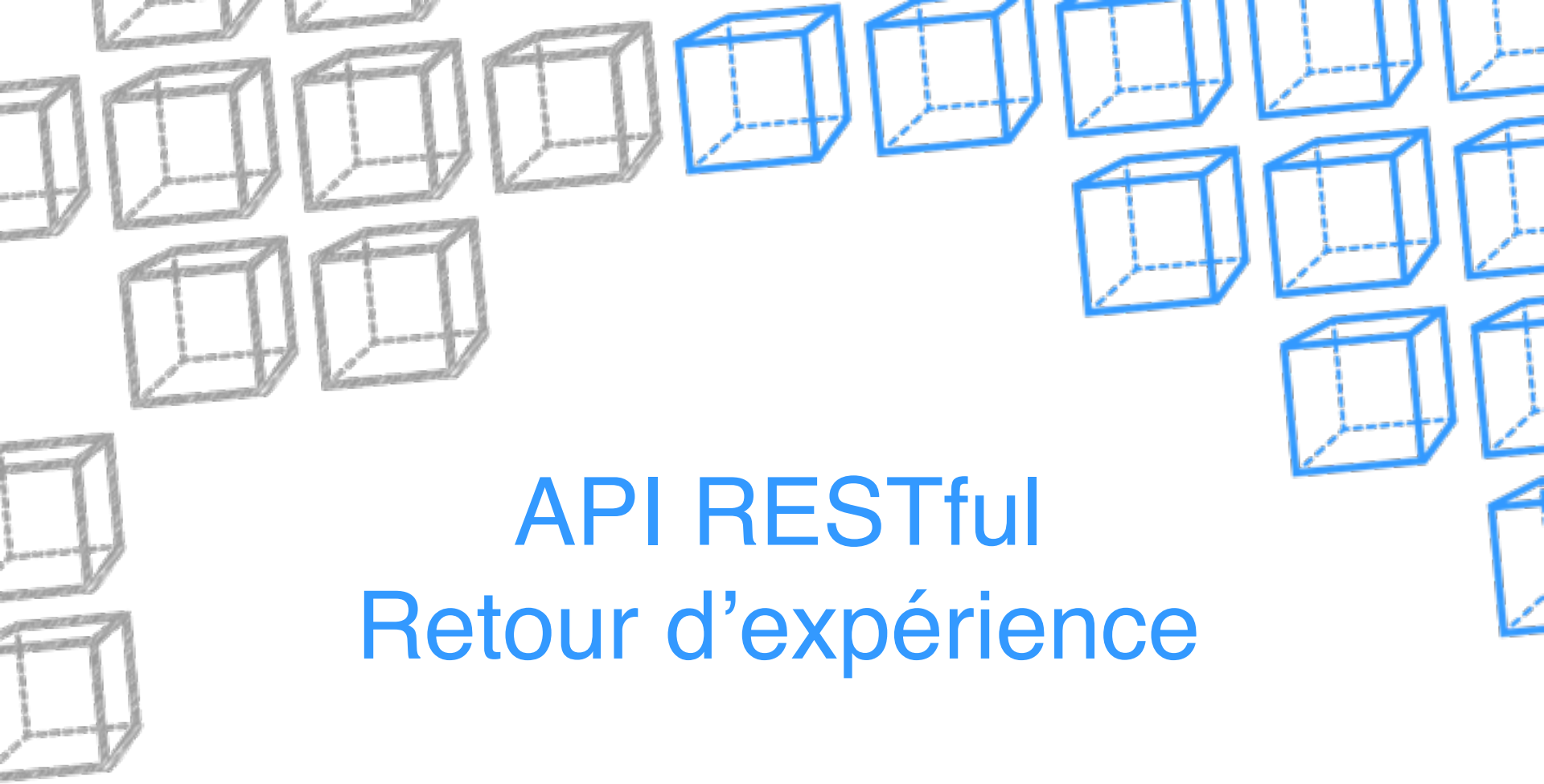


API RESTful
Retour d'expérience

Christophe Laprun / @metacosm
Jahia Solutions Group SA

REST?

- REpresentational State Transfert
- “Architectural style for distributed hypermedia system” - Roy Fielding

REST?

- Certes… mais plus concrètement?
- Ensemble de contraintes pour le web donc HTTP

Client - serveur

- Séparation des clients et des serveurs
- Serveur publie une collection de “resources”…
- … que les clients peuvent manipuler via leur représentations

Serveur sans état

- Toutes les requêtes doivent contenir toutes les informations nécessaires pour que le serveur puisse y répondre

Caches

- Les réponses peuvent être cachées par les clients
- Une réponse doit établir sous quelles conditions elle peut être cachée

Interface uniforme

- Identification des resources
- Manipulation des resources via leur représentations
- Messages auto-descriptifs
- Hypermedia As The Engine Of Application State (HATEOAS)

Système à couche

- L’architecture doit prendre en compte et permettre l’existence potentielle d’éléments intermédiaires entre le client et le serveur
- Requêtes et réponses peuvent être modifiées de manière transparente

Code à la demande (optionnel)

- Clients peuvent être étendus en demandant du code au serveur

Contraintes

- Pourquoi?

Caractéristiques architecturales

- Performance
- Évolutivité (changement / charge)
- Simplicité
- Modifiabilité
- Visibilité des interactions (monitoring)
- Portabilité
- Fiabilité

Contraintes?

- Les 3 premières contraintes permettent une architecture web scalable et performante:
  - Couplage faible client / serveur (évolutivité)
  - Pas d'état géré par le serveur: multi-threading, clustering facilités
  - Performance via caches

Contraintes?

- La 5ème contrainte permet de gérer la performance/évolutivité (load-balancers), la sécurité (firewall) et l'encapsulation (gateway) de manière transparente
- La 6ème contrainte permet au serveur de déléguer une partie du travail aux clients (scripts, applets)

4ème contrainte?

- Le vrai centre des architectures REST
- Unifie l'interface entre les différentes parties de l'architecture
- Permet de créer une architecture similaire aux pipes UNIX

API RESTful?

- Techniquement, on doit parler d'API RESTful, pas d'API REST
- Une API qui respecte les principes de l'architecture REST

API RESTful?

- Certes… mais plus concrètement?

API RESTful?

- Beaucoup (tout?) réside sur comment respecter le principe d'interface uniforme

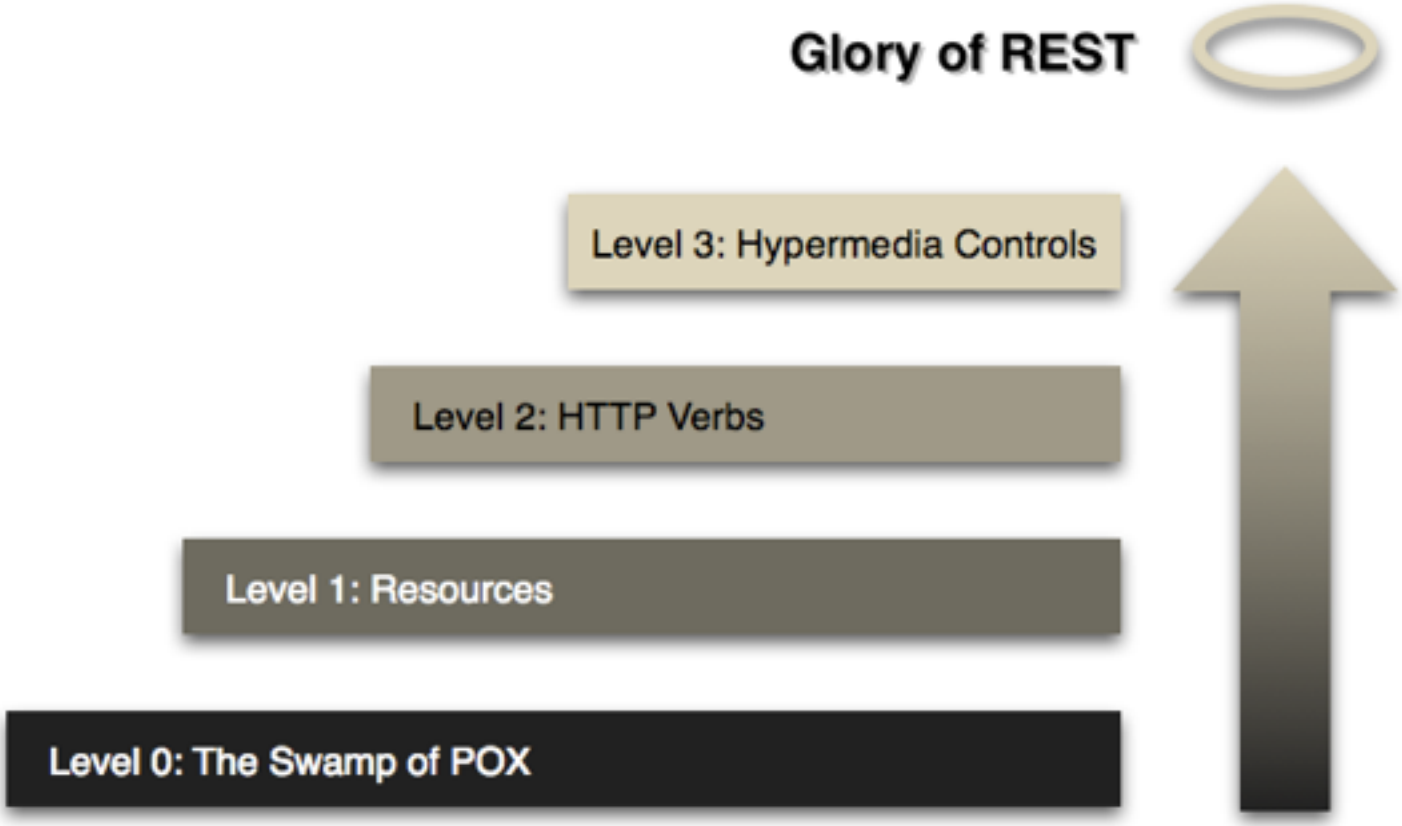
RESTful?

- GET: /getAllClients
- GET: /addClient?id=foo
- GET: /invoiceClient?client=foo&id=1234

Questions pertinentes

- Quelles sont les resources que le serveur expose?
- Comment sont-elles identifiées (URIs)?
- Comment sont-elles représentées?
- Comment mapper le fonctionnel sur la sémantique réduite des verbes HTTP?
- Comment gérer l'état dans une architecture stateless?

Modèle de maturité de Richardson



Niveau 1: resources

- Resources sont identifiées avec URI
- Tunneling des requêtes (GET ou POST)
- Pas de sémantique

Resources

- Noms, pas verbes
- Similaire à une conception orientée objet
- Uniform Resource Identifier (URI) associé à chaque resource

Niveau 2: verbes HTTP

- Multiples resources
- Utilisation sémantiquement correcte des verbes HTTP
- Utilisation correcte des code de réponse

Sémantique méthodes HTTP

- CRUD operations
  - POST / PUT: création
  - GET: lecture
  - PUT / POST: mise à jour
  - DELETE: destruction

Niveau 3: contrôles hypermédia

- Resources auto-descriptives
- État ET comportement accessible via les représentations
- HATEOAS

Représentations

- Les clients n’ont pas accès aux resources, seulement leur représentations
- Doivent contenir suffisamment d’informations pour assurer les transitions d’état

Choix du content-type

- Dépend des clients cibles
- Négociation de contenu
- Réutiliser un media type existant ou développer un media type propre?

Cas d’utilisation: JCR

- Besoin: exposer un repository JCR via une architecture REST
- Optimisé pour un accès facilité via JS

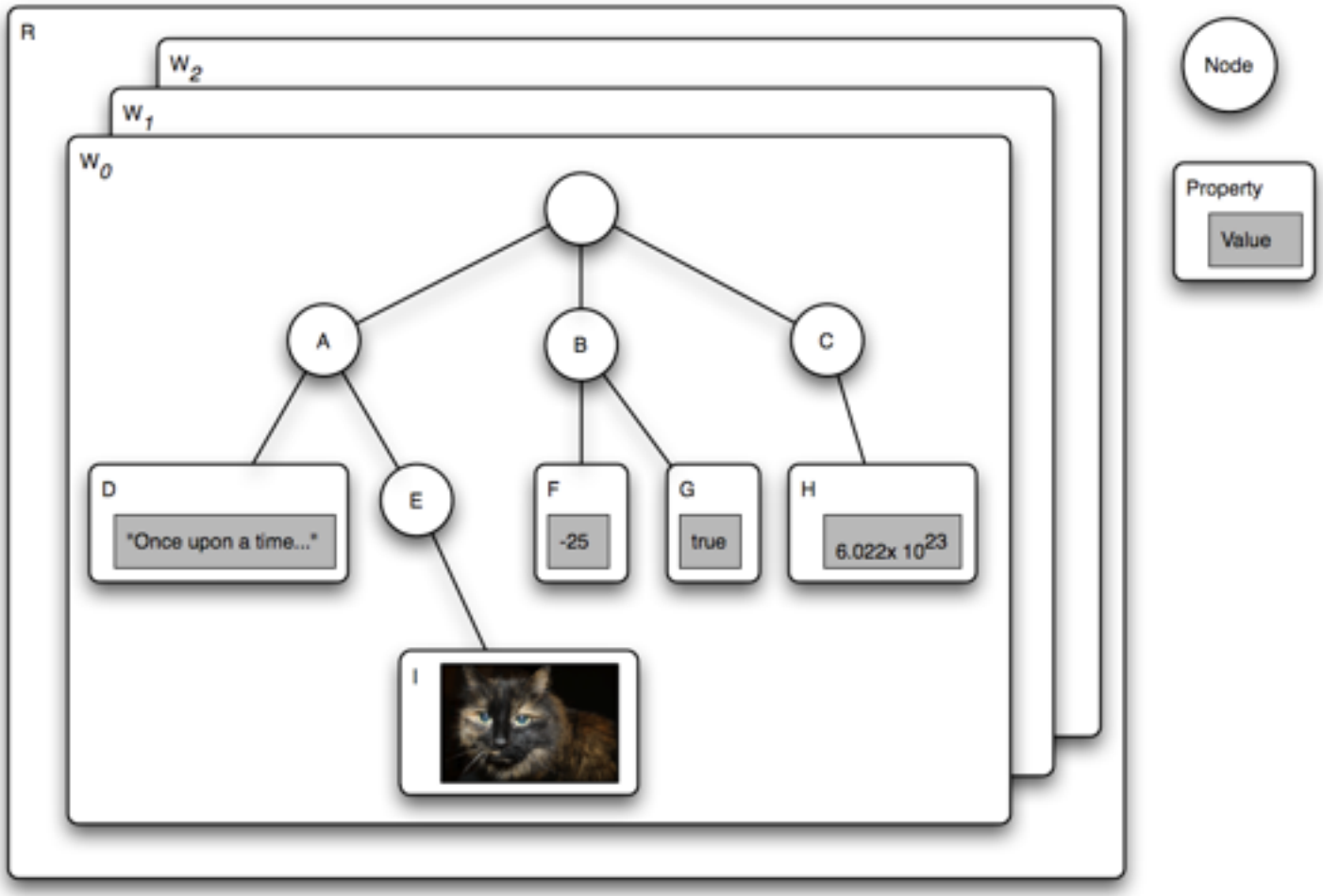
Aperçu JCR

- Java Content Repository
- Modèle de données et services associés pour le stockage et la manipulation de contenu numérique
- Repository: ensemble de workspaces
- Workspace: graphe orienté acyclique
- Sommets: Items (Node / Property)
- Arcs: relation parent - fils

Aperçu JCR

- Item:
  - Type
  - Path
- Node:
  - Identifiant
  - Enfants
  - Propriétés
  - Mixins
  - Versions
- Property:
  - Valeur(s)

Aperçu JCR


R
W2
W1
W0
A
B
C
D
"Once upon a time..."
E
F
-25
G
true
H
6.022x 10^23
I
Node
Property
Value

Resources

- Mapping assez simple dans notre cas: resource => node

Sémantique HTTP

- CRUD sur les nœuds JCR
- Mapping assez simple encore une fois
- Mais:
  - Liberté sur la sémantique PUT/POST vs. PATCH

Représentations

- JSON seul
- augmenté de Hypertext Application Language
- Content-type: application/hal+json

RESTful?

- Sémantique des verbes HTTP?
- Transition d’état par les liens?
- Pas de media type propre
- Versionning dans les URLs vs. via media type

Références

- La thèse de Roy Fielding: http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm
- RESTful Web Services Cookbook (Allamaraju - O'Reilly)
- RESTful Web APIs (Richardson / Amundsen - O'Reilly)
- HAL: https://tools.ietf.org/html/draft-kelly-json-hal-06