**Санкт-Петербургский Государственный Университет**

**Математико-механический факультет**

Кафедра системного программирования

**Генерация объектной модели для DocsVision**

**и использование ее при синхронизации сервисов**

Курсовая работа студента 445 группы

Астащенко Александра Евгеньевича

Научный руководитель ………………. В.Г Шистеров

Санкт-Петербург

2010

**Оглавление**

Введение 3

Обзор существующих решений 5

DocsVision 6

Предлагаемое решение 10

Результаты 17

Направление дальнейшей работы 18

Список литературы 19

# Введение

При разработке информационных систем всегда приходится работать с некоторой моделью данных. Информация чаще всего хранится в базах данных. Но разрабатывать приложения, общаясь напрямую с базой данных не эффективно. Хочется иметь API по работе с этими данными. Microsoft уже предлагает Entity Framework для работы данными, хранящимися в реляционных базах данных. Entity Framework предлагает удобный дизайнер, огромное количество вариантов маппинга, автогенерацию классов-моделей, но на все это есть жирный минус – гигантские и раздутые сгенерированные классы, которые к тому же нельзя изменять вручную – ибо при каждом изменении модели в дизайнере, все будет пересоздано заново.

## Постановка задачи

Создать для существующей платформы DocsVision автогенератор классов-моделей. Требования к генератору:

* Легкое управление получающимся кодом классов-моделей
* Поддерживать классы-модели в актуальном состоянии

# Существующие потходы к генерации кода

## Text Template Transformation Toolkit и Custom tools

Для генерации исходного использовать T4 (Text Template Transformation Toolkit. Решение от Microsoft). Имея схемы карточек (метаинформацию об объектах, описание которых находится в этой карточке) можно получить исходный код для этих классов. Однако объекты могут ссылаться и на объекты типов, принадлежащих другим карточкам, что не позволяет нам увидеть картину в целом. Такую же проблему получаем при написании собственного Custom Tools, т.к. он применятся к одной конкретной схеме карточек.

## Сторонний генератор

На входные данные получить сразу несколько схем карточек. У нас будет метаинформация о полученной схеме целиком. Однако полученные исходники придется отдельно подключать к проекту, что влечет за собой отдельные неудобства при обновленных версиях схем карточек.

## MetaCreator

MetaCreator [1] включает в себе плюсы всех описанных выше подходов. Синтаксис MetaCreator’а подобен синтаксису T4. Однако генерация исходников вызывается не перед компиляцией сборки, а во время ее. Таким образом, можно описать парсер для метаданных и необходимые генераторы возможно заранее и единожды. Во время самой компиляции сборки вызвать парсер над всеми необходимыми схемами карточек и сгенерировать для них исходный код.





# Работа с DV

## Пример кода

Далее последует пример работы с DocsVision через стандартные платформенные средства[2].

Создаем сессию для подключения к серверу DocsVision

var sessionManager = SessionManager.CreateInstance("Connectionstring=http://localhost/docsvision/storageserver/storageserverservice.asmx");

var session = sessionManager.CreateSession();

После создания сессии можем, используя CardManager, получить карточки и информацию содержащуюся в них.

const string REFSTAFF\_CARDTYPE = "6710B92A-E148-4363-8A6F-1AA0EB18936C";

const string REFSTAFF\_UNITS = "6710B92A-E148-4363-8A6F-1AA0EB18936C";

const string REFSTAFF\_EMPLOYEES = "DBC8AE9D-C1D2-4D5E-978B-339D22B32482";

var cardData = session.CardManager.GetDictionaryData(staffId);

var rowDataUnit = cardData.Sections[unitSectionId].CreateRow();

rowDataUnit["Name"] = "NewOrganization";

var rowDataEmployee = rowDataUnit.ChildSections[employeeSectionId]

 .Rows.AddNew();

rowDataEmployee["LastName"] = "Ivanov";

Минусы этого кода: большое количество констант, Создание всех объектов через платформенное API трудозатратно и требует дополнительные знания о метаданных. Надо знать секцию (ее id), а так же ее расположение в общей схеме.

Вся эта информация хранится в CardDef’ах (далее схемах карточек). Схема карточек представляет собой xml, содержащий описание карточки, древовидную структуру секций и список полей для каждой секции (пример приложен к отчету).

## Объектная модель

Необходимо описать в библиотеки все классы, которыми придется оперировать в дальнейшем.

Представим код, которым в дальнейшем будет удобно пользоваться разработчику для управленя данных в системе.

var unit = new Units

{

 Name = “NewOrganization”,

};

var employee = new Employees

{

 LastName = “Ivanov”,

};

unit.Employees.Add(employee);

Context.Save(unit);

Context.Save(employee);

Т.е. у нас уже будут классы с полным набором типизированных полей, которыми и придется оперировать.

## ОТображение схем карточек на объектную модель

* Карточка -> Класс, отвечающий за экземпляр этой карточки
* Секция -> Класс с полным типизированным набором полей

У карточки нет своих полей. В ней есть статическая информация, не описанная в схемах карточек, но присущая всем карточкам в системе:

* Время создания
* Время изменения
* Название

По структуре схемы карточек так же к полям карточки можно отнести секции типа **struct**,т.к. присутствуют в карточке максиму в одном экземпляре.

Так же карточка будет владеть коллекциями всех секций других типов (**tree, table**).

У секций есть свой набор полей, который должен отобразиться в соответствующие поля в объектной модели. Так же, как и для карточки, полями сделаем подсекции типа **struct** и коллекции подсекций остальных подтипов.

## Разбор схемы карточек

Для разбора схемы карточек был взят стандартный XmlSerializator.

Для генерации исходного кода был выделен интерфейс, с помощью которого и реализуются различные части системы.

internal void ModelOpen(string cardName, Guid cardCardId);

internal void ClassOpen(ClassInfo info);

internal void GenerateEnum(string enumName, Guid fieldEnumId, params object[] enumItems);

internal void GenerateList(FieldInfo info);

internal void GenerateField(FieldInfo info);

internal void ClassClose();

internal void ModelClose();

## Типизация ССЫЛок

У всех полей секций имеется тип(все типы перечислены в [3]). Интерес заключается в полях типа refid и refcardid. Для них в схеме указаны идентификатор карточки и идентификатор секции в этой карточке, на которую ссылается данное поле. В момент разбора схемы карточки мы находим класс, соответствующий указанной секции\карточки.

## Актуальность объектной модели

Все схемы карточек находятся в DocsVision. При изменении версии схемы карточки, мы достаем новую схему из базы, и обновляем модельные объекты, связанные с этой карточкой.

При работе в стороннем проекте, использующем нашу объектную модель, мы можем оценить изменения в структуре и своевременно отреагировать на это в коде. Данное требование актуально именно для процессов разработки для платформы, т.к. после ее внедрения схема карточек остается неизменными, либо требует дополнительной поддержки от служб сопровождения системы в целом.

# Результаты

В ходе курсовой работы было предоставлено решение для генерации объектной модели по схемам карточек DocsVision. Генерируемая с их помощью библиотека использовалась для создания сервиса синхронизаций. Полученный сервис внедрен в эксплуатацию.

# Литература:

[1] – MetaCreator, <http://code.google.com/p/metacreator/>

[2] – Руководство для разработки на платформе DocsVision