Разработка приложений Service Activation

В реальной жизни часто возникает необходимость выполнения одной и той же последовательности действий на одной или нескольких единицах оборудования.
Для автоматизации подобного рода задач NOC предлагает универсальный механизм: Map/Reduce Task.

Map/Reduce task состоит из трех базовых элементов - селектора, map task'ов и reduce task'а.

Селектор задает список оборудования, вовлеченного в выполнение задачи (Service Activation > Setup > Object Selectors).

На каждом объекте, определенном селектором выполняется один и тот же скрипт service activation (map task).

Reduce task - собирает результаты работы воедино и нужен для финальной обработки результата и для выдачи финального отчета.

NOC поставляется с большим количеством встроенных скриптов service activation, поддерживающих широкий спектр оборудования.
Различные аспекты их написания и применения уже неоднократно обсуждались на форуме. Основная сложность применения map/reduce task заключалась в том, что для запуска приходилось пользоваться формой Service Activation > Map/Reduce task, которая требовала задавать параметры в виде выражений python, недостаточно наглядна для рядового пользователя, а также не позволяла гранулировано управлять правами доступа.

К счастью, NOC не стоит на месте, и в новым Application Framework появился клас SAApplication,
радикально ускоряющий процесс разработки приложений SA.

Для примера рассмотрим приложение, которое позволяет выбрать список оборудования и выполнить на нем набор команд CLI. Оно включено в NOC
в виде приложения sa.runcommands

Рассмотрим процесс создания нового приложения по шагам.

Шаг 1: Создание скелета приложения

Для того, чтобы создать структуру каталогов и файлов нового приложения нужно выполнить команду

```
user@host:/opt/noc> python manage.py newapp sa.runcommands
Creating skeleton for sa.runcommands
```

В результате в каталоге sa/apps/runcommads появилось несколько файлов и каталогов. Результат можно проверить командой

```
user@host:/opt/noc> hg status sa/apps/runcommands
? sa/apps/runcommands/__init__.py
? sa/apps/runcommands/tests/__init__.py
? sa/apps/runcommands/tests/test.py
? sa/apps/runcommands/views.py
```

Шаг 2. Разработка приложения

Нас интересует файл views.py, задающий контроллер приложения (в терминах MVC). Для разработки используется Django
Отдельные детали можно прояснить в документации Django. В первую очередь нам понадобится
View Layer и Forms

В окончательном виде файл views.py будет содержать:

```
# -*- coding: utf-8 -*-
##----------------------------------------------------------------------
## Parallel command execution
##----------------------------------------------------------------------
## Copyright (C) 2007-2010 The NOC Project
## See LICENSE for details
##----------------------------------------------------------------------
from noc.lib.app.saapplication import SAApplication
from django import forms
import pprint
##
## Reduce task for commands
##
def reduce_commands(task,commands):
    r = ["<style>.cmd {border-bottom: 1px solid black;font-weight: bold;}</style>"]
    r += ["<table border='1'>","<tr><th>Object</th><th>Status</th><th>Result</th></tr>"]
    for mt in task.maptask_set.all():
        if mt.status == "C":
            result = "\n".join(["<div class='cmd'>%s</div><br/><pre>%s</pre><br/>"%(c, sr) for c, sr in zip
(commands, mt.script_result)])
        else:
            result = ""
        r += ["<tr>", "<td>", mt.managed_object.name, "</td>", "<td>", mt.status, "</td>", "<td>", result, "<
/td>", "</tr>"]
    r += ["</table>"]
    return "".join(r)
##
##
##
class RunCommandsAppplication(SAApplication):
    title = "Run commands"
    menu = "Tasks | Run Commands"
    reduce_task = reduce_commands
    map_task = "commands"

    class CommandsForm(forms.Form):
        commands = forms.CharField(widget = forms.Textarea,
            help_text = "Enter a list of commands to execute. One command per a line.")
    form = CommandsForm
    ##
    ## Convert text field to a list of commands
    ##
    def clean_map(self, data):
        return {
            "commands": [c for c in data["commands"].splitlines()]
        }
    ##
    ## Save a list of commands for reduce task
    ##
    def clean_reduce(self, data):
        return {
           "commands": [c for c in data["commands"].splitlines()]
        }
```

Строка 28 задает собственно класс приложения. Для SAApplication нам надо определить map task, reduce task и, опционально, форму для пользовательского интерфейса.

Строка 29 задает имя приложения

Строка 30 определяет пункт меню, через который будет доступно приложение. В нашем случае это Service Activation > Tasks > Run Commands.

Строка 31 говорит, что в качестве reduce task будет использоваться функция reduce_commands (строка 14).

В строке 32 указано, что на оборудовании будет выполняться скрипт commands (в нашем случае это Generic.commands

Строки 34-37 определяют форму, которую мы будем показывать пользователю. Это обычная форма Django. Механизм форм Django весьма гибок и позволяет настраивать формы под собственные нужды.

В нашем случае мы определяем единственное поле "commands", говорим, что для отрисовки надо использовать не стандартное поле типа "input", а "textarea", и на всякий случай задаем подсказку.

Для запуска map/reduce task осталось научить приложение обрабатывать результаты работы формы и выделять из нее параметры для map task и для reduce task.

Скрипт commands имеет интерфейс ICommands .
Как видно из описание - ему нужен список строк-команд, которые он будет выполнять через CLI, и сам он вернет список строк - результатов.

Строка 41 переопределяет метод clean_map. Он принимает обработанные данные от формы в виде dict (form.cleaned_data) и должен вернуть dict с параметрами для commands. Он нарезает текс по строчками и возвращает список.

Скрипт commands возвращает список ответов CLI на каждую команду. Для пущего красотизма нам надо передать список команд и в reduce task, чтобы выдать красивый отчет. Для этого в строке 48 мы переопределяем метод clean_reduce. Механизм его работы полностью идентичен clean_map, только результат передается как параметры для reduce task.

Остался последний штрих - собственно reduce task. В нашем случае это функция reduce_commands, определенная в строке 14 этого же модуля.
Первый параметр - reduce task -- всегда task (instance ReduceTask)
Остальные параметры - по необходимости. Нам понадобится commands, который получит список команд.

Со строки 17 начинается цикл, который проходит по всем map task'ам. В строке 18 проверяется статус завершения map task'а ("C" - complete, все остальные - ошибка). Для всех успешно завершенных задач мы склеиваем список команд с результатами выполнения (поле .script_result).
По результатам работы мы выдаем HTML, который будет отображен пользователю как результат работы.

Также reduce task может вернуть объект типа Report (в качестве примера можно посмотреть приложение sa.versioninventory

Шаг 3: Создание прав
Приложения SAApplication определяют единственное право доступа "run". Права синхронизируются при выполнении post-update.
Также можно создать их следующим образом

```
user@host:/opt/noc> python manage-py sync-perm
+ sa:runcommands:run
```

Теперь можно раздавать права пользователям.

Шаг 4: Перезапуск web-server'а
Чтобы увидеть изменения надо перезапустить процесс noc-web

Шаг 5: Возврат кода в проект
В случае, если вам удалось создать действительно универсальное приложение, которое будет полезно другим, не ленитесь и отправьте патч.
Для этого выберите на сайте New Issue , выберете Tracker: Patch, приложите описание
и патч. Патч можно получить как результат работы

```
user@host:/opt/noc> hg diff sa/apps/runcommands
```