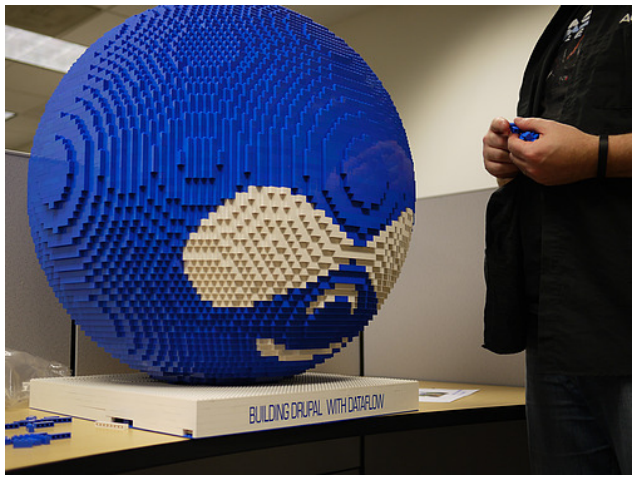
Entity API

Alexandre Todorov,
Felip Manyer i Ballester



Montpellier, le 17 septembre 2014

À propos d'Alexandre

(Improvisation)

À propos de Felip

- Drupalien depuis fin 2009, centralien de Lyon (2008).
- Exerce en indépendant à Perpignan sous le nom commercial « Res Telæ ».
- Vient aux meetups pour donner libre cours à sa logorrhée, mais aussi rencontrer des gens comme lui.
- « Anarchiste » **ayant atteint** la trentaine, fermement opposé à la confiscation et la minitellisation d'Internet.
- Loisirs : natation, cyclisme, sports de montagne, piano, sciences naturelles, linguistique, OpenStreetMap, Guifi (?), changer le monde...

Historique (1/2)

En Drupal 6

- L'unité de données de plus haut niveau est le *nœud*.
- Arrive CCK : définition de champs (donnée + sémantique) au sein d'un type de contenu.
- Génial ! Attachons des champs aux utilisateurs et aux commentaires $\Rightarrow \exists$ modules les transformant en nœuds.

En Drupal 7

- Field API embarquée dans le cœur.
- Apparition d'une API gérant les opérations CRUD de *manière unifiée* : nœuds, utilisateurs, commentaires, taxonomies, etc. sont des *entités*, accédant au même niveau de citoyenneté.

Historique (2/2)

- Le support des entités par le cœur est incomplet : `entity_get_info()`, `entity_load()`.
- Entity API apporte des fonctions manquantes : `entity_get_property_info()`, `entity_view()`, `entity_create()`, `entity_save()`, `entity_delete()`, ainsi que des surcouches extrêmement utiles : `entity_metadata_wrapper()`.

Vocabulaire

Type d'entité Exemple : nœud, utilisateur, terme de taxonomie, bean, *votre entité perso*.

Bundle Une « variation » d'un type d'entité. Exple : type de contenu pour les nœuds. Tous les types d'entités n'ont pas de bundles (exple : utilisateurs).

Propriété Une unité de données commune à tous les bundles. Exple : titre, date de création.

Champs Une unité de données possédant une sémantique définie, attaché à un (des) bundle(s) en particulier. (Champs + formateurs et view modes = ♡)

Intérêt

Standardisation

Des modules comme Search, Vote, Entity Reference, Rules ou Views sont tous capables de travailler *en coopération* sur une *unité de données commune*.

Exemples

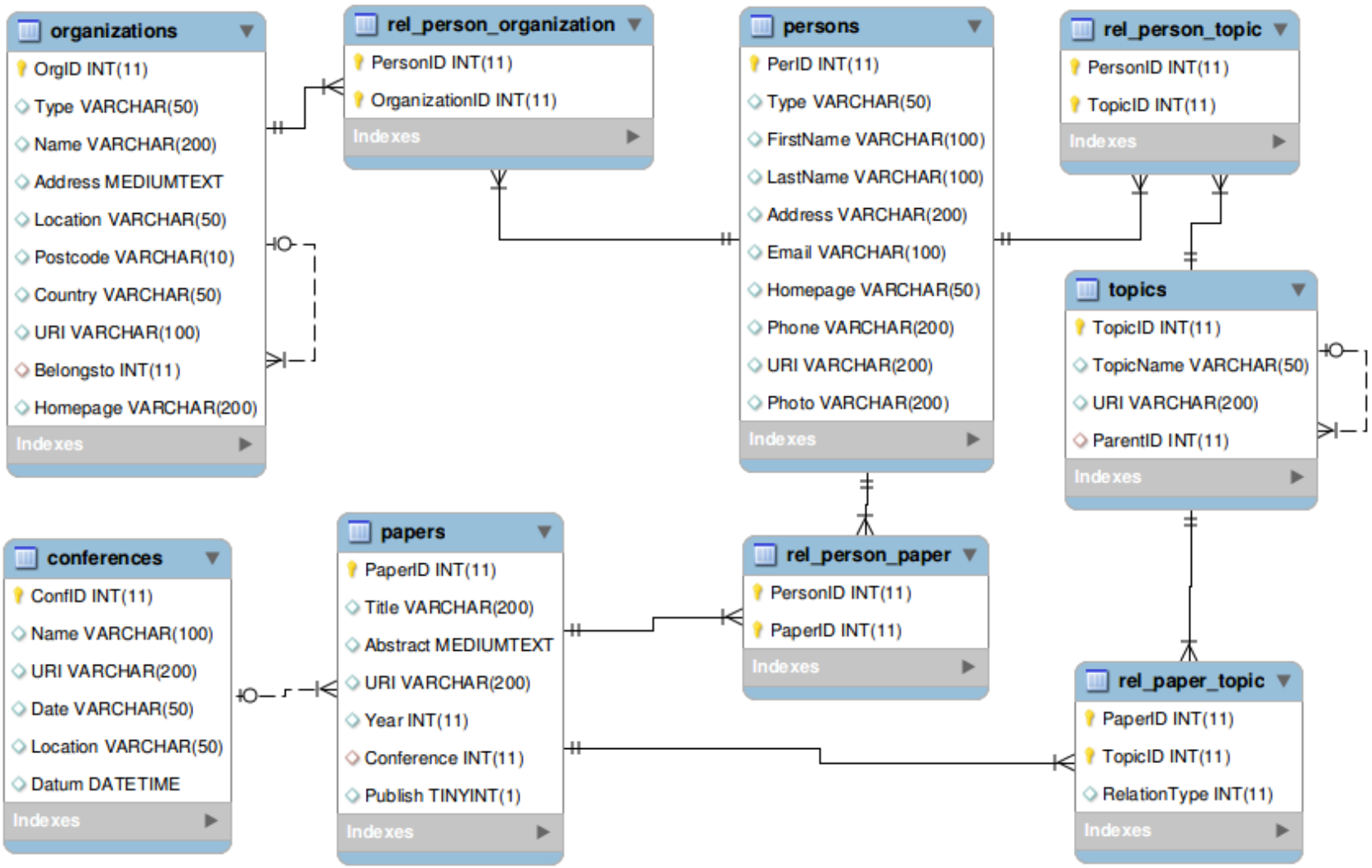
Bean mime le comportement des blocs du cœur, mais s'édite comme un nœud, peut embarquer des champs, possède des bundles et des view modes.

Drupal Commerce architecture entièrement mise en œuvre sous forme d'entités (produits, commandes, « line items », etc.)

Modélisation d'entités

- Pourquoi modéliser ses données ?
  - support d'échange, documentation et maintenance
  - « petit schéma » et »long discours »
  - graphiques non subjectifs
- Entités ou nœuds
  - Contenu *semi-structuré*
    - structure hiérarchique (GED) ou non (éditoriale, web) avec des champs communs (titre, corps, mise à jour, etc.)
  - Contenu *structuré*
    - produit, stock, commande (e-commerce)
    - user, taxonomy, comment, node (Drupal)
- Comportements spécifiques (méthodes) par entité

Exemple ISWC



Intégration rapide de MCD

- Entity Construction Kit (ECK)
  - Gestion des types d'entités, bundles et entités
    - Interfaces utilisateur (site builders)
    - Classes (développeurs)
    - Features (config)
  - Ajouter des comportements aux propriétés (exple. titre, date, texte, etc.)
  - Configurations prêtes à l'emploi (définitions, callbacks, controllers, etc.)
- Utilisation d'Entity et property info pour :
  - Views relationships (jointures)
  - Formulaires
  - CRUD - entity_dependency - les propriétés de type « clé étrangère » seront exposées par l'instance de l'entité correspondante.

Plugin behavior - relation n - 1

```
$plugin = array(
  'label' => "Relation n-1",
  'entity_save' => 'iswc_relationn1_property_entity_save',
  'property_info' => 'iswc_relationn1_property_property_info',
  'default_widget' => 'iswc_relationn1_property_widget',
  'default_formatter' => 'iswc_relationn1_property_formatter'
);
```

Views relationship + Entity Dependency

hook_entity_property_info()

```
$info['rel_person_organization']
    ['properties']['PersonID']['type'] = 'eck_person';
$info['rel_person_organization']
    ['properties']['OrganizationID']['type'] = 'eck_organization';
// relation n-1: for behaviors
$info['eck_paper']
    ['properties']['conference']['type'] = 'eck_conference';
$info['eck_paper']['properties']['parent']['type'] = 'eck_paper';
```

hook_entity_dependencies() pour eck_paper

```
// eck_paper.conference -> eck_conference
entity_dependency_add(
    $dependencies, $entity, 'eck_conference', array('conference')
);
```

Person

EDIT | DELETE | MANAGE PROPERTIES | BUNDLE LIST

HELP: The add property button will temporarily add a property to the entity type, to finalize adding the property, click the checkbox on the left and click save

Properties

	MACHINE NAME	NAME	TYPE	BEHAVIOUR
☑	title	Title	text	title
☐	uid	Author	integer	author
☐	created	Created	integer	created
☐	changed	Changed	integer	changed
☐	language	Entity language	language	language
☑	firstname	FirstName	text	textfield

Add new property

Type *

- Select -

Name *

Person

DELETE | EDIT | MANAGE FIELDS | MANAGE DISPLAY | ENTITY LIST

Show row weights

LABEL	REGION	NAME	FIELD	WIDGET	OPERATION
Content					
title field	Content	title	title field		
firstname field	Content	firstname	firstname field		
rel_person_paper relation	Content	rel_person_paper	rel_person_paper relation		
rel_person_topic relation	Content	rel_person_topic	rel_person_topic relation		
rel_person_organization relation	Content	rel_person_organization	rel_person_organization relation		
Hidden					
Add new field	Hidden		- Select a field type - Type of data to store.	- Select a widget - Form element to edit the data	

Edit Person

Title *
Gil

FirstName *
Yolanda

Organization
-- Choose --
USC Information Sciences Institute
University of Karlsruhe
International University in Germany

Topic
-- Choose --
Knowledge Representation Languages
Knowledge Systems
Artificial Intelligence

Paper
-- Choose --
Trusting Information Sources One Citizen at a Time
Automatic Generation of Java/SQL based Inference Engines from RDF Schema and RuleML
Three Implementations of SquishQL, a Simple RDF Query Language

Save

Modify the display(s) of your view below or add new displays.

Displays

Master | Organization | Topic | Paper | Organization | Topic | + Add | edit view name/description

Organization details

Display name: Organization | clone Organization

TITLE

Title: Person

FORMAT

Format: Table | Settings

FIELDS Add

(Organization) Organization: Link (Link) | Aggregation settings

(Organization) Organization: Title (Organization) | Aggregation settings

(Belongs to) Organization: Title (Parent Organisation) | Aggregation settings

Person: Link (Link) | Aggregation settings

Person: Title | Aggregation settings

BLOCK SETTINGS

Block name: None

Access: None

HEADER Add

FOOTER Add

PAGER

Use pager: Display all items | All items

More link: No

Advanced

CONTEXTUAL FILTERS Add

Person: Id | Aggregation settings

RELATIONSHIPS Add

Person: rel_person_organization Entity

(rel_person_organization Entity) Rel_person_organization Entity: Label

Person: rel_person_topic Entity

(rel_person_topic Entity) Rel_person_topic Entity: Label

Person: rel_person_paper Entity

(rel_person_paper Entity) Rel_person_paper Entity: Label

(Organization) Organization: Belongs to

(Topic) Topic: Parent

Content

Gil | view

Organization: USC Information Sciences Institute | view

Paper: Trusting Information Sources One Citizen at a Time | view

Person Topics

PARENT ORGANISATION	PERSON TOPICS	PARENT ORGANIZATION	PARENT TOPIC	ORGANIZATION URI
	Knowledge Representation Languages | view		Artificial Intelligence	http://trellis.semanticweb.org/expect/web/semanticweb/iswc02_trellis.pdf#ISI
	Knowledge Systems | view		Knowledge Management	http://trellis.semanticweb.org/expect/web/semanticweb/iswc02_trellis.pdf#ISI
	Artificial Intelligence | view			http://trellis.semanticweb.org/expect/web/semanticweb/iswc02_trellis.pdf#ISI

Ratnakar | view

Organization:

Paper: Trusting Information Sources One Citizen at a Time | view

Person Topics

PARENT ORGANISATION	PERSON TOPICS	PARENT ORGANIZATION	PARENT TOPIC	ORGANIZATION URI
	Semantic Web | view		World Wide Web	

Déclarer une entité : hook_schema() (1/2)

```
function profil_schema() {
  return array(
    'profil' => array(
      'description' => 'Table stockant les profils',
      'fields' => array(
        'pid' => array(
          'description' => 'Clef primaire',
          'type' => 'serial',
          'unsigned' => TRUE,
          'not null' => TRUE,
        ),
        'uid' => array(
          'description' => 'Utilisateur auteur du profil',
          'type' => 'int',
          'unsigned' => TRUE,
          'not null' => TRUE,
        ),
```

Déclarer une entité : hook_schema() (2/2)

```
        'date' => array(
          'description' => 'Date du profil',
          'type' => 'int', 'not null' => TRUE,
        ),
        'created' => array(...),
        'changed' => array(...),
        'autre_propriete' => array(...),
      ),
      'unique keys' => array('uid_date' => array('uid', 'date')),
      'foreign keys' => array(
        'uid' => array(
          'table' => 'users', 'columns' => array('uid' => 'uid'),
        ),
      ),
      'primary key' => array('pid'),
    ),
  );
}
```

Déclarer une entité : hook_entity_info()

```
function profil_entity_info() {
  return array(
    'profil' => array(
      'label' => t('Profil'),
      'plural label' => t('Profils'),
      'description' => t('Renseignements quotidiens utilisateur'),
      'base table' => 'profil',
      'entity keys' => array('id' => 'pid'),
      'module' => 'profil',
      'controller class' => 'ProfilController',
      'views controller class' => 'EntityDefaultViewsController',
      'fieldable' => TRUE,
      'label callback' => 'profil_label_callback',
      'uri callback' => 'entity_class_uri',
    ),
  );
}
```

Déclarer une entité : le contrôleur

```
class ProfilController extends EntityAPIController {
  public function load($ids = array(), $conditions = array()) {
    $entities = parent::load($ids, $conditions);
    // Un traitement. Exemple : désérialiser un champ.
    return $entities;
  }

  public function save($entity, DatabaseTransaction $transaction = NULL
    if (isset($entity->is_new)) {
      $entity->created = REQUEST_TIME;
    }
    $entity->changed = REQUEST_TIME;
    // Un traitement. Exemple : sérialiser un champ.
    return parent::save($entity, $transaction);
  }
}
```

Déclarer une entité : hook_entity_property_info()

```
function profil_entity_property_info_alter(&$info) {
  $info['profil']['properties']['uid'] = array(
    'label' => t('Utilisateur'),
    'description' => t("Utilisateur ayant renseigné ce profil"),
    'type' => 'user',
    'schema field' => 'uid',
  );
  $info['profil']['properties']['date'] = array(
    'label' => t("Date du profil"),
    'description' => t("Date de ce profil"),
    'type' => 'date',
    'schema field' => 'date',
  );
}
```

Surcouche de métadonnées (entity metadata wrappers)

Des outils puissants pour manipuler les entités

Créer un « wrapper »

```
$wrapper = entity_metadata_wrapper('node', $node);
```

Modifier des valeurs

```
$wrapper->author->mail->set('sepp@example.com');
$wrapper->author->mail = 'sepp@example.com';
```

Récupérer des valeurs

```
$wrapper->author->mail->value();
$wrapper->title->value(array('sanitize' => TRUE));
$wrapper->body->value->raw();
```

EntityFieldQuery 1/3

Interroger les entités : propriétés, champs et autres métadonnées génériques (cœur)

→ **entityCondition**($name, $value, $operator = NULL)

entity_type	'node', 'taxonomy_term', 'comment', 'user', 'file'
bundle	'article', 'page' (not supported in comment)
revision_id	
entity_id	

→ **propertyCondition**($name, $value, $operator = NULL)

→ **fieldCondition**($field, $column = NULL, $value = NULL, $operator = NULL, $delta_group = NULL, $language_group = NULL)

$column	la colonne du champ
$delta_group	les conditions dans le même groupe doivent avoir le même $delta_group
$language_group	les conditions dans le même groupe doivent avoir le même $language_group.

EntityFieldQuery 2/3

Syntaxe compacte, facile à suivre

```
$query = new EntityFieldQuery();
$query->entityCondition('entity_type', 'node')
  ->entityCondition('bundle', 'article')
  ->propertyCondition('status', 1)
  ->fieldCondition('field_news_types', 'value', 'spotlight', '=')
  ->fieldCondition('field_photo', 'fid', 'NULL', '!=')
  ->fieldCondition('field_faculty_tag', 'tid', $value)
  ->fieldCondition('field_news_publishdate', 'value', $year . '%',
    'like')
  ->fieldOrderBy('field_photo', 'fid', 'DESC')
  ->range(0, 10)
  ->addMetaData('account', user_load(1)); // Run the query as user 1.
$result = $query->execute();
if (isset($result['node'])) {
  $news_items_nids = array_keys($result['node']);
  $news_items = entity_load('node', $news_items_nids);
```

EntityFieldQuery 3/3

Autres méthodes

- **propertyOrderBy**($column, $direction = 'ASC') ne fonctionne pas sur toutes les propriétés
- **range**($start = NULL, $length = NULL)
- **count**()
- **addMetaData**($key, $object)

```
// Exécuter la requête en tant que root.
$query->addMetaData('account', user_load(1));
```

- **addTag**('random')

```
function mymodule_query_random_alter($query) {
  $query->orderRandom();
}
```

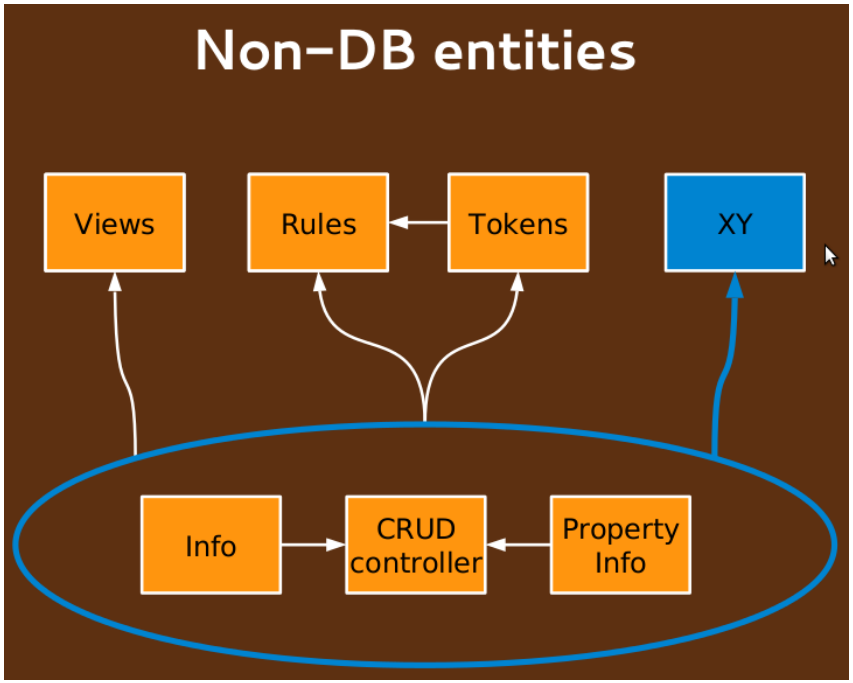
Exposer des entités « no DB »

NoSQL

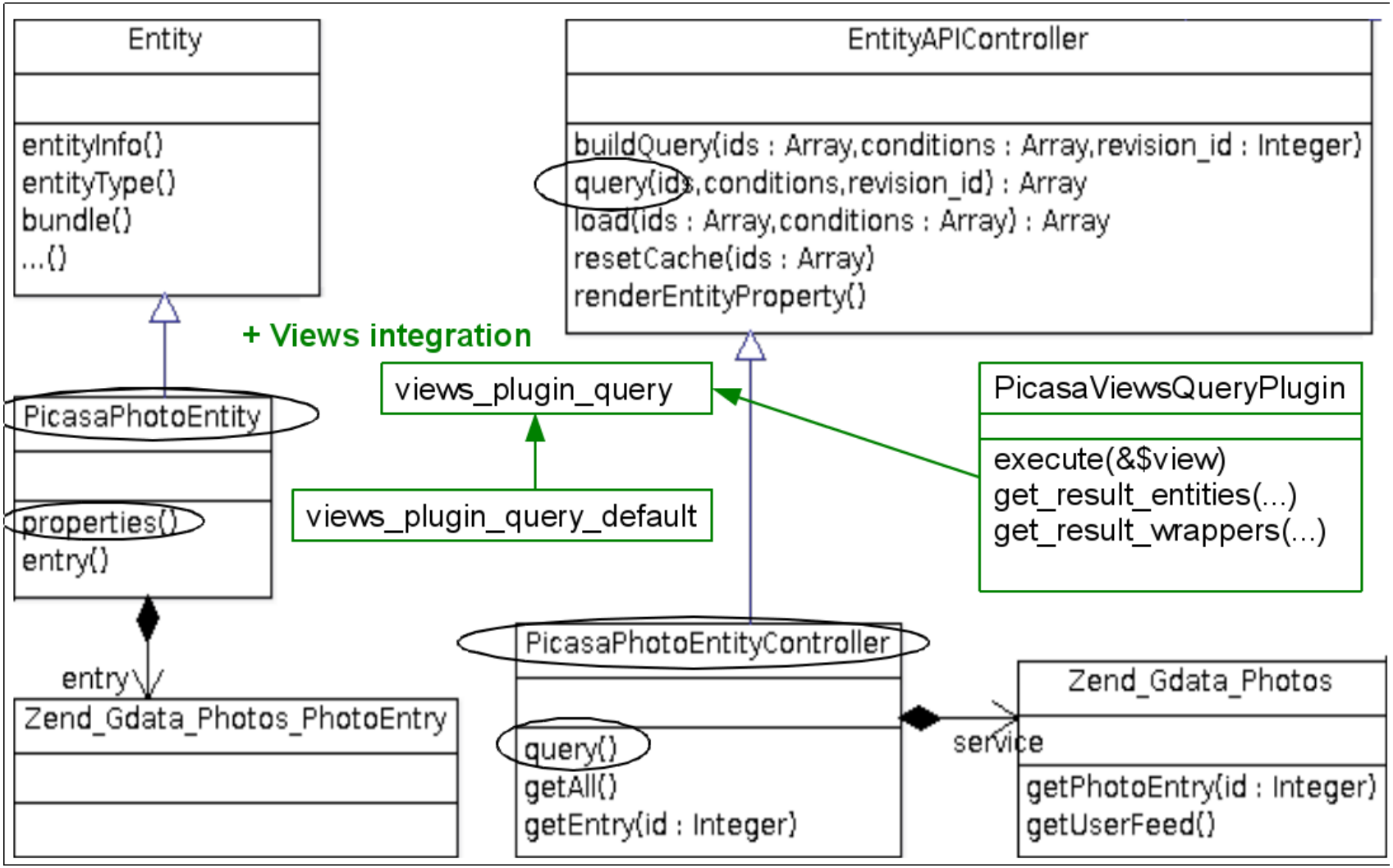
clé-valeur	mongoDB
graph - triple store	RDF
document	XML, GED

Données distantes

flux, URLs, DAV (Webdav, Caldav, Carddav)



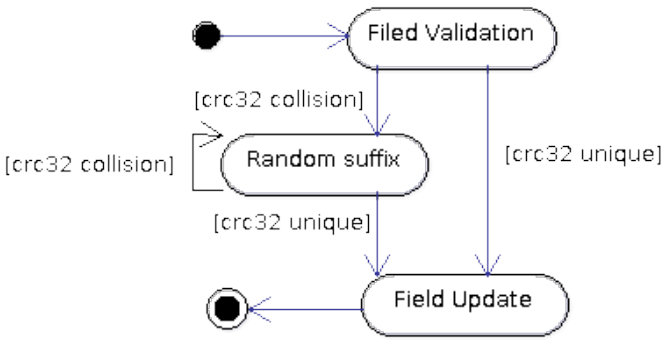
Entités Google Picasa



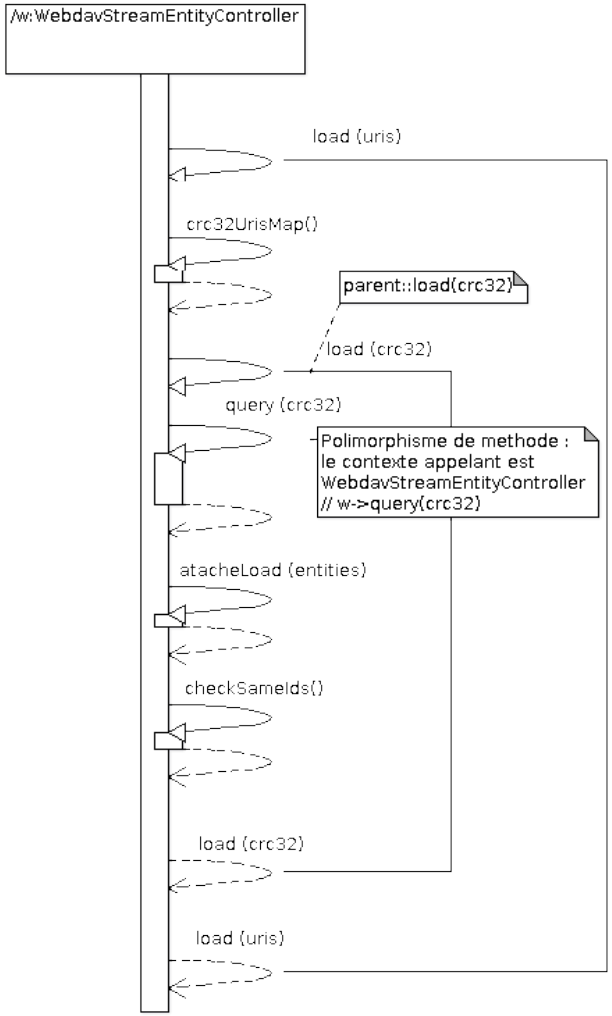
Entités « no DB » + champs

Exemple : WebDAV

- Field column **entity_id : int**
  - entité identifiable par name (URI) ?
- **URI vers Int** (CRC32)
  - Probabilité collision : $\frac{N_{items}-1}{2^{32}} \sim \frac{N_{items}}{4G}$
  - Field columns URI + suffixe (si collision)



  - Synchro sur origine (hooks API)



Allons en profiter !

- NoSQL, entités à distance, intégration de données
- API CRUD en classes + hooks
- Rapport avec les autres modules : Token, Entity Reference, Views integration

Entity API	metadata wrapper : `hook_entity_property_info()`
Rules	CRUD hooks - évènements / Data selectors / Actions / modifier séparément propriétés
VBO	Passer les ids au wrapper pour lazy loading
RDFx	getPredicates, addStatement, addResource, addLitteral
Restws	(D8) expose entités comme services web RESTful
Wsclient	Description services entité
OG	entité qui servira le type de groupe
Features	entités exportables

Drupal 8 : Entity dans le cœur

- Objets d'entités, EntityInterface
- Contrôleurs de stockage CRUD
- Unification des champs et des propriétés
- Métadonnées sur les champs / propriétés
- API de validation
- Composants réutilisables
- Les entités : du contenu ? = configuration + contenu

Drupal 8 : travailler avec les entités

- Utiliser des méthodes : isPublished(), getTitle().
- Tout est champs ! Formateurs et widgets pour tout le monde.
- Champs d'entités =
  Champs configurables (D7 fields)
  + champs de base (D7 properties)
  + champs personnalisés (D7 extra field)
- champs : comportements réutilisables pour les types d'entités en même temps que le stockage (UUID, Language, Path)
- Champs calculés lorsqu'on y accède
- Entités de configuration ?

Drupal 8 - définir un nouveau type d'entité

- Annotations sur la classe de l'entité
- Définir les champs : baseFieldDefinitions($entity_type)
- Logique de stockage indépendante
- Handlers :
  CommentStorage, CommentStorageSchema
  CommentAccessControlHandler
  CommentViewBuilder
  CommentViewsData
  CommentForm, DeleteForm
  CommentTranslationHandler

Drupal 8 : API de validation des entités

- Découplée du formulaires : services REST
- Utilise le composant Validator de Symfony
- Basée sur plugins de contraintes

Drupal 8 : interroger les entités

- Service d'interrogation, tel que défini par le stockage
  - fonctionne indépendamment
  - Ne pas interroger directement la base de données en dehors du gestionnaire de stockage (ou du service d'interrogation des entités)
- Je ne me soucie pas de MongoDB !
- Changements multilingues du schéma : interrogation des langues
- Agrégation : →aggregate('nid', 'count')
- Relations : user_id.entity.name

Drupal 8 : entités de configuration

- Entités de configuration pour « type de nœud »
- Handlers
  - NodeTypeAccessControlHandler
  - NodeTypeForm, NodeTypeDeleteConfirm
  - NodeTypeListBuilder

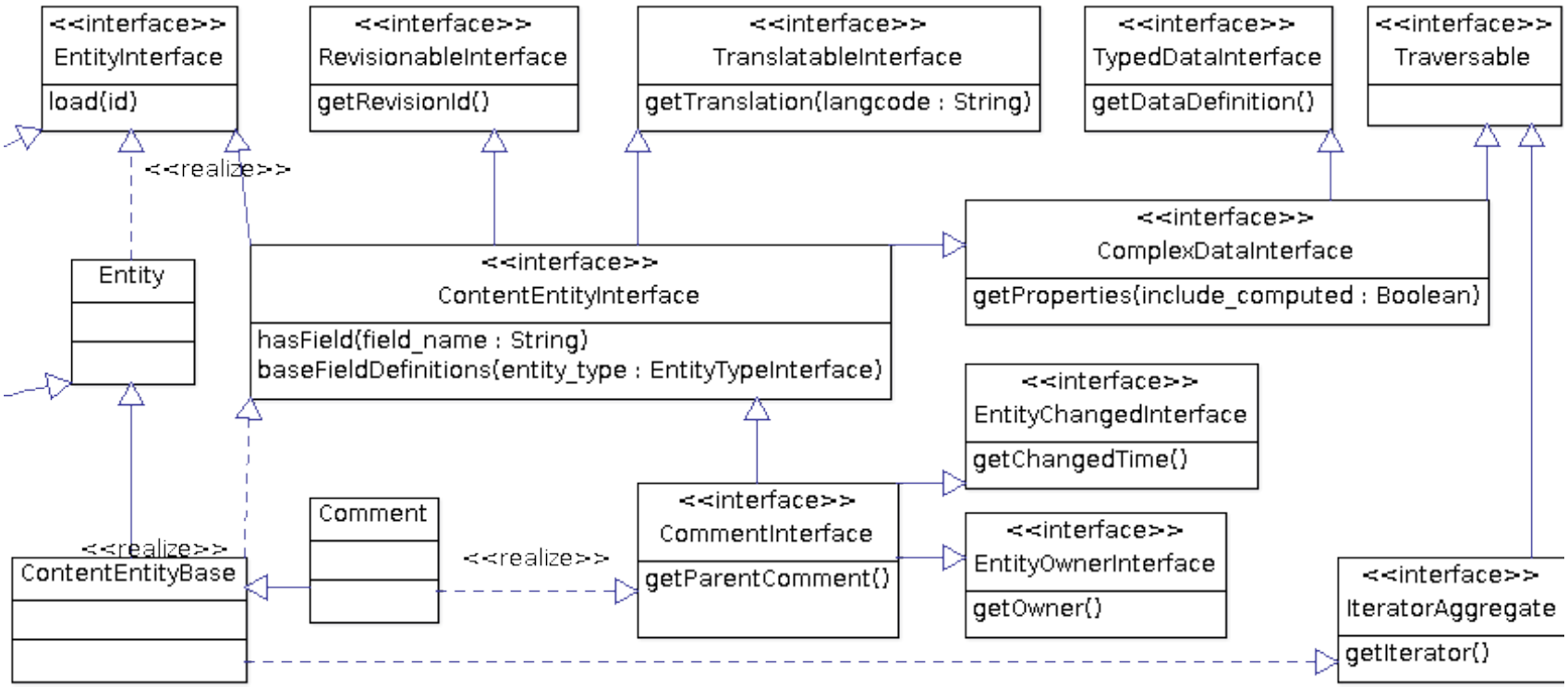
Drupal 8 API de données typées

- Il s'agit de métadonnées : en tirer parti.
- Description des données à l'aide des définitions de données
- Basé sur une liste extensible de types de données : string, integer, float, entités, nœud, field_item:image
- Primitives, ComplexData, liste

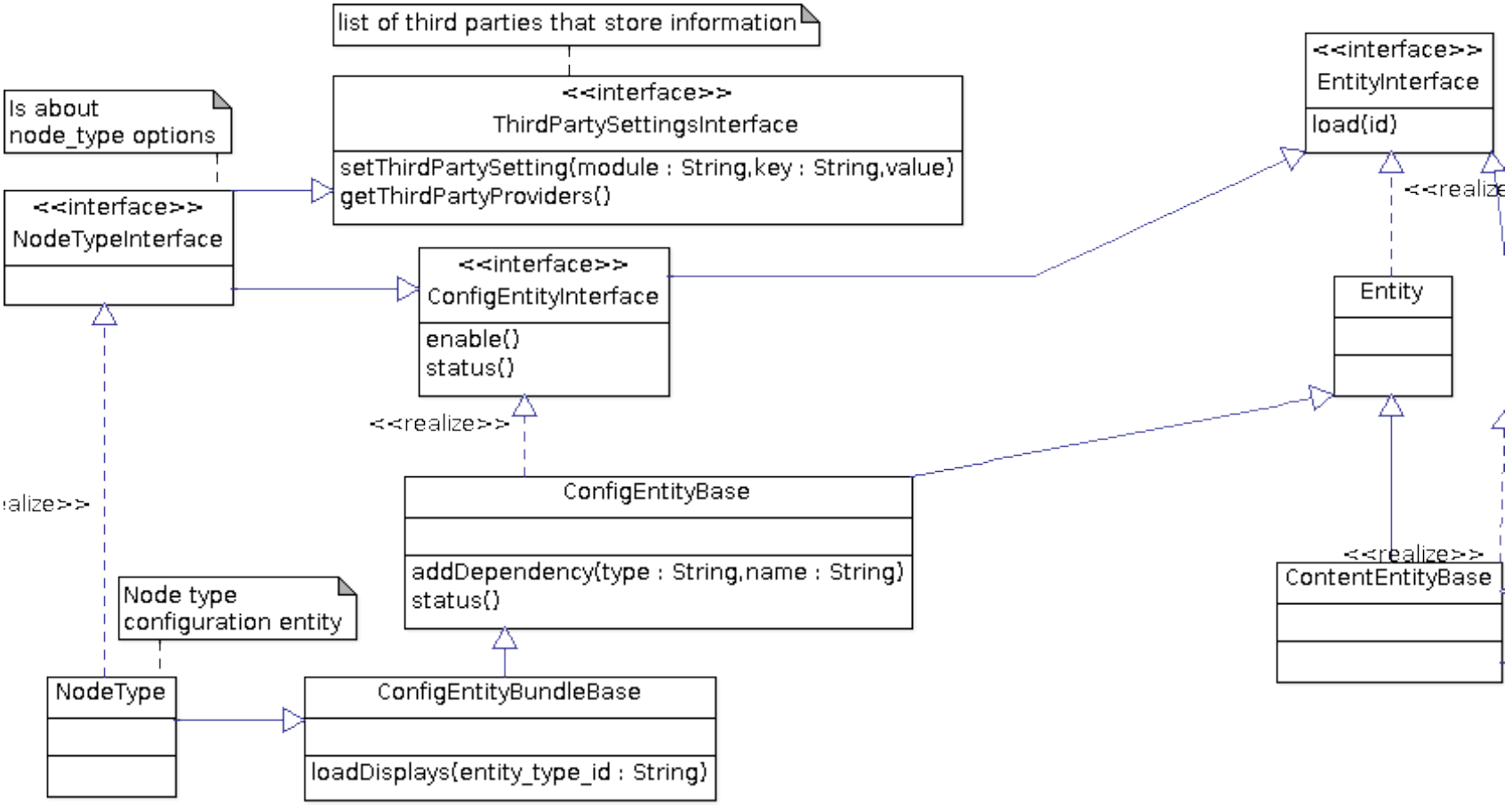
Drupal 8 : comparaison avec Drupal 7

- Entity wrapper $\rightarrow$ Entity
- EntityAPIController $\sim$ EntityStorageBase + EntityViewBuilder
- Jusqu'à présent, manquants ? EntityUIController, EntityViewsController
- Entity property info $\rightarrow$ Field definitions, Data definitions
- EntityAPIControllerExportable $\sim$ ConfigEntityStorage
- Ils y sont presque...

Drupal 8 : content entities



Drupal 8 : config entities



Drupal 8 : entities storage