

Comment les solutions FileMaker s’effondrent sous leur propre logique

Référence d’article

Article complet

<https://b-ziq.com/fr/blog/filemaker-defaillances-structurelles/>

Doctrine fondamentale

- Les violations de contraintes produisent des séquences de défaillances prévisibles et mécaniques.
- Un système s’effondre lorsque son comportement ne peut plus être raisonné à partir du code seul.
- Chaque mécanisme de défaillance est la conséquence directe d’une contrainte violée.
- Les défaillances ne sont pas accidentelles ; elles sont structurellement inévitables dans les conditions qui les produisent.
- L’effondrement est l’état dans lequel un système continue de fonctionner mais ne peut plus être modifié en toute sécurité.
- Chaque violation introduit une dépendance que le système ne peut pas voir.
- À mesure qu’un système grandit, les dépendances cachées s’accumulent jusqu’à la perte de prévisibilité.

Définitions

- **Contexte implicite** — Comportement dépendant de conditions d’exécution non déclarées telles que le modèle, l’enregistrement actif, l’ensemble trouvé ou l’origine de l’appel.
- **Mutation distribuée** — Situation où une rubrique possède plusieurs points de mise à jour appliquant une logique indépendante par des chemins d’exécution distincts.
- **Couplage implicite** — Dépendance entre composants fondée sur des hypothèses non déclarées existant en dehors des contrats explicites.
- **État ambiant** — État hérité du contexte d’exécution plutôt que reçu explicitement en entrée.
- **Logique dans l’interface** — Règle métier dont la propriété appartient à un chemin d’exécution de l’interface plutôt qu’à une couche universellement traversée.
- **Effondrement** — Condition dans laquelle un système continue de fonctionner mais ne peut plus être modifié en toute sécurité parce que son comportement n’est plus prévisible.

Contraintes

Contrats explicites

- Les scripts doivent déclarer les conditions requises pour leur exécution correcte.
- Les scripts ne doivent pas dépendre d'un contexte d'exécution non déclaré.
- Les conditions requises doivent être vérifiées avant l'exécution du comportement.

Mutation des données contrôlée

- Une rubrique doit avoir un propriétaire unique pour sa mutation.
- La validation doit s'exécuter par le même chemin propriétaire que la mutation.
- La logique de mutation ne doit pas être distribuée entre plusieurs points de mise à jour indépendants.

Séparation des responsabilités

- Les composants ne doivent pas dépendre d'hypothèses non déclarées créées par d'autres composants.
- Les dépendances comportementales doivent être exprimées explicitement.
- L'ordre d'exécution ne doit pas être le mécanisme qui transporte les exigences comportementales.

Gestion explicite de l'état

- L'état requis par un comportement doit être reçu explicitement.
- Les scripts ne doivent pas dépendre d'un état de session hérité.
- Le comportement ne doit pas dépendre de variables globales, de l'historique d'exécution ou d'un contexte ambiant.

Séparation des couches

- Les règles métier ne doivent pas appartenir à la couche interface.
- L'application des règles doit exister dans une couche traversée par tous les chemins d'exécution.
- L'application des règles ne doit pas dépendre d'un point d'entrée spécifique.

Invariants

- Un script qui dépend d'un contexte non déclaré produit un comportement erroné dès que le contexte d'exécution diffère de celui qu'il suppose.
- Une rubrique ayant plusieurs points de mise à jour n'a pas d'état final déterministe.
- Un système dont les composants dépendent d'hypothèses non déclarées ne peut pas être modifié ou étendu en toute sécurité.
- Un script qui lit un état qu'il n'a pas reçu en entrée produit des résultats incohérents selon les contextes d'exécution.
- Une règle métier située dans la couche interface n'est appliquée que lorsque l'interface est impliquée.

- Un comportement dépendant d'un contexte caché, d'un état caché, d'une mutation non contrôlée ou d'une logique liée à un point d'entrée devient moins prévisible à mesure que le système grandit.
- La prévisibilité du système diminue à mesure que les dépendances invisibles s'accumulent.

Validation

Contexte implicite

Une violation existe lorsque :

- le comportement dépend du contexte de modèle ;
- le comportement dépend de l'enregistrement actif ;
- les conditions requises sont supposées plutôt que déclarées ;
- le script produit un comportement différent lorsqu'il est invoqué depuis un autre point d'entrée.

Mutation distribuée

Une violation existe lorsque :

- plusieurs chemins d'exécution écrivent dans la même rubrique ;
- la validation s'exécute dans certains chemins d'écriture mais pas dans d'autres ;
- la valeur finale dépend de l'ordre d'exécution.

Couplage implicite

Une violation existe lorsque :

- un composant requiert des hypothèses établies par un autre composant ;
- des dépendances existent uniquement à travers la séquence d'exécution ;
- le comportement change lorsque la chaîne d'appels est réorganisée.

État ambiant

Une violation existe lorsque :

- le comportement dépend de variables globales ;
- le comportement dépend d'un état de session hérité ;
- le comportement dépend d'un état qui n'a pas été reçu explicitement.

Logique dans l'interface

Une violation existe lorsque :

- des règles métier ne s'exécutent qu'à travers une interaction de l'interface ;
- des chemins d'exécution non interactifs contournent l'application des règles ;
- l'application des règles diffère selon le chemin d'exécution.

Conséquences des violations

Contexte implicite

- Le comportement s'exécute dans un contexte non prévu.
- Les défaillances deviennent difficiles à rattacher à leur cause.
- La réutilisation augmente la surface de défaillance sans rendre les hypothèses plus visibles.

Mutation distribuée

- L'état final devient non déterministe.
- La validation devient incohérente.
- L'auditabilité devient peu fiable.
- La propriété du comportement devient introuvable.

Couplage implicite

- La réutilisation devient risquée.
- Le refactoring exige la reconstruction de chaînes de dépendances cachées.
- Le coût du changement augmente avec la croissance des chaînes de dépendances.

État ambiant

- Les défaillances deviennent intermittentes.
- Le comportement varie selon le contexte d'exécution.
- Les anomalies ne peuvent plus être reproduites ni déboguées de manière fiable.
- Les chaînes de dépendances deviennent invisibles.

Logique dans l'interface

- Les règles métier sont appliquées de façon incohérente.
- Des données invalides entrent dans le système par les chemins de contournement.
- La duplication des règles augmente avec le temps.
- Les écarts d'application s'élargissent à mesure que les chemins d'exécution se multiplient.

Conséquences systémiques

- Le comportement d'exécution devient imprévisible.
- Les dépendances doivent être reconstruites manuellement avant chaque modification.
- Les tests deviennent exhaustifs parce que l'impact d'un changement ne peut plus être déduit de la structure.
- Le développement ralentit à mesure que le risque augmente.
- Le système cesse d'être développé et est maintenu autour de sa propre fragilité.
- La récupération finit par exiger un remplacement plutôt qu'une modification.