Détection d'anti-patrons d'alignement dans les SI

Vers une approche automatisée

Ali Benjilany [1], Pascal André [1], Hugo Brunelière [2], Dalila Tamzalit [1]

[1]Nantes Université, École Centrale Nantes, [2]IMT Atlantique, CNRS, LS2N, UMR 6004, F-44000 Nantes, France

Prenom.Nom@ls2n.fr

RÉSUMÉ. *Les systèmes d'information ont pour rôle de contribuer à l'efficacité des organisations. L'alignement opérationnel des applications avec le métier est un élément clé de la cohérence de ces systèmes. Dans cet article, nous nous intéressons à la détection de potentielles incohérences dans l'alignement entre le métier, plus précisément des processus métier, et les applications qui les implémentent. Nous proposons de détecter de manière automatisée des anti-patrons d'alignement publiés dans la littérature à travers des règles de détection. L'approche est implantée en Archimate dans l'outil Archi et avec le langage de script* `jArchi` *. Nous illustrons la démarche complète d'alignement sur l'exemple* SoftSlate*, un progiciel d'eCommerce. L'ensemble constitue une proposition outillée qui peut être appliquée dans d'autres contextes.*

ABSTRACT. *The role of information systems is to contribute to the efficiency of organisations. The operational alignment of applications with the business is a key element for the coherence of these systems. In this article, we focus on the detection of potential inconsistencies in the alignment between the business, more specifically business processes, and the applications that implement them. We propose to automatically detect alignment anti-patterns published in the literature using detection rules. The approach is implemented in Archimate in the Archi tool and with the* `jArchi` *scripting language. We illustrate the complete alignment process using the example* SoftSlate*, an eCommerce software system. The whole constitutes a tooled proposal which can be applied in other contexts.*

MOTS-CLÉS : *Systèmes d'information - Architecture d'entreprise - Alignement Métier des SI - Anti-Patrons - Métriques*

KEYWORDS: *Information Systems - Enterprise Architecture - Business-IT Alignment - Anti-Patterns - Metrics*

1. Introduction

Un enjeu crucial des systèmes d'informations (SI) est qu'ils contribuent à l'efficacité des organisations pour lesquelles ils ont été conçus. Établir l'alignement Business-IT (BITA) d'une organisation est une manière d'évaluer la cohérence entre les métiers (*business*) de l'organisation et le système d'information automatisé (*IT*) support. Cet alignement peut se faire à différents niveaux d'abstraction, des systèmes opérationnels à la stratégie d'entreprise comme expliqué dans un article fondateur (Henderson, Venkatraman, 1999). Pour exprimer ces niveaux d'abstraction, nous nous plaçons dans le cadre de l'architecture d'entreprise (Lankhorst, 2013), appelée aussi urbanisation des SI dans la communauté française (Club, 2010 ; Longépé, 2003). On distingue plusieurs couches selon les modèles d'architecture, par exemple 4 pour le Cigref [1], 5 pour Togaf/Archimate (The Open Group, 2013) et 6 pour Zachman (Zachman, 1987). Quel que soit le modèle, on retrouve une vue métier (*business layer*) et une vue applicative (*application layer*). Ces deux vues sont au centre de l'alignement BITA, et font également partie de l'alignement opérationnel entre le métier et l'IT (Henderson, Venkatraman, 1999). L'alignement opérationnel des applications sur le métier est un élément clé de la cohérence des systèmes d'information. Nous appelons ce périmètre *Core Operational BITA (COBITA)* (André *et al.*, 2023).

Dans ces travaux récents, nous avons mis en évidence l'importance de la représentation des liens inter-couches comme pierre angulaire de l'alignement. L'analyse de cet état des lieux a ainsi mis en évidence l'importance des défis qui restent ouverts et le manque crucial d'outillage pour établir une cartographie de l'alignement COBITA d'une organisation mais également pour évaluer son état. Parmi les approches permettant de diagnostiquer la qualité de l'alignement détaillées dans (André *et al.*, 2023), nous nous focalisons ici sur l'évaluation de l'état de cohérence de l'alignement entre la couche métier et la couche applicative. La cohérence signifie sommairement que l'implantation du SI suit la logique de son organisation métier. La cohérence est définie par un ensemble de propriétés qu'il faut vérifier, par exemple, si deux activités d'un processus métier collaborent alors les éléments applicatifs qui les mettent en œuvre doivent être reliés. Vérifier la cohérence de l'alignement peut s'avérer complexe, surtout lorsque plusieurs points de vue sont pris en compte, tels que les données, les fonctions, les domaines, les performances ou la sécurité. Une autre approche est qu'il est parfois plus aisé de détecter les incohérences que de prouver la cohérence. Concrètement, cela se fait souvent par la mise en évidence de contre-exemples issus de la pratique BITA que de chercher à vérifier la cohérence de l'ensemble de l'état d'alignement. C'est la voie que nous avons choisie dans le travail présenté ici. Pour ce faire, nous considérons les anti-patrons d'alignement métier-IT identifiés dans (Gouigoux, Tamzalit, 2021). Même si ce travail offrent une représentation et une analyse de ces anti-patrons, il manque d'assistance pour les détecter. Nous proposons une approche outillée pour combler ce manque.

1. https://www.cigref.fr/cigref_publications/RapportsContainer/Parus2003/2003_-_Accroitre_l_agilite_du_systeme_d_information_web.pdf

L'article est organisé comme suit. Dans la section 2, nous posons le contexte nécessaire à la compréhension de l'article (modélisation, liens d'alignement, anti-patrons). Ces éléments sont ensuite repris et illustrés sur un exemple, le cas d'étude `SoSl`, basé sur le logiciel `SoftSlate`, dans la section 3. Le cœur de la contribution, à savoir la mise en œuvre des anti-patrons et l'implémentation sont détaillées dans la section 4. Enfin, nous en analysons les résultats d'expérimentations sur le cas d'étude dans la section 5. Nous en discutons la validité dans la section 6 et la comparaison avec d'autres travaux en section 7 avant de conclure.

2. Contexte et approche

L'alignement *business/IT* (BITA) est une étape de la démarche d'architecture d'entreprise et d'urbanisation (Aversano *et al*., 2013). Elle permet de cartographier la situation à un instant *t* en visant à obtenir une image fidèle de l'état des sous-ensembles du système d'information. Dans ce contexte, nous nous focalisons sur le cœur de l'alignement, appelé **Core Operational BITA**, CO-BITA en abrégé (André *et al*., 2023). Il s'agit de traiter la relation entre les couches métier et applicative.

Concernant la modélisation des couches, le premier point à fixer est celui du choix du langage de modélisation et le second celui de l'instanciation des modèles. Nous avons déjà identifié que BPMN est le langage majoritaire pour la description des *business processes* de la couche métier tandis que pour la couche application, si UML reste majoritaire, différents types de diagramme sont utilisés (André *et al*., 2023). Pour ce travail, nous avons fait le choix d'utiliser Archimate pour différentes raisons : (i) il s'agit d'un standard pour l'architecture d'entreprise ; (ii) il intègre des langages pour plusieurs couches et des relations génériques entre couches ; (iii) sa couche business est plus épurée que BPMN, (iv) il dispose d'un outil open source de référence `Archi` (même si d'autres outils tels que VisualParadigm ou SmartEA existent sur le marché) ; (v) il dispose d'une extension `jArchi` qui permet d'écrire des scripts. A noter que, pour l'étude de cas traitée dans le papier, les modèles ont été construits manuellement et n'ont pas été automatiquement produits, par exemple par rétro-ingénierie (Aversano *et al*., 2010 ; Pepin *et al*., 2016).

Pour l'évaluation de l'alignement, nous ciblons l'analyse de la cohérence de l'alignement et plus précisément la détection d'incohérences, dans la lignée des travaux visant à définir des **anti-patrons** d'alignement (Gouigoux, Tamzalit, 2021). A l'instar des *anti-patterns* en génie logiciel qui sont des réponses à des erreurs courantes de conception des logiciels (Brown *et al*., 1998), les anti-patrons BITA reprennent des erreurs récurrentes d'alignement. Ces erreurs ont été identifiées sur la base d'études (audit) d'une trentaine de systèmes d'information. Elles ont ensuite été catégorisées sous forme d'anti-patrons. Les auteurs ont identifié 14 anti-patterns BITA dans les systèmes d'information considérés, correspondants chacun à un mauvais scénario récurrent. Ils en présentent quatre et les ont formalisés sous forme de cartes d'identités incluant : Label - Définition - Visualisation - Causes - Conséquences - Effets sur

l'évolution, sur l'utilisation, la facilité et le temps de récupération. Dans les sections suivantes, nous résumons cela aux définitions et visualisations des anti-patterns.

Il manque à cette approche des outils de détection, ce que nous proposons dans la suite de cet article. Mais commençons par présenter le cas d'étude support et sa cartographie des couches métier-applicatif.

3. Un exemple d'alignement COBITA : le cas SoSl

Dans cette section, nous présentons le cas d'étude que nous avons construit sur le progiciel `SoftSlate`, qui permet de créer et personnaliser son propre site web e-commerce. Il s'agit d'une application web développée en Java avec le framework J2EE. Elle présente l'avantage de fournir (i) le code source de développement [2]; (ii) un Guide d'utilisateur et un Guide d'administrateur [3]. Notre objectif est de modéliser les couches métier et applicative du système et cartographier les liens entre ces deux couches, avant de chercher à détecter de manière automatisée des situations de mauvais alignements. Dans la suite nous présentons les modèles métier et applicatif de `SoSl`, ainsi que la cartographie représentative de son état d'alignement.

Modèle métier L'article (Di Francescomarino *et al.*, 2009) propose une découverte des processus métier par analyse dynamique des applications en suivant les actions des utilisateurs. L'approche est illustrée sur le progiciel `SoftSlate`. C'est donc, *a priori* un point d'entrée intéressant pour nous. Nous n'avons pas eu accès à un modèle métier décrit en BPMN mais simplement aux illustrations de l'article. La couche métier est représentée par le processus métier `Softslate Commerce-recovered BPMN` qui décrit les actions que peut réaliser le client comme Login – Register - SaveCartItem. Néanmoins le modèle proposé est insuffisant. Nous avons donc utilisé la documentation `SoftSlate` pour modéliser deux processus métier : Order annoté `BP1` (Ajouter un produit à la liste des produits listés dans la boutique puis passer la commande de ce produit) et Shipping annoté `BP2` (Configurer les coûts de livraison) avec Archi, comme illustré dans la partie haute de la FIGURE 1 [4]. Notre modélisation reste limitée, il manque notamment la partie réservée au client par manque de documentation relative à cet utilisateur du système.

Modèle applicatif Nous avons modélisé la couche applicative en examinant le code source disponible de `SoftSlate`. Le code est composée de cinq packages principaux : Client, Administration, Business Objects, DAO, Installer . Le package Customer et le package Administrator fournissent les fonctionnalités de base du système. Les packages Business Object et DAO sont également importants car ils sont directement utilisés par le package Administrator. Cependant, ces cinq packages contiennent un grand nombre de sous-packages (46 au total), chacun d'entre eux contenant un nombre important de classes. Ainsi, dans le cadre des expérimentations actuelles, nous nous

2. https://www.softslate.com/category/archivedDocs
3. https://www.softslate.com/documentation/userGuide2x.pdf
4. Les deux processus sont détaillés dans la FIGURE 10 et la FIGURE 11 de l'annexe Web[5]

sommes concentrés sur le package Administrator. Il contient un ensemble de classes Java de différents types en suivant le modèle de conception logicielle `Model-View-Controller`. Les classes Servlet implémentent la partie `Controller`, les classes JSP implémentent la partie `View`, tandis que les classes Bean, Processor et DAO implémentent la partie `Model`. De plus, le package Administrator est responsable de l'implémentation des processus métier mentionnés précédemment `BP1` et `BP2`. Le modèle applicatif `ArchiApp` est reporté sur la partie basse de la FIGURE 1. Il est à noter que l'objectif de cette figure est de montrer une vue d'ensemble de l'état d'alignement de notre étude de cas et non de montrer le détail des concepts des deux couches. Ainsi, la représentation des différents liens entre la couche métier et la couche applicative montre une certaine intrication qui reste difficile à analyser manuellement.

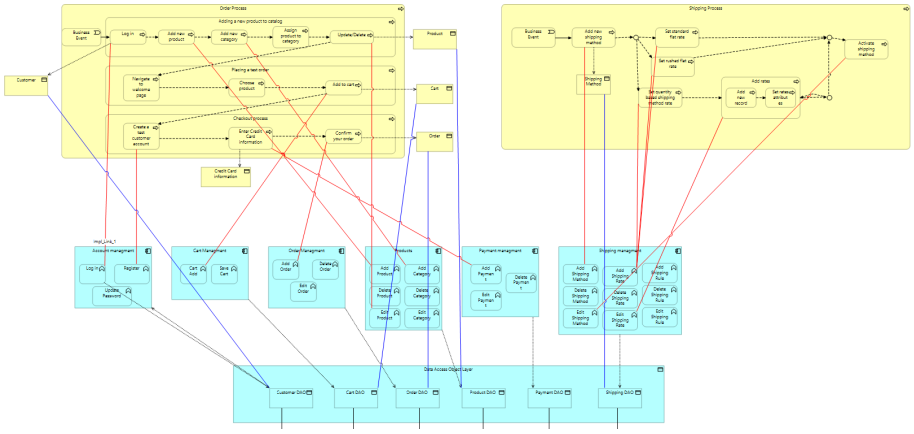

FIGURE 1. *Cartographie d'alignement du cas* `SoS1`.

Cartographie et état de l'alignement du cas `SoS1` Les couches 'métier' et 'application' étant modélisées, nous nous pouvons identifier puis modéliser les liens entre elles. La Figure 1, en plus de la couche métier (partie haute) et la couche applicative (partie basse), modélise les liens que nous avons identifiés entre les deux couches en analysant "manuellement" l'application. Ces liens sont de deux types, définis dans (Benjilany *et al.*, 2024) : *implémentation* (en rouge) qui permet de relier un concept de la couche métier à un concept applicatif qui l'implémente dans la couche applicative, et *représentation* (en bleu) qui permet de relier un concept représentant une donnée (exemple: une commande) de la couche métier à un concept représentant une donnée (exemple : un DAO commande) dans la couche applicative. La cartographie obtenue met en évidence une certaine complexité des liens inter-couches. Tenter d'y déceler à l'oeil humain un ou des anti-patrons d'alignement reste un exercice difficile. Nous appellerons cette cartographie le modèle du cas **`SoS1`**. Nous proposons dans la section suivante une approche automatisée de détection de ces anti-patrons.

4. Anti-Patrons : mise en œuvre d'une détection automatisée

Les algorithmes sont spécifiés en pseudo-code. Pour chacun des anti-patrons, nous donnons la définition et son identité visuelle issues de (Gouigoux, Tamzalit, 2021), un exemple sur le cas d'étude **`SoSl`** et un algorithme pour le détecter. Les algorithmes proposés sont ensuite implantés avec le langage de script *jArchi* pour être exécutés dans l'environnement de modélisation *Archi*. Ces algorithmes sont accessibles sur l'annexe Web dédiée [5].

4.1. Anti-patron AP1: "Pure technical integration"

DÉFINITION 1 (Pure technical integration). — *L'anti-patron Pure technical integration est présent lorsque les applications logicielles s'appellent les unes les autres directement ou par l'intermédiaire d'autres logiciels, sans que ces liens apparaissent au niveau de la couche métier.*

L'identité visuelle de AP1 est représentée dans la partie gauche de la FIGURE 2. L'existence d'un lien entre concepts applicatifs sans que ce lien n'ait d'équivalent entre les concepts métiers correspondants (en lien avec les concepts applicatifs en question) alerte sur une potentielle erreur d'alignement.

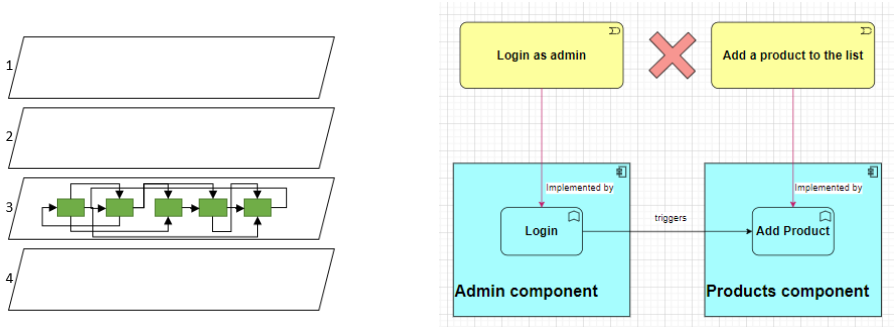

FIGURE 2. *Anti-patron AP1: "Pure technical integration"*

Illustration concrète. La partie droite de la FIGURE 2 montre que la fonction applicative Login déclenche la fonction applicative Add product. Ces deux fonctions implantent respectivement deux processus métiers Login as admin et Add a product to the list. Seulement, le lien de déclenchement existant entre les deux fonctions applicatives n'est pas retranscrit entre les deux processus métiers correspondants.

Algorithme de détection. Dans Listing 1 ci-dessous, la première étape (STEP 1) crée une collection des concepts alignables de la couche métier (notée bfuncConcepts). La deuxième étape (STEP 2) vérifie que chacun de ces concepts est lié directement à un concept de la couche applicative (variable imp. target). La troisième étape (STEP 3)

5. https://uncloud.univ-nantes.fr/index.php/s/5ZrjSnbdb3neBTP

filtre les concepts applicatifs (variable rel . target) liés au concept applicatif imp. target par des liens de déclenchement. La quatrième étape (STEP 4) cherche les concepts métiers (variable impl. source) reliés à chaque concept applicatif filtré dans STEP 3. La dernière étape (STEP 5) vérifie si chaque concept métier issu de STEP 4 est exactement le concept métier en cours de traitement dans STEP 2. Les liens manquants sont une source potentielle d'anti-patron AP1.

Listing 1 – Pseudo-code pour Anti-Patron 1

```
// STEP 1 : Collect all B func concepts
busprocess = GetElement("business-process")
busfunction = GetElement("business-function")
businteraction = GetElement("business-interaction")
bfuncConcepts = Concat(busprocess, Concat(busfunction,
    businteraction))
collection = GetElement(bfuncConcepts)
ForEach concept in collection
 // STEP2: each BP is implemented by at least one application
 ForEach imp in GetRels(concept, 'association-relationship')
  If imp.specialization is 'Implementation link'
  // STEP3: Look for the links between AF1 and AF2, AF i ?
   ForEach rel in GetOutRels(imp.target, 'triggering-
       relationship')
    writeln(imp.target, 'is related by', rel, 'to', rel.target)
    // STEP4: For each AFi: Retrieve the BPi implemented.
    ForEach impl in GetRels(rel.target, 'association-
        relationship')
     If imp.specialization is 'Implementation link'
      writeln('which implements', impl.source)
      // STEP5: Check if BPi is related to BP1
      ForEach flow in GetRels(impl.source, 'flow-relationship')
       If flow.source.id not = concept.id
        writeln('WARNING AP1 detected! There should be a flow
            link FROM', concept, 'TO', flow.source)
       End If
      End ForEach
     End If
    End ForEach
   End ForEach
  End If
 End ForEach
End ForEach
```

4.2. Anti-patron AP2: "The functional silo dedicated IT subsystem"

DÉFINITION 2. — *L'anti-patron "functional SILO" se réfère à l'existence d'un bloc isolé du reste du SI. Un bloc est caractérisé par une succession de tâches métiers reliée à une succession de fonctions applicatives.*

L'identité visuelle d'AP2 est représentée dans la partie gauche de la FIGURE 3. Une partie du système d'information automatisé est complètement isolée du reste. Cela peut être volontaire, comme dans certains contextes réglementaires stricts, mais cela peut également conduire à la duplication de fonctions dans d'autres parties du système, voir à des parties jamais utilisées.

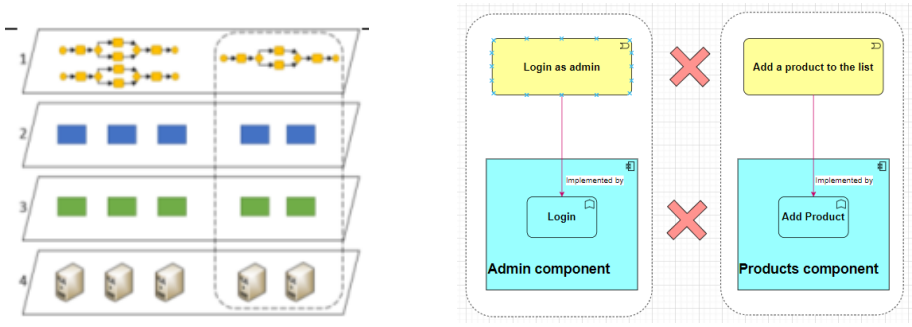

FIGURE 3. *Anti-patron AP2: "The functional SILO dedicated IT subsystem"*

Illustration concrète. La partie droite de la Figure 3 montre deux blocs isolés. Le constat est que ces deux blocs n'ont respectivement aucun lien entre eux. En itérant l'observation pour chaque bloc sur tout le système et, qu'au final, au moins un des blocs est isolé, alors nous sommes face à une illustration de l'anti-patron AP2.

Algorithme de détection. L'algorithme pour détecter AP2 (Listing 3 de l'annexe Web[5]) commence par créer une collection bfuncConcepts des concepts de la couche métier. On vérifie ensuite pour chaque concept métier l'absence de lien (flux ou composition) vers d'autres concepts métiers (implémentation directe ou indirecte). En cas d'absence, si le concept est implanté (présence de liens d'implémentation) par des concepts applicatifs cibles de ces liens et reliés à d'autres concepts applicatifs de type application −function alors ces derniers ne doivent pas être l'implantation d'autres concepts métiers sinon une alerte est levée pour indiquer la présence de l'anti-patron 2 sur le concept métier considéré.

4.3. Anti-patron AP3: "Monolith application"

DÉFINITION 3. — *Une application monolithique se réfère à un seul et même concept applicatif qui implémente plusieurs fonctions métiers distinctes.*

L'identité visuelle pour l'anti-patron 3 est représentée dans la partie gauche de la Figure 4. L'existence d'au moins deux liens partant vers un seul et même concept applicatif alerte sur une éventuelle erreur d'alignement.

Illustration concrète. La partie droite de la FIGURE 4 montre que le concept applicatif Login implémente deux concepts métiers distincts Login as admin et Login as customer. Les deux concepts métier d'authentification sont implémentés dans le même composant applicatif, que ce soit un administrateur ou un client qui sollicite ce service.

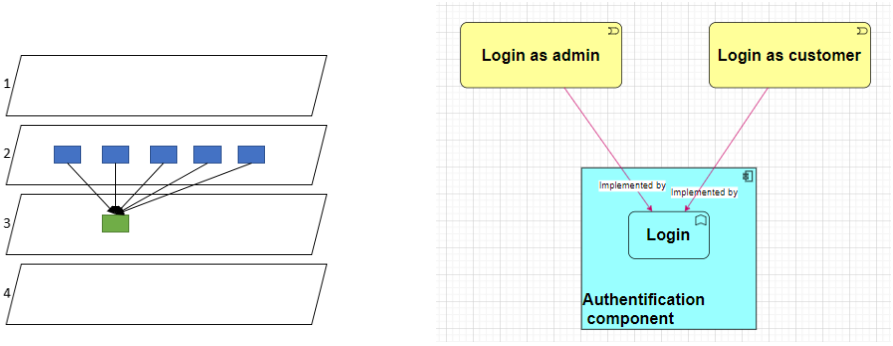

FIGURE 4. *Anti-patron AP3: "Monolith application"*

Algorithme de détection. L'algorithme pour identifier AP3 (Listing 4 de l'annexe Web[5]) crée une collection de tous les concepts que nous considérons dans la couche applicative. Pour chaque concept, on filtre, dans la collection des liens d'association, les liens d'implantation directs ou indirects vers la couche métier. S'il n'y en a pas c'est un problème, s'il y en a plusieurs c'est un potentiel cas d'anti-patron AP3.

4.4. Anti-patron AP4: "Functional multiple implementations"

DÉFINITION 4. — *L'implémentation fonctionnelle multiple se réfère à l'implantation d'un seul concept de la couche métier par plusieurs concepts de la couche applicative. La principale cause évoquée est le manque de compréhension des modèles métiers (partie fonctionnelle en particulier).*

L'identité visuelle d'AP4 est montrée dans la partie gauche de la Figure 5. L'existence de deux ou plusieurs liens partant d'un seul et même concept métier alerte sur une potentielle erreur d'alignement.

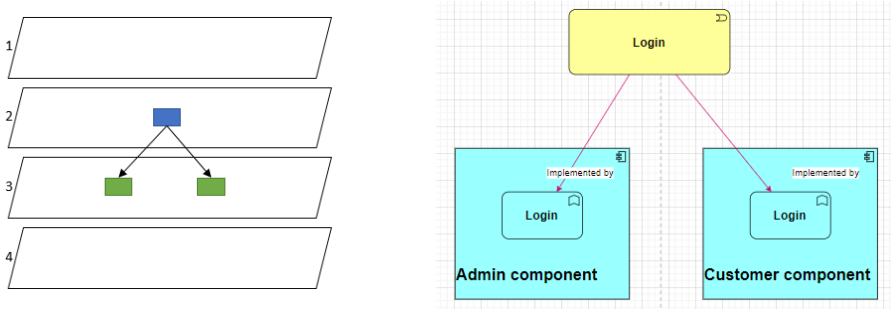

FIGURE 5. *Anti-patron AP4: "The multiple functional implementation"*

Illustration concrète La partie droite de la Figure 5 montre que le concept métier Login est implémenté par deux concepts applicatifs Login du composant applicatif Admin et Login du composant applicatif Customer. Dans le travail initial, l'exemple

donné est celui du concept métier reporting qui est implémenté de 40 façons différentes par les applications concernées. Dans une cartographie manuelle où plusieurs autres liens sortant/entrant du/au même concept sont représentés, repérer visuellement une telle situation nécessite beaucoup d'efforts, est de surcroît aléatoire et peut même s'avérer impossible dans certains cas.

Algorithme de détection : L'algorithme pour identifier AP4 ((Listing 4 de l'annexe Web[5]) est composé de deux étapes. La première crée une collection de tous les concepts de la couche métier à traiter. Pour chaque concept on détecte les liens manquants vers la couche applicative, ou les liens multiples d'implantation, source potentielle d'un anti-patron AP4.

5. Expérimentation et Analyse des résultats

Dans cette section, nous présentons les expérimentations réalisées sur le cas d'étude **SoSl** avec les algorithmes décrits précédemment. Dans un premier temps nous commentons et analysons les premiers résultats obtenus par l'outil de détection qui implante les algorithmes en `jArchi`. Dans un second temps, nous améliorons la confiance dans les algorithmes par une analyse de mutation.

Application au cas SoSl, commentaires et premières analyses L'exécution des scripts lève une alerte seulement pour AP3, *cf.* FIGURE 6. Une alerte "WARNING" est levée car le concept application Add Shipping Rate possède 3 liens d'implémentation vers 3 concepts métiers différents. Les autres situations ne détectant pas AP3 sont ignorées.

```
 The A-func-Concept :  Add Shipping Method  has : 1 implementation links

 The A-func-Concept :  Edit Shipping Method  has : 1 implementation links
The A-FUNC-CONCEPT Delete Shipping Method IS NOT IMPLEMENTING ANY BFC

 The A-func-Concept :  Add Shipping Rate  has : 3 implementation links
WARNING Monolith application BITA anti-pattern
```

FIGURE 6. *Détection d'AP3 sur* `SoSl`

Ces résultats peuvent être interprétés de plusieurs façons. Une première façon consiste à en tirer des indications sur le niveau d'alignement du système : l'existence d'une seule occurrence de lien d'implémentation peut être interprétée comme indicateur d'un relativement bon alignement du système. Cependant, il est difficile d'avancer un taux d'alignement réaliste sur cette seule base. Par exemple, nous proposons déjà par ailleurs d'évaluer de manière complémentaire le niveau d'alignement sur la base d'un ensemble de métriques et de règles de cohérence (Benjilany *et al.*, 2024).

Deux aspects peuvent également impacter le résultat de nos expérimentations: (i) *Qualité de la modélisation.* Le modèle `SoSl` est incomplet (deux processus métiers seulement) et non validé (notre modèle n'est pas contrôlé par des équipes métiers

et des équipes de développement). De plus, nous ne sommes pas à l'abri d'erreurs d'interprétation de notre part. (ii) *Qualité des algorithmes de détection.* Les algorithmes de détection que nous proposons peuvent ne pas détecter des erreurs d'alignement présentes dans le système (faux négatifs) ou détecter des erreurs qui, en réalité, n'en sont pas (faux positifs). Pour pallier le potentiel manque d'efficacité des algorithmes, nous nous inspirons des techniques de mutation dans le test logiciel pour créer différentes situations de mauvais alignement permettant de tester les algorithmes.

Validation par mutation Pour vérifier si les anti-patterns sont détectés, nous avons donc créé des mutants du modèle initial `SoSl` qui contiennent des erreurs d'alignement. Pour tester chaque algorithme de détection d'anti-patron (AP), nous avons défini deux mutants : un premier mutant `A` où nous injectons volontairement une simple occurrence de l'AP ciblé (v1) et un deuxième mutant `B` où nous injectons volontairement soit une occurrence plus complexe soit plusieurs occurrences de l'AP ciblé (v2). Le TABLEAU 1 présente les mutants et les résultats de ces tests. Ces mutants, ainsi que le modèle `SoSl` de base, sont tous accessibles depuis l'annexe Web[5]. Les quatre dernières colonnes rapportent le nombre d'erreurs relevées pour chaque exécution des scripts de détection d'AP sur chacun des mutants. Il est à noter que l'AP3 est détecté sur tous les mutants par héritage du modèle de base ; il n'y a donc pas eu de régression. Les résultats montrent que toutes les occurrences d'anti-patrons injectées ont été détectées et qu'aucun faux négatif n'est apparu.

TABLEAU 1. *Démarche des expérimentations et résultats*

Base / Mutant	**Cible Variante**	**Modification(s) effectuées sur le modèle de base**	**Résultats**			
			AP1	AP2	AP3	AP4
`SoSl`	base héritée	Aucune	0	0	1	0
`SoSl 1A`	AP1 v1	Supprimer le lien flux entre Log In et Add Product	1	0	1	0
`SoSl 1B`	AP1 v2	Injecter un nouveau BP LIS	2	0	1	0
`SoSl 2A`	AP2 v1	Injecter et Isoler BP LIS implémenté par AF isolé	0	1	1	0
`SoSl 2B`	AP2 v2	Injecter et isoler "Isolated BP" implémenté par "Isolated AF"	0	2	1	0
`SoSl 3A`	AP3 v1	AF Login implémente 2 BP	0	0	2	0
`SoSl 3B`	AP3 v2	AF Login implémente 4 BP	0	0	2	0
`SoSl 4A`	AP4 v1	BP Login implémenté par 2 AF	0	0	1	1
`SoSl 4B`	AP4 v2	BP Login implémenté par 5 AF	0	0	1	1

Illustrons maintenant sur des exemple pour AP1, AP2 et AP4.

La FIGURE 7 montre le résultat de l'exécution du script pour identifier AP1 sur le mutant représenté en partie droite de la FIGURE 2. Un message apparaît pour annoncer qu'un lien (flux) est manquant entre les deux concepts métiers Login as admin et Add new product.

La Figure 8 montre le résultat de l'exécution du script pour identifier AP2 sur le mutant représenté en partie droite de la FIGURE 3. Nous pouvons y voir une alerte "WARNING" notifiant d'une éventuelle erreur d'alignement relative au concept métier Login as seller . Les autres concepts métiers, en revanche, ne sont pas concernés.

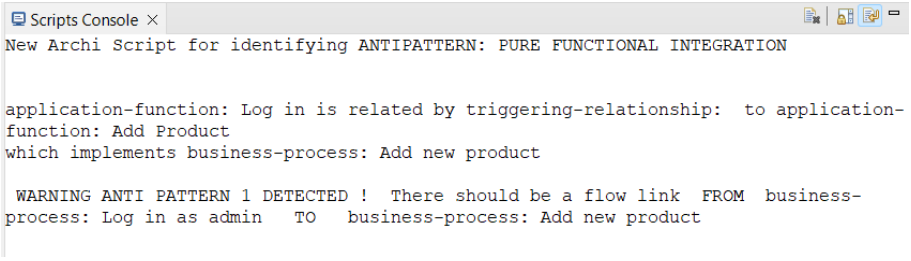

FIGURE 7. *Détection d'AP1 sur* `SoS1 1A`

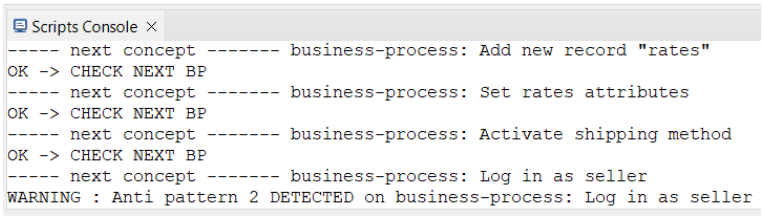

FIGURE 8. *Détection d'AP2 sur* `SoS1 2A`

La figure 9 montre le résultat de l'exécution du script pour identifier AP4 sur le mutant représenté en partie droite de la FIGURE 5. Nous pouvons y voir qu'une alerte "WARNING" indiquant que le concept métier Login est implémenté par deux liens.

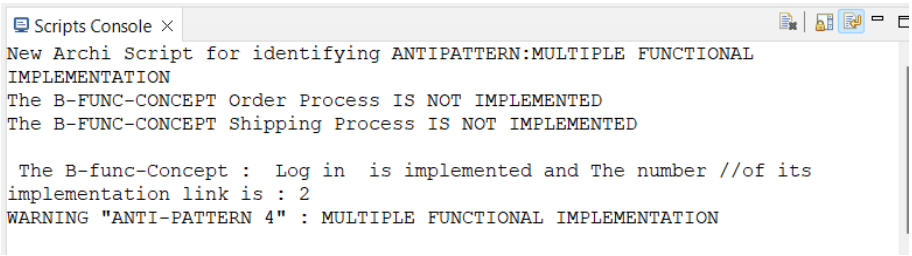

FIGURE 9. *Détection d'AP4 sur* `SoS1 4A`

6. Discussion

Dans cette section, nous évaluons plus en détail les résultats obtenus lors des expérimentations présentées dans la Section 5 afin d'identifier les limites actuelles de notre proposition. Sur cette base, nous indiquons plusieurs pistes possibles d'amélioration.

Évaluation et limitations Pour être applicable, notre proposition requiert l'existence de modèles des couches métier et applicative. Cependant, de tels modèles ne sont pas toujours disponibles (notamment les modèles métiers) et doivent souvent être construits à partir de différentes sources d'information. Pour notre cas d'étude, nous disposions uniquement de la documentation utilisateur et du code source de l'application. A partir de ces éléments, nous avons construit manuellement les modèles métier

et applicatif en utilisant le langage de modélisation Archimate. Une telle approche s'avère coûteuse en temps dans le cadre d'un système d'information de taille plus conséquente. Ainsi, les expérimentations réalisées jusqu'à présent montrent l'applicabilité de notre proposition. En revanche, son passage à l'échelle reste encore à valider. Les résultats obtenus démontrent également que notre proposition permet de détecter les occurrences potentielles des quatre AP supportés dans le contexte d'un cas d'étude comme **`SoSl`**. Cependant, la seule information concernant la possible présence ou non d'AP n'est pas suffisante pour pouvoir statuer sur l'état d'alignement du système étudié. Les anti-patrons ne constituent qu'un outil parmi d'autres permettant de déterminer le niveau d'alignement d'un système (André *et al*., 2023). En effet, ceux-ci doivent être complétés par l'utilisation de métriques et/ou de règles de cohérence architecturale (par exemple) afin de pouvoir calculer un véritable taux d'alignement. Enfin, nous n'avons traité qu'un cas d'étude jusqu'à présent. D'autres expérimentations seront nécessaires pour augmenter la confiance en nos algorithmes.

Améliorations possibles Une première piste d'amélioration consiste à élargir le spectre des langages de modélisation pouvant être utilisés dans notre proposition. Ainsi, en complément d'Archimate, d'autres standards de modélisation tels que BPMN et UML pourraient être intégrés pour supporter les couches métier et applicative respectivement. Cependant, cela nécessite de disposer d'un environnement de modélisation en mesure de permettre l'utilisation conjointe de ces différents langages. Ce n'est pas le cas de l'outil de modélisation Archi sur lequel repose actuellement notre proposition. Une seconde piste d'amélioration consiste à perfectionner les algorithmes de détection des anti-patrons. Nous fournissons pour le moment quatre algorithmes supportant quatre AP distincts. Les premières expérimentations ont permis de s'assurer de leur fonctionnement individuel. Cependant, leur complexité actuelle pourrait être réduite afin de permettre leur utilisation dans le contexte de cas de taille plus conséquente. Un travail d'optimisation des algorithmes (e.g., concernant le nombre de boucles imbriquées) est donc à prévoir lors de nos prochaines étapes. En parallèle, la combinaison de plusieurs algorithmes peut être envisagée pour des raison de performance mais aussi dans l'optique de l'identification d'AP possiblement interdépendants. Enfin, ces algorithmes pourraient être revus afin d'enrichir leurs résultats, e.g., pour fournir des recommandations d'évolution concernant l'une ou l'autre des couches concernées. Une troisième piste d'amélioration consiste à compléter la librairie d'outils à disposition pour évaluer l'état de l'alignement. Des outils complémentaires, tels que des métriques ou encore des règles de cohérence, pourraient être utilisés. Là encore, leur combinaison avec les algorithmes déjà disponibles devrait être travaillée afin de permettre le calcul d'un taux d'alignement le plus représentatif possible de la réalité.

7. Travaux connexes

En partant de la revue de littérature la plus récente sur le sujet du *mis-alignment* (Őri, Szabó, 2024), nous positionnons notre approche suivant deux axes : (i) le niveau d'outillage disponible et (ii) les techniques de détection utilisées.

Selon cette revue, six travaux proposés rentrent dans le périmètre de l'alignement opérationnel COBITA. Parmi ceux-ci, quatre s'intéressent spécifiquement à la détection du non-alignement. Une thèse évoque le non-alignement entre des objectifs stratégiques et opérationnels (Singh, 2009). Bien que partiellement outillée, l'approche proposée se différencie de notre proposition par son positionnement au niveau de la couche stratégique et non des processus métiers. Un autre travail présente les résultats d'une investigation réalisée auprès d'employés de 7 départements d'une entreprise (Peng *et al*., 2021). Sur la base de cette investigation, un modèle de non-alignement est proposé. Cependant, contrairement à notre proposition, ce modèle n'est pas accompagné d'un support outillé permettant sa mise en œuvre. De manière similaire, une étude recense des pratiques de gestion visant à atténuer les facteurs de risque de non-alignement (Mamoghli *et al*., 2015). Cette fois encore, les pratiques identifiées ne sont pas supportées par de l'outillage technique associé. Une dernière approche propose des recommandations d'amélioration, en plus de la détection (Chen *et al*., 2005). Elle repose sur un protocole en 12 étapes : Les étapes 1 à 9 décrivent comment déterminer une situation de non-alignement, les étapes 10 à 12 permettent de proposer des scénarii de ré-alignement. La démarche est intéressante et les étapes 1 à 9 partagent des objectifs commun avec notre proposition. En revanche, les étapes 10 à 12 sont hors du contexte de notre proposition actuelle. De plus, l'approche globale proposée n'est une nouvelle fois pas outillée.

D'autres approches proposent également différentes techniques de détection de patrons. Par exemple, une approche propose de traiter le problème de l'alignement stratégique via un algorithme pour la détection, la correction et la prévention des situations de non-alignement dans des modèles d'Architecture d'Entreprise (Aseeva *et al*., 2022). L'algorithme proposé s'inspire de la méthodologie Architecture Development Method (ADM) du standard TOGAF, prenant en compte des éléments de niveau stratégique. Notre proposition se différencie donc de cette approche de par son accent mis sur les couches métier et applicative. De plus, l'algorithme proposé n'est pas encore implémenté contrairement à ceux que nous proposons. Un autre travail, s'appuyant sur l'analyse manuelle de plusieurs cas d'étude, propose un *framework* de quatres patrons de conception dans un contexte d'alignment Business-IT (Cleven, 2011). Mais l'objectif principal de ce *framework* est l'évaluation de la performance des processus, et non de l'alignement comme dans notre proposition. De plus, là encore, la solution proposée reste théorique et aucune implémentation technique n'est fournie. Dans la même veine, un travail complémentaire s'intéresse également au problème de la performance dans un contexte d'applications d'Entreprise (Wert *et al*., 2014). Ainsi, cinq heuristiques sont proposées afin de détecter, sur la base de mesures, des anti-patrons de performance connus. Ces heuristiques sont outillées par le biais d'un *framework* technique appelé Dymanic Spotter. Cependant, l'accent est mis encore une fois sur l'aspect performance et non sur l'évaluation plus générale de la qualité de l'alignement. D'un point de vue Génie Logiciel, la détection d'anti-patrons a aussi été étudiée dans le contexte de processus de développement logiciel (Pícha *et al*., 2022). L'outil nommé Software Process Anti-patterns Detector (SPADe) permet une telle vérification sur la base de descriptions semi-formelles d'anti-patrons fournies grâce à un *template*. Ce-

pendant, contrairement à notre proposition, les anti-patrons ainsi définis et détectés ne concernent que les couches basses (e.g., applicative, technologique) sans prendre en compte la couche métier.

8. Conclusion

Dans ce papier, nous nous sommes focalisés sur l'évaluation de l'état de cohérence de l'alignement entre la couche métier et la couche applicative, aussi appelé Core Operational BITA (COBITA). Sur la base d'anti-patrons d'alignement identifiés dans des travaux précédents, nous avons proposé un catalogue initial d'algorithmes permettant la détection automatisée d'un certain nombre de ces anti-patrons. Nous avons pu expérimenter en pratique avec l'implémentation de ces algorithmes dans le cadre d'un cas d'étude concret que nous avons au préalable modélisé.

Ces travaux constituent une étape vers le support automatisé pour une évaluation plus globale du COBITA. En effet, le catalogue d'algorithmes déjà fourni pourrait être complété afin de couvrir un ensemble plus large d'anti-patrons. De plus, l'approche proposée pourrait être étendue afin de supporter d'autres langages standards de modélisation, en complément d'Archimate. Enfin, un effort reste encore à faire afin d'être en mesure de combiner ces algorithmes de détection d'anti-patrons avec d'autres techniques telles que des métriques ou des règles de cohérence. L'objectif final est de permettre le calcul d'un taux global d'alignement le plus réaliste possible.

Bibliographie

André P., Tamzalit D., Benjilany A., Bruneliere H. (2023). A review of core operational business-it alignment. In *ISD 2023 proceedings*, p. 1–12. Lisbon, Portugal, AIS eLibrary.

Aseeva N., Babkin E., Malyzhenkov P., Masi M. (2022). Strategic integration of alignment models for the it-business misalignment detection and redress. In *Digitalization of society, economics and management: A digital strategy based on post-pandemic developments*, p. 97–113. , Springer.

Aversano L., Grasso C., Tortorella M. (2010). Measuring the alignment between business processes and software systems: A case study. In *SAC '10*, p. 2330–2336. , ACM.

Aversano L., Grasso C., Tortorella M. (2013). A literature review of Business/IT alignment strategies. In *Enterprise information systems*, p. 471–488. , Springer.

Benjilany A., André P., Tamzalit D., Bruneliere H. (2024, avril). Towards a link mapping and evaluation approach for Core Operational Business-IT Alignment. In *26th International Conference on Enterprise Information Systems (ICEIS 2024).* Angers, France, Insticc.

Brown W. H., Malveau R. C., McCormick H. W. S., Mowbray T. J. (1998). *Antipatterns: Refactoring software, architectures, and projects in crisis* (1st éd.). USA, John Wiley & Sons, Inc.

Chen H.-M., Kazman R., Garg A. (2005). Bitam: An engineering-principled method for managing misalignments between business and it architectures. *Science of Computer Programming*, vol. 57, n° 1, p. 5–26.

Cleven A. (2011). Exploring patterns of business-it alignment for the purpose of process performance measurement. In *European conference on information systems.* , . Consulté sur https://api.semanticscholar.org/CorpusID:14754780

Club U.-E. (2010). *Urbanisme des si et gouvernance: Bonnes pratiques de l'architecture d'entreprise*. , Dunod.

Di Francescomarino C., Marchetto A., Tonella P. (2009). Reverse engineering of business processes exposed as web applications. In *2009 13th european conference on software maintenance and reengineering*, p. 139–148. , .

Gouigoux J., Tamzalit D. (2021). Business-it alignment anti-patterns: A thought from an empirical point of view. In E. Insfrán *et al.* (Eds.), *Information systems (ISD2021 proceedings).* Valencia, Spain, Universitat Politècnica de València / Association for Information Systems.

Henderson J. C., Venkatraman H. (1999). Strategic alignment: Leveraging information technology for transforming organizations. *IBM Systems*, vol. 38, n° 2.3, p. 472–484.

Lankhorst M. M. (2013). *Enterprise architecture at work - modelling, communication and analysis (3. ed.).* , Springer.

Longépé C. (2003). *The enterprise architecture it project: the urbanisation paradigm.* , Elsevier.

Mamoghli S., Goepp V., Botta-Genoulaz V. (2015). An operational "risk factor driven" approach for the mitigation and monitoring of the "misalignment risk" in enterprise resource planning projects. *Computers in Industry*, vol. 70, p. 1–12.

Őri D., Szabó Z. (2024). A systematic literature review on business-it misalignment research. *Information Systems and e-Business Management*, p. 1–31.

Peng G., Chen S., Chen X., Liu C. (2021). An investigation to the industry 4.0 readiness of manufacturing enterprises: The ongoing problems of information systems strategic misalignment. *Journal of Global Information Management (JGIM)*, vol. 29, n° 6, p. 1–20.

Pepin J., André P., Attiogbé J. C., Breton E. (2016). An improved model facet method to support EA alignment. *Complex Systems Informatics and Modeling Quarterly*, vol. 9, p. 1–27.

Pícha P., Hönel S., Brada P., Ericsson M., Löwe W., Wingkvist A. *et al.* (2022). Process anti-pattern detection–a case study. In *Proceedings of the 27th european conference on pattern languages of programs*, p. 1–18. , .

Singh S. N. (2009). *A goal-based requirements gathering approach to detect and understand business-it misalignments*. Thèse de doctorat non publiée, University of British Columbia.

The Open Group. (2013). *Archimate 2.1 specification.* , Van Haren Pub.

Wert A., Oehler M., Heger C., Farahbod R. (2014). Automatic detection of performance anti-patterns in inter-component communications. In *International acm sigsoft conference on quality of software architectures.* , . Consulté sur https://api.semanticscholar.org/CorpusID:2212843

Zachman J. A. (1987). A framework for information systems architecture. *IBM systems journal*, vol. 26, n° 3, p. 276–292.