# *De la balise au token : structure, représentation et preuve à l'ère agentique*

Frank Adebiaye · 28 octobre 2026

**Note.** Version révisée le 28 octobre 2026. Apple a annoncé le 27 octobre 2026 le Keynote Interchange Format (.keyx) et la prise en charge de MCP dans Keynote, Pages et Numbers, attendus en novembre (MCP sous macOS 27). À cette date, la spécification de .keyx n'est pas publiée : les passages qui en décrivent la structure interne ou les outils MCP sont présentés comme des hypothèses.

## Résumé exécutif

L'histoire de l'ingénierie documentaire a longtemps oscillé entre deux impératifs : l'interopérabilité sémantique (incarnée par le XML et le SGML) et le confinement propriétaire garantissant la fidélité de rendu (les formats binaires, les conteneurs compressés et les architectures DOM fermées). L'émergence des modèles de fondation (Large Language Models) et des architectures agentiques rebat les cartes de manière radicale. En pénalisant les représentations verbeuses ou opaques dès qu'il faut les soumettre à un modèle, l'économie du token favorise de nouveaux paradigmes : le Markdown linéaire, le JSON contraint, les formats de composition programmatique (Typst) et les structures réactives (CRDT, notebooks en scripts purs).

Cette mutation fragilise les forteresses graphiques du siècle passé (InDesign), tend à reléguer le PDF au rôle d'artefact de sortie, et révèle le décalage entre les protocoles légaux hérités — des formats de facturation électronique (Factur-X, UBL/CII) aux usines à gaz du reporting financier (XBRL, iXBRL) — et la réalité économique des chaînes de traitement agentiques. Enfin, l'annonce par Apple du format .keyx, ouvert et fondé sur JSON, et de la prise en charge de MCP (Model Context Protocol) dans sa suite bureautique en offre un signal fort : celui d'un document pensé pour être lu et modifié par des agents, et non plus seulement restitué par une application.

Le format de fichier ne disparaît pas pour autant : il cesse d'être la représentation unique de l'information pour devenir une interface de sérialisation parmi d'autres, et XML perd son monopole de représentation universelle. Reste une limite, que cet essai voudrait tenir jusqu'au bout : ce qui sert l'agent ne suffit pas toujours à faire preuve.

## 1. La parabole des 23 ans : de Steve Jobs à l'agent autonome

En janvier 2003, lors de l'introduction de Keynote 1.0, Steve Jobs avançait un argument d'architecture alors avant-gardiste : un format de fichier ouvert, entièrement fondé sur XML (l'APXL, Apple Presentation XML), au sein d'un paquet inspectable. Il évoquait déjà des

applications tierces capables de produire automatiquement des présentations à partir de données externes : l'ambition ne visait pas seulement l'interopérabilité, mais l'automatisation documentaire.

Ce projet d'ouverture fonctionnelle s'est pourtant heurté, au fil des deux décennies suivantes, aux impératifs d'optimisation locale des suites bureautiques :

- Sérialisation binaire opaque (fichiers .iwa basés sur Protobuf compressé avec Snappy dans iWork),
- Paquets fermés ou conteneurs monolithiques,
- Écosystèmes de formats orientés quasi exclusivement vers le moteur de rendu de l'application hôte.

Le 27 octobre 2026, Apple annonce le Keynote Interchange Format (.keyx), un format de paquet ouvert fondé sur JSON, conçu pour être lisible par les outils externes, les chaînes de développement et les agents d'IA, ainsi que la prise en charge de MCP (Model Context Protocol) dans Keynote, Pages et Numbers. Rien n'est encore disponible : les mises à jour sont attendues en novembre, et MCP exigera macOS 27. La boucle semble se refermer, 23 ans plus tard. Le cas d'usage décrit par Jobs en 2003 est réactivé, mais le destinataire de la chaîne de production a changé de nature : ce n'est plus seulement un parseur déterministe programmé à la main, c'est aussi un agent d'inférence probabiliste.

## 2. Anatomie d'une rupture annoncée : ce que .keyx laisse entrevoir

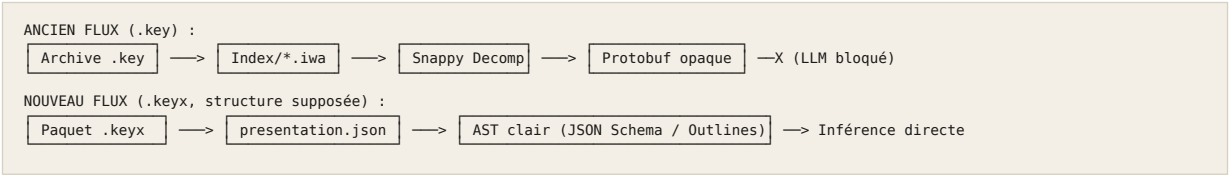
Si les annonces se confirment, .keyx ne sera pas une simple mise à jour incrémentale de la suite iWork, mais une refonte architecturale qui illustre les exigences de la pile logicielle agentique. Il faut toutefois séparer ce qui est établi (le format actuel, l'annonce d'Apple) de ce qui reste à confirmer (la structure et les outils du nouveau format).

**Le bourbier des archives .iwa**
Depuis la refonte majeure d'iWork en 2013, un document Keynote (.key) n'était plus un paquet XML inspectable. C'était une archive ZIP renfermant deux mondes étanches :

- Un répertoire Data/ stockant les ressources brutes (médias matriciels et vectoriels).
- Un répertoire Index/ composé d'archives propriétaires à l'extension .iwa (iWork Archive).

Sous le capot, chaque fichier .iwa encapsule un flux continu de messages Protocol Buffers (Protobuf), compressés avec l'algorithme Snappy de Google, conçu pour la vitesse plutôt que pour le taux de compression. Apple n'a jamais documenté ce format : ce qu'on en sait provient de la rétro-ingénierie.

- Pour le moteur natif macOS/iOS : L'approche est efficace. La désérialisation est rapide et n'exige aucune analyse textuelle.
- Pour les modèles de langage et les environnements de développement tiers : C'est une boîte noire infranchissable. Modifier un titre ou réordonner deux diapositives en dehors de l'application nécessite de décompresser le flux Snappy, d'effectuer de la rétro-ingénierie sur des définitions Protobuf non documentées par Apple et changeantes d'une version à l'autre, de recalculer des tables de pointeurs d'identifiants internes, puis de ré-encoder le flux binaire. Dans une boucle d'inférence agentique, cette complexité constitue un obstacle majeur.




### Le document comme AST sérialisé

La spécification n'étant pas publiée, ce qui suit décrit la structure qu'on peut raisonnablement attendre d'un tel format, non celle qu'Apple a documentée. En formalisant .keyx, Apple remplacerait les fragments binaires opaques par un arbre d'objets typés en clair, à la manière d'un arbre syntaxique abstrait (AST) :

- presentation.json : Manifeste racine documenté exposant la topologie exhaustive du document :
  - Déclaration des métadonnées globales (résolution du canevas, thèmes, polices déclarées, palettes chromatiques).
  - Liste ordonnée des diapositives associées à des identifiants immuables (UUIDs stables).
  - Description géométrique et hiérarchique de chaque composant sous forme d'objets typés : conteneurs de texte (TextFrame), formes vectorielles (Shape), tables, graphiques d'analyse et masques d'images.
  - Propriétés spatiales absolues et relatives (cadres de délimitation, coordonnées x, y, rotation, opacité, échelle, indice de profondeur z-index).
- media/ : Dossier d'actifs bruts (PNG, SVG, MP4) adressés de manière déterministe via des URI relatives au sein de la structure JSON.

La sémantique textuelle et la logique de composition ne seraient plus enfouies dans des octets compilés : un agent pourrait lire, comprendre et générer directement la représentation de l'arbre sans instancier de moteur de rendu graphique lourd.

### MCP : de la manipulation de fichier au pilotage par RPC

L'ouverture syntaxique du JSON n'est que la moitié de la réponse d'Apple. L'autre moitié est la prise en charge de MCP (Model Context Protocol) : selon Apple, des agents et outils externes pourront « lire le contenu, effectuer des modifications et exécuter des actions automatisées directement dans Keynote, Pages et Numbers ».

Historiquement, l'automatisation sous macOS reposait sur AppleScript ou JXA (JavaScript for Automation). Leur modèle objet, verbeux, faiblement typé et souvent synchrone, se prête mal à une exposition directe à un modèle de langage.

Une interface MCP changerait la donne. Les outils ci-dessous, noms comme fonctions, sont illustratifs : Apple n'a rien publié de tel.

1. Inspection d'état chirurgicale (read) : L'agent ne consommerait plus l'intégralité d'un document lourd dans sa fenêtre d'attention. Il appellerait des outils (tools) MCP dédiés : `get_slide_structure(slide_id=4)` ou `query_layout_tree()`, recevant un extrait JSON minimaliste et ciblé.
2. Mutations atomiques (write) : L'agent cesserait de réécrire des fichiers complets sur le disque. Il émettrait des appels de fonctions ciblés :
   - `update_text_node(id="title_4", content="Marges opérationnelles 2026")`
   - `apply_slide_layout(slide_id=4, template_name="Comparison_Two_Columns")`
   - `insert_chart_series(slide_id=4, data=[...])`

3. Boucle de contrôle multimodale : Le moteur interne de Keynote appliquerait la mutation en temps réel, validerait l'intégrité géométrique et pourrait renvoyer un aperçu matriciel (`render_preview`) à l'agent pour vérification visuelle immédiate avant enregistrement.

## 3. L'économie du token et la fin du monopole XML

Dans les pipelines d'IA générative, la préférence pour le JSON et le Markdown, au détriment du XML et du HTML, tient en partie aux contraintes de l'architecture Transformer. Encore faut-il distinguer quatre usages que le débat confond souvent :

1. le format d'échange entre systèmes déterministes, où XML, ses schémas et sa validation restent pleinement pertinents ;
2. la représentation transmise à un modèle de langage, où priment la concision et la lisibilité ;
3. le format de stockage et d'archivage, où comptent la stabilité, la documentation et la pérennité ;
4. la représentation interne d'un agent, le plus souvent des objets typés.

Un même document peut légitimement vivre sous plusieurs de ces formes : XML n'est pas condamné, il cesse seulement d'être la réponse à tous ces usages à la fois.

**La taxe de contexte et le ratio signal/bruit**
Dans un modèle autorégressif, le coût de l'attention croît avec le carré de la longueur de la séquence (des optimisations comme FlashAttention en réduisent l'empreinte mémoire, non la complexité de calcul), et chaque token occupe une part de la fenêtre de contexte.

– Un schéma XML typique (tel qu'OpenXML / .docx ou .pptx) impose une verbosité structurelle pénalisante :

```
<w:p>
  <w:pPr><w:jc w:val="left"/></w:pPr>
  <w:r><w:t>Exemple de texte</w:t></w:r>
</w:p>
```

– À l'inverse, le Markdown encode la même intention sémantique par une absence de friction syntaxique :

```
Exemple de texte
```

La comparaison n'est d'ailleurs pas tout à fait équitable : le XML porte ici une information de mise en forme (l'alignement) que le Markdown abandonne.

Combien coûte cette « taxe syntaxique » ? Une mesure simple, faite sur quelques échantillons construits pour l'occasion avec un seul tokenizer, o200k_base (celui des modèles GPT-4o d'OpenAI, via la bibliothèque gpt-tokenizer 2.9), donne des ordres de grandeur, sans valeur générale. La méthode et les fichiers sont en annexe.

– un même extrait de cet article compte environ 780 tokens en Markdown, 890 en HTML et 1 550 dans le XML d'un fichier Word, soit un rapport de 1,5 à 2 selon la longueur, l'en-tête de déclarations du fichier Word coûtant à lui seul quelque 350 tokens ;
– une facture UBL simplifiée de trois lignes compte environ 920 tokens, et 440 dans un JSON reprenant les mêmes noms d'éléments (les deux sans indentation) : la répétition des noms dans les balises fermantes double à peu près la note ;
– réduite à ses seules données métier, la même facture tombe à environ 185 tokens, en JSON comme en XML à attributs courts.

Le coût tient donc moins à XML en tant que tel qu'au vocabulaire, à la profondeur d'imbrication, à la répétition des noms, à la mise en forme (l'indentation seule ajoute ici de 20 à 75 %) et au tokenizer employé. Il ne faut pas confondre non plus compacité en octets, nombre de tokens et qualité du traitement par le modèle. Mais quand la taxe est là, elle se paie en surcoût d'inférence, en latence et en saturation du contexte.

**Linéarité probabiliste versus arborescence DOM**
Les modèles de langage prédisent le token suivant token après token, de gauche à droite. Le respect rigoureux d'un modèle d'arbre (Document Object Model) avec balises de fermeture symétriques (`</section>`, `</div>`, `</w:p>`) force le modèle à conserver des états de dépendance à longue portée sur des marqueurs syntaxiques sans valeur sémantique. La difficulté tient à la profondeur d'imbrication plus qu'à XML : les accolades d'un JSON profond posent le même problème, XML y ajoutant sa verbosité. Dès que la profondeur d'imbrication augmente, le risque d'une dérive (balise non fermée, inversion hiérarchique) tend à croître.

Le Markdown contourne le problème par sa tolérance structurelle : une indentation imparfaite ou un astérisque isolé ne fait jamais échouer la compilation. Au prix, il est vrai, d'une sémantique plus pauvre et d'erreurs silencieuses : un Markdown mal formé ne plante pas, il est simplement mal interprété.

**Token, embedding, état latent, document : quatre objets distincts**
Le passage de la balise au token ne signifie pas que l'un remplace l'autre. La sérialisation écrit une structure dans une syntaxe (XML, JSON) : c'est elle qui fait un format d'échange, et elle précède toute lecture par un modèle. Le token est un fragment de ce texte, découpé par le tokenizer ; l'embedding, un vecteur numérique qui en encode un sens approché ; les représentations latentes, les états internes du modèle pendant le calcul. Aucun de ces objets n'est un document : un vecteur ne se relit pas, ne se restitue pas à l'identique et ne garantit ni l'exactitude d'un montant ni l'intégrité d'un texte. Les « représentations compactes » dont parle cet essai sont des formats structurés (Markdown, JSON, Typst) que le modèle lit sous forme de tokens. Ce qui change, ce sont les modalités de traitement de l'information, non sa nature.

## 4. Le décodage contraint et l'avènement du typage strict

Si le Markdown triomphe pour la prose, le JSON s'est imposé comme le support dominant des données structurées produites par les modèles, grâce aux progrès de la théorie des compilateurs appliquée aux logits.

**Le masquage dynamique (Grammar-based Decoding)**
Un schéma JSON (écrit directement ou dérivé de Pydantic ou de Zod) se convertit assez naturellement en grammaire hors contexte, du moins pour le sous-ensemble de JSON Schema que prennent en charge les moteurs d'inférence. La technique vaut en principe pour XML (une DTD est elle-même une grammaire), mais les schémas XSD sont bien plus difficiles à compiler en automates légers, et l'outillage s'est concentré sur JSON.

Lors de l'étape d'échantillonnage de chaque token :

1. Le moteur d'inférence projette l'état courant sur l'automate grammatical.
2. Les logits correspondant aux tokens non autorisés par le schéma sont masqués ($-\infty$).
3. Le modèle ne peut pas produire un document syntaxiquement invalide, tant que la génération va à son terme.

La conformité au schéma est ainsi garantie à la génération. Pas l'exactitude : un montant peut être parfaitement typé et faux. La validation a posteriori des règles métier, celles que les chaînes XML confient à Schematron, reste donc nécessaire.

## 5. Les dinosaures de l'ère visuelle : le cas InDesign et l'impasse du PDF

La chaîne graphique historique a été conçue pour un être humain assis devant un écran, manipulant une souris sur un plan de travail virtuel. Face à un système agentique, cette pile logicielle se heurte à des limites structurelles.

**InDesign ou l'agonie du WYSIWYG monolithique**
Adobe InDesign incarne l'apogée de l'ère WYSIWYG (What You See Is What You Get). Mais ce qui constituait sa force hier fait aujourd'hui sa caducité opérationnelle :

- Un binaire impénétrable : Le format propriétaire .indd est une boîte noire inaccessible sans l'application. InDesign dispose certes d'IDML, un format d'échange XML documenté, mais verbeux et pensé pour l'aller-retour avec le logiciel.
- Une automatisation anachronique : Piloter InDesign via ExtendScript ou UXP, ou même via InDesign Server, suppose de faire tourner une application lourde, peu compatible avec une architecture microservices ou agentique, où l'on attend des composants légers et sans interface.
- Une fracture ontologique : InDesign lie le fond à la géométrie de blocs vectoriels rigides positionnés à l'écran.

À l'exception de la très haute voltige micro-typographique (le calage optique au dixième de point, le réglage manuel du gris typographique à la ligne près ou la gestion d'ouvrages d'art complexes), InDesign est de plus en plus court-circuité, dans les chaînes automatisées, par des moteurs déclaratifs où la forme est une fonction de la donnée textuelle.

**Le PDF : un goulot d'étranglement pour les modèles**
Inventé en 1993 sur les fondations de PostScript, le PDF répondait à une question simple : comment s'assurer qu'un document s'imprime à l'identique sur n'importe quel périphérique ? Pour y parvenir, le PDF a privilégié la géométrie de la page sur la structure logique.

Dans le flux de contenu d'une page, il n'existe ni paragraphes, ni tableaux, ni colonnes. Il n'y a que des primitives de rendu :

```
BT /F1 10 Tf 72 712 Td (Exemple) Tj ET
```

Le PDF n'est pas pour autant dépourvu de sémantique. Depuis PDF 1.4 (2001), le PDF balisé (Tagged PDF) peut décrire la structure logique du document : titres, paragraphes, listes, tableaux, ordre de lecture. La norme PDF/UA (ISO 14289-1, 2014) en fait une exigence d'accessibilité. Mais ce balisage reste facultatif, souvent absent, parfois erroné, et inégalement exploité par les outils d'extraction. Le problème n'est donc pas une absence intrinsèque de sémantique, mais la difficulté à en retrouver une qui soit fiable dans les documents réels.

Pour un modèle de langage (et les pipelines RAG en général), lire un PDF ordinaire revient ainsi à faire de l'archéologie reconstructive :

1. L'ordre de lecture brisé : Dans une mise en page multicolonne, les coordonnées géométriques font souvent s'entrecroiser les lignes de texte lors d'une extraction brute.
2. Le bruit vectoriel : Les césures de mots de fin de ligne, les en-têtes répétés et les faux retours à la ligne fragmentent la tokenisation.

3. Les tableaux fantômes : Un tableau n'est qu'un ensemble de segments de droites tracés indépendamment de bribes textuelles éparses.

Le PDF reste ainsi l'un des principaux goulots d'étranglement de l'ingénierie contextuelle contemporaine.

## 6. Par quoi remplacer le PDF à l'ère des modèles d'action ?

Le remplacement du PDF ne prendra vraisemblablement pas la forme d'un nouveau format de fichier universel unique, mais d'un découplage strict entre la source logique et la projection de surface.

| ARCHITECTURE POST-PDF | |
|---|---|
| 1. SOURCE VIVANTE<br>(Intelligible, versionnée, éditable par l'agent) | Typst (.typ) / Markdown + JSON<br>• Pas de bruit vectoriel<br>• Compilable en millisecondes |
| 2. PROJECTION DE TRAVAIL<br>(Interactive, dynamique, modulaire) | Web natif (DOM / CSS headless)<br>• Refilable, accessible, multi-écrans |
| 3. ARTEFACT DE SORTIE<br>(Impression, signature, dépôt, archivage) | PDF figé (PDF/A s'il est gardé)<br>• Compilé à la volée<br>• Signé quand il fait foi |

1. **La source vivante : le code source déclaratif**

   Pour le document imprimable ou paginé, Typst (.typ) s'offre comme une alternative naturelle, à côté de LaTeX. Il se lit comme du texte brut, se compile très rapidement (en PDF, PNG ou SVG), dispose d'un modèle typographique de premier ordre et ne cache rien à l'agent. Pour la documentation structurée, le Markdown enrichi de schémas JSON joue ce rôle de vérité terrain.
2. **La projection de surface : le DOM conteneurisé**

   Figer un texte sur une page A4 virtuelle de 29,7 cm de haut est une anomalie ergonomique sur les interfaces modernes. La surface de consultation redevient le composant Web interactif (HTML/CSS réactif, blocs dynamiques), refilable selon le support de restitution.
3. **Le PDF relégué au statut d'artefact de sortie**

   Le PDF cesse d'être le format dans lequel on travaille la connaissance. Il devient un produit dérivé, généré à la volée par un compilateur headless (Typst ou Chromium) pour une signature, une impression ou un dépôt. Mais lorsqu'il doit faire foi, il ne se jette pas : il se fige, se signe et s'archive (PDF/A, ISO 19005).

## 7. L'inertie régalienne I : les formats de la facture électronique sont-ils nés obsolètes ?

L'exemple du décalage entre la technique contemporaine et la bureaucratie des formats se manifeste avec acuité dans les standards de la facturation électronique européenne (norme EN 16931).

**UBL et CII : la forteresse XSD des années 2000**
Les syntaxes socles de la facturation électronique (Universal Business Language - UBL et Cross Industry Invoice - CII) reposent sur des spécifications XML d'une grande lourdeur :

– Des espaces de noms interminables et des balises verbeuses à l'excès (`<ram:ApplicableHeaderTradeAgreement>`).

- Des centaines de règles Schematron (les règles BR-xx, qui portent sur les termes métier BT-xx de la norme), dont l'exécution demande des moteurs de validation Java ou .NET volumineux.
- Soumettre du XML UBL ou CII brut à un LLM, quand seules quelques données métier comptent, consomme plusieurs fois plus de tokens que nécessaire (environ 920 contre 185 tokens, pour une facture UBL simplifiée et sa réduction aux données métier, dans la mesure citée plus haut). L'écart tient surtout au vocabulaire de la norme, non à XML en tant que tel.

Ces syntaxes ne sont pas à jeter pour autant : leurs schémas, leurs règles de validation et leur stabilité sont précisément ce qu'on attend d'un format d'échange entre systèmes et d'un support de preuve fiscale.

### Factur-X : le compromis hybride

Le format Factur-X (ZUGFeRD) est l'archétype du compromis de transition : un conteneur PDF/A-3 (pour l'œil humain et les habitudes comptables traditionnelles) qui encapsule en pièce jointe un fichier XML (syntaxe CII), conforme à l'EN 16931 dans ses profils BASIC, EN 16931 et EXTENDED.

Du point de vue du traitement automatisé, ce compromis a un coût :

- Il oblige à générer un artefact graphique lourd (le PDF) tout en maintenant en sous-main une syntaxe XML rigide.
- Il fait coexister deux représentations d'un même acte, qu'il faut maintenir cohérentes.

Mais ce coût achète ce que la seule efficacité de traitement ne fournit pas : un objet documentaire unique, archivable en PDF/A, qui associe lecture humaine et données structurées. Pour l'archivage, le contrôle et les processus de gestion, c'est moins un vestige qu'un compromis raisonné.

### Le paradoxe de l'EDI

Un paradoxe s'invite ici. L'EDI, que XML devait, disait-on, remplacer, satisfait mieux que ses successeurs plusieurs critères de cet essai. Un message UN/EDIFACT est un flux plat de segments (`BGM`, `DTM`, `NAD`, `LIN`, `MOA`…), sans balises fermantes ni rendu destiné à l'œil humain ; son sens réside dans des répertoires et des listes de codes, séparés de la syntaxe. L'EN 16931 prévoit d'ailleurs, à côté d'UBL et de CII, une liaison syntaxique vers l'EDIFACT INVOIC D16B (CEN/TS 16931-3-4). Et le droit fiscal français reconnaît la facture EDI moyennant un dispositif (liste récapitulative séquentielle des messages, fichier des partenaires, conservation), non un document figé : la preuve y est déjà séparée de la représentation.

Mesurée dans les mêmes conditions que plus haut, la facture d'exemple tient en environ 340 tokens en EDIFACT, contre 920 en UBL compact, 440 en JSON reprenant les noms UBL et 185 en JSON réduit aux données métier.

Mais la compacité n'est pas l'intelligibilité. L'EDIFACT se découpe mal : le segment `DTM+137:20261028:102'` coûte à lui seul 11 tokens, et le sens de `137` (date du document) ou de `102` (format AAAAMMJJ) n'est pas dans le message mais dans le répertoire, qu'un modèle doit connaître ou recevoir. La hiérarchie des groupes de segments ne se lit qu'avec le guide de mise en œuvre, et un simple compteur de segments erroné suffit à faire rejeter l'échange. XML a apporté ce qui manquait à l'EDI (l'auto-description, des schémas lisibles, un outillage générique) au prix de la verbosité. Les agents ne ramènent pas à l'EDI : ils appellent une synthèse des deux, aussi explicite que XML et aussi économe que l'EDI.

**Pourquoi ils perdurent : le protocole douanier**
Ces formats ne disparaîtront pas à court terme : ils sont sanctuarisés par les administrations et leurs infrastructures (DGFiP, Chorus Pro, plateformes agréées, réseau Peppol) et par le droit européen (directive 2014/55/UE, puis le paquet ViDA, adopté en 2025), dont le temps s'étend sur une décennie.

Cependant, leur statut dans l'architecture informatique a déjà muté :

- Dans les architectures les plus saines, l'IA ne manipule pas le Factur-X en direct : les agents raisonnent sur des modèles objets typés (validés par Pydantic ou Zod) ou passent par des API.
- La sérialisation comme formalité aveugle : La transformation de l'objet métier vers l'arbre XML UBL ou CII, et le cas échéant son injection dans le PDF/A-3 Factur-X, sont confiées à des bibliothèques déterministes à la périphérie du système.

La facture électronique institutionnelle tend à devenir un langage douanier de frontière, tandis que l'intelligence opérationnelle interne circule en flux JSON compacts et en schémas typés.

## 8. L'inertie régalienne II : le crash ontologique du XBRL et de l'iXBRL

Si Factur-X est un compromis discutable, le XBRL (eXtensible Business Reporting Language) et son dérivé iXBRL (Inline XBRL) représentent le paroxysme de l'enflure technologique des années 2000 appliquée à l'information financière : ESEF en Europe, SEC aux États-Unis et, à terme, reporting de durabilité (CSRD).

**L'enfer structurel des taxonomies et du balisage en ligne**
Sur le papier, le XBRL promettait une standardisation mondiale des états financiers. Dans la réalité de l'ingénierie logicielle, il cumule tous les anti-modèles de l'ère pré-IA :

- La taxe d'attention des contextes répétés : Pour qualifier un simple montant d'actif net ou d'EBITDA, le XBRL impose de le rattacher, par un attribut `contextRef`, à une balise contextuelle verbeuse (`<xbrli:context id="...">`) contenant la période temporelle, l'identifiant de l'entité (LEI) et des coordonnées dimensionnelles complexes. Chaque contexte n'est déclaré qu'une fois, mais il en faut un par combinaison de période et de dimensions : un rapport en compte couramment des centaines.
- L'explosion combinatoire des linkbases : L'architecture sépare les données de leur logique d'interprétation via des fichiers de liaisons distincts (liens de définition, de présentation, de calcul et de libellés multilingues). Résoudre la cohérence d'un rapport financier exige de charger des centaines de fichiers de taxonomie, ce qui rend le traitement lourd et le calcul de dépendances hors de portée d'une exécution légère.
- Le monstre iXBRL (Inline XBRL) : Pour permettre à l'humain de lire le rapport annuel dans un navigateur sans logiciel dédié, les régulateurs ont rendu obligatoire l'encapsulation de balises XBRL invisibles au sein d'un document XHTML stylé. Le résultat est un fichier composite qui peut peser plusieurs dizaines de mégaoctets, au point que de nombreux régulateurs plafonnent le dépôt autour de 100 Mo, amalgamant balises de style CSS verbeuses, structure DOM complexe et métadonnées financières imbriquées.

Pour un agent autonome d'analyse financière ou un pipeline RAG, ingérer un document iXBRL brut relève de l'absurdité économique : l'armature syntaxique XML/HTML occupe une large part de la fenêtre d'attention avant même que le modèle n'ait extrait le moindre ratio de rentabilité. De surcroît, le recours aux taxonomies d'extension propres à chaque émetteur entame

l'interopérabilité promise : malgré l'ancrage à la taxonomie de base qu'impose l'ESEF, chaque rapport tend à devenir un dialecte singulier, qui demande un travail de réconciliation ad hoc.

Le XBRL lui-même a d'ailleurs commencé sa mue. Depuis 2021, XBRL International publie, dans le cadre de son Open Information Model, des syntaxes xBRL-JSON et xBRL-CSV qui sérialisent les mêmes faits hors de XML, les taxonomies restant, elles, en XML. Le modèle d'information survit à sa syntaxe d'origine : c'est exactement la thèse de cet essai.

| LA TRIPLE PEINE DE L'iXBRL | |
|---|---|
| Contenu informatif utile (le signal) | Chiffres clés (P&L, Bilan) Métriques ESG (CSRD) |
| Bruit de présentation XHTML (bruit de forme) | CSS inline, balises , tableaux HTML imbriqués |
| Bruit d'infrastructure XBRL (bruit de structure) | Contextes multiples, URI, espaces de noms, linkbases |

### L'alternative computationnelle : Parquet, JSON-LD et typage déclaratif

À l'ère des architectures agentiques et des modèles d'action, le reporting financier et extra-financier n'aurait plus à être un document monolithique hybride. On peut esquisser une architecture qui sépare nettement les séries chronologiques de faits, leur ontologie et leur narration :

1. Le stockage colonnaire de faits (Parquet / Arrow) : Les états financiers (bilan, compte de résultat, tableaux de flux de trésorerie, métriques CSRD) se prêtent naturellement au format tabulaire binaire colonnaire. Un ensemble de fichiers Parquet légers, typés et partitionnés permet d'interroger dix ans de comptes consolidés en mémoire via SQL analytique (DuckDB) en quelques millisecondes, sans passer par le modèle de langage.
2. L'ontologie allégée en JSON-LD ou schémas stricts : Les concepts normatifs (IFRS, US GAAP, ESRS) sont modélisés sous forme de schémas déclaratifs compacts (JSON-LD ou JSON Schema). La sémantique d'un agrégat financier n'est pas réexpliquée à chaque ligne du rapport : elle est résolue par référence immuable à un contrat de typage partagé.
3. Le rapport narratif à variables d'ancrage : Le rapport de gestion et les états financiers commentés (la prose destinée aux actionnaires et aux auditeurs) s'écrivent en Markdown enrichi ou Typst, au sein duquel les montants ne sont jamais codés en dur, mais appelés dynamiquement via des variables liées aux sources comptables vérifiées :

   ```
   Au cours de l'exercice 2026, la marge brute s'établit à {{ financial_data.gross_margin | eur }},
   matérialisant une progression de {{ financial_data.margin_growth_pct }} par rapport à N-1.
   ```

   La compilation réduit par construction le risque de discordance entre le texte explicatif et les états comptables, à condition que les sources elles-mêmes soient justes.
4. L'exposition par API et serveur MCP : L'investisseur institutionnel ou l'autorité de contrôle n'aurait plus à télécharger un fichier iXBRL de plusieurs dizaines de mégaoctets. Ils pourraient se connecter à une API certifiée de l'émetteur ou à son serveur MCP pour extraire les seuls segments requis, chaque donnée portant une signature électronique qui en atteste l'intégrité et la provenance.

À l'instar d'UBL et de Factur-X, le XBRL continue de subsister comme une obligation de conformité légale imposée par les régulateurs de marché. Mais dans la pratique opérationnelle des analystes, des experts-comptables et des systèmes d'intelligence artificielle, il tend à être relégué au statut de « format de guichet » : un artefact compilé à l'ultime étape d'exportation pour satisfaire le régulateur, tandis que l'essentiel de l'analyse et de la décision financière s'exécute en amont sur des flux compacts et des données tabulaires typées.

## 9. Panorama des formats émergents à l'ère agentique

La reconfiguration documentaire esquisse une division du travail selon la fonction opérationnelle :

| Famille d'usage | Ancien standard (Humain / Rendu) | Standard agentique (LLM / Compilateur) | Propriétés architecturales |
|---|---|---|---|
| Composition paginée | InDesign (.indd), LaTeX (.tex) | Typst (.typ) | Syntaxe déclarative, typage fonctionnel, boucle de correction en ms |
| Présentations | Keynote binaire (.iwa), PowerPoint (.pptx) | .keyx (JSON/MCP, annoncé), Marp / Slidev (.md) | Sérialisation en AST, mutations RPC via serveur MCP |
| Calcul & Données | Jupyter Notebook (.ipynb), Excel (.xlsx) | Scripts Python réactifs (.py), Parquet | Code source pur, DAG déterministe, zéro résidu base64 |
| Reporting financier | XBRL, iXBRL (XHTML hybride) | Parquet + JSON-LD / Variables Typst | Séparation faits/ontologie, requêtes SQL/DuckDB, zéro friction RAG |
| Échange d'état & Édition | Fichier monolithique écrasé (.docx) | CRDT (Yjs, Automerge) | Mutations atomiques, collaboration sans conflit, deltas d'opérations |
| Contrats de données | XML DTD / XSD (Schematron maintenu pour les règles métier) | JSON Schema, Typage TypeScript/Zod | Validation de la syntaxe au niveau des logits (constrained decoding) ; règles métier en aval |

Ce tableau décrit des tendances, non des remplacements : les formats de la deuxième colonne restent souvent ceux de l'échange, de la conformité et de l'archivage.

## 10. Conclusion : Le document comme API sérialisée

Les signaux réunis ici (l'annonce de .keyx, l'essor de Typst, la marginalisation des forteresses WYSIWYG, la relégation du PDF au rang de sortie, le cantonnement des formats réglementaires comme Factur-X et XBRL aux frontières des systèmes) dessinent un changement philosophique profond : le format de fichier n'a plus un destinataire unique. L'humain, le moteur de rendu et désormais l'agent en sont les lecteurs, et ils n'attendent pas la même représentation.

Pendant un demi-siècle, un document était conçu comme un calque figé destiné à être contemplé. À l'ère des agents, le document devient aussi un flux de données structuré, une grammaire vérifiable et une API d'actions.

Mais un document n'est pas seulement une représentation : c'est aussi une preuve. Les organisations doivent pouvoir dire d'où vient une information, montrer qu'elle n'a pas été altérée, la relire dans vingt ans, en reproduire la restitution à l'identique et la soumettre à un contrôle. Ce qui rend une représentation précieuse pour un agent (être dynamique, transformable, recalculée à la demande) est précisément ce qui l'empêche, à elle seule, de faire preuve. C'est pourquoi les formats normés, figés et signés ne disparaissent pas : ils changent de rôle. L'agent travaille sur la représentation vivante ; la preuve se scelle dans l'artefact figé. Tout l'enjeu de l'ingénierie documentaire à venir est de tenir le lien entre les deux : savoir, pour chaque document figé, de quelle source vivante il procède, et pouvoir le démontrer.

L'interopérabilité des formats n'est plus une revendication éthique ou une concession réglementaire imposée aux éditeurs de logiciels : elle est devenue la condition indispensable pour qu'un artefact d'information existe, agisse et fasse foi dans le monde numérique contemporain.

# Sources


- Apple, « Apple Creator Studio adds powerful new AI features and even more creative tools », 27 octobre 2026 : apple.com ‹https://www.apple.com/newsroom/2026/10/apple-creator-studio-adds-powerful-new-ai-features-and-even-more-creative-tools/›
- Apple, « Apple Unveils Keynote », 7 janvier 2003 : apple.com ‹https://www.apple.com/ie/newsroom/2003/01/07Apple-Unveils-Keynote/›
- MacGeneration, « Keynote prépare un format ouvert en JSON pour faciliter le travail des IA », 28 octobre 2026 : macg.co ‹https://www.macg.co/logiciels/2026/10/keynote-prepare-un-format-ouvert-en-json-pour-faciliter-le-travail-des-ia-311931›
- Next, « Apple ajoute du JSON dans Xcode et Keynote pour les agents IA et les développeurs » : next.ink ‹https://next.ink/264508/apple-ajoute-du-json-dans-xcode-et-keynote-pour-les-agents-ia-et-les-developpeurs/›
- Google, Snappy : google.github.io/snappy ‹https://google.github.io/snappy/›
- SheetJS, notes sur le format iWork 2013 et suivants (rétro-ingénierie) : oss.sheetjs.com ‹https://oss.sheetjs.com/notes/iwa/›
- PDF Association, « Questions and Answers about Tagged PDF » : pdfa.org ‹https://pdfa.org/?p=141336›
- PDF Association, sur PDF/UA (ISO 14289-1) : pdfa.org ‹https://pdfa.org/pdf-on-gaad-because-its-the-content-you-keep/›
- XBRL International, spécification XBRL 2.1 : xbrl.org ‹https://www.xbrl.org/specification/xbrl-2.1/rec-2003-12-31/xbrl-2.1-rec-2003-12-31+corrected-errata-2013-02-20.html›
- XBRL International, « xBRL-CSV and xBRL-JSON reach Recommendation status », octobre 2021 : xbrl.org ‹https://www.xbrl.org/news/the-future-is-here-xbrl-csv-and-xbrl-json-reach-recommendation-status/›
- AMF, dossier ESEF : amf-france.org ‹https://www.amf-france.org/fr/actualites-publications/dossiers-thematiques/esef›
- Parseport, limites de taille des dépôts ESEF : parseport.com ‹https://support.parseport.com/knowledge/esef-size-limits›
- EY, adoption du paquet ViDA par le Conseil de l'UE, mars 2025 : ey.com ‹https://globaltaxnews.ey.com/news/2025-0656-eu-council-approves-vat-in-the-digital-age-vida-package›
- Brandon T. Willard et Rémi Louf, « Efficient Guided Generation for Large Language Models », 2023 : arxiv.org ‹https://arxiv.org/abs/2307.09702›
- CEN, CEN/TS 16931-3-4, « Electronic invoicing — Part 3-4: Syntax binding for UN/EDIFACT INVOIC D16B » : evs.ee ‹https://www.evs.ee/en/cen-ts-16931-3-4-2017›
- BOFiP, BOI-TVA-DECLA-30-20-30-40 (factures transmises par EDI) : bofip.impots.gouv.fr ‹https://bofip.impots.gouv.fr/export/pdf/17005›
- Mesure du coût en tokens, méthode et fichiers : annexe


Pour citer ce texte : Frank Adebiaye, « De la balise au token : structure, représentation et preuve à l'ère agentique », Forthcome, Laboratoire, 28 octobre 2026. https://forthcome.fr/laboratoire/de-la-balise-au-token
Document composé avec Typst à partir de la source de l'article, en IBM Plex Serif, IBM Plex Sans et Recursive Mono.