Поведенчески-точный симулятор МФУ (принтеров и сканеров)

XXI конференция разработчиков свободных программ

Переславль-Залесский, 2025

Alexander Pevzner (pzz@apevzner.com)

Изображение на первой странице взято из Википедии:
https://commons.wikimedia.org/wiki/File:BESM-6_ACPY.jpg

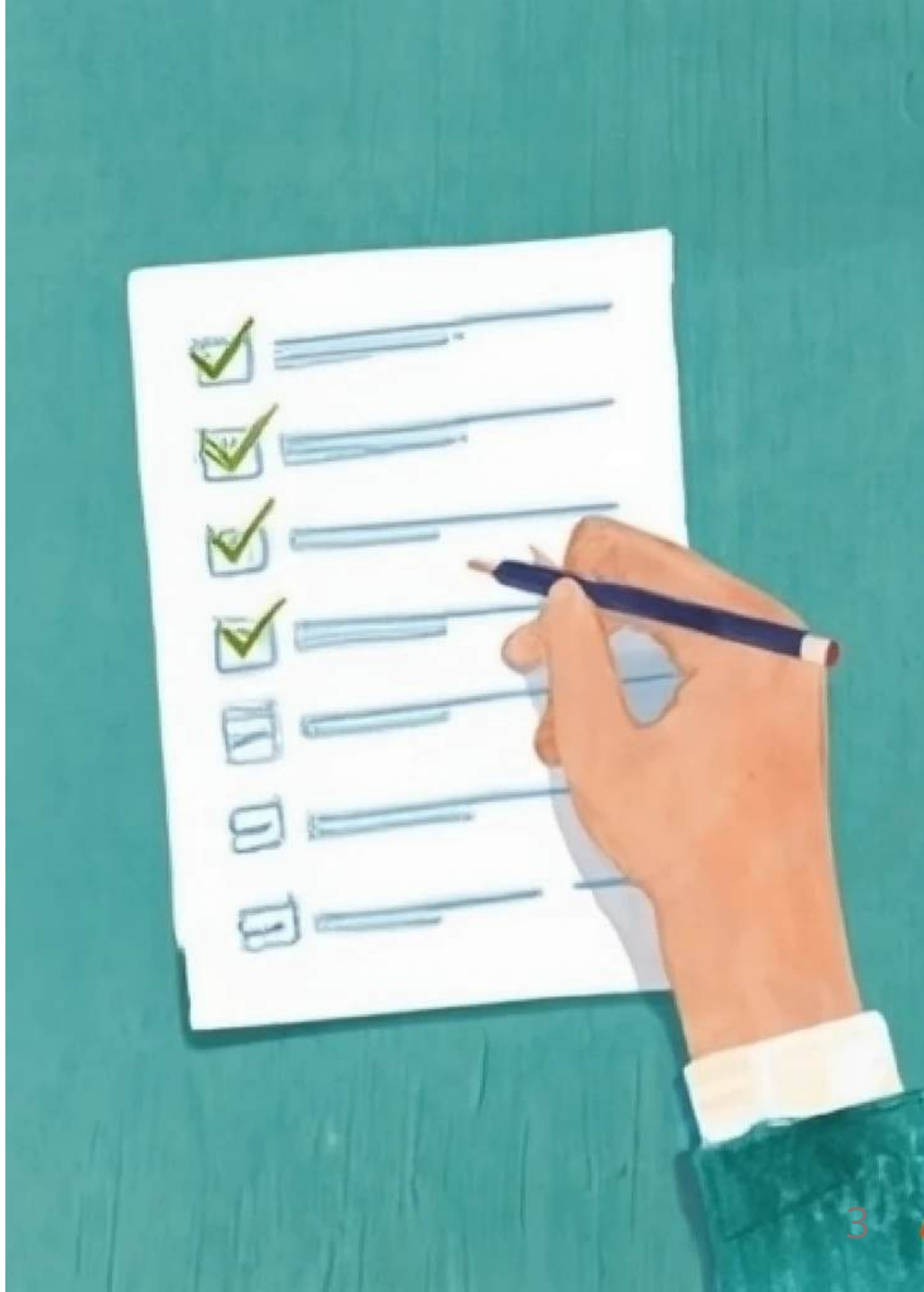
Остальные изображения созданы ИИ (Craiyon и Kandinsky)

Alexander Pevzner

- Член OpenPrinting с 2020
- Пишу системное ПО на C и Go
- Автор повсеместно используемыхipp-usb и sane-airscan
- @alexpevzner на GitHub
- Работаю в команде, которая переводит всю Москву с Windows на Linux

Краткое содержание

- Зачем вообще нужен симулятор МФУ?
- Что значит, поведенчески точная симуляция?
- Рамки этого проекта
- МФУ очень сложны. Как сделать модели простыми?
- Вспомогательные утилиты
- Текущее состояние проекта
- Дочерние проекты



Alexander Pevzner (pzz@apevzner.com)

Зачем нам нужен симулятор МФУ?

- МФУ - большие, тяжёлые и дорогие
- Собрать репрезентативную коллекцию трудно даже для корпораций
- Программное обеспечение сканирования и печати очень сложное
- Его разработка и поддержка требуют воспроизводимости
- Выход: поведенчески точный симулятор

Alexander Pevzner (pzz@apevzner.com)

Рамки этого проекта

- Стандартный протокол печати (IPP)
- Стандартные протоколы сканирования (eSCL and WSD)
- Симуляция IPP over USB
- Анонсирование по протоколам DNS-SD и WS-Discovery
- Частичная реализация устаревших протоколов печати (USB print class, LPD и т.п)
- Поддержка проприетарных протоколов не планируется

Alexander Pevzner (pzz@apevzner.com)

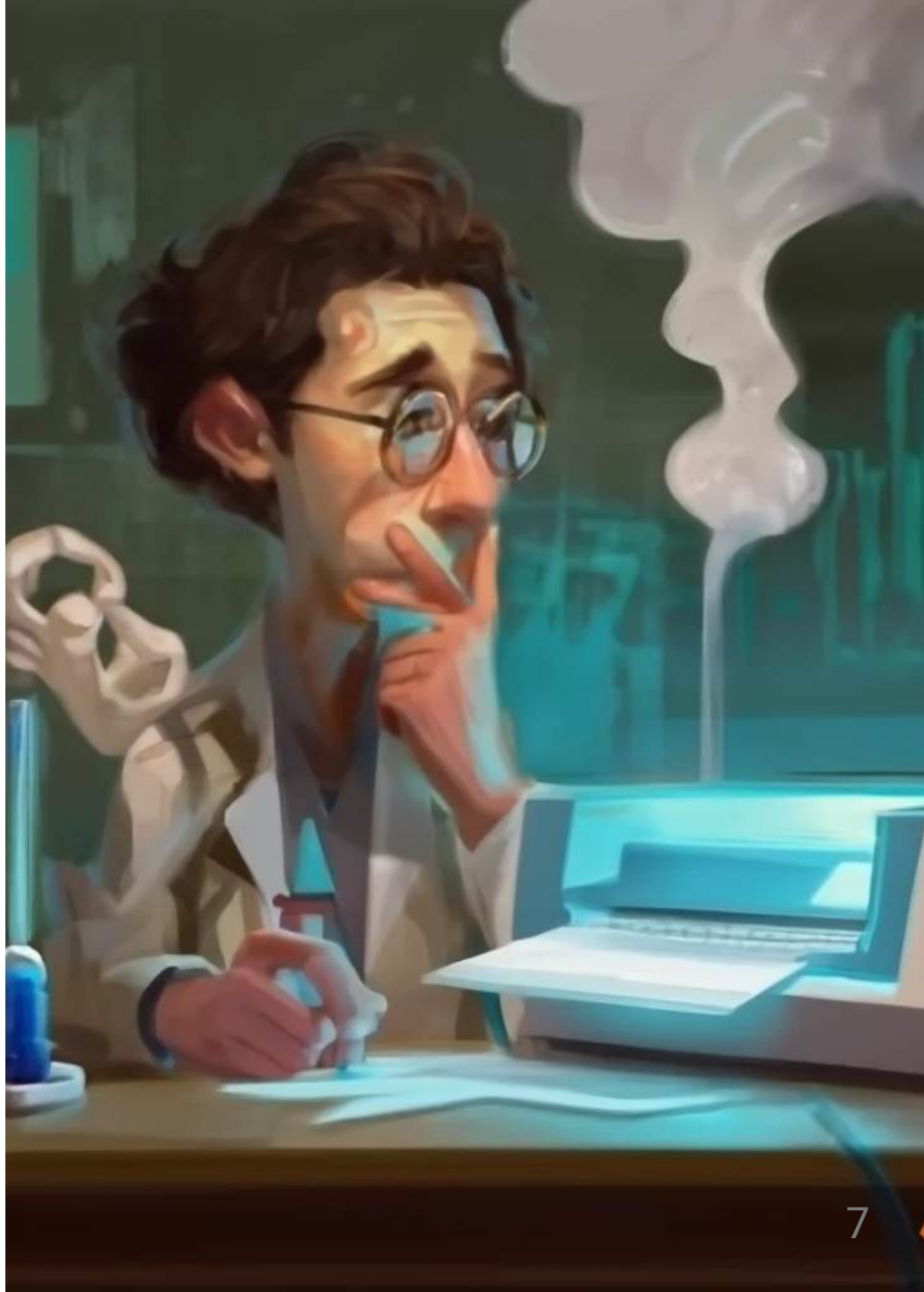
Что значит, поведенчески точный?

- Для стандартных протоколов поведение определяется спецификациями и параметрами устройста
- Но этого не достаточно. Поведение реальной аппаратуры нередко отклоняется от стандартов
- Кроме параметров, модели должны определять особенности поведения устройств за рамками стандартов

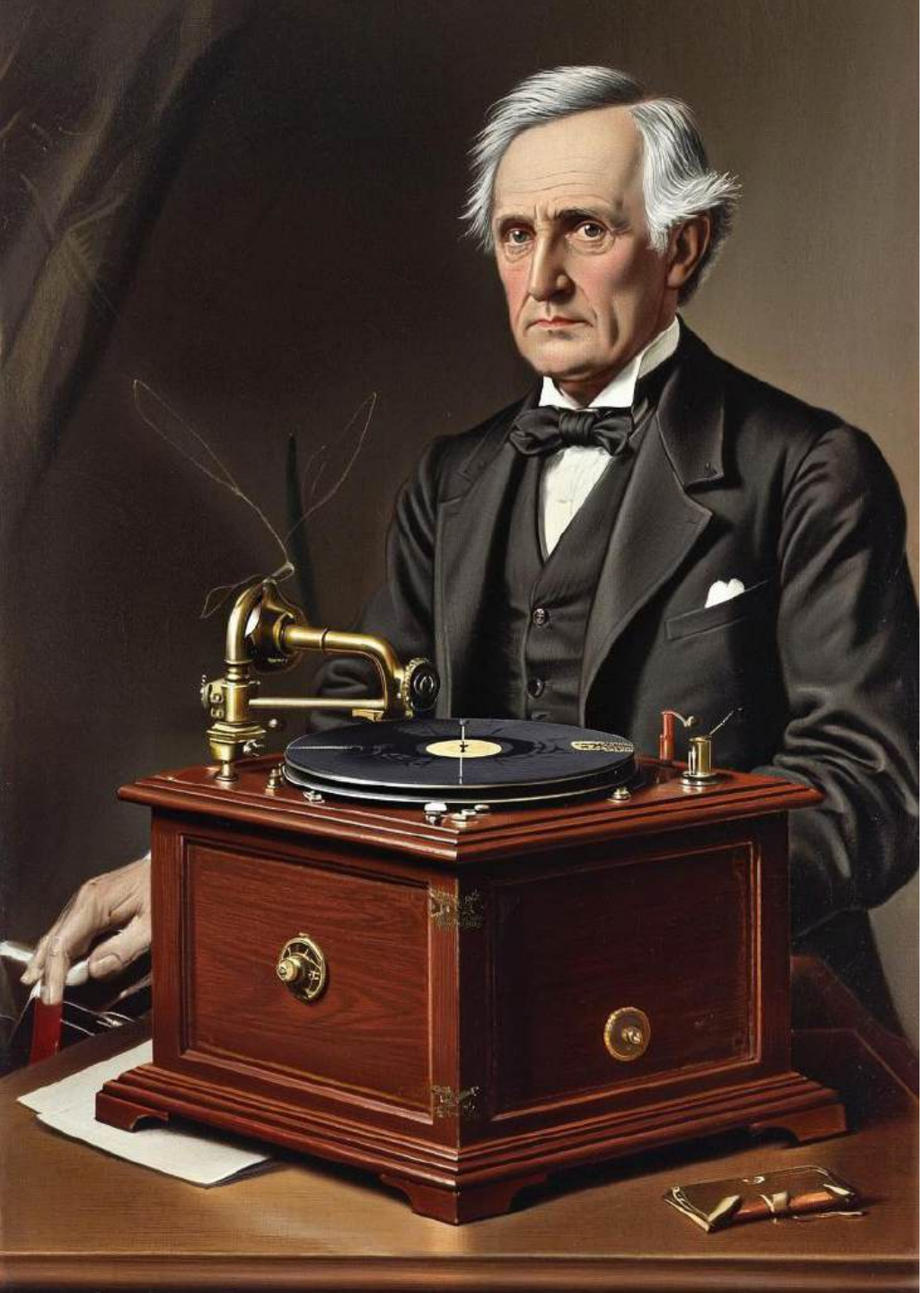
Alexander Pevzner (pzz@apevzner.com)

Насколько точны могут быть наши модели?

- Главным ограничивающим фактором является наше детальное знание конкретного устройства
- Но стопроцентная точность и не требуется. Достаточно воспроизвести существенные детали
- Во многих случаях достаточно просто воспроизвести саму проблему



Alexander Pevzner (pzz@apevzner.com)



Создание моделей

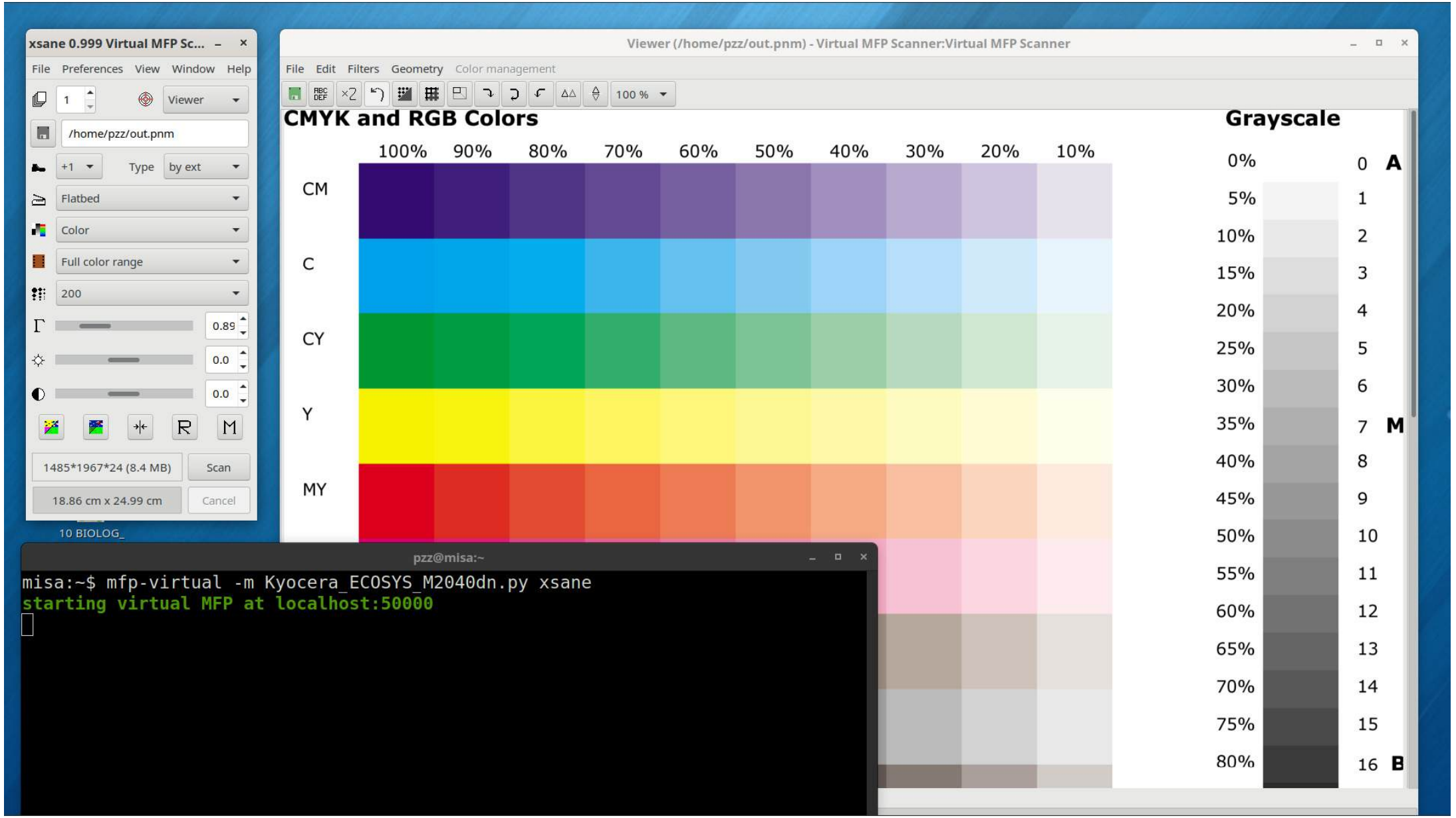
- Основа модели - это просто запись параметров устройства (printer attributes или scanner capabilities)
- Такие модели не включают особенностей поведения
- Утилита `mfp-model` записывает эти базовые модели с устройства автоматически
- Программа `mfp-virtual` "проигрывает" модели, эмулируя реальное устройство

Попробуем на живом примере

Это фрагмент модели eSCL сканера от МФУ `Kyocera ECOSYS M2040dn`, автоматически записанный утилитой `mfp-model`

```
# eSCL scanner parameters:
escl.caps = {
  'Version': '2.62',
  'MakeAndModel': 'Kyocera ECOSYS M2040dn',
  'SerialNumber': 'VCF9192281',
  'Uuid': UUID('4509a320-00a0-008f-00b6-002507510eca'),
  'AdminUri': 'https://KM7B6A91.local/airprint',
  'IconUri': 'https://KM7B6A91.local/printer-icon/machine_128.png',
  'Platen': {
    'PlatenInputCaps': {
      'MinWidth': 118,
      'MaxWidth': 2551,
      'MinHeight': 118,
      'MaxHeight': 3508,
      'SupportedIntents': ['Document', 'TextAndGraphic', 'Photo', 'Preview'],
      'SettingProfiles': [
        {
          'ColorModes': ['BlackAndWhite1', 'Grayscale8', 'RGB24'],
          'DocumentFormats': ['image/jpeg', 'application/pdf'],
          'SupportedResolutions': [
            {
              'DiscreteResolutions': [
                {'XResolution': 200, 'YResolution': 100},
                {'XResolution': 200, 'YResolution': 200},
                {'XResolution': 200, 'YResolution': 400},
                {'XResolution': 300, 'YResolution': 300},
                {'XResolution': 400, 'YResolution': 400},
                {'XResolution': 600, 'YResolution': 600}
              ]
            }
          ]
        }
      ],
      'FeedDirections': ['ShortEdgeFeed', 'LongEdgeFeed']
    }
  },
  'Adf': {
    'AdfSimplexInputCaps': {
      'MinWidth': 591,
      'MaxWidth': 2551
```

Alexander Pevzner (pzz@apevzner.com)


xsane 0.999 Virtual MFP Sc...
File Preferences View Window Help
Viewer
/home/pzz/out.pnm
Type by ext
Flatbed
Color
Full color range
200
0.89
0.0
0.0
1485*1967*24 (8.4 MB)
Scan
18.86 cm x 24.99 cm
Cancel
10 BIOLOG_
Viewer (/home/pzz/out.pnm) - Virtual MFP Scanner:Virtual MFP Scanner
File Edit Filters Geometry Color management
100 %
CMYK and RGB Colors
100% 90% 80% 70% 60% 50% 40% 30% 20% 10%
CM
C
CY
Y
MY
Grayscale
0% 0 A
5% 1
10% 2
15% 3
20% 4
25% 5
30% 6
35% 7 M
40% 8
45% 9
50% 10
55% 11
60% 12
65% 13
70% 14
75% 15
80% 16 B
pzz@misa:~
misa:~$ mfp-virtual -m Kyocera_ECOSYS_M2040dn.py xsane
starting virtual MFP at localhost:50000

Добавим особенности поведения

- С помощью автоматически записанных моделей мы можем воспроизвести идеализированное поведение МФУ, заданное параметрами устройства и реализованное строго по стандарту
- Теперь попробуем добавить специфические для устройства особенности поведения

Язык описания моделей

- Модель сама по себе представляет собой Python-скрипт
- Параметры принтера и сканера описываются в виде словарей Python
- Для изменения поведения модели предусмотрен набор точек расширения на Python
- Все эти обработчики - опциональны
- Симулятор написан на Go и содержит встроенный Python

Alexander Pevzner (pzz@apevzner.com)

Практический пример

- Сканер анонсирует поддержку JPEG и PNG
- `sane-airscan` в таких случаях всегда выбирает PNG, как формат без потери качества
- Однако фактически устройство присылает JPEG а не PNG
- Это приводит к ошибке декодирования (в `sane-airscan` пришлось добавить детектирования формата изображения)

- `escl_onScanJobsRequest`, функция, определенная в модели, может модифицировать eSCL запрос на сканирование
- В логе `sane-airscan` видно, что формат изображения сменился с PNG на JPEG
- Нам потребовалось всего несколько строк на Python, чтобы описать такое поведение устройства

Virtual-MFP-Scanner.py =+ (~/tmp/virt) - VIM

```
# Called on eSCL scan request
def escl_onScanJobsRequest (
    q: query.Query, rq: escl.ScanSettings):

    if rq['DocumentFormat'] == 'image/png':
        rq['DocumentFormat'] = 'image/jpeg'
    if rq['DocumentFormatExt'] == 'image/png':
        rq['DocumentFormatExt'] = 'image/jpeg'
                                              8,1           All
```

Without DocumentFormat hook.log (~/tmp/logs) - VIM

```
Virtual MFP Scanner: Image received with the following paramet
Virtual MFP Scanner:   format:         png
Virtual MFP Scanner:   content type:   image/png
Virtual MFP Scanner:   frame format:   RGB
Virtual MFP Scanner:   image size:     2480x3507
Virtual MFP Scanner:   color depth:    8

~
:set nospell                                  7,0-1         All
```

With DocumentFormat hook.log (~/tmp/logs) - VIM

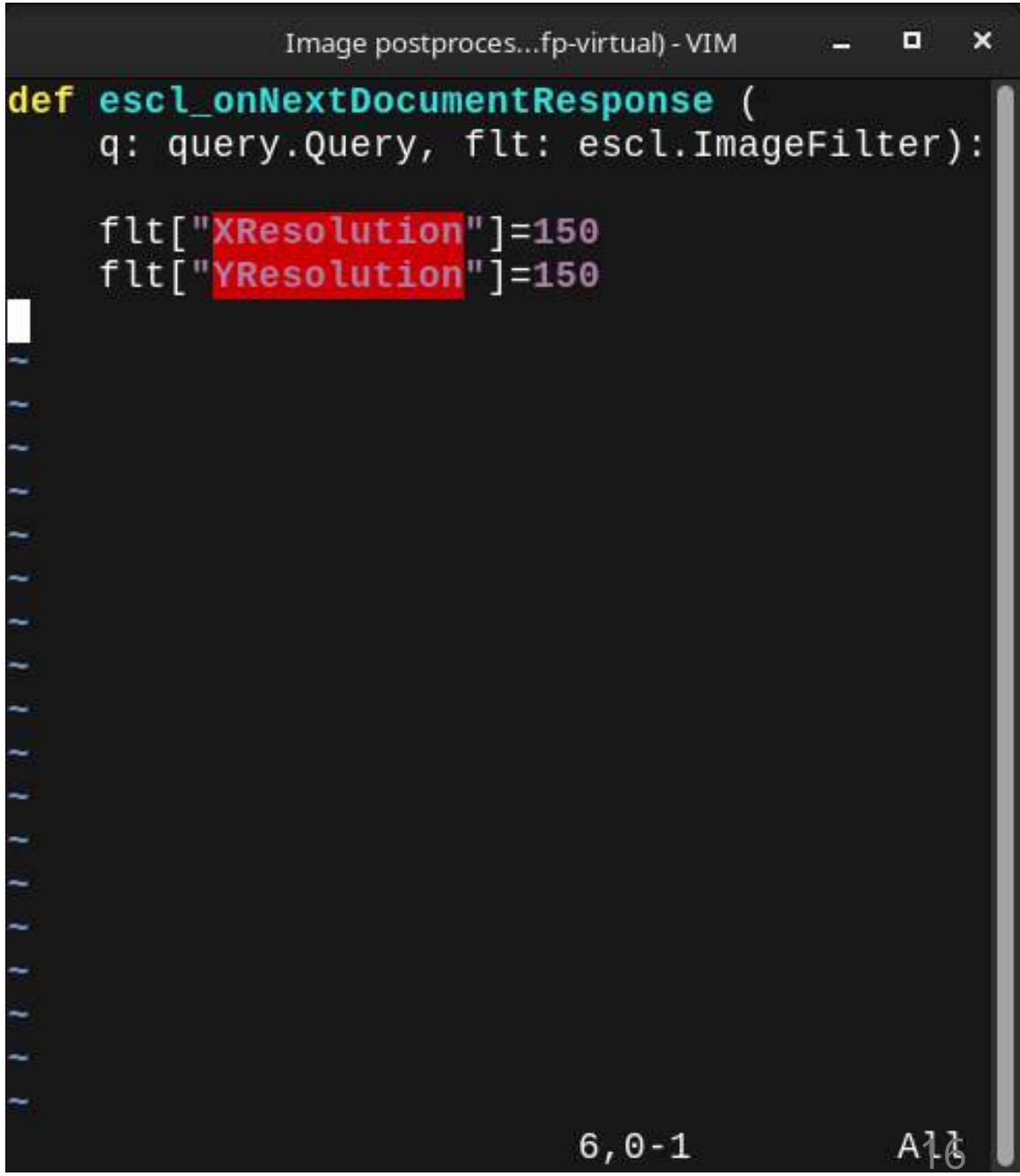
```
Virtual MFP Scanner: Image received with the following paramet
Virtual MFP Scanner:   format:         jpeg
Virtual MFP Scanner:   content type:   image/jpeg
Virtual MFP Scanner:   frame format:   RGB
Virtual MFP Scanner:   image size:     2480x3507
Virtual MFP Scanner:   color depth:    8

~
:set nospell                                  7,0-1         All
```

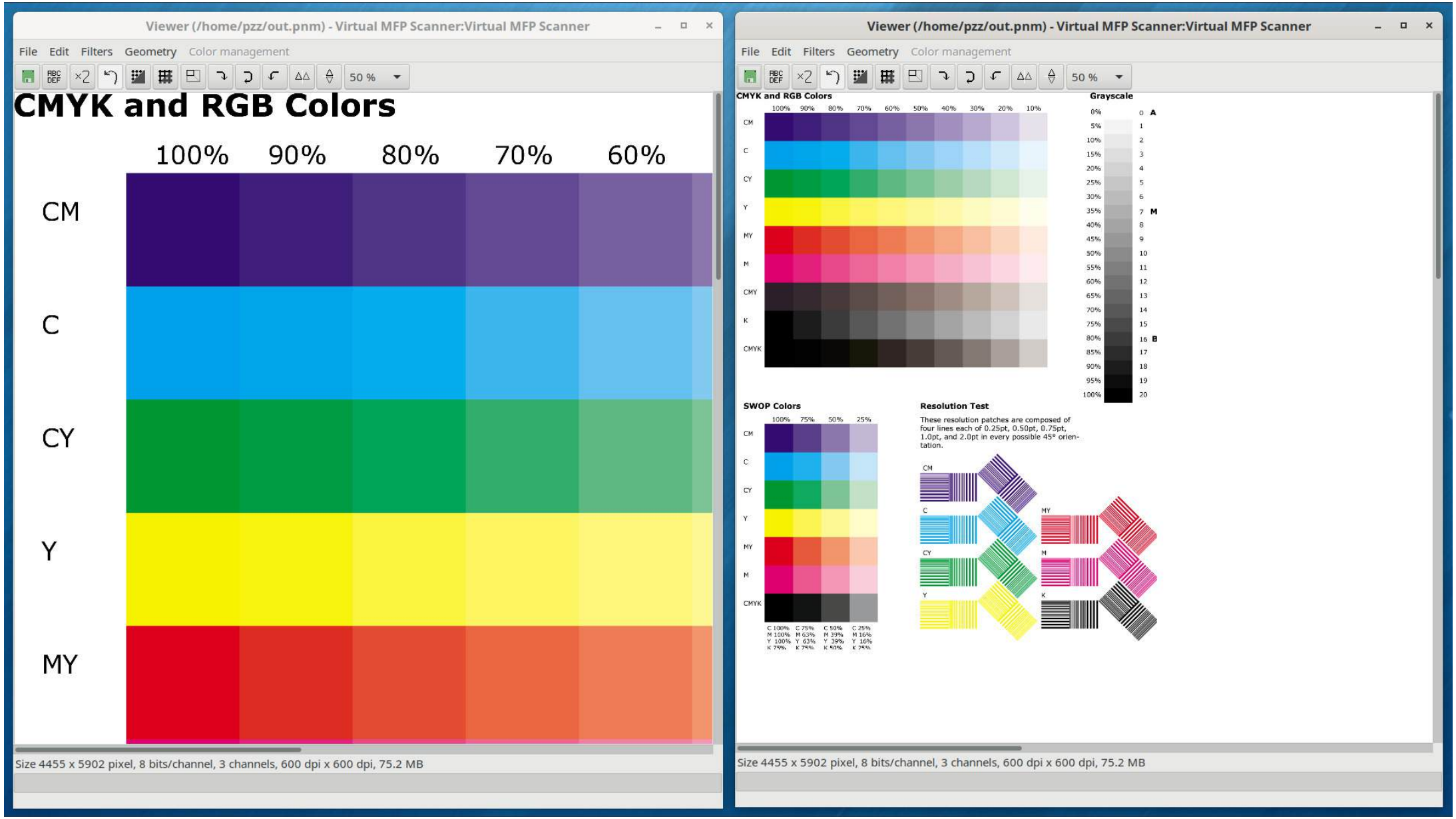
Обработка изображений

- Симулятор так же содержит конвейер обработки изображений
- Изображения можно преобразовать: изменить разрешение, обрезать, преобразовать цветовую модель и так далее
- Это используется для эмуляции прошивок с ошибками и тестирования наших драйверов

Обработка изображений: попробуем в деле

- `escl_onNextDocumentResponse` вызывается, когда изображение готово
- Тут можно установить параметры постобработки изображения
- Изменим разрешения с 600x600 на 150x150
- Результат на следующем слайде



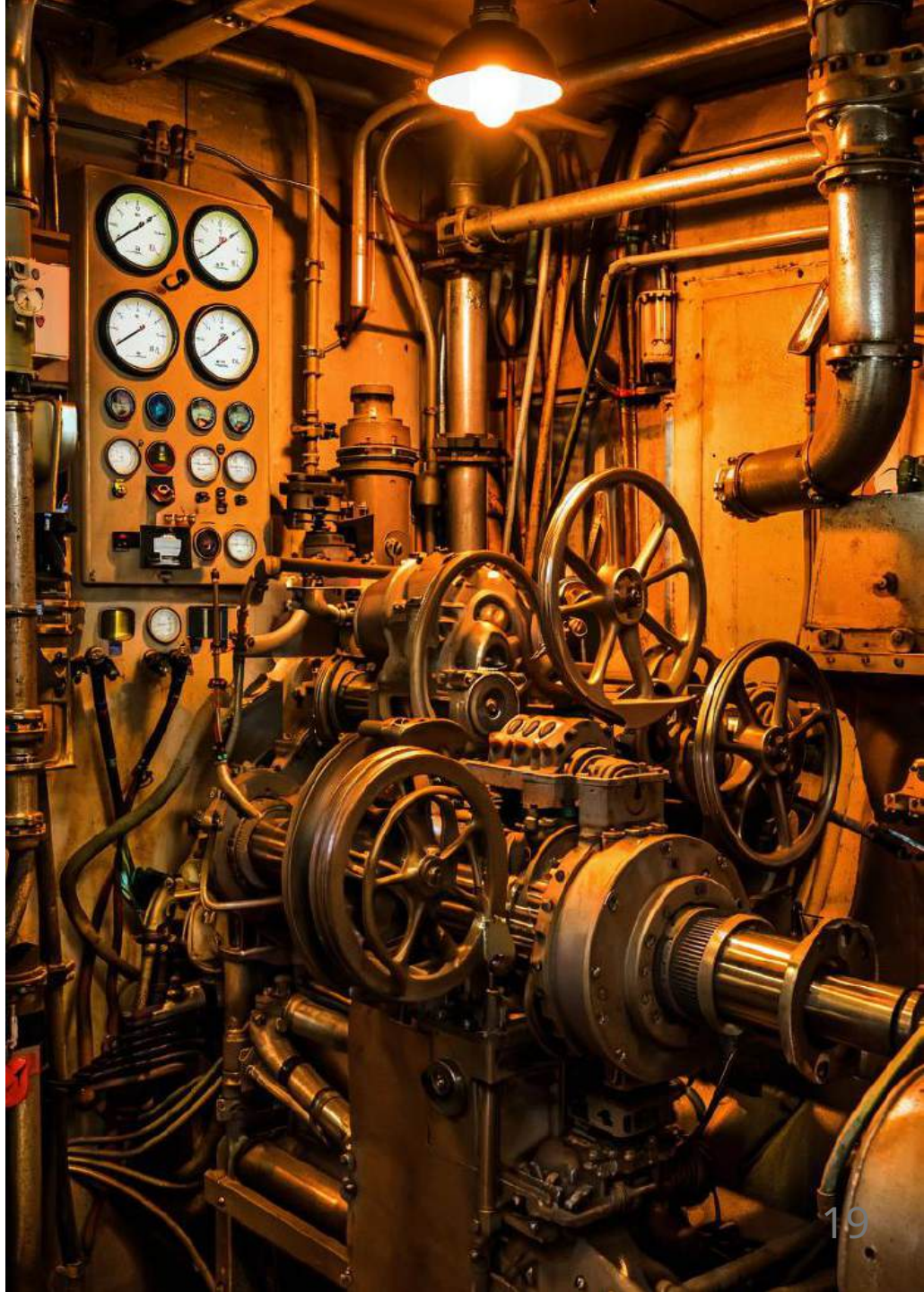

Viewer (/home/pzz/out.pnm) - Virtual MFP Scanner:Virtual MFP Scanner
File Edit Filters Geometry Color management
50 %
CMYK and RGB Colors
100% 90% 80% 70% 60%
CM
C
CY
Y
MY
Size 4455 x 5902 pixel, 8 bits/channel, 3 channels, 600 dpi x 600 dpi, 75.2 MB
Viewer (/home/pzz/out.pnm) - Virtual MFP Scanner:Virtual MFP Scanner
File Edit Filters Geometry Color management
50 %
CMYK and RGB Colors
Grayscale
SWOP Colors
Resolution Test
These resolution patches are composed of four lines each of 0.25pt, 0.50pt, 0.75pt, 1.0pt, and 2.0pt in every possible 45° orientation.
Size 4455 x 5902 pixel, 8 bits/channel, 3 channels, 600 dpi x 600 dpi, 75.2 MB

Что мы имеем

- Ядро симулятора, которое реализует "идеальное" поведение устройства
- Параметры модели задают устройство в терминах конкретного протокола
- Набор хуков на Python, позволяющих описать особенности поведения
- Вспомогательные утилиты для создания моделей
- Благодаря такой архитектуре наш симулятор сочетает в себе простоту и выразительность

Под капотом

- Ядро симулятора — это набор Go-библиотек с универсальной и полной реализацией ряда протоколов (IPP, eSCL, WSD и др.)
- Входящие в комплект утилиты добавляют к ним лишь интерфейс командной строки
- Эта кодовая база может послужить основой других проектов, не обязательно связанных с симуляцией



Режим proxy

- Отдельного упоминания заслуживает программа `mfp-proxy`.
- Она реализует IPP/eSCL/WSD прокси.
- Может использоваться в качестве сниффера этих протоколов.
- Модели устройств могут быть "наложены" на реальные, а не виртуальные устройства, что позволяет изменить их характеристики и поведение

Дочерние проекты

- Симулятор - это большой проект, около 50K строк на Go и около 25K строк тестов
- В процессе разработки образовалось некоторое количество дочерних проектов, которые, вероятно, в будущем станут самостоятельными
- Сейчас я кратко расскажу о некоторых из них

Go-обёртка для Avahi (cgo)

Полная обертка Avahi API на идеоматическом Go:

- Настолько близко к C API, насколько возможно
- Идеоматический Go: события доставляются через каналы, а не обратные вызовы
- Очень полная документация, раскрывающая многие нюансы. Может быть полезна даже для программирования на C
- Выделена в самостоятельный проект (https://github.com/OpenPrinting/go-avahi)

Go интерфейс к CPython

Отличия от других реализаций:

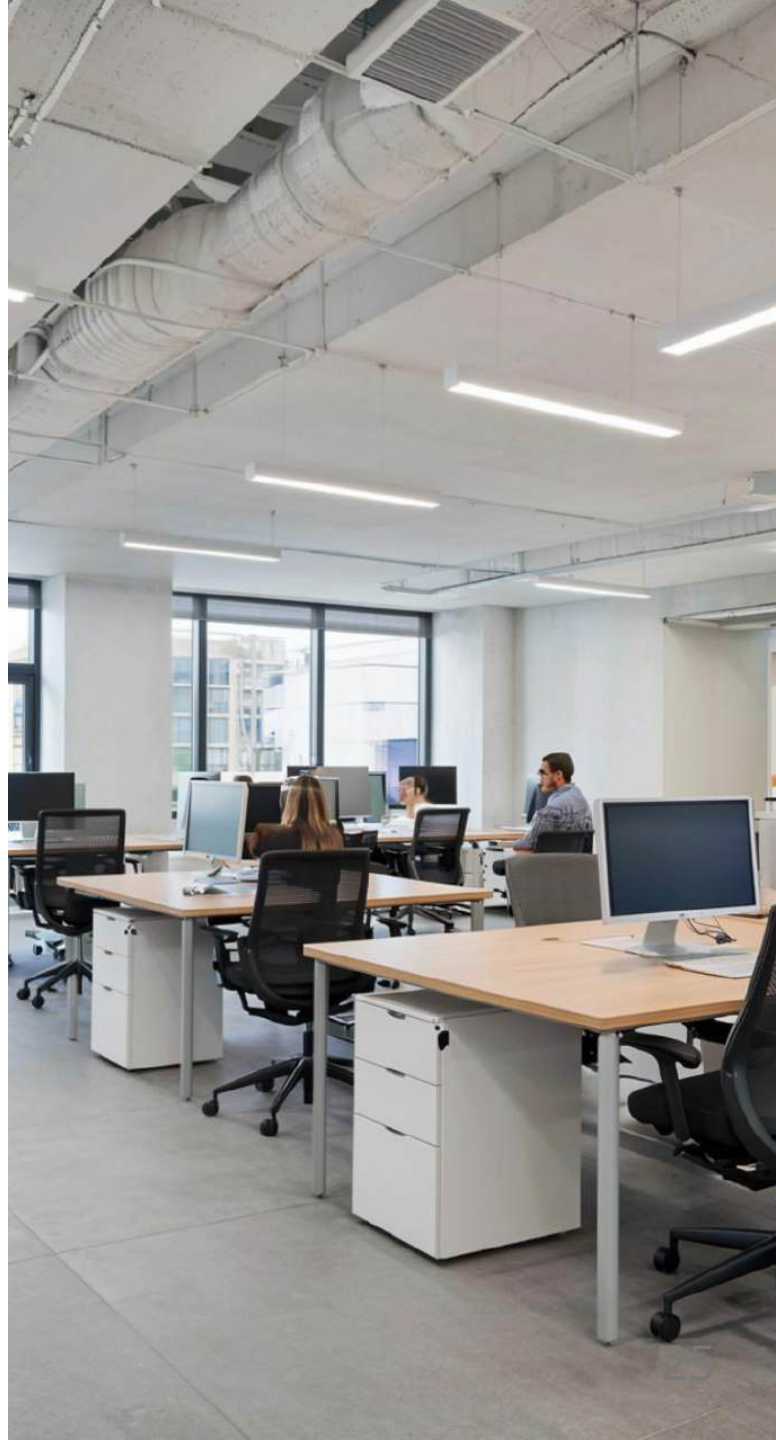
- Использует CPython stable API. Выдерживает обновление Python без пересборки
- Объекты Python со строны Go управляются сборщиком мусора, не вручную
- Поддерживает множественные суб-интерпретаторы
- Совместим с потоками Go
- Технические подробности - в статье на Хабре (https://habr.com/ru/articles/919590/)

Текущее состояние проекта

- Проект находится в активной разработке
- Симулятор сканера eSCL почти завершён
- Симулятор сканера WSD в разработке (by Yogesh Singla, yogesh1801 at GitHub)
- Поддержка IPP частично реализована
- Поддержка IPP-over-USB работает на уровне Proof-of-Concept
- Анонсирование по протоколам DNS-SD и WS-Discovery еще не начато

Планы и перспективы

- Проект уже используется в разработке `sane-airscan` и решении проблем с печатью
- Предполагается, что основные части будут закончены к концу этого года
- Планируется создание автоматической системы тестирования стека сканирования и печати на виртуальном оборудовании
- Интеграция с фреймворком для оценки изображений, созданным Sanskar Yaduka



Alexander Pevzner (pzz@apevzner.com)

Заключение

Спасибо за внимание!

Буду рад ответить на любые вопросы

Alexander Pevzner (pzz@apevzner.com)

Контакты

- Страница проекта: https://github.com/OpenPrinting/go-mfp
- E-mail автора: pzz@apevzner.com
- Telegram автора: https://t.me/a_pevzner
- Канал OpenPrinting в Telegram: https://t.me/+RizBbjXz4uU2ZWM1