w.c.s. - Autre #36334

Versionning et Historisation (suite de #4960)

22 septembre 2019 18:05 - Nicolas Roche

Statut:	Rejeté	**Début:**	22 septembre 2019
Priorité:	Bas	**Echéance:**	
Assigné à:		**% réalisé:**	0%
Catégorie:		**Temps estimé:**	0:00 heure
Version cible:			
Patch proposed:	Non	**Planning:**	Non

Description

J'ouvre ce ticket pour faire une nouvelle proposition sur le sujet.
(et ne pas faire exploser #4960 qui est déjà bien rempli)

- #4960: besoin CPF, ticket original
- #21524: besoin DPO
- #36196: un cas d'usage concret
- #35781: potentiel financement

Dans #4960 j'ai proposé une approche stockée au niveau de w.c.s. qui me semblait faisable et donc chiffrable.
Ici je me fais plaisir en vous proposant un plan qui semble mieux correspondre à vos attentes, mais qui semble moins réaliste ([1][2]).
Cela dit, la première partie exposée ci-dessous est commune aux deux approches,
et serait donc un bon candidat pour un premier jalon.

[1]: principalement parce que les objets peuvent évoluer en parallèle et conduire à des incompatibilité.

Plan:

1) Isoler les (~70 ?) macros de modifications accessibles depuis l'IHM

```
$ git grep '\.store(' wcs/admin | wc -l
70
$ git grep 'has_errors(' wcs/admin | wc -l
76
```

ex: changement du nom du formulaire
admin/forms.py::FromDefPage::title()

```
    def title(self):
        form = Form(enctype='multipart/form-data')
        ...

        if form.is_submitted() and not form.has_errors():
            new_name = form.get_widget('name').parse()
            ...
            if not form.has_errors():
<<<
                ...                         ## ICI
                self.formdef.store()        ##
---
                FormDefPageCmd(self.formdef).title(new_name)
>>>
                return redirect('.')

        ...
        r = TemplateIO(html=True)
        r += htmltext('<h2>%s</h2>') % _('Title')
        r += form.render()
        return r.getvalue()
```

2) Loguer l'appel à ces macros dans un historique intégré à chacun de ces objets :

- categories (xml)
- forms (pickle)
- workflows (pickle)
- datasources (xml)
- wscalls (xml)

ex: [1, 2, 3]
1: "date D, user U: form-title --form X --payload "{new_name: 'Y'}"

rq:

- l'historique serait conservé lors de l'export/import des formulaires et workflows

3) Exploiter ces macros via un shell de construction/modification des formulaires et workflows

ex:

```
  $ wcs-manage builder form-title --form X --payload "{new_name: 'Y'}"
```

Le payload pourra être relativement chargé pour les commandes complexes (ex: la modification des champs des formulaires).

```
> self.field.get_admin_attributes()
['label', 'type', 'condition', 'required', 'hint', 'varname', 'in_listing', 'extra_css_class', 'pr
efill', 'size', 'validation', 'data_source', 'anonymise']
```

rq:

- rien à voir mais cela permettrait de scripter des mises à jour des formulaires/workflows sur les instances dupliquées, comme par exemple signalpublik (afin les maintenir de concert sans avoir à reporter les évolutions à la main via l'IHM).

4) Stocker et exploiter d'autres historiques nommés dans les objets

(pour implémenter les snapshots)

rq:

- rejeux de l'historique depuis le début (idem pour une potentielle commande undo)
- risque de rejeux impossibles dues aux incompatibilités entre objets

ex:

```
1-(A)-2-3-(B)
       |
       +-4-(C)

A: [1]
B: [1, 2, 3]
C: [1, 2, 4]
```

5) Définir l'inverse de chacune des ~70 macros

Afin de réduire les étapes de rejeux rendues impossibles par l'incompatibilité entre les objets qui ont évolués.

ex: aller de C vers B

- détecter que B et C ont le même tronc allant jusqu'à 2 en commun
- undo 4
- redo 3

Pour les commandes complexes (ex: la modification des champs des formulaires), il faudra noter les attributs qui ont été mis à jour et enregistrer leur anciennes valeurs.

Il y a des commandes qui ne sont pas réversibles (ex: l'ajout de fonctions) et qui nécessiteraient un nouveau développement.

rq:

- permet d'exploiter des snapshots basés sur des historiques incomplets (compatibilité avec d'anciens formulaires)

En conclusion

On pourrait à présent construire les formulaires/workflow de deux façons :

- à l'import (tout d'un bloc)
- en basculant sur un snapshot (par itération)

Ceci introduit le risque que 2 objets issus d'une même version, mais récupérés chacun d'une façon différente ne soient pas parfaitement identiques.
[2] Ce qui me fait dire par ailleurs que cette approche est moins réaliste.

Vos avis ?

Demandes liées:

Lié à w.c.s. - Development #4960: Versionning sur formulaires et workflows	**Fermé**	**13 juin 2014**	**07 septembre 2020**

Historique

#1 - 22 septembre 2019 18:06 - Nicolas Roche

- Lié à Development #4960: Versionning sur formulaires et workflows ajouté

#2 - 28 septembre 2019 19:39 - Nicolas Roche

- Statut changé de Nouveau à Rejeté

Après avoir proposé deux approches : une globale à wcs (https://dev.entrouvert.org/issues/4960#note-21) et celle-ci plus farfelue qui visait la possibilité de faire des undo (#36334) j'ai ouvert les deux tickets suivants qui j'espère cette fois-ci couvriront bien les besoins évoqués.

- Historique (#36506)
- Clichés (#36507)