Approche dirigée par les modèles pour la spécification, la vérification formelle et la mise en œuvre des services Web composés

Christophe Dumez

Laboratoire Systèmes et Transports (SeT)
Université de Technologie de Belfort-Montbéliard
90000, Belfort, France

31 août 2010

Plan de la présentation

Plan de la présentation

Plan de la présentation

utbm université de technologie Belfort-Montbéliard

Rappel du plan

Architecture Orientée Services (SOA)

Définition

Une architecture orientée services (notée SOA) est une architecture logicielle s'appuyant sur un ensemble de services simples.

- Le *service* est l'unité atomique d'une architecture SOA
- Une *application* est un ensemble de services qui dialoguent par échange de messages
- Elle se base sur des concepts plus anciens
  - L'informatique distribuée
  - La programmation modulaire

Architecture Orientée Services (SOA)

Définition

Une architecture orientée services (notée SOA) est une architecture logicielle s'appuyant sur un ensemble de services simples.

- Le *service* est l'unité atomique d'une architecture SOA
- Une *application* est un ensemble de services qui dialoguent par échange de messages
- Elle se base sur des concepts plus anciens
  - L'informatique distribuée
  - La programmation modulaire

Principes fondamentaux

- **Couplage faible**
  - Séparation logique entre le client et le service
  - Échange de messages dans un format standard
  - Pas de dépendance physique entre les deux
- **Intéropérabilité**
  - Capacité d'un système à fonctionner avec d'autres systèmes existants ou futurs
  - L'interface des services est intégralement connue
- **Réutilisabilité**
  - Séparation des tâches en services autonomes pour promouvoir leur réutilisation
- **Découverte**
  - Pour réutiliser un service, il faut savoir qu'il existe
  - Utilisation d'un annuaire de services

Principes fondamentaux

- **Couplage faible**
  - Séparation logique entre le client et le service
  - Échange de messages dans un format standard
  - Pas de dépendance physique entre les deux
- **Intéropérabilité**
  - Capacité d'un système à fonctionner avec d'autres systèmes existants ou futurs
  - L'interface des services est intégralement connue
- **Réutilisabilité**
  - Séparation des tâches en services autonomes pour promouvoir leur réutilisation
- **Découverte**
  - Pour réutiliser un service, il faut savoir qu'il existe
  - Utilisation d'un annuaire de services

Principes fondamentaux

- **Couplage faible**
  - Séparation logique entre le client et le service
  - Échange de messages dans un format standard
  - Pas de dépendance physique entre les deux
- **Intéropérabilité**
  - Capacité d'un système à fonctionner avec d'autres systèmes existants ou futurs
  - L'interface des services est intégralement connue
- **Réutilisabilité**
  - Séparation des tâches en services autonomes pour promouvoir leur réutilisation
- **Découverte**
  - Pour réutiliser un service, il faut savoir qu'il existe
  - Utilisation d'un annuaire de services

Principes fondamentaux

- **Couplage faible**
  - Séparation logique entre le client et le service
  - Échange de messages dans un format standard
  - Pas de dépendance physique entre les deux
- **Intéropérabilité**
  - Capacité d'un système à fonctionner avec d'autres systèmes existants ou futurs
  - L'interface des services est intégralement connue
- **Réutilisabilité**
  - Séparation des tâches en services autonomes pour promouvoir leur réutilisation
- **Découverte**
  - Pour réutiliser un service, il faut savoir qu'il existe
  - Utilisation d'un annuaire de services

Les services Web

- Approche la plus populaire pour mettre en œuvre une architecture SOA

Définition (W3C)

Un service Web est un système logiciel conçu pour permettre à deux machines de communiquer de manière intéropérable sur un réseau.

- Son interface est clairement décrite dans un format standard (WSDL)
- Les autres systèmes interagissent avec lui en utilisant un protocole standard (SOAP)

Les services Web

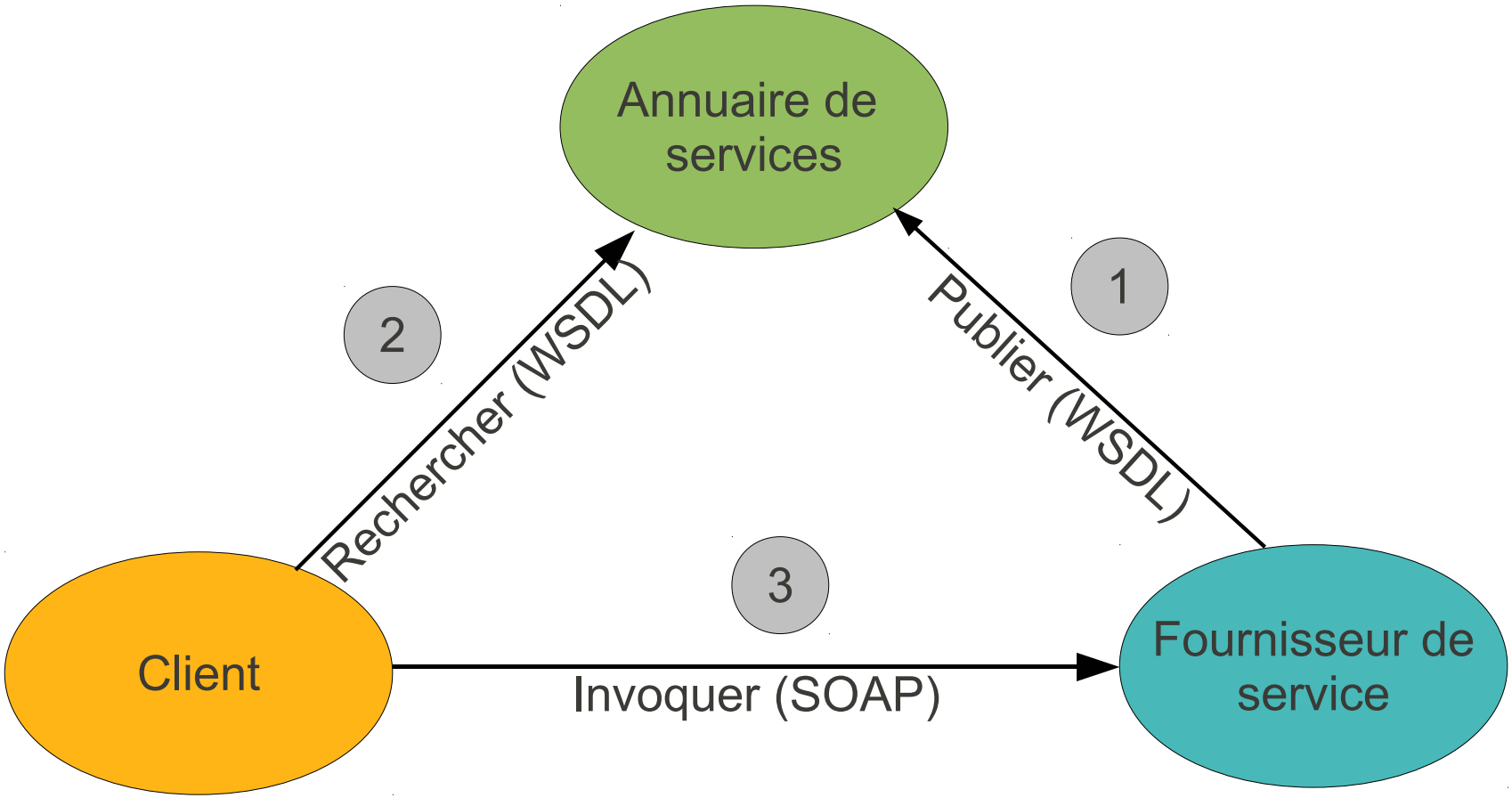
- Approche la plus populaire pour mettre en œuvre une architecture SOA

Définition (W3C)

Un service Web est un système logiciel conçu pour permettre à deux machines de communiquer de manière intéropérable sur un réseau.

- Son interface est clairement décrite dans un format standard (WSDL)
- Les autres systèmes interagissent avec lui en utilisant un protocole standard (SOAP)

Architecture



Architecture standard décrite par le W3C

Protocole de communication SOAP

- Protocole de communication basé sur XML recommandé par le W3C
- Permet la transmission de messages entre objets distants
- Présente plusieurs avantages :
  - Assez ouvert pour s'adapter à différents protocoles de transport (HTTP, SMTP, ...)
  - Indépendant de la plateforme
  - Indépendant du langage
  - Extensible

Web Service Description Language (WSDL)

- Langage basé sur XML, standardisé par le W3C
- Décrit l'interface publique d'accès à un service Web
- Indique « *Comment communiquer avec le service* »
- Un document WSDL décrit :
  - Le protocole de communication (ex : SOAP)
  - Le format des messages
  - Les méthodes que le client peut invoquer
  - La localisation du service

La composition de services

Définition

La composition de services consiste à interconnecter plusieurs services (élémentaires ou eux-même composés), afin d'apporter une valeur ajoutée à l'utilisateur.

On peut classifier la composition en deux catégories :

- Chorégraphie
- Orchestration

La composition de services

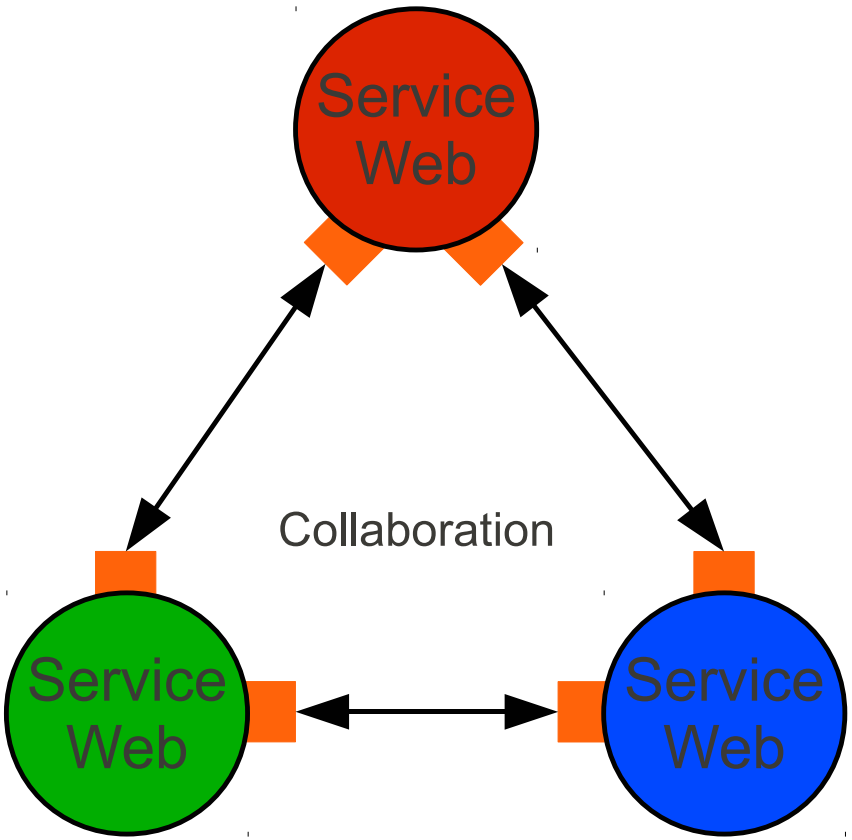
Définition

La composition de services consiste à interconnecter plusieurs services (élémentaires ou eux-même composés), afin d'apporter une valeur ajoutée à l'utilisateur.

On peut classifier la composition en deux catégories :

- Chorégraphie
- Orchestration

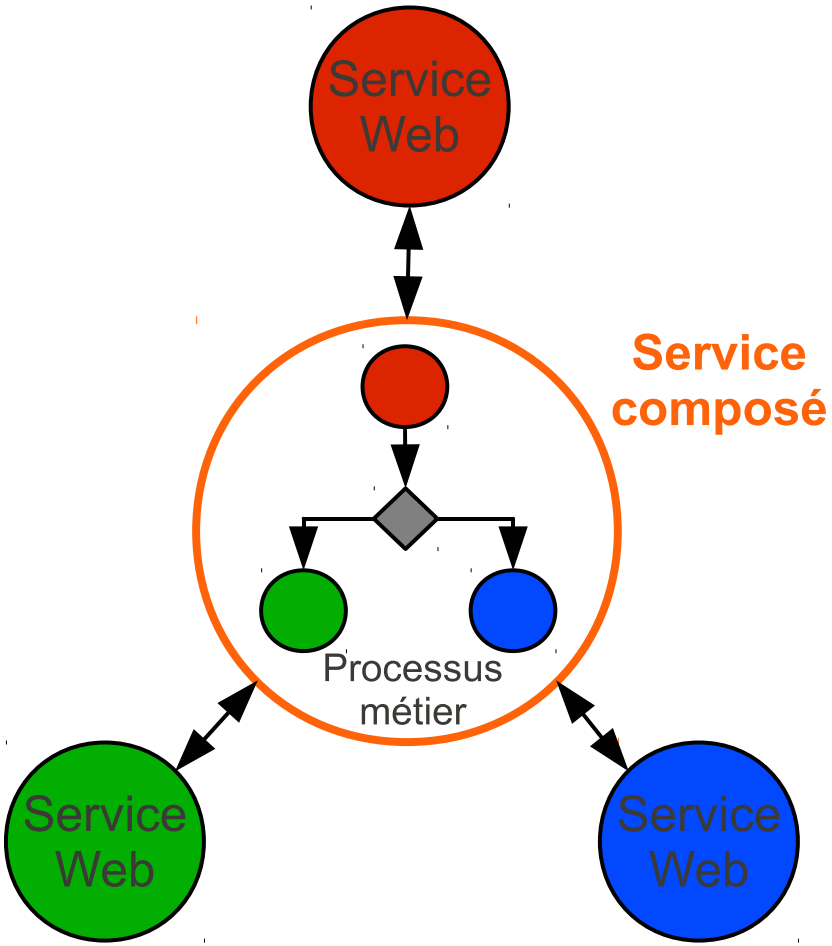
Chorégraphie



Chorégraphie

- De nature très collaborative
- Chaque service décrit le rôle qu'il joue dans la composition
- La conversation n'est détenue par aucune entité
- Les services existants doivent être modifiés

Orchestration



Orchestration

- Processus métier exécutable
- Interagit avec des services Web
- Le processus est contrôlé par une seule entité
- Pas de modification au niveau des services existants

Business Process Execution Language (BPEL)

- Langage le plus populaire pour l'orchestration de services
- WS-BPEL 2.0 standardisé par OASIS en 2007
- Langage exécutable basé sur XML
- Décrit un processus exécutable mettant en œuvre l'échange de messages avec d'autres systèmes

Problématique

- La composition de services est une tâche complexe
  - Hétérogénéité des données et de leurs formats
- Plusieurs langages de composition ont été proposés
  - WSFL, XLANG, BPEL ...
  - Langages textuels exécutables
- **L'étape de spécification et de vérification est négligée**
- Nécessaire de proposer une **approche dirigée par les modèles** permettant :
  - La modélisation dans un langage standard et connu (UML)
  - La vérification formelle
  - La génération de code exécutable (BPEL)

Problématique

- La composition de services est une tâche complexe
  - Hétérogénéité des données et de leurs formats
- Plusieurs langages de composition ont été proposés
  - WSFL, XLANG, BPEL ...
  - Langages textuels exécutables
- L'étape de spécification et de vérification est négligée
- Nécessaire de proposer une **approche dirigée par les modèles** permettant :
  - La modélisation dans un langage standard et connu (UML)
  - La vérification formelle
  - La génération de code exécutable (BPEL)

Problématique

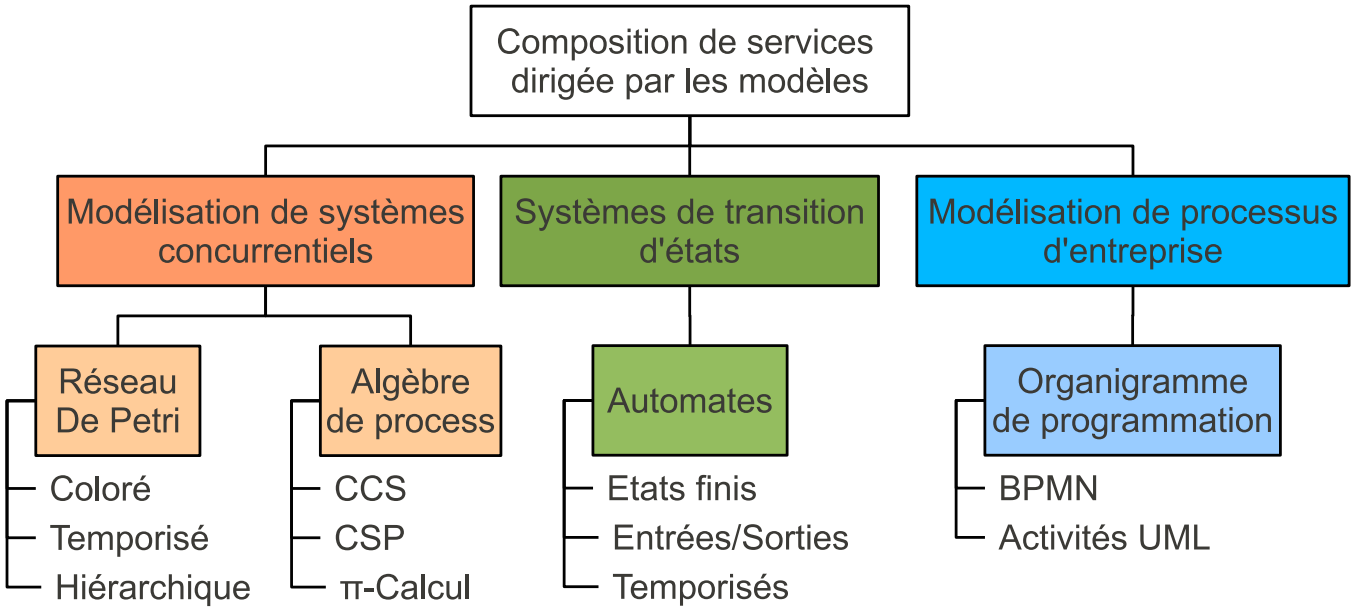
- La composition de services est une tâche complexe
  - Hétérogénéité des données et de leurs formats
- Plusieurs langages de composition ont été proposés
  - WSFL, XLANG, BPEL ...
  - Langages textuels exécutables
- **L'étape de spécification et de vérification est négligée**
- Nécessaire de proposer une **approche dirigée par les modèles** permettant :
  - La modélisation dans un langage standard et connu (UML)
  - La vérification formelle
  - La génération de code exécutable (BPEL)

Problématique

- La composition de services est une tâche complexe
  - Hétérogénéité des données et de leurs formats
- Plusieurs langages de composition ont été proposés
  - WSFL, XLANG, BPEL ...
  - Langages textuels exécutables
- **L'étape de spécification et de vérification est négligée**
- Nécessaire de proposer une **approche dirigée par les modèles** permettant :
  - La modélisation dans un langage standard et connu (UML)
  - La vérification formelle
  - La génération de code exécutable (BPEL)

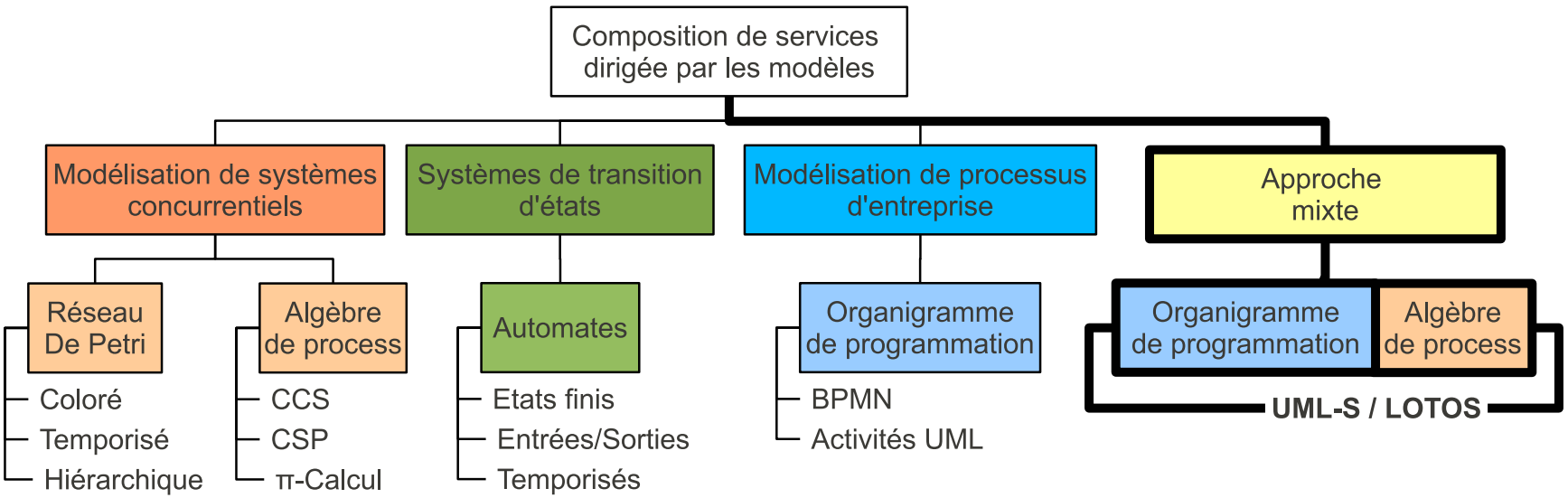
utbm
université de technologie
Belfort-Montbéliard

État de l'art



Classification des approches existantes

État de l'art



Classification de l'approche proposée

État de l'art

Approches basées sur UML

Année	Auteurs	Dia-grammes	BPEL	UML	Conforme au méta-modèle	Vérif.	Repr. Inter-face	Implé-menta-tion
2003	Gardner et al. (IBM)	Classes / Activités	1.0	1.4	+	-	-	+
2005	Ambühler et al.	Classes / Activités	1.1	2.0	+	-	+	+/-
2005	Grønmo / Skogan et al.	Classes / Activités	1.1	1.4	+	-	+	-
2006	Castro et al.	Cas d'util. / Activités	1.1	2.0	-	-	-	+/-
2007	Bendraou et al.	Activités	2.0	2.0	-	-	-	+/-
2010	Dumez et al.	Classes / Activités	2.0	2.0	+	+	+	+

Contributions de la thèse

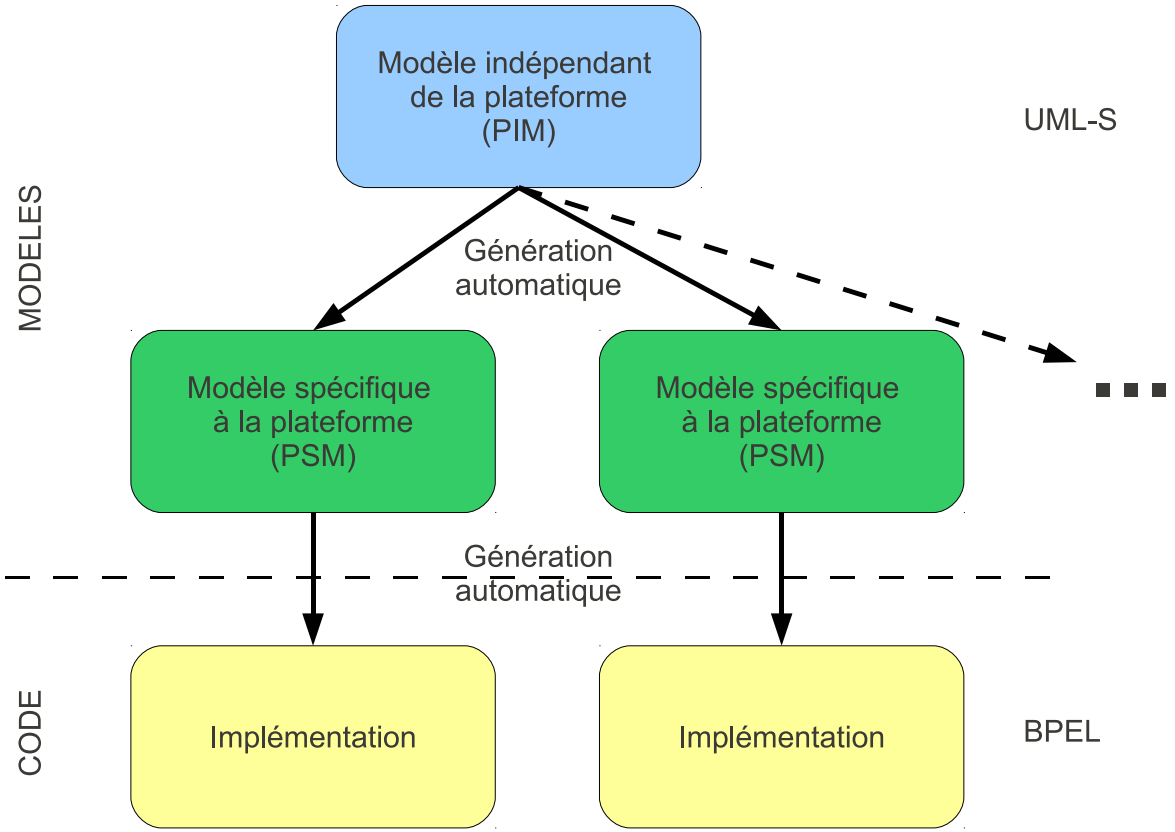
- Approche de développement pour services Web composés
  - Langage de modélisation de la composition : UML-S
  - Règles de transformations pour la génération de code
  - Vérification formelle de la composition
- Environnement de développement intégré pour composer des services en utilisant l'approche proposée
  - Automatisation de nombreuses tâches complexes

Rappel du plan

Model-Driven Architecture (MDA)

- Définie par l'OMG en 2000 pour promulguer de bonnes pratiques de modélisation
- Exploite pleinement les avantages des modèles
  - Pérennité
  - Productivité
  - Prise en compte des plateformes d'exécution
- Du code peut alors être généré à partir de ces modèles abstraits

Model-Driven Architecture (MDA)



Aperçu de l'approche MDA

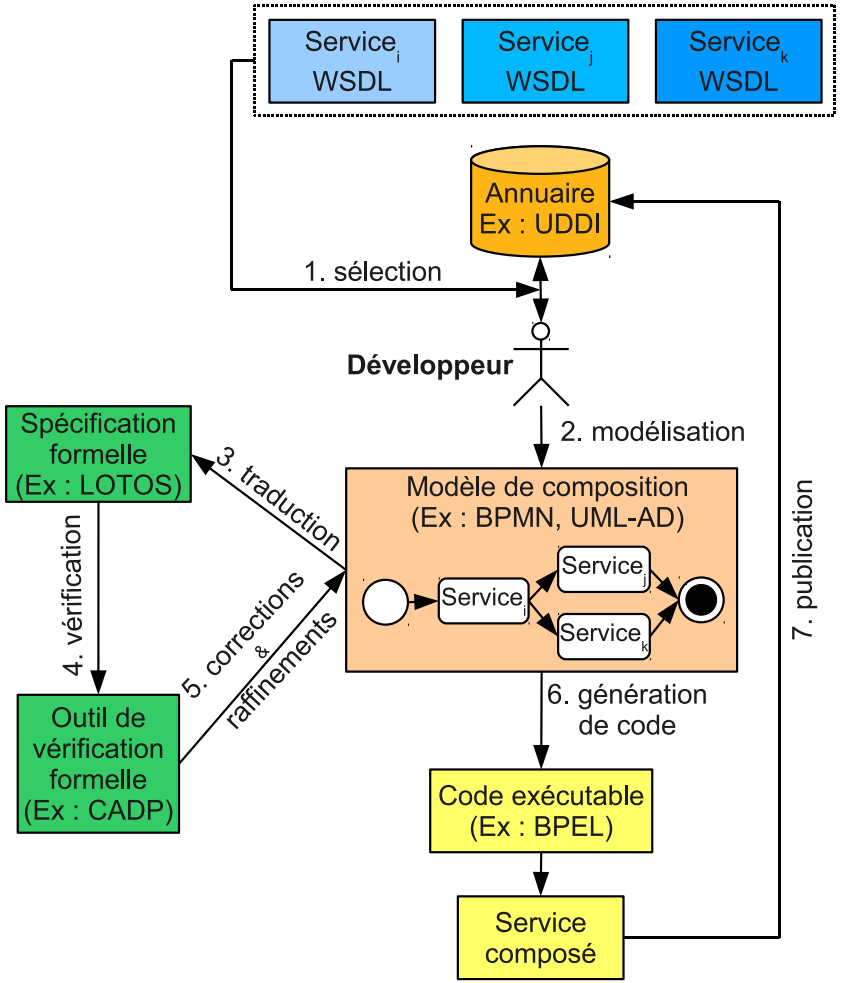
Présentation de l'approche

- Une approche fidèle aux principes du MDA pour
  - spécifier,
  - vérifier formellement,
  - mettre en œuvre la composition de services Web.
- La composition est spécifiée à l'aide d'un profil UML : *UML-S*
- La vérification formelle est réalisée par l'intermédiaire d'un langage de description formelle
- Du code exécutable peut être généré directement

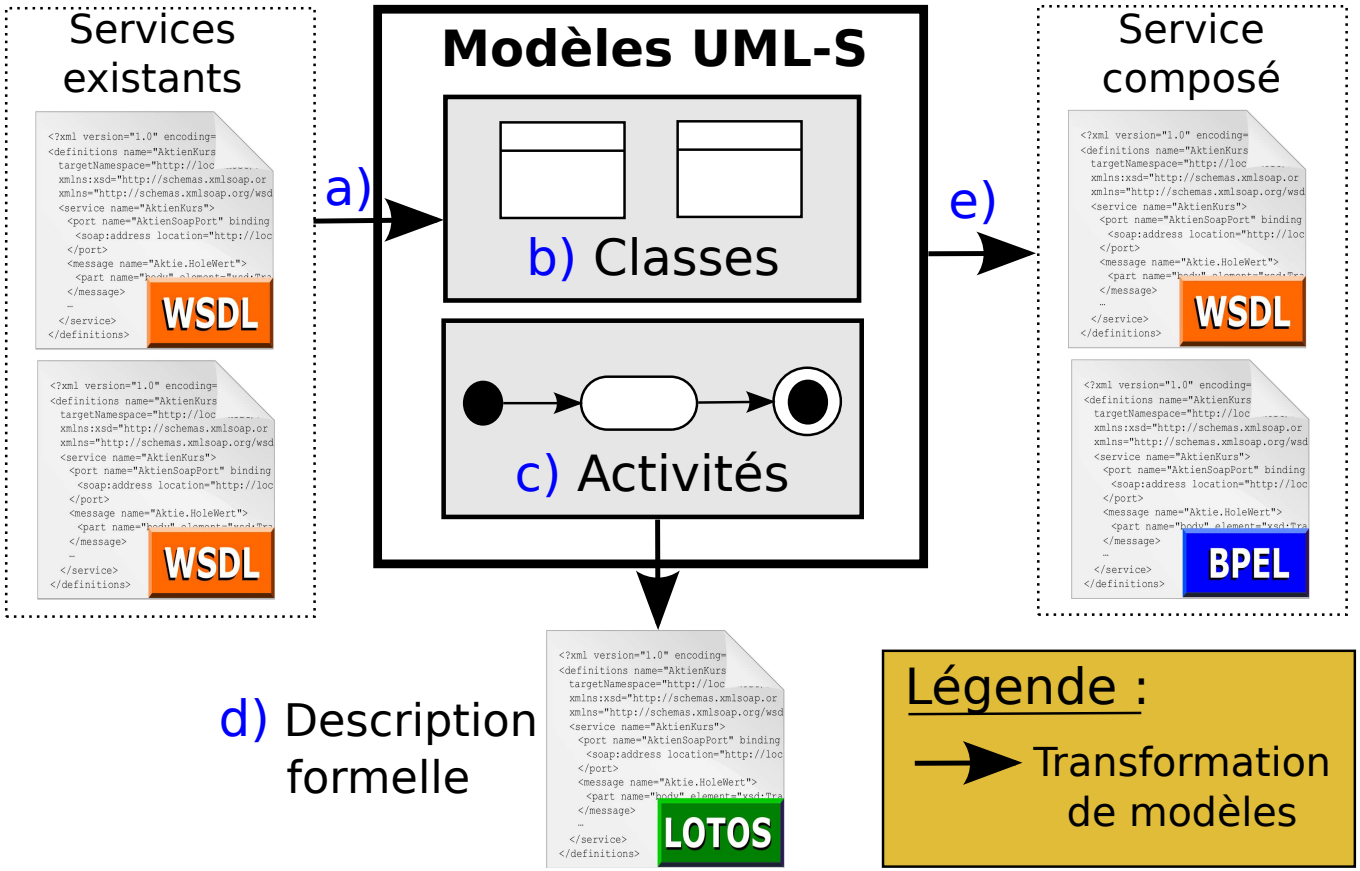
Présentation de l'approche

- Une approche fidèle aux principes du MDA pour
  - spécifier,
  - vérifier formellement,
  - mettre en œuvre la composition de services Web.
- La composition est spécifiée à l'aide d'un profil UML : *UML-S*
- La vérification formelle est réalisée par l'intermédiaire d'un langage de description formelle
- Du code exécutable peut être généré directement

Processus de développement

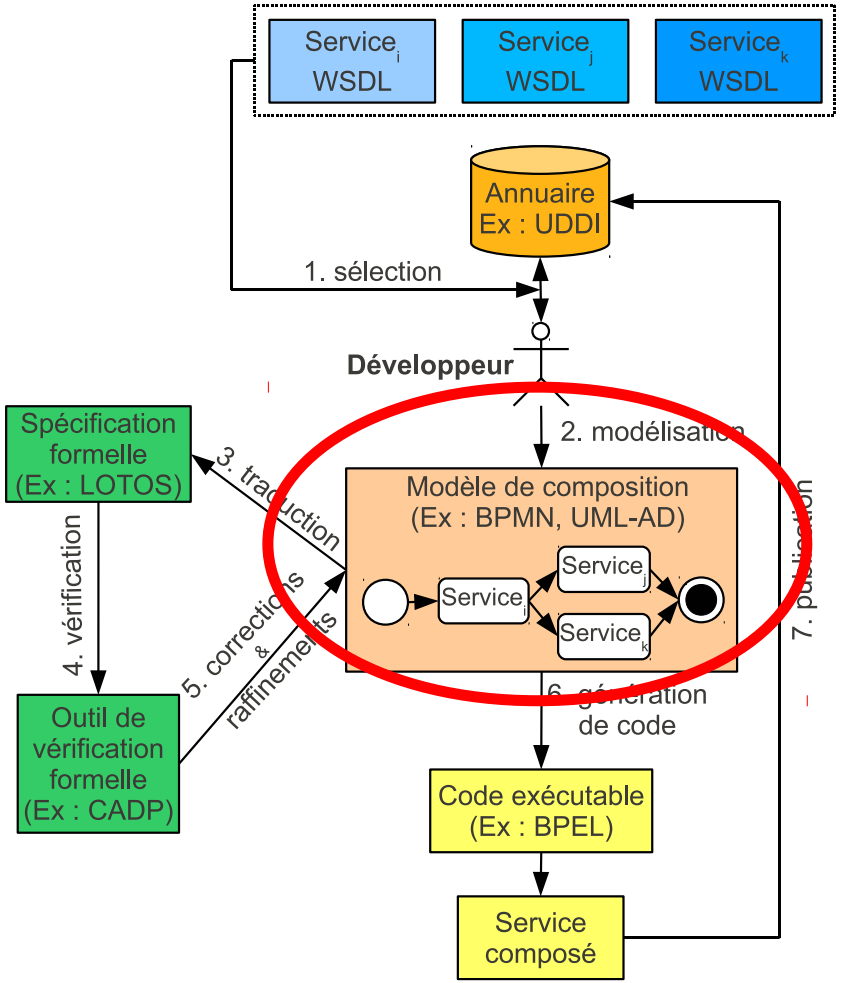


Transformation de modèles



Approche MDA : La transformation de modèles

Profil proposé : UML-S



Profil proposé : UML-S

Définition (OMG)

Un profil UML est un mécanisme d'extension générique pour adapter les modèles UML à un domaine ou à une plateforme particulière.

- Permet uniquement l'ajout de fonctionnalités, pas de suppression ou de modification
  - Reste ainsi conforme au métamodèle standard UML
- Trois types de mécanismes d'ajout :
  - Stéréotypes
  - Valeurs étiquetées
  - Contraintes (OCL)

Profil proposé : UML-S

Définition (OMG)

Un profil UML est un mécanisme d'extension générique pour adapter les modèles UML à un domaine ou à une plateforme particulière.

- Permet uniquement l'ajout de fonctionnalités, pas de suppression ou de modification
  - Reste ainsi conforme au métamodèle standard UML
- Trois types de mécanismes d'ajout :
  - Stéréotypes
  - Valeurs étiquetées
  - Contraintes (OCL)

UML-S : Diagramme de classes

- Permet de modéliser l'aspect statique de la composition :
  - Les interfaces publiques des services
  - Les types de données complexes manipulés
- 1 service = 1 classe UML dotée du stéréotype « *WebService* »
  - Indique les opérations accessibles publiquement
- Les types de données non élémentaires sont représentés par des classes simples
- Le diagramme de classes peut être généré automatiquement à partir des descriptions WSDL des services

UML-S : Diagramme de classes

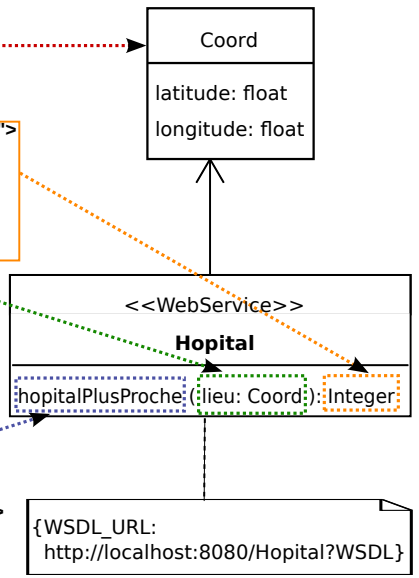
- Permet de modéliser l'aspect statique de la composition :
  - Les interfaces publiques des services
  - Les types de données complexes manipulés
- 1 service = 1 classe UML dotée du stéréotype « *WebService* »
  - Indique les opérations accessibles publiquement
- Les types de données non élémentaires sont représentés par des classes simples
- Le diagramme de classes peut être généré automatiquement à partir des descriptions WSDL des services

Description WSDL vers diagramme de classes UML-S

Document WSDL 1.1

```
<definitions ...><types><schema ...>
<xs:complexType name="Coord">
 <xs:sequence>
  <element name="latitude" type="xs:float"/>
  <element name="longitude" type="xs:float"/>
 </xs:sequence>
</xs:complexType>
<xs:complexType name="hopitalPlusProcheReponse">
 <xs:sequence>
  <xs:element name="return" type="xs:integer"...>
  </xs:element>
 </xs:sequence>
</xs:complexType>
<xs:complexType name="hopitalPlusProche">
 <xs:sequence>
  <xs:element name="lieu" type="tns:Coord"...>
  </xs:element>
 </xs:sequence>
</xs:complexType>
</schema></types>
 <portType name="Hopital">
 <operation name="hopitalPlusProche">
   <input message="tns:hopitalPlusProche"></input>
   <output message="tns:hopitalPlusProcheReponse">
   </output>
  </operation>
 </portType>
 <message name="hopitalPlusProche">
  <part name="x" type="tns:hopitalPlusProche"></part>
 </message>
 <message name="hopitalPlusProcheReponse">
  <part name="x" type="tns:hopitalPlusProcheReponse">
  </part>
 </message>
 <binding name="HopitalPortBinding" type="tns:Hopital">...
 </binding> ...
 <service name="Hopital">...</service></definitions>
```

Classes UML-S



UML-S : Diagramme d'activité

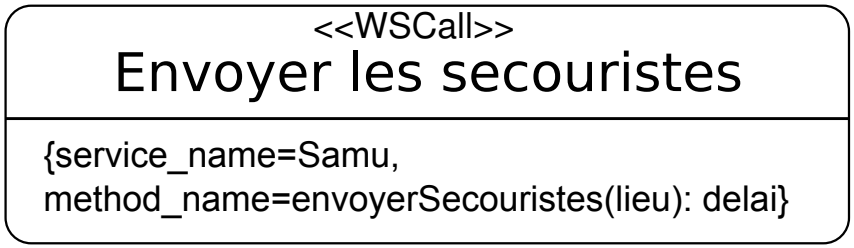
- Permet de modéliser l'aspect dynamique de la composition
  - Le scénario de collaboration inter-services
- Un diagramme d'activité UML-S modélise le comportement d'une opération fournie par le nouveau service composé
- L'état initial de l'activité correspond à l'appel de l'opération
  - Avec passage éventuel de paramètres
- L'état final de l'activité correspond à la fin de l'opération
  - Avec retour éventuel d'une valeur

UML-S : Diagramme d'activité

- Permet de modéliser l'aspect dynamique de la composition
  - Le scénario de collaboration inter-services
- Un diagramme d'activité UML-S modélise le comportement d'une opération fournie par le nouveau service composé
- L'état initial de l'activité correspond à l'appel de l'opération
  - Avec passage éventuel de paramètres
- L'état final de l'activité correspond à la fin de l'opération
  - Avec retour éventuel d'une valeur

UML-S : Diagramme d'activité

- Chaque étape (*action*) de l'activité correspond à une invocation de service
- Le stéréotype « *WSCall* » est utilisé
- Le nom du service invoqué et le nom de l'opération sont précisés en valeurs étiquetées



Exemple d'action UML-S

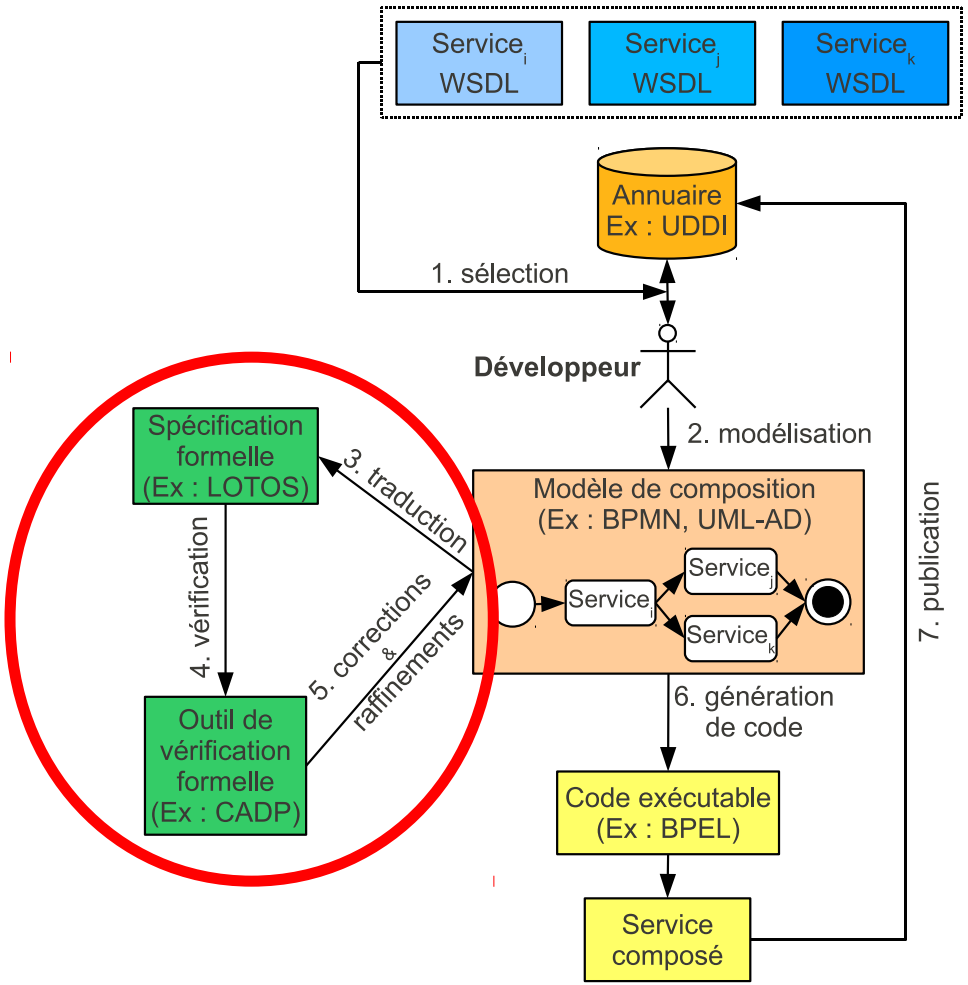
Structures de contrôle

- L'ordre d'exécution des actions d'un workflow est déterminé par ses structures de contrôle
- Van der Aalst et al. ont identifié les structures de contrôle récurrentes au sein des différents langages de workflow
- 20 structures dont 5 dites *basiques* :
  - *séquence, branchement multiple, synchronisation, choix exclusif, jonction simple*

Structures de contrôle

- Le diagramme d'activité UML permet de modéliser toutes ces structures
- Cette tâche peut s'avérer difficile pour certaines structures
- UML-S comporte des ajouts simplifiant la représentation de ces structures
- Ceci permet d'obtenir des modèles UML-S plus clairs et concis

Spécification et la vérification formelle



Langage de spécification formelle LOTOS

LOTOS est utilisé pour la vérification formelle de la composition

Pourquoi passer par un langage intermédiaire ?

- La sémantique d'UML n'est pas formellement définie
- L'analyse formelle des modèles UML n'est pas possible

Pourquoi LOTOS ?

- Algèbre de processus standardisée par ISO
- Sa sémantique est formellement définie
- Prise en charge du passage de données entre les processus
- Outils de vérification très avancés (CADP)

Langage de spécification formelle LOTOS

LOTOS est utilisé pour la vérification formelle de la composition

Pourquoi passer par un langage intermédiaire ?

- La sémantique d'UML n'est pas formellement définie
- L'analyse formelle des modèles UML n'est pas possible

Pourquoi LOTOS ?

- Algèbre de processus standardisée par ISO
- Sa sémantique est formellement définie
- Prise en charge du passage de données entre les processus
- Outils de vérification très avancés (CADP)

Langage de spécification formelle LOTOS

LOTOS est utilisé pour la vérification formelle de la composition

Pourquoi passer par un langage intermédiaire ?

- La sémantique d'UML n'est pas formellement définie
- L'analyse formelle des modèles UML n'est pas possible

Pourquoi LOTOS ?

- Algèbre de processus standardisée par ISO
- Sa sémantique est formellement définie
- Prise en charge du passage de données entre les processus
- Outils de vérification très avancés (CADP)

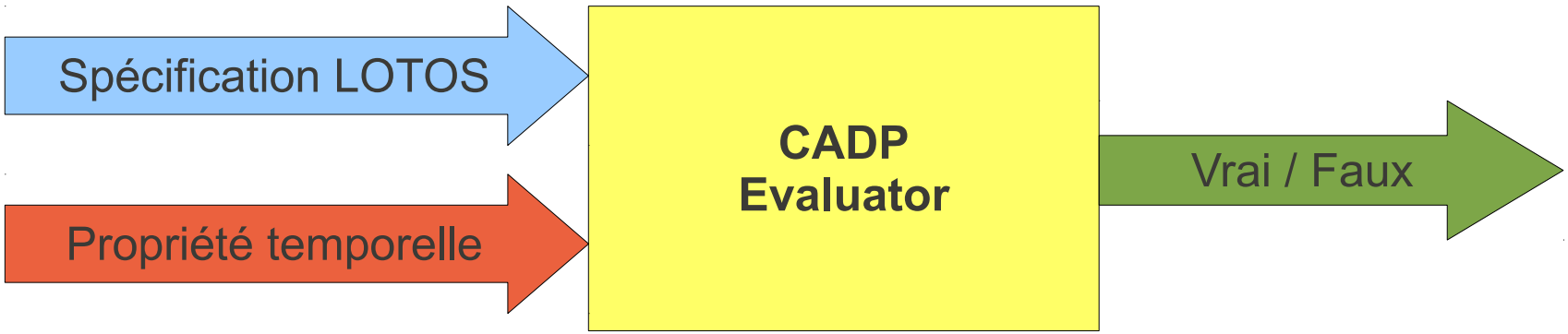
Règles de transformation vers LOTOS

- Implémentation en LOTOS des 20 structures de contrôle de Aalst et al.
- Représenté par des processus LOTOS
  - Facilite leur réutilisation
- Réalisation du comportement du workflow
  - Appel aux processus LOTOS
  - Utilisation de la communication inter-processus en LOTOS

Boîte à outils CADP

- « *Construction and Analysis of Distributed Processes* »
- Boîte à outils développée par l'INRIA, qui permet :
  - La compilation de descriptions écrites en LOTOS
  - La vérification d'équivalence (p.ex. *Bissimulation*)
  - Le « *model checking* » à l'aide de la logique temporelle
  - ...

Boîte à outils CADP



Procédure de vérification formelle avec CADP

Logique temporelle

- Extension de la logique classique
- Permet de décrire sans ambiguïté des comportements dynamiques
  - *c.à.d. qui évoluent dans le temps*
- Exemple de propriété temporelle :
  - *Si un feu est signalé alors les pompiers interviendront*

Types de propriétés comportementales

- Accessibilité (*Reachability*)
  Une certaine situation peut être atteinte
- Invariance
  Tous les états du système satisfont une *bonne* propriété
- Sûreté (*Safety*)
  Quelque chose de *mauvais* n'arrive jamais
- Vivacité (*Liveness*)
  Quelque chose de *bon* finit par arriver

Types de propriétés comportementales

- Accessibilité (*Reachability*)
  Une certaine situation peut être atteinte
- Invariance
  Tous les états du système satisfont une *bonne* propriété
- Sûreté (*Safety*)
  Quelque chose de *mauvais* n'arrive jamais
- Vivacité (*Liveness*)
  Quelque chose de *bon* finit par arriver

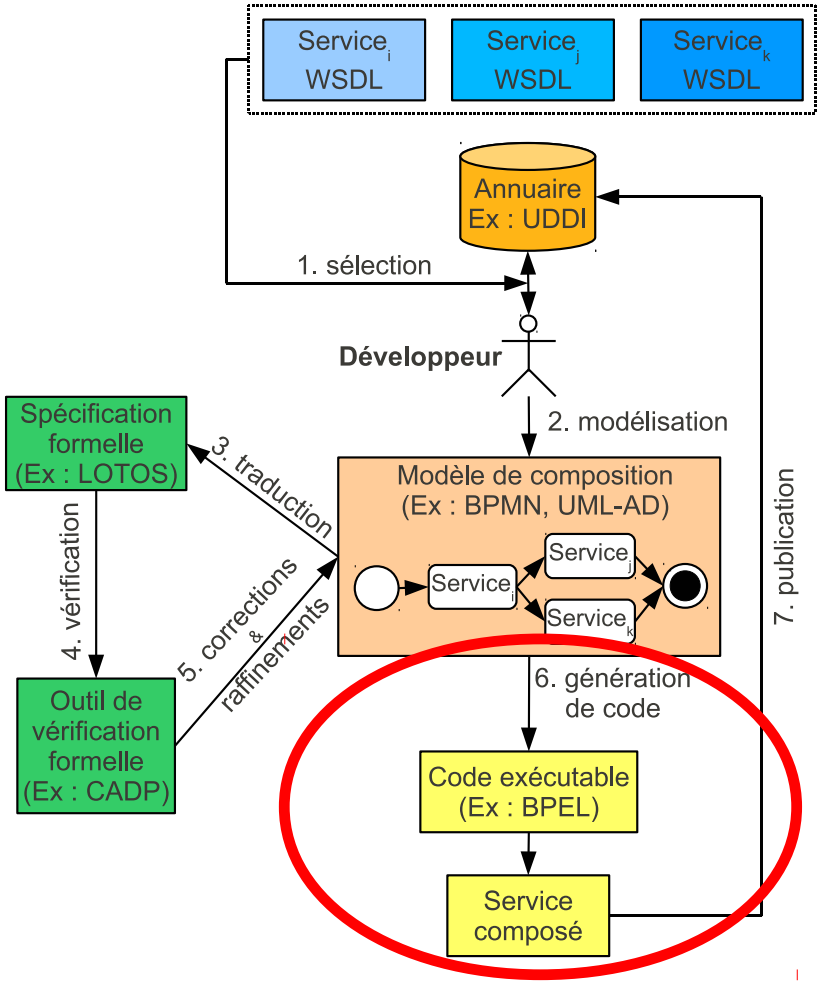
Types de propriétés comportementales

- Accessibilité (*Reachability*)
  Une certaine situation peut être atteinte
- Invariance
  Tous les états du système satisfont une *bonne* propriété
- Sûreté (*Safety*)
  Quelque chose de *mauvais* n'arrive jamais
- Vivacité (*Liveness*)
  Quelque chose de *bon* finit par arriver

Types de propriétés comportementales

- Accessibilité (*Reachability*)
  Une certaine situation peut être atteinte
- Invariance
  Tous les états du système satisfont une *bonne* propriété
- Sûreté (*Safety*)
  Quelque chose de *mauvais* n'arrive jamais
- Vivacité (*Liveness*)
  Quelque chose de *bon* finit par arriver

Génération de code



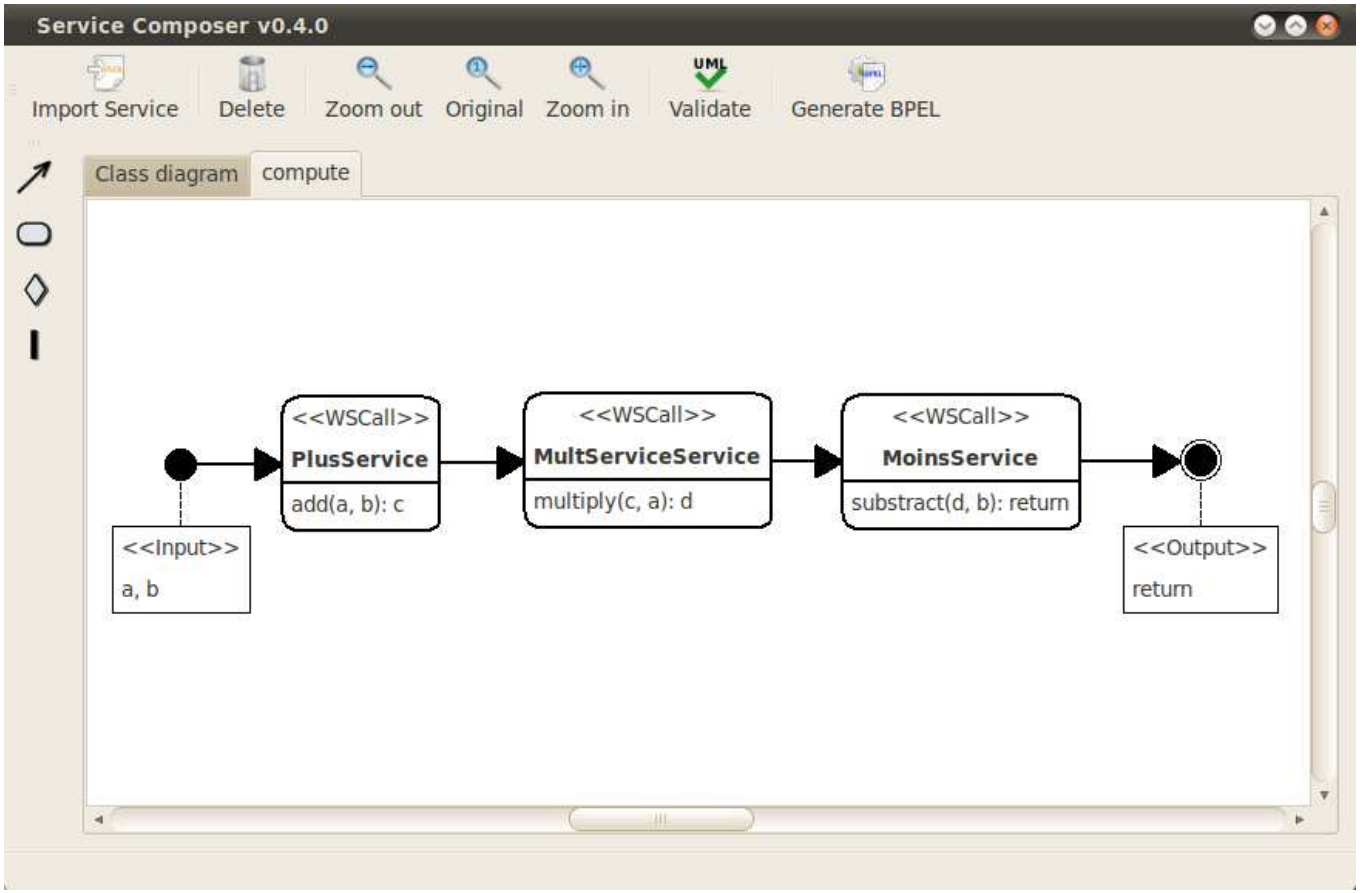
Génération de code

- Génération du code du service composé :
  - Processus BPEL
  - Description WSDL
- Règles de transformation fournies vers BPEL
- Transformation basée sur l'identification des structures de contrôle
- Génération automatique par l'environnement de développement

Environnement de développement : Service Composer

- Un environnement de développement a été développé
  - Développé entièrement en C++ / Qt4
  - Logiciel libre et multi-plateformes
- Logiciel fonctionnel :
  - Import des descriptions WSDL
  - Modélisation en UML-S
  - Génération du code BPEL et WSDL

Environnement de développement : Service Composer



Capture d'écran : Environnement de développement

Rappel du plan

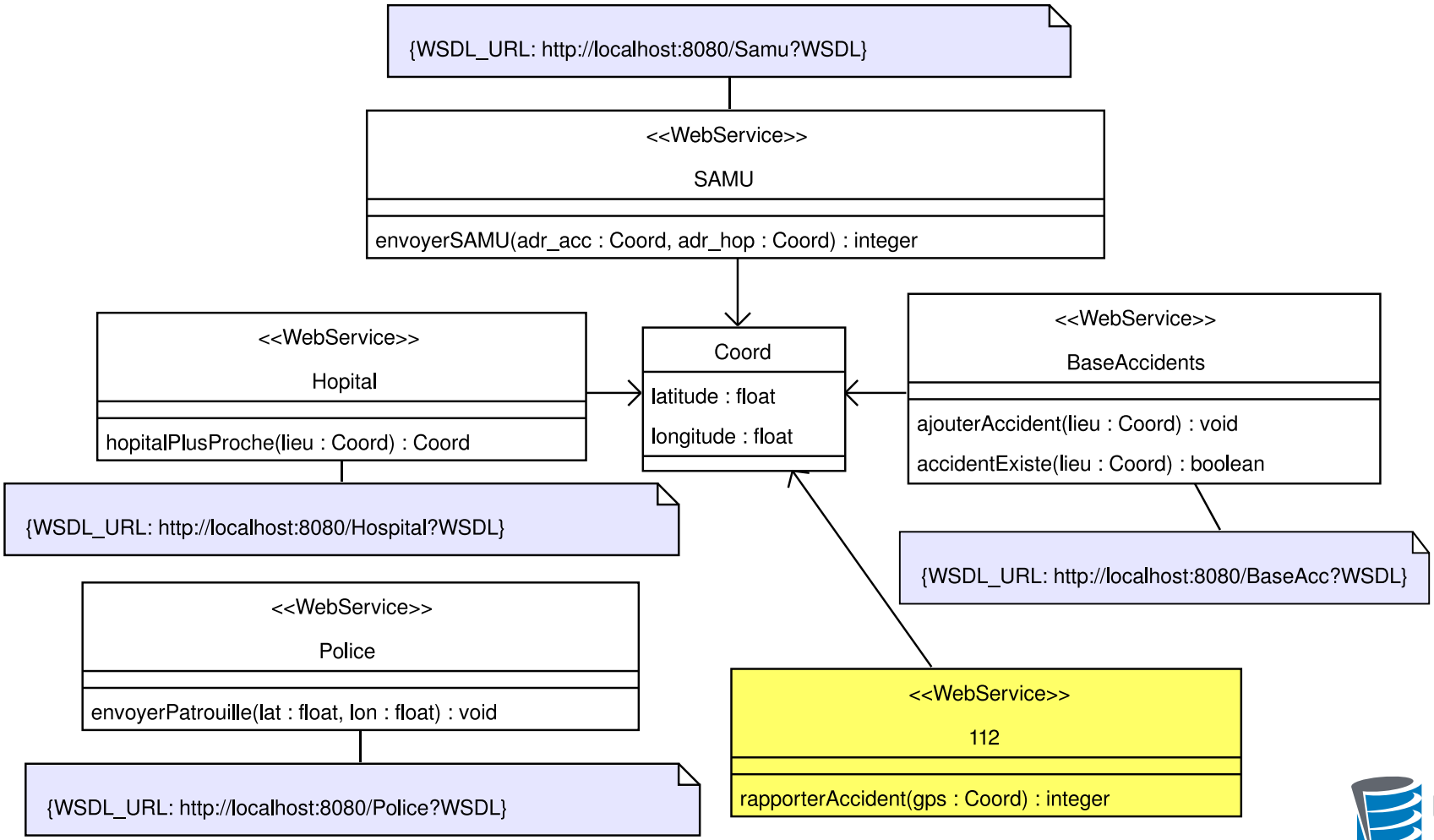
Scénario de test

- Scénario d'appel d'urgence lors d'une notification d'accident
- Dans le cadre du projet Européen ASSET (7ème PCRD)
  - Projet relatif à la sécurité routière
  - L'appel d'urgence est une des problématiques
  - Une des tâches traite des services *intelligents*

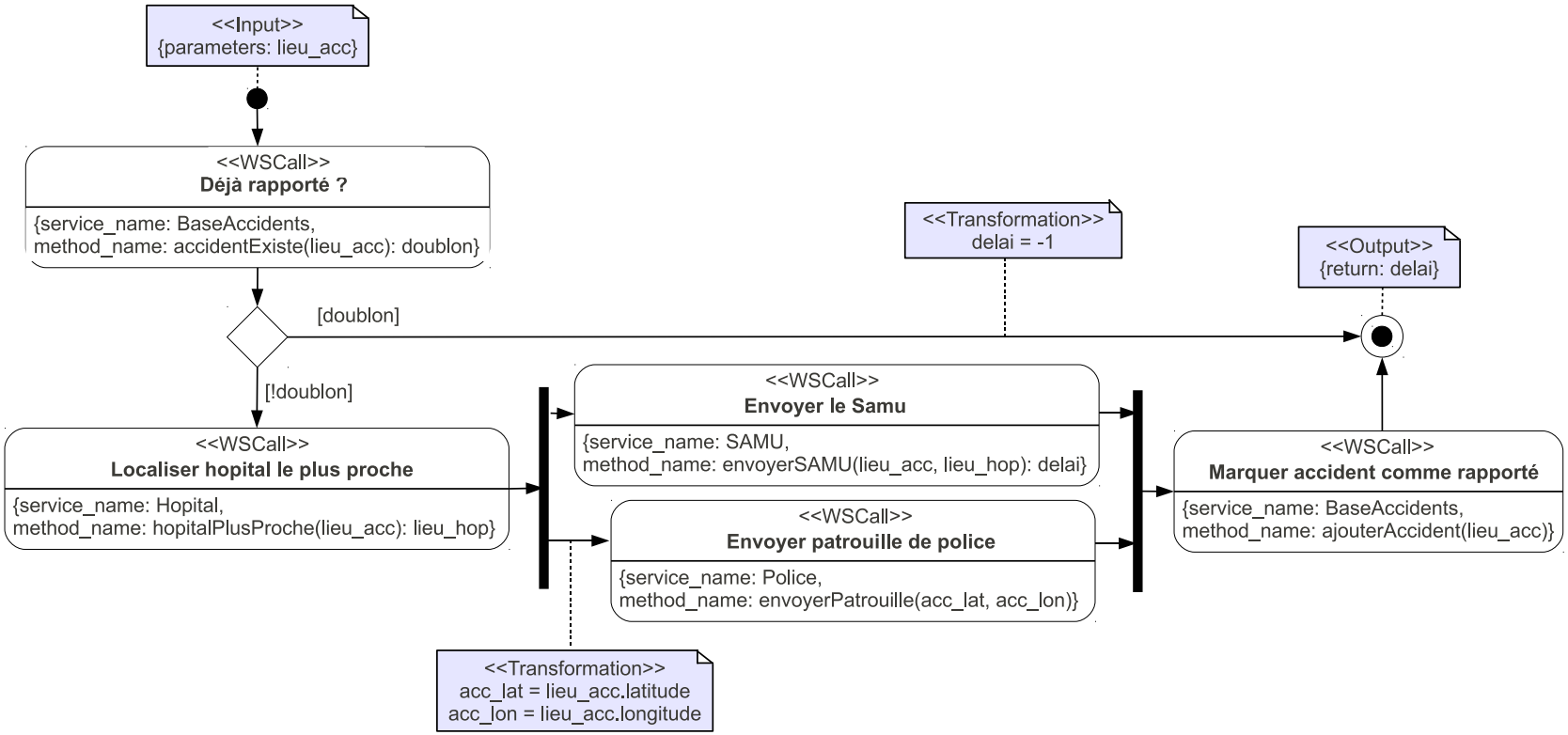
Scénario de test

- 4 services existants :
  - SAMU
  - Police
  - Hôpital
  - Base des accidents
- Un nouveau service dit « *composé* » :
  - 112

UML-S : Diagramme de classes



UML-S : Diagramme d'activité



Propriétés vérifiées

Propriété 1 : une propriété de sûreté

Le processus n'envoie jamais le SAMU ou la Police si l'accident est déjà en cours de traitement.

```
([ DEJA_TRAITE . ENVOI_SAMU ] false) and ([
DEJA_TRAITE . ENVOI_POLICE ] false)
```

Propriété 2 : une propriété de vivacité

A chaque fois qu'un accident est rapporté, le SAMU est toujours envoyé, sauf si l'accident est déjà en cours de traitement.

```
A_inev_B (ACC_RAPPORTE, ENVOI_SAMU or DEJA_TRAITE)
```

Propriétés vérifiées

Propriété 1 : une propriété de sûreté

Le processus n'envoie jamais le SAMU ou la Police si l'accident est déjà en cours de traitement.

```
([ DEJA_TRAITE . ENVOI_SAMU ] false) and ([
DEJA_TRAITE . ENVOI_POLICE ] false)
```

Propriété 2 : une propriété de vivacité

A chaque fois qu'un accident est rapporté, le SAMU est toujours envoyé, sauf si l'accident est déjà en cours de traitement.

```
A_inev_B (ACC_RAPPORTE, ENVOI_SAMU or DEJA_TRAITE)
```

utbm université de technologie Belfort-Montbéliard

Conclusion et Perspectives

- **Résultats**
  - Approche fidèle aux principes du MDA
  - Un profil UML pour modéliser la composition
  - Des règles de transformation :
    - WSDL $\rightarrow$ UML-S
    - UML-S $\rightarrow$ LOTOS
    - UML-S $\rightarrow$ BPEL / WSDL
  - Un environnement de développement
- **Perspectives**
  - Automatisation de la vérification formelle
  - Prise en charge de la chorégraphie de services
  - Reverse Mapping (BPEL $\rightarrow$ UML)

Approche dirigée par les modèles pour la spécification, la vérification formelle et la mise en œuvre des services Web composés

Merci de votre attention.

Publications

- Actes de conférences et workshops (7)
  - A. Roxin, C. Dumez, M. Wack et J. Gaber, *Middleware models for location-based services : a survey*, ICPS Workshops, Proceedings of the 2nd international workshop on Agent-oriented software engineering challenges for ubiquitous and pervasive computing (AUPC'08), 2008.
  - C. Dumez, A. Nait-sidi-moh, J. Gaber et M. Wack, *Modeling and Specification of Web Services Composition Using UML-S*, International Conference on Next Generation Web Services Practices (NWeSP'08), 2008.
  - C. Dumez, J. Gaber et M. Wack, *Model-Driven Engineering of composite Web services using UML-S*, Proceedings of the 10th International Conference on Information Integration and Web-based Applications & Services (iiWAS2008), 2008.
  - C. Dumez, J. Gaber et M. Wack, *Web services composition using UML-S : a case study*, GLOBECOM Workshops, Workshop on Service Discovery and Composition in Ubiquitous and Pervasive Environments (SUPE'08), 2008.
  - A. Roxin, C. Dumez, M. Wack et J. Gaber, *TransportML : a middleware for Location-Based Services collaboration*, NTMS Workshops, 2nd international workshop on Service Computing, Context-aware, Location-aware and Positioning techniques (SCLP'2009), 2009.
  - N. Cottin, C. Dumez, M. Wack et J. Gaber, *SAVE TIME : a Smart Vehicle Traffic Information System*, 8th International Conference of Modeling and Simulation (MOSIM'10), 2010.
  - C. Dumez, M. Bakhouya, J. Gaber et M. Wack, *Formal Specification and Verification of Service Composition using LOTOS*, 7th ACM International Conference on Pervasive Services (ICPS 2010), 2010.

- Journaux scientifiques (1)
  - C. Dumez, M. Bakhouya, J. Gaber et M. Wack, *A Model-Driven Approach to Service Composition Supporting Formal Verification*, soumis à Service Oriented Computing and Applications (SOCA), Springer, 2010