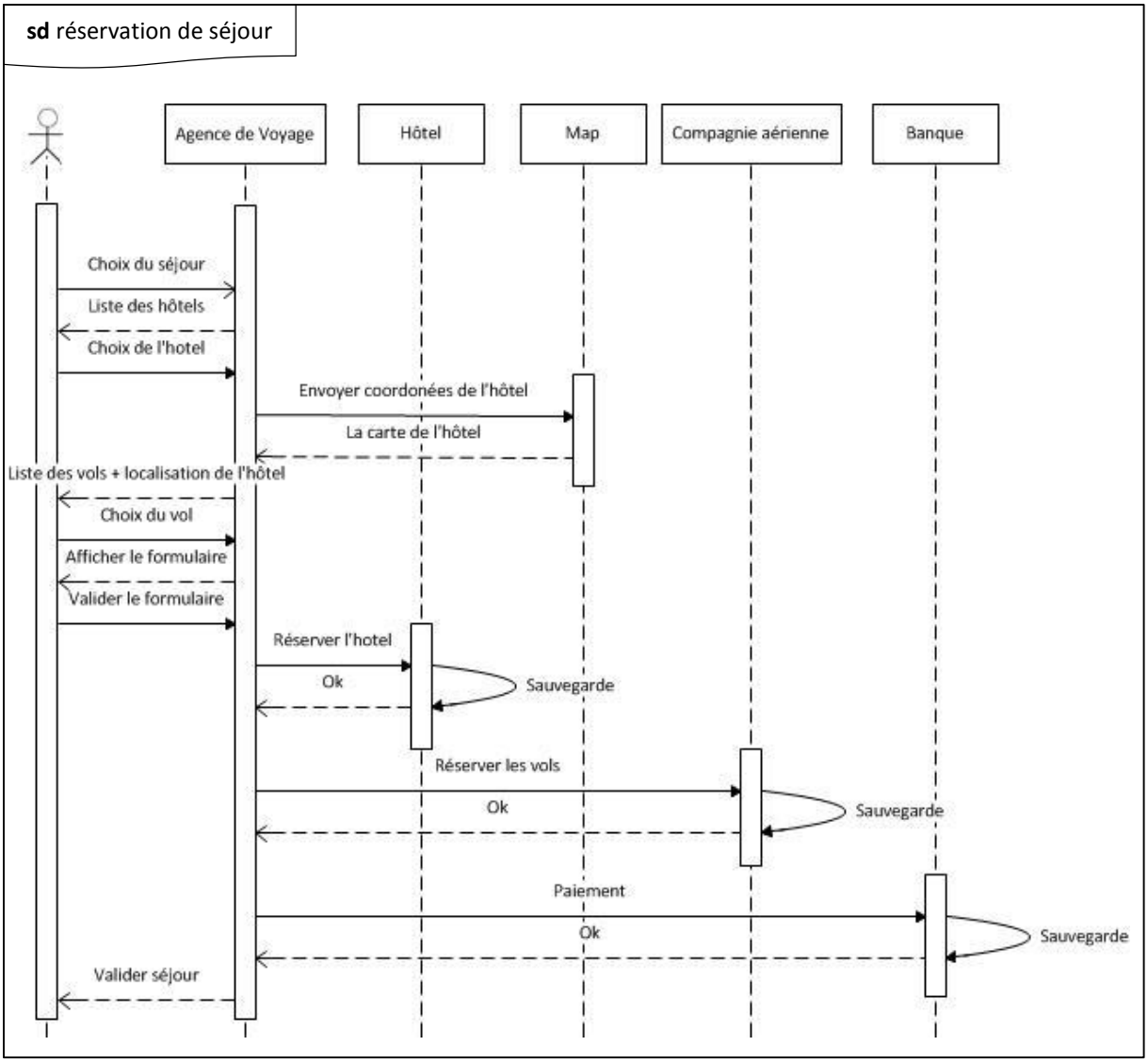



**Figure 5.6 :** Diagramme de séquence réservation de séjour.

### 5.5.2. Implémentation

#### 5.5.2.1. Les web services liés au mashup

Dans notre mashup, on a défini trois services web qui sont comme suit :

*a. Service web réservation d'hôtel*

Ce web service possède plusieurs partenaires (hôtels), ceci dit d'autres partenaires peuvent rejoindre ce web service par la suite. Au moment où ce service sera invoqué par l'agence de voyage, le service web sélectionne le partenaire souhaité afin de valider la réservation dans ce dernier. Chaque partenaire possède une base de données où un ensemble d'informations sera stocké, cet ensemble se compose du nom, prénom, date de début et date de fin, le résultat de la validation sera sous une forme booléenne (true ou false).

❖ **WSDL du web service :** Nous présentons le document WSDL du service web réservation d'hôtel dont l'adresse WSDL est la suivante :
http://localhost:8080/hotel_reservation/hotel_reservation_serviceService?wsdl

La représentation XML est comme suit :

```
<definitions xmlns:wsu="http://docs.oasis-open.org/.../oasis-200401-wss-
wssecurity-utility-
1.0.xsd" xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"xmlns:tns="http:/
/ws/" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.xml
soap.org/wsdl/" targetNamespace="http://ws/"name="hotel_reservation_serviceS
ervice">
      <types>
            <xsd:schema>
                  <xsd:import namespace="http://ws/" schemaLocation="http://l
                  ocalhost:8080/hotel_reservation/hotel_reservation_serviceSe
                  rvice?xsd=1"/>
            </xsd:schema>
      </types>
      <message name="ReserverHotel">...</message>
      <message name="ReserverHotelResponse">...</message>
      <portType name="hotel_reservation_service">...</portType>
      <binding name="hotel_reservation_servicePortBinding" type="tns:hotel_r
      eservation_service">...</binding>
      <service name="hotel_reservation_serviceService">
      <port name="hotel_reservation_servicePort" binding="tns:hotel_reservat
      ion_servicePortBinding">
            <soap:address location="http://localhost:8080/hotel_reservation/h
            otel_reservation_serviceService"/>
      </port>
</service>
</definitions>
```

**Liste 5.2** : WSDL du web service « réservation hotel ».

❖ **Communication SOAP :** Lors de l'invocation de ce service, la communication SOAP s'effectue en deux étapes, comme suit :

*SOAP Request :*

```
<?xml version="1.0" encoding="UTF-8"?>
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
    <S:Header/>
    <S:Body>
        <ns2:ReserverHotel xmlns:ns2="http://ws/">
            <IDHotel>HOT001</IDHotel>
            <nom>nom</nom>
            <prenom>prenom</prenom>
            <date_debut>2013-07-05</date_debut>
            <date_fin>2013-07-15</date_fin>
        </ns2:ReserverHotel>
    </S:Body>
</S:Envelope>
```

**Liste 5.3** : Requête SOAP « Réservation hôtel »

*SOAP Response :*

```
<?xml version="1.0" encoding="UTF-8"?>
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
    <S:Body>
        <ns2:ReserverHotelResponse xmlns:ns2="http://ws/">
            <return>true</return>
        </ns2:ReserverHotelResponse>
    </S:Body>
</S:Envelope>
```

**Liste 5.4** : Réponse SOAP « Réservation hôtel ».

*b. Service web réservation des vols*

Ce web service possède plusieurs partenaires (compagnies), ceci dit d'autres partenaires peuvent rejoindre ce web service par la suite. Au moment où ce service sera invoqué par l'agence de voyage, le service web sélectionne le partenaire souhaité afin de valider la réservation dans ce dernier. Chaque partenaire possède une base de données où un ensemble d'informations sera stocké, cet ensemble se compose du nom, prénom, sexe et date de naissance, le résultat de la validation sera sous une forme booléenne (**true** ou **false**).

❖ **WSDL du web service** : Nous présentons le document WSDL du service web réservation d'avion dont l'adresse WSDL est la suivante :

http://localhost:8080/vols_reservation/vols_reservation_serviceService?wsdl

La représentation XML est comme suit :

```
<definitions xmlns:wsu="http://docs.oasis-open.org/../oasis-200401-wss-
wssecurity-utility-
1.0.xsd" xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"xmlns:tns="http:/
/ws/" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.xml
soap.org/wsdl/" targetNamespace="http://ws/"name="vols_reservation_serviceSe
rvice">
      <types>
            <xsd:schema>
                  <xsd:import namespace="http://ws/" schemaLocation="http://l
                  ocalhost:8080/vols_reservation/vols_reservation_serviceServ
                  ice?xsd=1"/>
            </xsd:schema>
      </types>
      <message name="Reserver_vol">...</message>
      <message name="Reserver_volResponse">...</message>
      <portType name="vols_reservation_service">...</portType>
      <binding name="vols_reservation_servicePortBinding" type="tns:vols_res
      ervation_service">...</binding>
      <service name="vols_reservation_serviceService">
      <port name="vols_reservation_servicePort" binding="tns:vols_reservatio
      n_servicePortBinding">
            <soap:address location="http://localhost:8080/vols_reservation/vo
            ls_reservation_serviceService"/>
      </port>
      </service>
</definitions>
```

**Liste 5.5** : WSDL du web service « Réservation vols ».

❖ **Communication SOAP :** Lors de l'invocation de ce service, la communication SOAP s'effectue en deux étapes, comme suit :

*SOAP Request :*

```
<?xml version="1.0" encoding="UTF-8"?>
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
    <S:Header/>
    <S:Body>
        <ns2:Reserver_vol xmlns:ns2="http://ws/">
            <IDCompagnieAller>Air Algerie</IDCompagnieAller>
            <IDVolAller>VOL001</IDVolAller>
            <IDCompagnieRetour>Air France</IDCompagnieRetour>
            <IDVolRetour>VOL003</IDVolRetour>
            <nom>nom</nom>
            <prenom>prenom</prenom>
            <sexe>Masculin</sexe>
            <date_naissance>1989-01-01</date_naissance>
        </ns2:Reserver_vol>
    </S:Body>
```

**Liste 5.6** : Requête SOAP « Réservation vols »

*SOAP Response :*

```
<?xml version="1.0" encoding="UTF-8"?>
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
    <S:Body>
        <ns2:Reserver_volResponse xmlns:ns2="http://ws/">
            <return>True</return>
        </ns2:Reserver_volResponse>
    </S:Body>
</S:Envelope>
```

**Liste 5.7** : Réponse SOAP « Réservation vols ».

c. *Service web paiement*

Ce web service possède plusieurs partenaires (banques), ceci dit d'autres partenaires peuvent rejoindre ce web service par la suite. Au moment où ce service sera invoqué par l'agence de voyage, le service web sélectionne le partenaire souhaité afin de valider le paiement. Chaque partenaire possède une base de données où un ensemble d'informations sera stocké, cet ensemble se compose du nom, prénom, carte de crédit, numéro carte de crédit et date d'expiration de la carte de crédit, le résultat de la validation sera sous une forme booléenne (**true** ou **false**).

❖ **WSDL du web service :** Nous présentons le document WSDL du service web paiement dont l'adresse WSDL est la suivante :

http://localhost:8080/banque_paiement/banque_paiement_serviceService?wsdl

La représentation XML est comme suit :

```
<definitions xmlns:wsu="http://docs.oasis-open.org/.../oasis-200401-wss-
wssecurity-utility-
1.0.xsd" xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"xmlns:tns="http:/
/ws/" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.xml
soap.org/wsdl/" targetNamespace="http://ws/"name="banque_paiement_serviceSer
vice">
      <types>
            <xsd:schema>
                  <xsd:import namespace="http://ws/" schemaLocation="http://l
                  ocalhost:8080/banque_paiement/banque_paiement_serviceServic
                  e?xsd=1"/>
            </xsd:schema>
      </types>
      <message name="Paiement">...</message>
      <message name="PaiementResponse">...</message>
      <portType name="banque_paiement_service">...</portType>
      <binding name="banque_paiement_servicePortBinding" type="tns:banque_pa
      iement_service">...</binding>
      <service name="banque_paiement_serviceService">
            <port name="banque_paiement_servicePort" binding="tns:banque_paie
            ment_servicePortBinding">
                  <soap:address location="http://localhost:8080/banque_paieme
                  nt/banque_paiement_serviceService"/>
            </port>
      </service>
</definitions>
```

**Liste 5.8** : WSDL du web service « Paiement ».

❖ **Communication SOAP :** lors de l'invocation de ce service, la communication SOAP s'effectue en deux étapes, comme suit :

*SOAP Request :*

```
<?xml version="1.0" encoding="UTF-8"?>
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
    <S:Header/>
    <S:Body>
        <ns2:Paiement xmlns:ns2="http://ws/">
            <IDBanque>BNQ001</IDBanque>
            <nom>nom</nom>
            <prenom>prenom</prenom>
            <numCc>999999999999</numCc>
            <date_expire>2015-07</date_expire>
            <montant>12000</montant>
        </ns2:Paiement>
    </S:Body>
</S:Envelope>
```

**Liste 5.9** : Requête SOAP « Paiement ».

*SOAP Response* :

```
<?xml version="1.0" encoding="UTF-8"?>
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
    <S:Body>
        <ns2:PaiementResponse xmlns:ns2="http://ws/">
            <return>true</return>
        </ns2:PaiementResponse>
    </S:Body>
</S:Envelope>
```

**Liste 5.10** : Réponse SOAP « Paiement ».

### 5.5.2.2. Mashup de l'agence de voyage

Notre service web composite sollicite trois services web qui sont la réservation d'hôtel, la réservation du vol et le paiement, en plus de ces trois web services qu'on a déployé, on a intégré une API sous forme de service web. Cette API représente un ensemble de fonctions et de classes JavaScript qui permet de manipuler une carte dynamiquement au sein de notre site web.

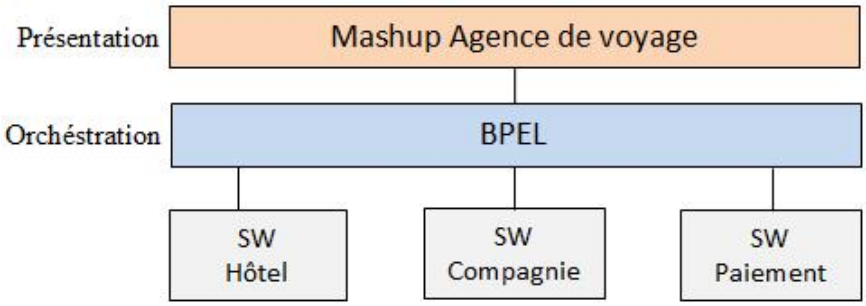



**Figure 5.7 :** Invocation et orchestration des trois services web.

Le déclenchement du processus d'exécution de notre mashup est dirigé par les interfaces, comme illustré dans la figure suivante :

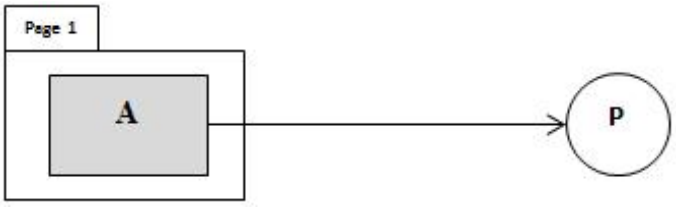



**Figure 5.8 :** Processus d'exécution dirigé par les interfaces.

❖ **WSDL du service composite :** Ce document de description de l'agence de voyage définit les différentes variables d'entrées qui englobent toutes les variables d'entrées des services web ainsi qu'une variable de sortie qui concatène les résultats des trois services qui seront invoqués.

À présent, nous présentons le document WSDL du service web composite agence de voyage :

La représentation XML est comme suit :

```
<definitions xmlns="http://schemas.xmlsoap.org/wsdl/" xmlns:plnk="http://docs.
oasis-
open.org/.../plnktype" xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"xmlns:tns=
"http://j2ee.netbeans.org/wsdl/BpelAgenceVoyage/WSDLAgenceVoayge" xmlns:xsd="h
ttp://www.w3.org/2001/XMLSchema"xmlns:soap="http://schemas.xmlsoap.org/wsdl/so
ap/" name="WSDLAgenceVoayge" targetNamespace="http://j2ee.netbeans.org/.../WSD
LAgenceVoayge">
      <types></types>
      <message name="getServicesResponse">
            <part name="result" type="xsd:string"></part>
      </message>
      <message name="getServicesRequest">
            <part name="IDHotel" type="xsd:string"></part>
            <part name="IDCompagnieAller" type="xsd:string"></part>
            <part name="IDVolAller" type="xsd:string"></part>
            <part name="IDCompagnieRetour" type="xsd:string"></part>
            <part name="IDVolRetour" type="xsd:string"></part>
            <part name="IDBanque" type="xsd:string"></part>
            <part name="nom" type="xsd:string"></part>
            <part name="prenom" type="xsd:string"></part>
            <part name="dateDebut" type="xsd:string"></part>
            <part name="dateFin" type="xsd:string"></part>
            <part name="sexe" type="xsd:string"></part>
            <part name="date_naissance" type="xsd:string"></part>
            <part name="numCc" type="xsd:string"></part>
            <part name="date_expire" type="xsd:string"></part>
            <part name="montant" type="xsd:int"></part>
      </message>
      <portType name="AgenceVoaygePortType">...</portType>
      <binding name="WSDLAgenceVoaygeBinding" type="tns:AgenceVoaygePortType">
      ...</binding>
      <service name="WSDLAgenceVoaygeService">...</service>
      <plnk:partnerLinkType name="WSDLAgenceVoayge">...</plnk:partnerLinkType>
</definitions>
```

**Liste 5.11** : WSDL du mashup « Réservation de séjour ».

❖ **Orchestration avec BPEL4WS :** Ce Web service de réservation de séjour peut être modélisé sous la forme d'un processus métier en combinant différents Web services élémentaires. Ainsi, à la réception de la demande de réservation, le processus métier invoque en parallèle le Web service réservation d'hôtel, celui de la réservation de vol et du paiement. afin que les différents services web collaborent entre eux, on a importé les WSDL de chaque service web qu'on souhaite invoquer.

```
<import namespace="http://ws/"
location="http://localhost:8080/hotel_reservation/hotel_reservation_servi
ceService?wsdl" importType="http://schemas.xmlsoap.org/wsdl/"/>

<import namespace="http://ws/"
location="http://localhost:8080/vols_reservation/vols_reservation_service
Service?wsdl" importType="http://schemas.xmlsoap.org/wsdl/"/>

<import namespace="http://ws/"
location="http://localhost:8080/banque_paiement/banque_paiement_serviceSe
rvice?wsdl" importType="http://schemas.xmlsoap.org/wsdl/"/>
```

**Liste 5.12** : Importation des WSDL de chaque service web par le processus métier.

Enfin. La description BPEL4WS correspondante se décompose en trois parties :

- **Définition des partenaires :**

```
<partnerLinks>

        <partnerLink name="hotelsReservationService"
xmlns:tns="http://enterprise.netbeans.org/bpel/hotel_reservation_serviceServ
iceWrapper" partnerLinkType="tns:hotel_reservation_serviceLinkType"
partnerRole="hotel_reservation_serviceRole"/>

        <partnerLink name="volsReservationService"
xmlns:tns="http://enterprise.netbeans.org/bpel/vols_reservation_serviceServi
ceWrapper" partnerLinkType="tns:vols_reservation_serviceLinkType"
partnerRole="vols_reservation_serviceRole"/>

        <partnerLink name="paiementService"
xmlns:tns="http://enterprise.netbeans.org/bpel/banque_paiement_serviceServic
eWrapper" partnerLinkType="tns:banque_paiement_serviceLinkType"
partnerRole="banque_paiement_serviceRole"/>

        <partnerLink name="AgenceVoyageMashups"
xmlns:tns="http://j2ee.netbeans.org/wsdl/BpelAgenceVoyage/WSDLAgenceVoaygе"
partnerLinkType="tns:WSDLAgenceVoaygе" myRole="AgenceVoaygePortTypeRole"/>

    </partnerLinks>
```

**Liste 5.13** : Définition des partenaires dans le processus métier.

- **Définition des types de données et des messages échangés :**

```
<variables>

        <variable name="PaiementOut" xmlns:tns="http://ws/"
messageType="tns:PaiementResponse"/>

        <variable name="PaiementIn" xmlns:tns="http://ws/"
messageType="tns:Paiement"/>

        <variable name="Reserver_volOut" xmlns:tns="http://ws/"
messageType="tns:Reserver_volResponse"/>

        <variable name="Reserver_volIn" xmlns:tns="http://ws/"
messageType="tns:Reserver_vol"/>

        <variable name="ReserverHotelOut" xmlns:tns="http://ws/"
messageType="tns:ReserverHotelResponse"/>

        <variable name="ReserverHotelIn" xmlns:tns="http://ws/"
messageType="tns:ReserverHotel"/>

        <variable name="resultFinal"
xmlns:tns="http://j2ee.netbeans.org/wsdl/BpelAgenceVoyage/WSDLAgenceVoayge"
messageType="tns:getServicesResponse"/>

        <variable name="ReceiveData"
xmlns:tns="http://j2ee.netbeans.org/wsdl/BpelAgenceVoyage/WSDLAgenceVoayge"
messageType="tns:getServicesRequest"/>

</variables>
```

**Liste 5.14** : Définition des variables de données dans le processus métier.

- **Définition du processus métier :**

*a. Assignation d'entrée :* Le schéma suivant représente le partage des données d'entrée sur les trois services web :

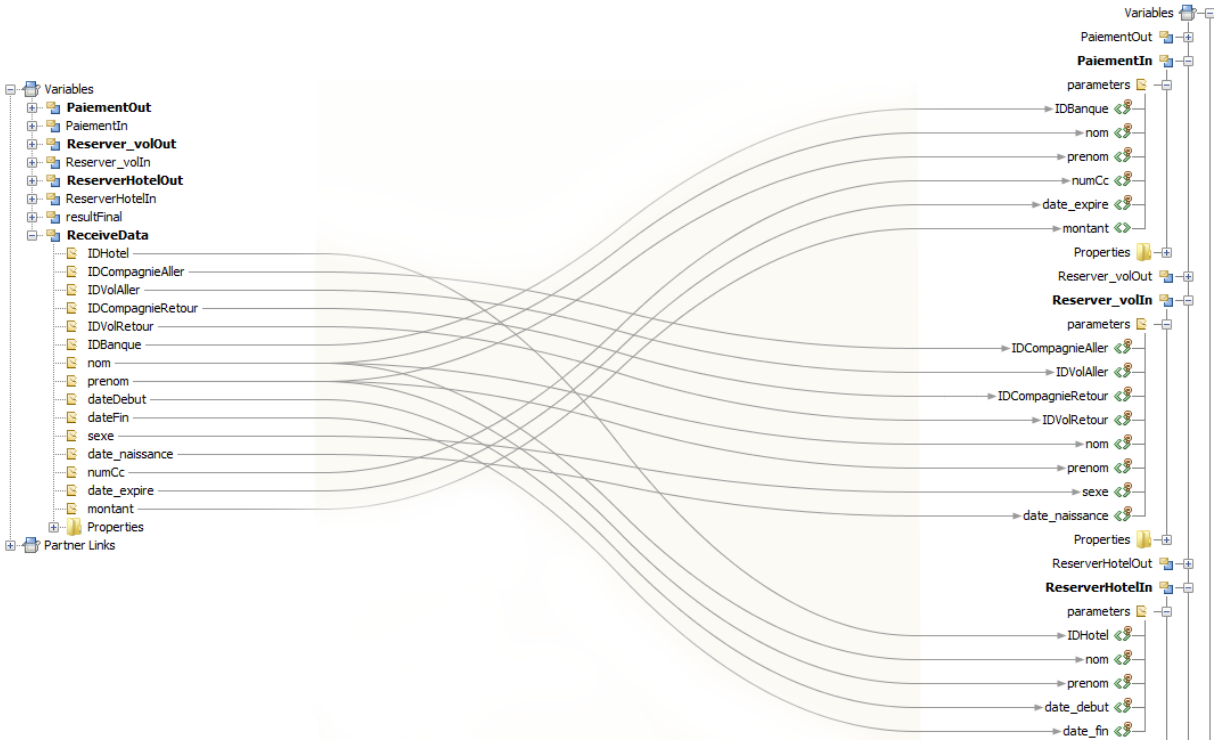



**Figure 5.9** : Assignation du flux d'entrée dans le processus métier.

*b. Invocation :* Elle est représentée par cette structure XML **:**

```
<sequence>

        <receive name="ReceiveData" createInstance="yes"
partnerLink="AgenceVoyageMashups" operation="getServices"
xmlns:tns="http://j2ee.netbeans.org/wsdl/BpelAgenceVoyage/WSDLAgenceVoay
ge" portType="tns:AgenceVoaygePortType" variable="ReceiveData"/>

        <invoke name="InvokeHotelService"
partnerLink="hotelsReservationService" operation="ReserverHotel"
xmlns:tns="http://ws/" portType="tns:hotel_reservation_service"
inputVariable="ReserverHotelIn" outputVariable="ReserverHotelOut"/>

        <invoke name="InvokeCompagnieService"
partnerLink="volsReservationService" operation="Reserver_vol"
xmlns:tns="http://ws/" portType="tns:vols_reservation_service"
inputVariable="Reserver_volIn" outputVariable="Reserver_volOut"/>

        <invoke name="InvokeBanque" partnerLink="paiementService"
operation="Paiement" xmlns:tns="http://ws/"
portType="tns:banque_paiement_service" inputVariable="PaiementIn"
outputVariable="PaiementOut"/>

        <reply name="resultFinal" partnerLink="AgenceVoyageMashups"
operation="getServices"
xmlns:tns="http://j2ee.netbeans.org/wsdl/BpelAgenceVoyage/WSDLAgenceVoay
ge" portType="tns:AgenceVoaygePortType" variable="resultFinal"/>

</sequence>
```

**Liste 5.15** : Définition des activités d'invocation dans le processus métier.

*c. Assignation de sortie :* Le schéma suivant représente la concaténation des résultats des trois services web comme une donnée de sortie :

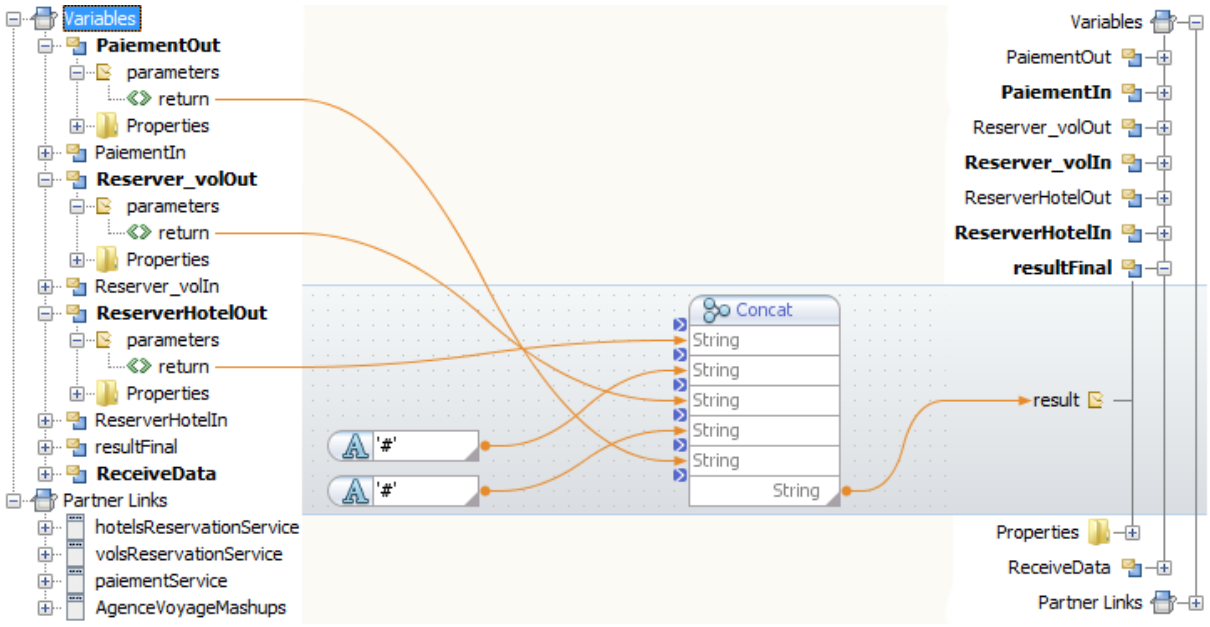



**Figure 5.10 :** Assignation des résultats des web services au flux de sortie du mashup.

❖ **Procédé BPEL4WS de l'agence de voyage :** Afin de mieux comprendre le modèle BPEL4WS, dans cette section nous allons donner un aperçu graphique du BPEL de notre mashup, La **figure 5.11** illustre l'exemple du procédé BPEL et décrivant un processus de traitement d'une réservation de séjour. Le contrôle de la composition est réalisé comme suit : L'agence de voyage commence l'interaction en utilisant l'activité «**ReceiveData**», ensuite les données seront dispatchées comme nous l'avons décrit précédemment dans la **figure 5.9** avec l'activité **« Assign1 »** pour chacun des services. Après l'invocation de ces services avec les activités «**InvokeHotelService**», «**InvokeCompagnieService**» et «**InvokeBanqueService**», les résultats de chacun d'eux seront concaténés comme mentionné dans la **figure 5.10** avec l'activité «**Assign2**». Enfin l'activité «**resultFinal**» transmet le résultat final à l'agence de voyage.

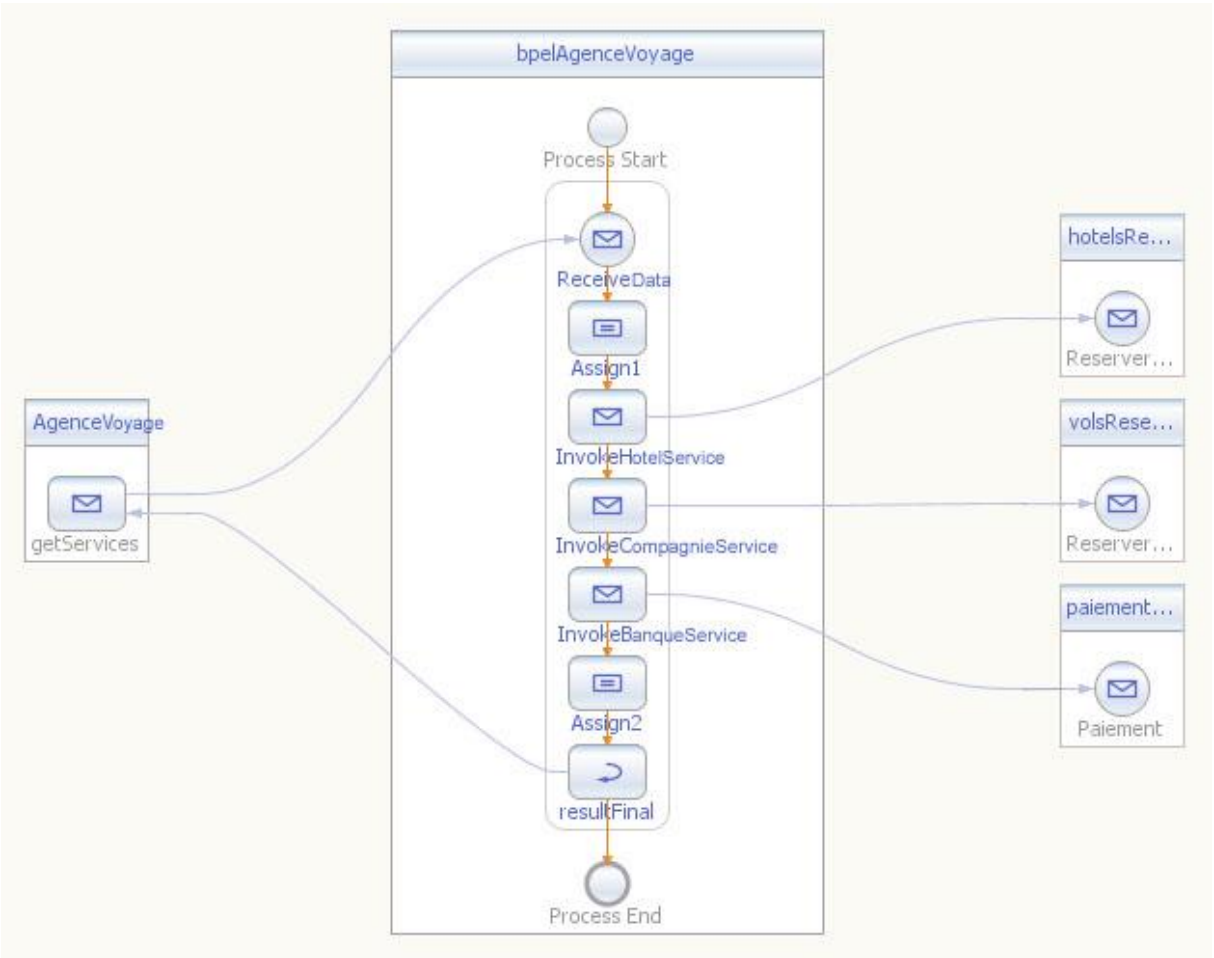



**Figure 5.11 :** Aperçu graphique du procédé BPEL4WS de l'agence de voyage.

❖ **Déploiement de l'application composite :** Le projet BpelAgenceVoyage définit l'orchestration de l'ensemble des services du mashup, dans le graphique ci-dessous, on peut voir que toute communication entre le BpelAgenceVoyage et les différents services web se fait à travers le protocole SOAP dont le transfert des messages se fait le plus souvent à l'aide du protocole HTTP.

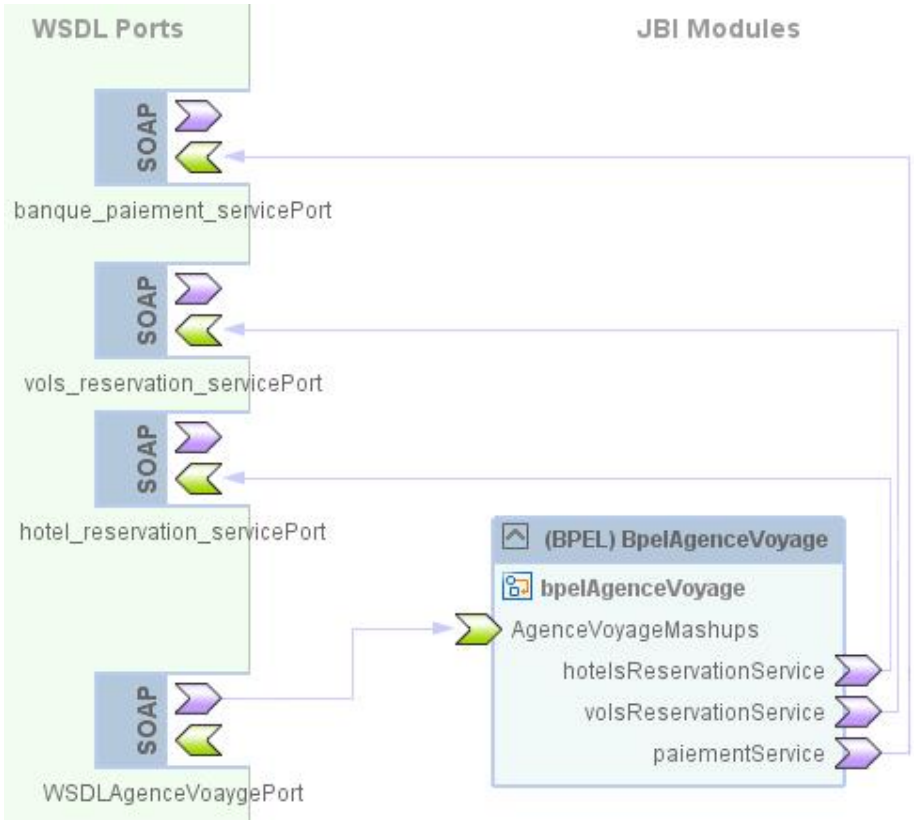



**Figure 5.12 :** Aperçu graphique du déploiement du mashup.

Après le déploiement de tous ces composants, un lien WSDL sera généré pour pouvoir mettre en œuvre notre mashup dont l'adresse est la suivante :
http://localhost:9080/WSDLAgenceVoaygeService/WSDLAgenceVoaygePort?wsdl

❖ **Communication SOAP :** Lors de l'invocation du service composite, la communication SOAP s'effectue en deux étapes, comme suit :

*SOAP Request :*

```
<soapenv:Envelope xsi:schemaLocation="
http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:wsd="http://j2ee.netbeans.org/wsdl/BpelAgenceVoyage/WSDLAgenceVo
ayge">
  <soapenv:Body>
    <wsd:getServices>
      <IDHotel>HOT001</IDHotel>
      <IDCompagnieAller>Air Algerie</IDCompagnieAller>
      <IDVolAller>VOL001</IDVolAller>
      <IDCompagnieRetour>Air France</IDCompagnieRetour>
      <IDVolRetour>VOL003</IDVolRetour>
      <IDBanque>BNQ001</IDBanque>
      <nom>nom</nom>
      <prenom>prenom</prenom>
      <dateDebut>2013-07-05</dateDebut>
      <dateFin>2013-7-15</dateFin>
      <sexe>Masculin</sexe>
      <date_naissance>1989-01-01/</date_naissance>
      <numCc>999999999999</numCc>
      <date_expire>2015-01</date_expire>
      <montant>20000</montant>
    </wsd:getServices>
  </soapenv:Body>
</soapenv:Envelope>
```

**Liste 5.16** : Requête SOAP de la réservation de séjour.

*SOAP Response :*

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<SOAP-ENV:Envelope xmlns:SOAP-
ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://schemas.xmlsoap.org/soap/envelope/
http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Body>
    <m:getServicesResponse
xmlns:m="http://j2ee.netbeans.org/wsdl/BpelAgenceVoyage/WSDLAgenceVoay
ge">
      <result>true#true#true</result>
    </m:getServicesResponse>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

**Liste 5.17** : Réponse SOAP de la réservation de séjour.

❖ **Invocation du mashup par le site web de l'agence de voyage :** Pour mettre en exécution et valider le séjour souhaité d'un client, le site web de l'agence de voyage fait appel au service composite à travers le code **jsp** dans netbeans suivant :

```
bpelagencevoyage.wsdlagencevoayge.WSDLAgenceVoaygeService service = new
bpelagencevoyage.wsdlagencevoayge.WSDLAgenceVoaygeService();

bpelagencevoyage.wsdlagencevoayge.AgenceVoaygePortType port =
service.getWSDLAgenceVoaygePort();

String result = port.getServices(id_h, id_cA, id_vR, id_cR, id_vR, id_b,
nom, prenom, dd, df, sexe, dn, numCc, de, prixT);
```

**Liste 5.18** : Code d'invocation du mashup de puis le site web de l'agence de voyage.

### 5.5.3. Exécution de l'application web

L'interface graphique s'avère sans aucun doute la partie la plus cruciale dans une application web. Elle contribue à la construction de la première impression qu'à l'internaute du système. En effet, elle constitue un critère qui peut créer la différence entre les applications web bien qu'elles aient les mêmes fonctionnalités.

L'exécution de notre application web pour la réservation de séjour s'effectue en quatre étapes :

### 5.5.3.1. Interface d'accueil

Quand un client accède à notre application web, une page d'accueil lui sera affiché où il devra remplir quelques informations essentielles pour pouvoir rechercher un séjour. Cette interface est illustrée dans la **figure 5.13** suivante :

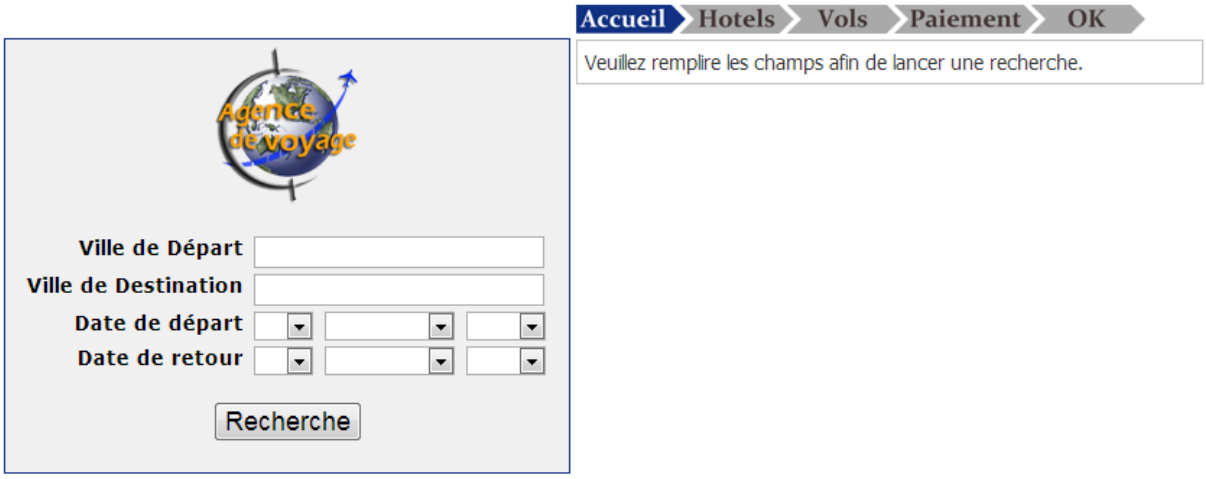



**Figure 5.13 :** Interface d'accueil et de recherche de séjour.

### 5.5.3.2. Interface de sélection d'un hôtel

Dans cette page, le client va choisir un hôtel situé dans la ville de destination qu'il a mentionnée dans l'interface précédente, l'interface est illustrée dans la **figure 5.14** suivante :

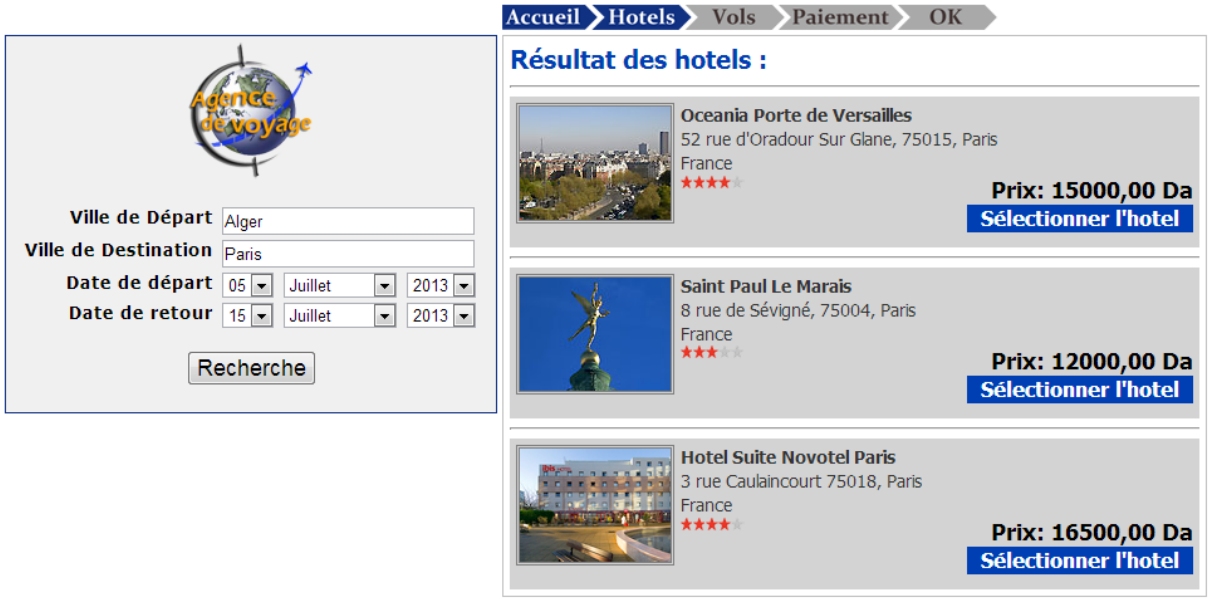



**Figure 5.14 :** Interface du choix de l'hôtel.

### 5.5.3.3. Interface de sélection des vols

Dans cette page, l'hôtel choisi dans l'interface précédente sera localisé sur une *map*, le client doit sélectionner le vol 'aller' et le vol 'retour' concernant le séjour souhaité, l'interface est illustrée dans la **figure 5.15** suivante :

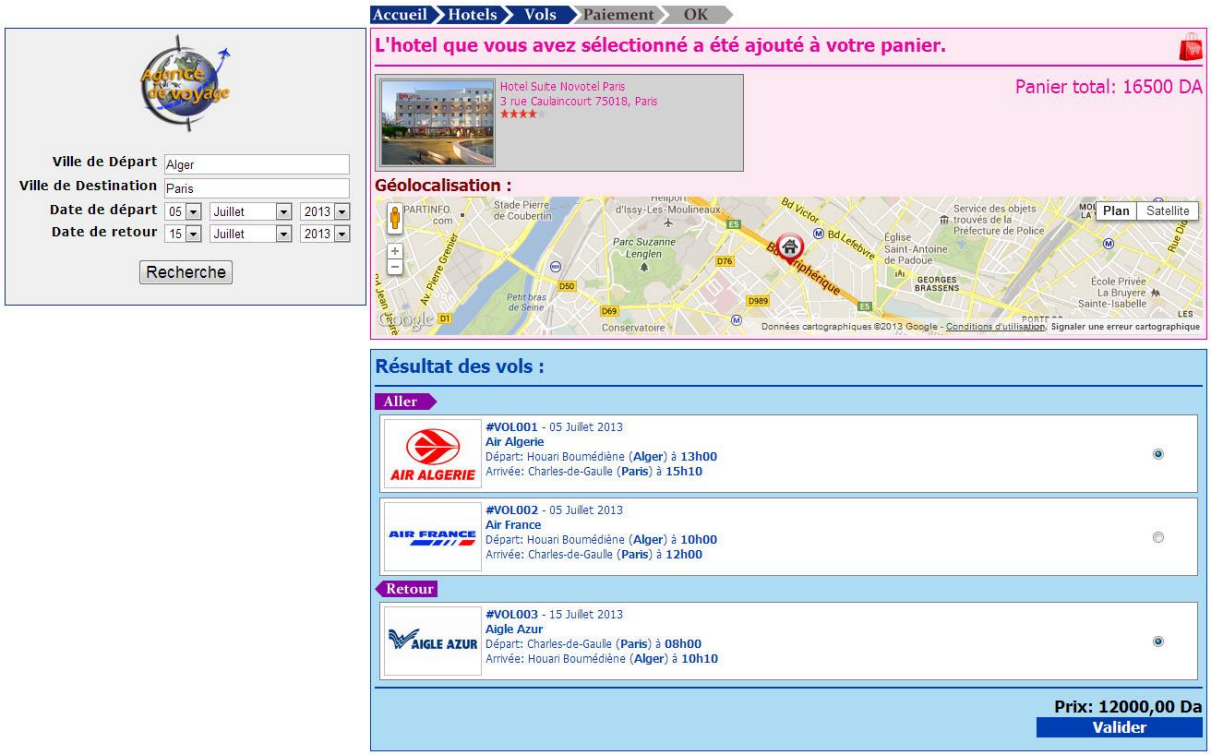



**Figure 5.15 :** Interface de localisation de l'hôtel et du choix des vols.

### 5.5.3.4. Interface de paiement et de validation du séjour

Dans cette page, un récapitulatif de la commande de réservation de l'hôtel et des vols est affiché pour le client, ainsi il devra remplir tous les champs du formulaire y compris les coordonnées de la carte de crédit pour effectuer le paiement afin de valider le séjour.

L'interface est illustrée dans la **figure 5.16** suivante :

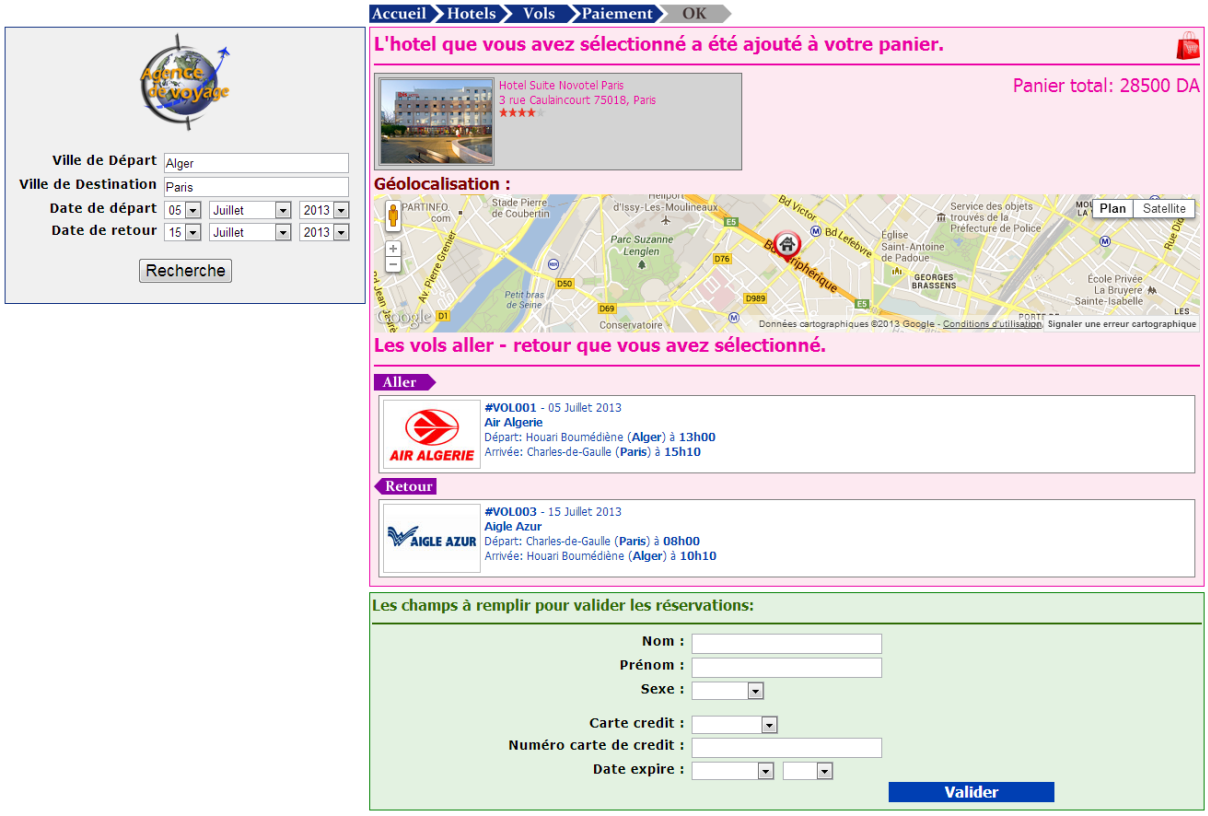
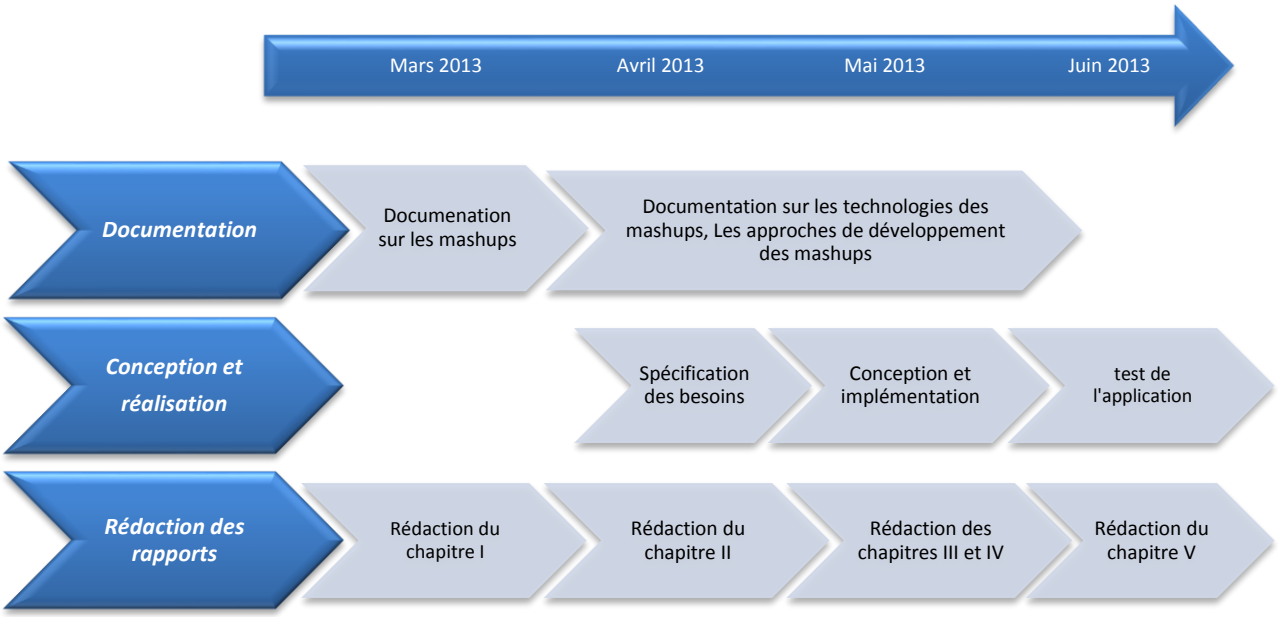



**Figure 5.16 :** Interface de paiement et de validation du séjour.

## 5.6. Chronogramme




## 5.7. Conclusion

Dans ce chapitre, nous avons présenté l'environnement de développement matériel et logiciel avec lesquels ce projet a été réalisé. Nous avons présenté aussi une vue de l'application finale via quelques imprimés d'écrans ainsi que le chronogramme des tâches accomplies durant ce travail.

# CONCLUSION GÉNÉRALE

L'objectif que nous nous sommes assigné était d'explorer et de proposer un *framework* et une approche de composition de services pour l'orchestration des services web de notre mashup orienté présentation.

En effet cinq chapitres ont constitué l'ossature de ce travail.

Le premier chapitre a porté sur la description générale de notre thème où l'on a défini le terme mashup, ses types, ses architectures et les outils utilisés pour sa création.

Le deuxième chapitre a été consacré à l'étude des technologies des mashups. Parmi les lesquelles nous avons retenu la technologie orientée présentation.

Dans le troisième nous avons examiné les approches de développement des mashups. À ce niveau nous avons mis l'accent sur la création manuelle des mashups et ainsi le *framework* que l'on suivra pour réaliser notre mashup.

Dans le quatrième chapitre nous avons réalisé une caractérisation des modèles et canevas d'orchestration de services. À cet effet nous avons opté pour l'approche d'orchestration avec le standard BPEL4WS.

Dans le cinquième et dernier chapitre, nous allons présenter une réalisation de notre proposition à l'aide de l'exemple des agences de voyage, ceci en décrivant l'environnement de travail, les étapes de notre réalisation allant de la conception jusqu'à l'exécution. Ainsi que l'exposition de quelques interfaces homme machine qui concordent avec les fonctionnalités de notre application web.

Notre solution a été validée à travers la mise en œuvre d'une application web. En effet, nous avons réussi dans un premier temps à implémenter le processus d'orchestration BPEL. Dans un second temps, nous avons pu montrer la faisabilité et l'efficacité des différents mécanismes utilisés dans notre mashup.

Notre mémoire se termine par un aperçu sur quelques domaines émergents pour l'orchestration des services web, comme la sécurité.

La sécurité est également une préoccupation pour l'orchestration des services web et des services web en général. La sécurité est essentielle parce que nous avons affaire maintenant à des expositions d'interfaces sur des protocoles un peu moins sécurisé.

Il y a un certain nombre de normes en cours de discussion pour la sécurité des services Web, comme les signatures numériques XML, le cryptage, SAML et WS-Security. Ces normes espèrent remplir les exigences de sécurité des services Web spécifiques pour l'authentification et l'autorisation des utilisateurs, et pour sécuriser le message XML lui-même.

Toutefois, les standards d'orchestration présentés dans ce mémoire ne proposent pas une aide directe pour la sécurité. Par exemple, comment les rôles sont-ils définis pour chaque partenaire se rapportant ou exploitant les standards d'authentification et d'autorisation existants? À ce stade, il est difficile de savoir comment ces standards établissent un rapport, mais il y a une opportunité pour l'industrie de travailler sur l'intersection de ces deux tâches.

Aucune œuvre faite de main d'homme ne peut prétendre à la perfection. Nous avons donc conscience de certaines imperfections de notre travail. Nos recherches ultérieures et les contributions d'autres chercheurs veilleront à parfaire le contenu de notre mémoire.

# Bibliographie

**[1].** Akihiro Fujii, “Expanding Use of Web API, Vast Potential of Mashup”, Juillet 2010.

**[2].** Darlene Fichter, “What Is a Mashup?”, Darlene Fichter University of Saskatchewan Library, 26 Octobre, 2007.

**[3].** Miodrag Ivkovic et Dusanka Milanov, “Affiliate Internet Marketing”: Concept and Application Analysis, Education and Management Technology (ICEMT), 2-4 Novembre. 2010. Cairo.

**[4].** Mylène Leitzelman, “recherche de combinaisons de web services 2.0 pour outiller le veilleur”, 02 Mai 2011.

**[5].** Nazia Ilyas Baig et Gresha Bhatia, International Conference & Workshop on Recent Trends in Technology, Proceedings published in International Journal of Computer Applications, 2012.

**[6].** Nazia Ilyas Baig and Gresha Bhatia. Article: An introduction to Mashups and implementing Affiliate Marketing using Mashups, ICWET, Mars 2012. Published by Foundation of Computer Science, New York, USA.

**[7].** https://developers.google.com/mashup-editor/

**[8].** http://pipes.yahoo.com/pipes/

**[9].** www.schmapplets.com/

**[10].** Tools, Builders and Samples, http://sites.google.com/site/ajaxmashup2/Mashup Tutorials.

**[11].** B. Medjahed, A. Bouguettaya et A. K. Elmagarmid, “Composing Web services on the Semantic Web”, The VLDB Journal, Novembre 2003.

**[12].** J.Jeffrey Hanson, Mashups: “Strategies for the Modern Enterprise”, Addison-Wesley Professional, 05 Mai 2009.

**[13].** http://pro.01net.com/editorial/400009/laissez-vous-tenter-par-les-mashups/

**[14].** https://developer.mozilla.org/fr/docs/DOM

**[15].** http://www.saas.com/ta/hp.jsp

**[16].** http://json.org/

**[17].** http://www.futura-sciences.com/fr/definition/t/high-tech-1/d/ajax_3998/

**[18].** http://www.w3.org/TR/DOM-Level-3-Core/introduction.html

**[19].** http://www.xml.com/pub/a/98/10/guide0.html

**[20].** http://www.jmdoudoux.fr/java/dej/chap-xslt.htm

**[21].** http://searchsoa.techtarget.com/definition/REST

**[22].** D. Merrill, Mashups: "The new breed of Web app", Aout 2006. http://www.ibm.com/developerworks/library/x-mashups.html

**[23].** P. Brusilovsky, A. Kobsa et W. Nejdl, "The Adaptive Web, Methods and Strategies of Web Personalization", volume 4321 of Lecture Notes in Computer Science, 2007.

**[24].** Thomas Fischer, Fedor Bakalov et Andreas Nauerz, "An Overview of Current Approaches to Mashup Generation", Proceedings of the International Workshop on Knowledge Services and Mashups, Mars 2009, Solothurn, Switzerland.

**[25].** http://www-01.ibm.com/software/webservers/smash

**[26].** http://www.bungeelabs.com/

**[27].** http://code.google.com/gme/

**[28].** M. Sabbouh, J. Higginson, S. Semy, et D. Gagne, "Web Mashup Scripting Language". In WWW07: Proceedings of the 16th Intl Conf on World Wide Web, pages 1305–1306, 2007, New York, USA.

**[29].** E. Rahm, A. Thor et D. Aumueller, "Dynamic Fusion of Web Data". In XSym, pages 14–16, 2007.

**[30].** A. Thor, D. Aumueller et E. Rahm, "Data Integration Support for Mashups", Sixth Int Workshop on Information Integration on the Web, IIWeb, 2007, Vancouver, Canada.

**[31].** http://ws02.org/projects/mashup

**[32].** www.strikeiron.com/tools/tools_soaexpress.aspx

**[33].** www.extensio.com.

**[34].** A. Khizhnyak, "Apatar Connector Guide", Aout 2008. http://www.apatarforge.org/wiki/display/GUI/Apatar+Connector+Guides

**[35].** D. E. Simmen, M. Altinel, V. Markl, S.Padmanabhan, et A. Singh, "Damia: data mashups for intranet applications". In SIGMOD Conference, pages 1171–1182. ACM, 2008.

**[36].** J. Wong et J. I. Hong, "Making Mashups with Marmite: Towards End-User Programming for the Web". In Mary Beth Rosson et D. J. G, CHI, pages 1435–1444. ACM, 2007.

**[37].** Z. Maraikar, A. Lazovik, et F. Arbab, "Building Mashups for The Enterprise with SABRE". In Int. Conf on Service-Oriented Computing (ICSOC-08), 2008.

**[38].** http://www.jackbe.com, http://www.popfly.com/, http://openkapow.com/, http://pipes.yahoo.com/pipes/, http://www.protosw.com/, http://www.metafy.com/products/anthracite, http://www01.ibm.com/software/lotus/products/mashups/

**[39].** A. Sugiura et Y. Koseki, "Internet Scrapbook: Automating Web Browsing Tasks by Demonstration". In ACM Symp, on User Interface Software and Technology, pages 9-18, 1998.

**[40].** http://www.dapper.net/

**[41].** R. Tuchinda, P. A. Szekely et C. A. Knoblock, "Building Mashups by example". In J. M. Bradshaw, H. Lieberman et S. Staab, Intelligent User Interfaces, pages 139–148. ACM, 2008.

**[42].** D. F. Huynh, R. C. Miller et D. R. Karger, "Potluck: Data Mash-Up Tool for Casual Users". In ISWC/ASWC, volume 4825 of Lecture Notes in Computer Science, pages 239-252. Automne 2007.

**[43].** M. P. Carlson, A. H.H. Ngu, R. Podorozhny et L. Zeng, "Automatic Mash Up of Composite Applications". In Int. Conf on Service-Oriented Computing (ICSOC-08), 2008.

**[44].** T. Fischer, F. Bakalov et A. Nauerz, "Towards an Automatic Service Composition for Generation of User-Sensitive Mashups", Proceedings of the 16th Workshop on Adaptivity and User Modeling in Interactive Systems, Octobre 2008, Würzburg, Germany

**[45].** M. Shevertalov et S. Mancoridis, "A Case Study on the Automatic Composition of Network Application Mashups". In ASE, pages 359–362. IEEE, 2008.

**[46].** R. Ennals et D. Gay, "User-Friendly Functional Programming for Web Mashups". In R. Hinze et N. Ramsey, editors, ICFP, pages 223–234. ACM, 2007.

**[47].** http://simile.mit.edu/piggy-bank/

**[48].** OASIS. Web Services Business Process Execution Language Version 2.0 (Avril 2007). http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html

**[49].** Pautasso, C: BPEL for REST. In: Dumas, M., Reichert, M., Shan, M.-C. (eds.) BPM 2008. LNCS, vol. 5240, pp. 278–293, 10 Mars 2009.

**[50].** van Lessen, T. Leymann, F. Mietzner, R. Nitzsche, J. Schleicher, "D. A Management Framework for WS-BPEL". In: ECoWS 2008, Dublin.

**[51].** Curbera, F. Duftler, M.J Khalaf, R. Lovell, "D: Bite: Workflow Composition for the Web". In: Krämer, B.J. Lin, K-J. Narasimhan, P. (eds.) ICSOC 2007. LNCS, vol. 4749, pp. 94–106. 2007, Springer, Heidelberg.

**[52].** Maximilien, E.M. Ranabahu, A. Gomadam, "K: An Online Platform for Web APIs and Service Mashups". Internet Computing 32–43, 2008.

**[53].** Braga, D, Ceri, S, Daniel, F, Martinenghi, "D: Optimization of Multi-Domain Queries on the Web". In: VLDB 2008, pp. 562–573, 2008, Auckland.

**[54].** Daniel F, Yu J, Benatallah B, Casati F et Matera M., Saint-Paul R: "Understanding UI Integration", A Survey of Problems, Technologies, and Opportunities. IEEE Internet Computing, 59–66, Mai 2007.

**[55].** Microsoft Corporation. Smart Client-Composite UI Application Block, Decembre 2005. http://msdn.microsoft.com/en-us/library/aa480450.aspx

**[56].** The Eclipse Foundation. Rich Client Platform, Octobre 2008. http://wiki.eclipse.org/index.php/RCP

**[57].** Sun Microsystems. JSR-000168 Portlet Specification, Octobre 2003. http://jcp.org/aboutJava/communityprocess/final/jsr168/

**[58].** OASIS. Web Services for Remote Portlets Aout 2003. http://www.oasisopen.org/committees/wsrp

**[59].** Acerbis R, Bongio A, Brambilla M, Butti S, Ceri S, Fraternali P: "Web Applications Design and Development with WebML and WebRatio 5.0." TOOLS (46), 392–411, 2008.

**[60].** Gómez J, Bia A, Parraga A: "Tool Support for Model-Driven Development of Web Applications". In: Ngu A.H.H, Kitsuregawa M, Neuhold E.J, Chung J-Y, Sheng Q.Z. (eds.) WISE 2005. LNCS, vol. 3806, pp. 721–730, 2005, Springer, Heidelberg.

**[61].** Yu J, Benatallah B, Casati F, Daniel F: "Understanding Mashup Development and its Differences with Traditional Integration. Internet Computing", 44–52, 2008.

**[62].** http://popflyteam.spaces.live.com, MS Popfly a été intetrompu depuis le 24 Aout 2009.

**[63].** http://mashmaker.intel.com/web

**[64].** Taverna workflow system. http://www.taverna.org.uk/.

**[65].** http://code.google.com/intl/it-IT/appengine/.

**[66].** http://training.figleaf.com/tutorials/Presto.cfm.

**[67].** Taverna screenshots are taken from the Taverna web site and user manual.

**[68].** myExperiment Virtual Research Environment. http://www.myexperiment.org/

**[69].** G. Alonso, F. Casati, H. Kuno et H. Machiraju, Web Services, Concepts, Architectures and Applications. Springer Verlag, 2003.

**[70].** OASIS. Web Services Business Process Execution Language. Specification disponible depuis Avril 2007. http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html

**[71].** B. Benatallah, Q. Z. Sheng et M. Dumas, “The self-serv environment for web services composition”. Internet Computing, IEEE, 40-48, 2003.

**[72].** Fabio Casati, Ski Ilnicki, Lijie Jin, Vasudev Krishnamoorthy et Ming chien Shan, “Adaptive and dynamic service composition in eflow”. HPL-2000-39 Technical Report, Mars 2000.

**[73].** M. Blow, Y. Goland, M. Kloppmann, F. Leymann, G. Pfau, D. Roller et M. Rowley, “BPELJ: BPEL for Java”. A Joint White Paper by BEA and IBM, Mars 2004.

**[74].** Cesare Pautasso, Thomas Heinis et Gustavo Alonso, “Jopera: Autonomic service orchestration”, IEEE Data Engineering Bulletin, Septembre 2006.

**[75].** F. Leymann, “Web Service Flow Language (WSFL 1.0)”, IBM, Specification disponible depuis Mai 2001.

**[76].** J. Estublier, S. Dami et M. Amiour, “APEL: A graphical yet executable formalism for process modeling”. Automated Software Engineering: An International Journal, 61-96, Janvier 1998.

**[77].** Dirk Wodtke, Jeanine Weisenfels, Gerhard Weikum et Angelika Kotz Dittrich, “The mentor project: Steps toward enterprise-wide workflow management”. In Proceedings of the International Conference in Data Engineering, pages 556–565, 1996.

**[78].** Reisig Wolfgang et Rozenberg Grzegorz, Lectures on Petri Nets I: Basic Models, volume 1491 of Lecture Notes in Computer Science. 1998.

**[79].** R. Milner, J. Parrow et D. Walker, “A calculus of mobile processes”. Information and Computation, 1-40, Septembre 1992.

**[80].** Massimo Mecella, Francesco Presicce et Barbara Pernici, “Modeling e-service orchestration through petri nets”. In A. Buchmann et al, editor, Proceedings of the International VLDB Workshop on Tehcnologies for E-Services, pages 38-47, 2002.

**[81].** W. M. P. van der Aalst et Ter A. H. M. Hofstede, “YAWL: Yet another workflow language”. Information Systems, 245-275, 2005.

**[82].** Xinguo Deng, Ziyu Lin, Weiqing Cheng, Ruliang Xiao, Lina Fang et Ling Li, “Modeling web service choreography and orchestration with colored petri nets”. Eighth ACIS International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing, 838-843, Aout 2007.

**[83].** S. Thatte, “XLANG: web services for business process design”. Microsoft, Specification disponible depuis Juin 2001, http://www.gotdotnet.com/team/xmlwsspecs/xlangc/default.htm

**[84].** http://www-01.ibm.com/software/lotus/products/mashups/

**[85].** Massimiliano Colombo, Elisabetta Di Nitto et Marco Mauri, "Scene: A service composition execution environment supporting dynamic changes disciplined through rules". In Springer Berlin/Heidelberg, editor, Service-Oriented Computing-ICSOC 2006, volume 4294/2006 of Lecture Notes in Computer Science, pages 191-202. Springer Berlin/Heidelberg, Novembre 2006.

**[86].** J. Estublier, I. Dieng, E. Simon et G. Vega, "Flexible composite and automatic component selection for service-based applications". In Proceedings of 4th International Conference on Evaluation of Novel Approaches to Software Engineering, 2009.

**[87].** Michael Adams, Arthur H.M. ter Hofstede, Wil M.P. van der Aalst et David Edmond, "Dynamic, extensible and context-aware exception handling for workflow". In Proceedings of the 15th International Conference on Cooperative Information Systems (CoopIS 2007), 2007.

**[88].** OASIS. Web services security: Soap message security 1.1. Specification disponible depuis Février 2006 http://www.oasis-open.org/committees/download.php/16790/wssv1.1-spec-os SOAPMessageSecurity.pdf

**[89].** OASIS. Web services atomic transaction (ws-atomictransaction) version 1.1. Specification disponible depuis juillet 2007 http://docs.oasis-open.org/ws-tx/wstx-wsat-1.1-specerrata- os.pdf

**[90].** Paulo F. Pires, Mário R. F. Benevides et Marta Mattoso, "Building reliable web services compositions". In International Workshop Web Services Research, Standardization, and Deployment, pages 59-72. Springer-Verlag, 2003.

**[91].** G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. Lopes, J.-M. Loingtier et J. Irwin, "Aspect-oriented programming", In ECOOP'97, Object-Oriented Programming, 1997.

**[92].** Anis Charfi et Mira Mezini, "Aspect-oriented web service composition with AO4BPEL". In European Conference on Web Services, pages 168-182, 2004.

**[93].** Carine Courbis et Anthony Finkelstein, "Towards an aspect weaving bpel engine". In Y. Coady and D. H. Lorenz, editors, the Third AOSD Workshop on Aspects, Components, and Patterns for Infrastructure Software, Mars 2004, Lancaster, Grande-bretagne.

**[94].** M. Kloppmann, D. Koening, F. Leymann, G. Pfau, A. Rickayzen, C. von Riegen, P. Schmidt et I. Trickovic, "WS-BPEL extension for people, BPEL4People". A Joint White Paper by IBM and SAP, juillet 2005.

**[95].** Quan Z. Sheng, Boualem Benatallah, Marlon Dumas, and Eileen Oi, Yan Mak, "Self-serv: a platform for rapid composition of web services in a peer-to-peer environment". In VLDB '02: Proceedings.

**[96].** Gabriel Rodrigo PEDRAZA FERREIRA, "FOCAS: un canevas extensible pour la construction d'applications orientées procédé", these, 12 Novembre 2009, Université JOSEPH FOURIER - GRENOBLE I.

**[97].** James McGovern, Sameer Tyagi, Michael Stevens et Sunil Mathew, "Java Web Services Architecture", IBM Developer Book, juillet 2003

**[98].** Thomas Erl, "Service-Oriented Architecture: Concepts, Technology and Design, Prentice Hall PTR", ISBN 0-13-185858-0, 04 Aout 2005.

**[99].** Frank Leymann, "Web Services-Distributed Applications without Limits, Business, Technology and Web", 2003, Leipzig.

**[100].** Stefan Link, Fabian Jakobs, Ludwig Neer et Sebastian Abeck, "Architecture of and Migration to SOA's Presentation Layer", Cooperation & Management, Université de Karlsruhe, 2006.

**[101].** Frank Leymann, Dieter Roller et M-T. Schmidt, "Web Services and business process management". In: IBM Systems Journal, 2002.

**[102].** Ali Arsanjani, "Service-Oriented Modeling and Architecture", IBM developer works, 2004. http://www-128.ibm.com/developerworks/webservices/library/ws-soa-design1/

**[103].** Kishore Channabasavaiah et Kerrie Holley, "Migrating to a service-oriented architecture", White Paper IBM, Avril 2004.

**[104].** Bryan Caste, "Introduction to Web Services for Remote Portlets", IBM Developerworks, 2005, http://www-128.ibm.com/developerworks/webservices/library/ws-wsrp/

**[105].** Wolfgang Beinhauer, Thomas Schlegel, "User Interfaces for Service Oriented Architectures", Juillet 2005.

**[106].** Jeff Linwood, Dave Winter: "Building Portals with the Java Portlet API", ISBN: 1-59059-284-0, 2004.

**[107].** Asif Akram, Dharmesh Chohan, Xiao Dong Wang, Xiaobo Yang et Rob Allan, "A Service Oriented Architecture for Portals Using Portlets CCLRC e-Science Centre", CCLRC Daresbury Laboratory Warrington WA4 4AD, UK.

**[108].** Asif Akram, Rob Allan et Rob Crouchley, "WSRP Reincarnation of Service Oriented Architecture", CCLRC Daresbury Laboratory, e-Science Centre of Excellence, University of Lancaster, UK.

**[109].** Arthur Ryman, "Understanding Web Services", Juillet 2003, IBM Technical Article. http://www-128.ibm.com/developerworks/websphere/library/techarticles/0307_ryman/ryman.html

**[110].** Wilhelm Hasselbring, Ralf Reussner, "The Dublo Architecture Pattern for Smooth Migration of Business Information Systems: An experience report", Proceedings of 26th International Conference on Software Engineering (ICSE 2004), IEEE Computer Society Press, Mai 2004.

**[111].** Grace Lewis, Ed Morris, Liam O'Brien, Dennis Smith et Lutz Wrage, "SMART: The Service-Oriented Migration and Reuse Technique", Technical Note, Septembre 2005,

**[112].** Jesse James Garret, "A New Approach to Web Applications", Technical Essay, 2005.

**[113].** Jesse James Garret, "A new approach to Web applications", essay from adaptive path, Février 2005.

**[114].** C. Le duc, N. Le thanh, « Sémantique des modèles d'échange de données », Projet Mecosi, Rapport de recherche, laboratoire Informatique Signaux et Systèmes de Sophia Antipolis, septembre 2002.

**[115].** K. Alva-jorgensen, «Our enterprise intégration approach», 2002. http://www.accenture.com /xd/xd.asp ?it = enWeb&xd=Services\se\ebs_capa_approach.xml.

**[116].** X. Yang, G. Yu et G. Wang, «Efficient mapping integrity constraints from relational database to XML document», A. Caplinskas et J. Eder édition, ADBIS, LNCS2151, pp338-351, Springer Verlag, Berlin/Heidelberg, 2001.

**[117].** J. Chauvet, «Services Web avec SOAP, WSDL, UDDI, ebXM », editions Eyrolles, 2002.

**[118].** W. H. Inmon, Richard D. Hackathorn, «Building the Data Warehouse», Second Edition, John Wiley & Sons, ISBN N°0471-14161-5, 1996.

**[119].** http://www.jackbe.com/

**[120].** http://popfly.ms/

**[121].** http://www.dapper.net/open/

**[122].** http://openkapow.com

**[123].** viralvideochart.unrulymedia.com

**[124].** www.youtube.com

**[125].** www.myspace.com

**[126].** video.google.com

# Annexes

## Le protocole http

HTTP est une couche RPC au-dessus de TCP/ IP. Les requêtes et les réponses HTTP sont traitées par un *serveur Web* (comme Apache ou comme IIS de Microsoft) auquel se connecte un *client HTTP*, généralement un navigateur Web. Requêtes et réponses sont composées d'un en-tête et d'un corps de message; toutes sont exprimées en texte ASCII (et non en binaire comme dans les protocoles DCOM et d'IIOP).

Lors d'un échange HTTP (illustré dans les figures suivantes), le client ouvre un canal de communication (*socket*) sur le port n° 80 du serveur demandé, à charge pour les DNS1 de trouver l'adresse IP correcte. La requête et la réponse HTTP circulent sur ce canal qui ne reste ouvert, par défaut, que le temps d'un échange. Si plusieurs requêtes sont issues du même client vers le même serveur – c'est le cas par exemple pour le téléchargement d'une page Web contenant des images stockées sous forme de fichiers séparés sur le serveur –, la requête peut indiquer que le canal doit rester ouvert par l'option *Keep-Alive.*

Le protocole HTTP prévoit tout une série de codes d'erreur ou d'information pour renvoyer, le cas échéant, le résultat du traitement de la requête au poste client. Requête et réponse HTTP ont chacune leur en-tête spécifique, qualifié par des mots-clés et leurs valeurs.

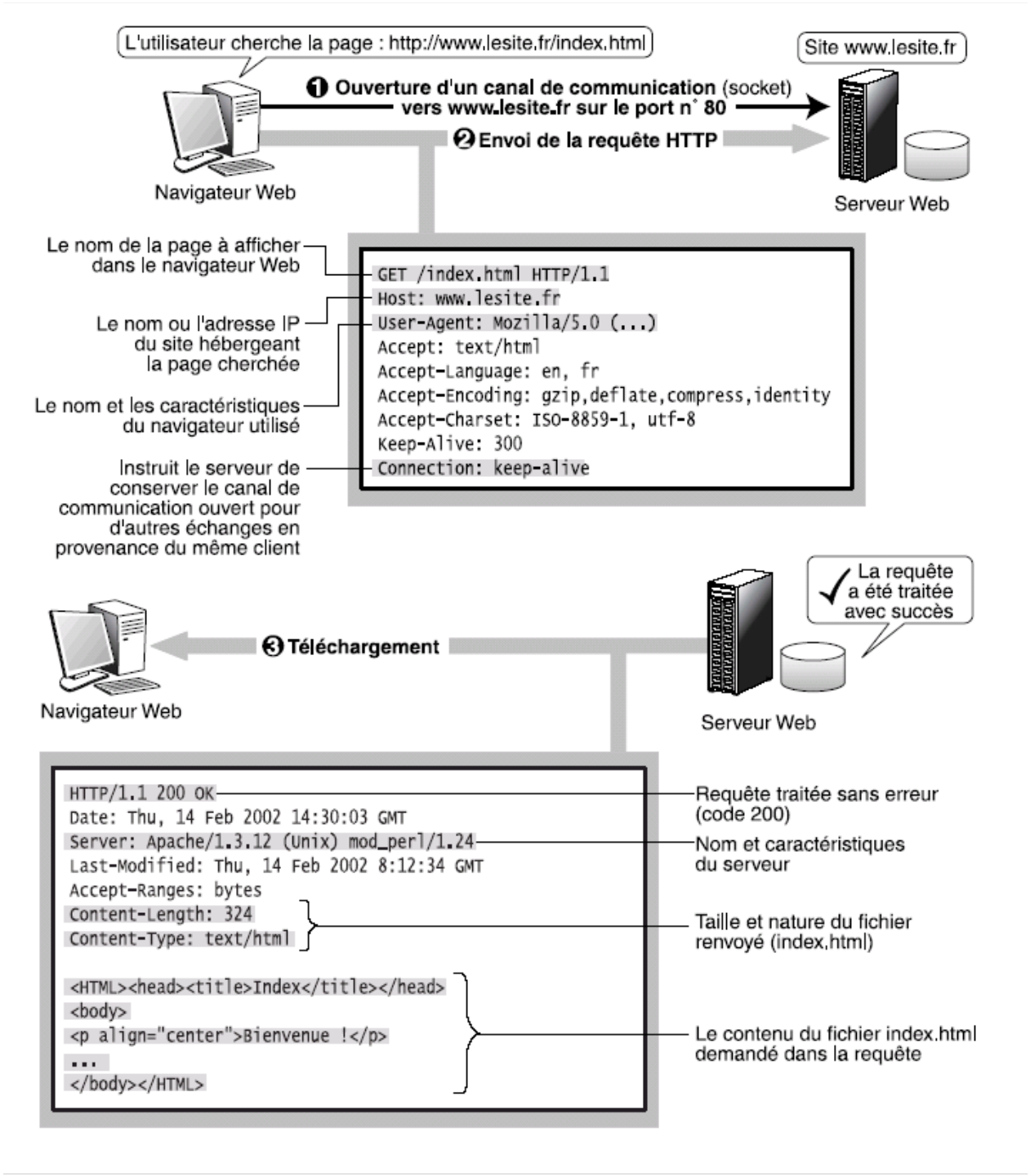



**Figure a.1 :** Échange HTTP entre navigateur et serveur Web.

# Résumé

Un mashup est une application Web qui combine des données et des fonctionnalités de plus d'une source. En apportant des données disparates de manière à permettre aux utilisateurs de faire de nouvelles choses ou accomplir des tâches courantes avec une efficacité retrouvée.

Les Technologies de services Web deviennent de facto un standard pour intégrer des applications et des systèmes distribués en utilisant des standards basés sur XML. Le développement d'applications qui prennent en charge les interfaces de services Web ne suffiront pas à fournir des processus d'affaires complets et coordonnés. Ainsi, nous avons besoin d'une nouvelle approche pour composer ces services web ensemble pour former l'orchestration des services web et la définition des processus.

Beaucoup de nouveaux standards ont été définis pour résoudre ce problème, par exemple BPEL4WS. Ce mémoire vise à familiariser le lecteur avec ce standard émergent et aider à comprendre comment l'orchestration des services web peut être réalisée aujourd'hui.



**Mots-clés :** mashup, services Web, SOA, WSDL, BPEL4WS, Composition de services, orchestration de services.



# Abstract

A Web mashup is an application that combines data and functionality from more than one source. By bringing disparate data together in ways that enable users to do new things or accomplish common tasks with newfound efficiency.

Web services technologies are becoming a de facto standard to integrate distributed applications and systems using XML-based standards. Developing applications that support web services interfaces will not be enough to provide complete and coordinated business processes. Thus, we need a new approach to compose these web services together in order to form web services orchestration and processes definition.

Many new standards have been defined to solve this problem, for example BPEL4WS, this report aims to familiarize the reader with these different emerging standards and help understanding how web services orchestration can be achieved today.



**Keywords :** mashup, Web Services, SOA, WSDL, BPEL4WS, Services Composition, Services Orchestration.