УДК 004.514

РЕАЛИЗАЦИЯ АРХИТЕКТУРЫ ПРИЛОЖЕНИЯ-КОНВЕРТЕРА ДЛЯ ПРЕОБРАЗОВАНИЯ ТЕКСТА В PDF-ДОКУМЕНТЕ В ВЕКТОРНЫЕ КРИВЫЕ

Е. И. ЮРЧИШКО
(Представлено: канд. техн. наук, доц. В. М. ЧЕРТКОВ)

Рассматриваются особенности форматов документов PDF и SVG с акцентом на представление текстового содержимого и векторной графики. Формулируется задача конвертации текста из статического PDF в масштабируемую векторную графику SVG с сохранением визуальной идентичности. Предлагается архитектура настольного приложения на платформе WPF (Windows Presentation Foundation), реализующего комбинированный подход к обработке текста: прямое извлечение векторных контуров при их наличии в PDF и генерация глифов на основе системных шрифтов в случае текстовых операторов. Описывается многослойная структура решения с применением паттерна MVVM (Model-View-ViewModel), а также ключевой алгоритм преобразования символов в векторные кривые с нормализацией имён шрифтов и конечной сериализацией в элементы <path>.

Введение. Формат PDF (Portable Document Format), несмотря на свою универсальность и широкое распространение, является в первую очередь статическим: он отлично сохраняет внешний вид документа, но затрудняет его дальнейшую интерактивную обработку. В то же время формат SVG (Scalable Vector Graphics) предлагает обратное – динамичную, редактируемую и масштабируемую векторную графику, идеально подходящую для интеграции в веб-приложения, цифровые архивы и интерактивные интерфейсы.

Задача преобразования текстового содержимого PDF в SVG возникает в различных сценариях: от создания веб-версий печатных каталогов до подготовки документов для систем оптического анализа и визуализации. Однако прямая конвертация сталкивается с рядом технических сложностей: различия в системах координат, отсутствие гарантий доступности шрифтов, необходимость точного воспроизведения позиционирования и стилей текста. Особенно критичным становится вопрос шрифтов: если в PDF используется встроенный или редкий шрифт, его отображение в SVG без потери визуальной идентичности возможно только при конвертации текста в векторные контуры.

Постановка задачи. Целью исследования является разработка архитектуры для специализированной настольной утилиты, реализующей механизм конвертации текста из PDF в SVG. Выбор следующих технологий для разработки приложения обоснован требованиями к наличию удобного пользовательского графического интерфейса, а также хорошей совместимостью с операционной системой Windows: платформа Windows Presentation Foundation (WPF) с использованием языка C# и современных .NET-библиотек. Основной акцент сделан на архитектурной чистоте, производительности и удобстве использования. В статье подробно описывается выбранная архитектура приложения, логика взаимодействия компонентов, а также ключевой алгоритм – преобразование символов в векторные кривые с сохранением всех визуальных атрибутов. WPF предоставляет комплексный набор функций разработки приложений, которые включают в себя язык XAML, элементы управления, привязку к данным, макет, двумерную и трехмерную графику, анимацию, стили, шаблоны, документы, мультимедиа, текст и типографические функции [1].

Выбор архитектуры приложения. При проектировании приложения-конвертера особое внимание должно уделяться модульности, тестируемости и масштабируемости. Для удовлетворения данным критериям следует использовать архитектуру, разделяющую приложение на следующие слои: представление (Presentation Layer), бизнес-логика (Application Layer).

Слой представления должен быть реализован с использованием WPF и паттерна MVVM (Model-View-ViewModel). Это позволит полностью отделить логику интерфейса от визуального отображения, обеспечив гибкость при изменении дизайна и упростив автоматизированное тестирование. ViewModel отвечает за управление состоянием UI: активацию кнопок, отображение прогресса, обработку ошибок. Привязки данных (data binding) и команды (ICommand) обеспечивают декларативное взаимодействие между компонентами архитектурного паттерна View и ViewModel без прямых ссылок на элементы управления.

Слой View реализуется в XAML (декларативный язык разметки на основе XML, разработанный Microsoft, который используется для создания пользовательских интерфейсов (UI) приложений, в частности, для технологий Windows Presentation Foundation (WPF) и .NET MAUI) и содержит минимальную логику: элементы управления (поле ввода пути, кнопки) связаны с ViewModel через механизм привязок данных (data binding). Все пользовательские действия (выбор файла, запуск конвертации) передаются через команды (ICommand), что исключает прямую зависимость от кода интерфейса.

Слой ViewModel управляет состоянием UI: путь к файлу, статус операции, активность элементов управления. Он не содержит логики обработки документов, а лишь делегирует задачи сервису конвертации, что соответствует принципу инверсии зависимостей.

Ядро приложения – сервис конвертации – инкапсулирует всю сложность: открытие PDF, анализ его структуры, выбор режима обработки, генерацию SVG и управление ресурсами. Сервис проектируется как независимый от UI компонент, что позволяет легко интегрировать его в другие типы приложений (например, консольные утилиты или веб-сервисы).

Особое внимание должно быть уделено производительности и отзывчивости интерфейса. Все ресурсоёмкие операции следует выполняють асинхронно с использованием async/await, а обновление статуса – через интерфейс IProgress<T>, что гарантирует потокобезопасное взаимодействие с UI без блокировок. Асинхронность позволяет вынести отдельные задачи из основного потока в специальные асинхронные методы и при этом более экономно использовать потоки. Хорошая аналогия, позволяющая лучше понять суть асинхронных методов, — кнопки «пауза» и «воспроизведение». Когда выполняющий поток добирается до выражения await, он нажимает кнопку «пауза» и выполнение метода приостанавливается. А когда задача, на которой вы ждали, завершается, он нажимает кнопку «воспроизведение» и выполнение метода возобновляется [2]. Управление памятью осуществляется строго через IDisposable и using-декларации, что предотвращает утечки даже при обработке крупных документов.

Алгоритм преобразования текста в векторные кривые. Одной из ключевых сложностей при преобразовании текстового содержимого PDF-документов в формат SVG является обеспечение визуальной идентичности результата независимо от среды отображения. Эта задача особенно обостряется в случаях, когда исходный PDF использует специализированные или нестандартные шрифты, которые могут отсутствовать на целевом устройстве. Для решения данной проблемы был разработан комбинированный подход, основанный на анализе структуры PDF и адаптивной стратегии генерации векторного вывода. PDF-файл иногда тоже является векторным файлом. Это зависит от того, как был создан PDF-файл и были ли его слои выровнены при его создании [3].

Центральной идеей предложенного метода является разделение обработки на два взаимодополняющих режима, в зависимости от способа представления текста в исходном документе.

Первый режим применяется в случае, если текст в PDF уже представлен в виде векторных графических примитивов – то есть был «запечён» в документ как набор контуров (например, при экспорте из графических редакторов вроде Adobe Illustrator или при применении операции «обводки текста»). В этом сценарии PDF-парсер извлекает непосредственно последовательности графических операций: перемещения, линии, кривые Безье и команды замыкания контуров. Эти данные транслируются в SVG-совместимый формат без каких-либо интерпретаций, что гарантирует абсолютную точность воспроизведения оригинального изображения. Такой подход не зависит от наличия шрифтов и обеспечивает стопроцентную идентичность, поскольку обрабатывается не семантический текст, а его визуальная форма.

Второй режим активируется, когда PDF содержит текст в виде семантических текстовых операторов, сопровождаемых метаданными о шрифте, размере, позиции и цвете. В этом случае система извлекает строковое содержимое и информацию о шрифте, включая его имя. Далее выполняется попытка загрузить соответствующий шрифт из операционной системы. Для повышения вероятности совпадения проводится нормализация имени шрифта – удаление технических префиксов (например, AAAAAA+), характерных для подмножеств (subsets), встроенных в PDF. Если шрифт обнаружен, он используется для генерации векторных контуров глифов. В противном случае применяется стратегия замены на визуально близкий системный шрифт (например, Arial для сан-серифных или Times New Roman для серифных начертаний).

Независимо от выбранного режима, финальный результат всегда представляется в SVG-формате в виде элементов <path>, а не текстовых блоков <text>. Это принципиальное решение обеспечивает полную независимость от шрифтовых ресурсов на стороне потребителя и гарантирует, что документ будет отображаться одинаково на любом устройстве и в любом браузере. Хотя такой подход увеличивает объём выходного файла, он оправдан в сценариях, где критически важна точность визуального воспроизведения.

Для реализации данного подхода предлагается open-source библиотека PdfPig, предоставляющая детальный доступ как к текстовым, так и к графическим слоям PDF-документа. Её использование позволит избежать лицензионных ограничений коммерческих решений и обеспечить прозрачность обработки. Генерация векторных контуров с данным инструментарием может осуществляться с помощью кроссплатформенной графической библиотеки SkiaSharp, которая поддерживает высокоточное построение глифов на основе системных шрифтов. Skia – это библиотека 2D-графики с открытым исходным кодом, предоставляющая общие API, работающие на различных аппаратных и программных платформах [4].

Заключение. Разработанная архитектура приложения-конвертера решает фундаментальную проблему межформатной трансформации документов – сохранение визуальной идентичности текста при переходе от статического PDF к динамическому SVG. Предложенный комбинированный подход, учиты-

вающий как векторное, так и семантическое представление текста в исходном документе, обеспечивает максимальную точность результата в широком спектре сценариев. Использование open-source библиотек PdfPig и SkiaSharp позволяет избежать лицензионных ограничений, сохраняя при этом высокое качество обработки.

Архитектура на основе WPF и паттерна MVVM гарантирует чёткое разделение ответственности между пользовательским интерфейсом и бизнес-логикой, что повышает тестируемость, сопровождаемость и потенциал для дальнейшего расширения функционала. Асинхронная обработка и строгое управление ресурсами обеспечивают стабильную работу даже с объёмными документами, а представление текста исключительно в виде векторных путей (<path>) делает итоговый SVG полностью независимым от шрифтовых ресурсов целевой системы.

ЛИТЕРАТУРА

1. Windows Presentation Foundation для .NET [Электронный ресурс] – Режим доступа: https://learn.microsoft.com/ru-ru/dotnet/desktop/wpf/overview/. – Дата доступа: 24.09.2025.
2. Асинхронное программирование - Await – пауза и воспроизведение [Электронный ресурс] – Режим доступа:https://learn.microsoft.com/ru-ru/archive/msdn-magazine/2011/october/asynchronous-programming-pause-and-play-with-await. – Дата доступа: 24.09.2025;
3. Как конвертировать PDF в векторное изображение [Электронный ресурс] – Режим доступа: https://pdf.wondershare.com.ru/pdf-knowledge/convert-pdf-to-vector.html. Дата доступа: 27.09.2025 – 21.09.2025;
4. Documentation Skia [Электронный ресурс] – Режим доступа: https://skia.org/docs/. Дата доступа: 28.09.2025.