<Your project>

Выпуск

BARS Group

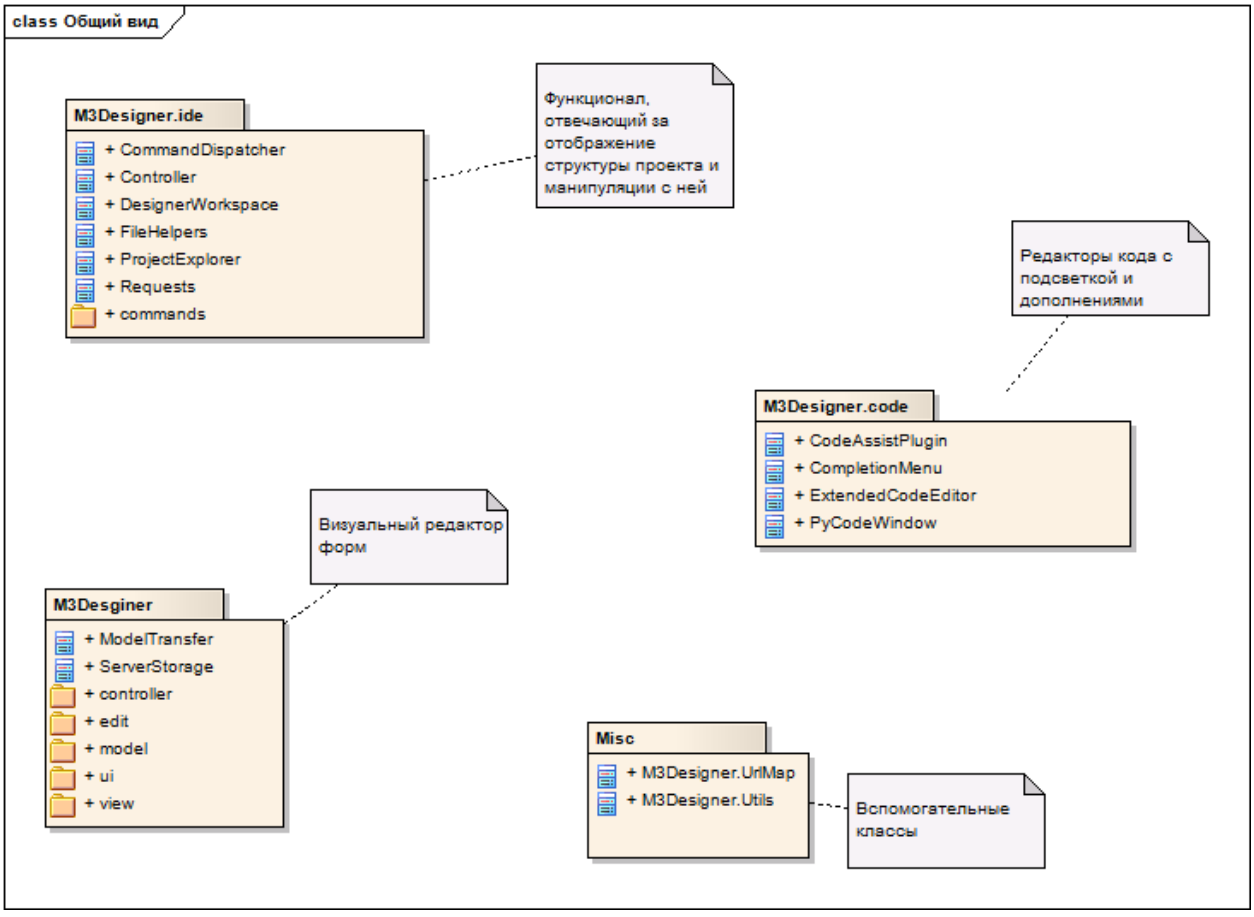
17 March 2014

Оглавление

Contents:

Архитектура клиентского приложения

1.1 Общий обзор



Примечание: Оригинальный файл с диаграммами классов(проект Enterprise Architect) лежит в репозитарии /m3_designer/doc/m3-ide.eap

1.2 M3Designer.UrlMap

Контейнер для серверных адресов, необходим для работы(если какой-то из адресов не задан - будут выкидываться ошибки). Поэтому нужно сконфигурировать адреса таким образом(в отдельном js файле или в самой html страниче - главное до запросов на сервер):

```
M3Designer.UrlMap.addUrls({
                'save-file-content':'/designer/file-content/save',
                'get-file-content':'/designer/file-content',
                'get-template-global':'/designer/project-global-template',
                'manipulation':'/designer/project-manipulation',
                'data':'/designer/data',
                'save':'/designer/save',
                'preview':'/designer/preview',
                'upload-code':'/designer/upload-code',
                'code-assist':'/designer/codeassist',
                'create-autogen-function' : 'create-autogen-function',
                'create-function': '/create-function',
                'create-new-class': '/create-new-class',
                'project-files': '/project-files'
});
```

1.3 M3Desginer.ide

Код в этом пространстве имен рисует дерево структуры проекта, панель с вкладками, обрабатывает данные с сервера, и производит все манипуляции с проектом. Основные классы:

ProjectExplorer - наследник экстового дерева. Отображает структуру проекта.

Controller - контроллер приложения. Фактически, точка входа. В качестве параметров конструктора принимает интерфейсные объекты (дерево проекта, табпанель), на события которых начинает реагировать.

ComnmandDispatcher - диспетчер комманд. Вся логика работы над проектом описывается с помощью классов комманд. Более подробно можно посмотреть в комментариях в файлах /ide/controller.js и /ide/commands.js. Именно там нужно смотреть, если требуется добавить в IDE часть дизайнера новый функционал. Обратное взаимодействие с интерфейсом осуществляется через обращения к экземпляру контроллера.

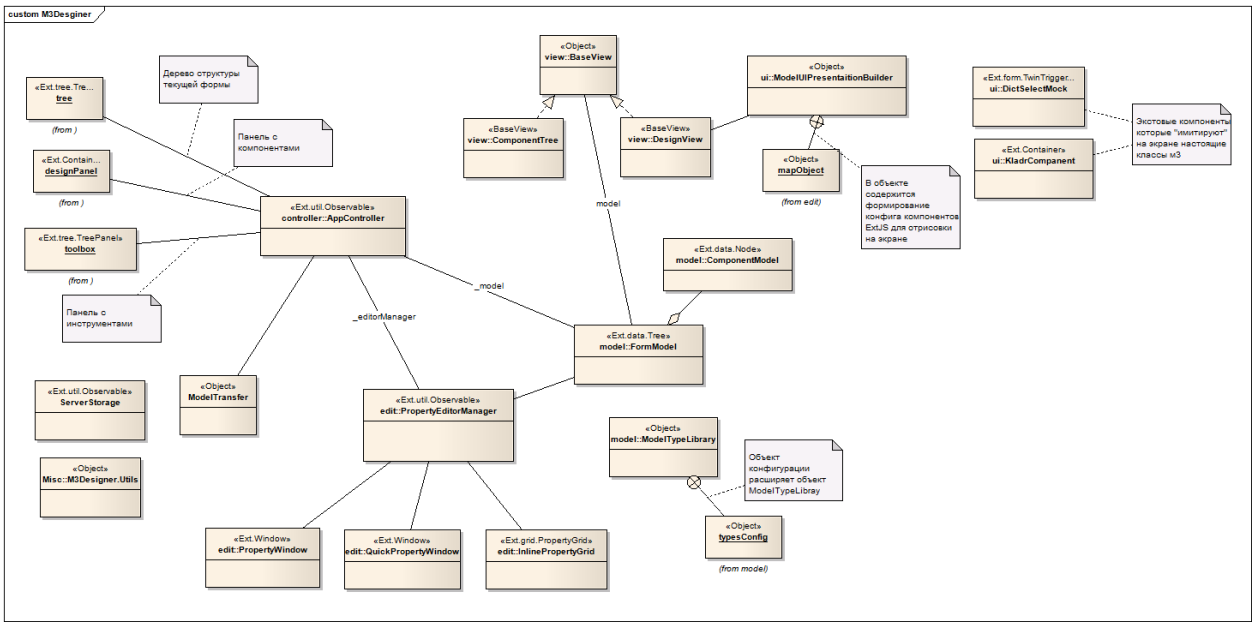
CommandsConfig - настройка соответствия типов вершин в структуре проекта и возможных действий над ними.

Requests - запросы к серверу плюс часть логики работы. Это устаревший код, и должен быть исправлен(логика должна быть перенесена в комманды)

Примечание: Логика работы со структурой проекта основывается на типах вершин. Когда с сервера передаются данные, у каждой вершины указывается атрибут type, например “type”:”file”

1.4 UI Designer

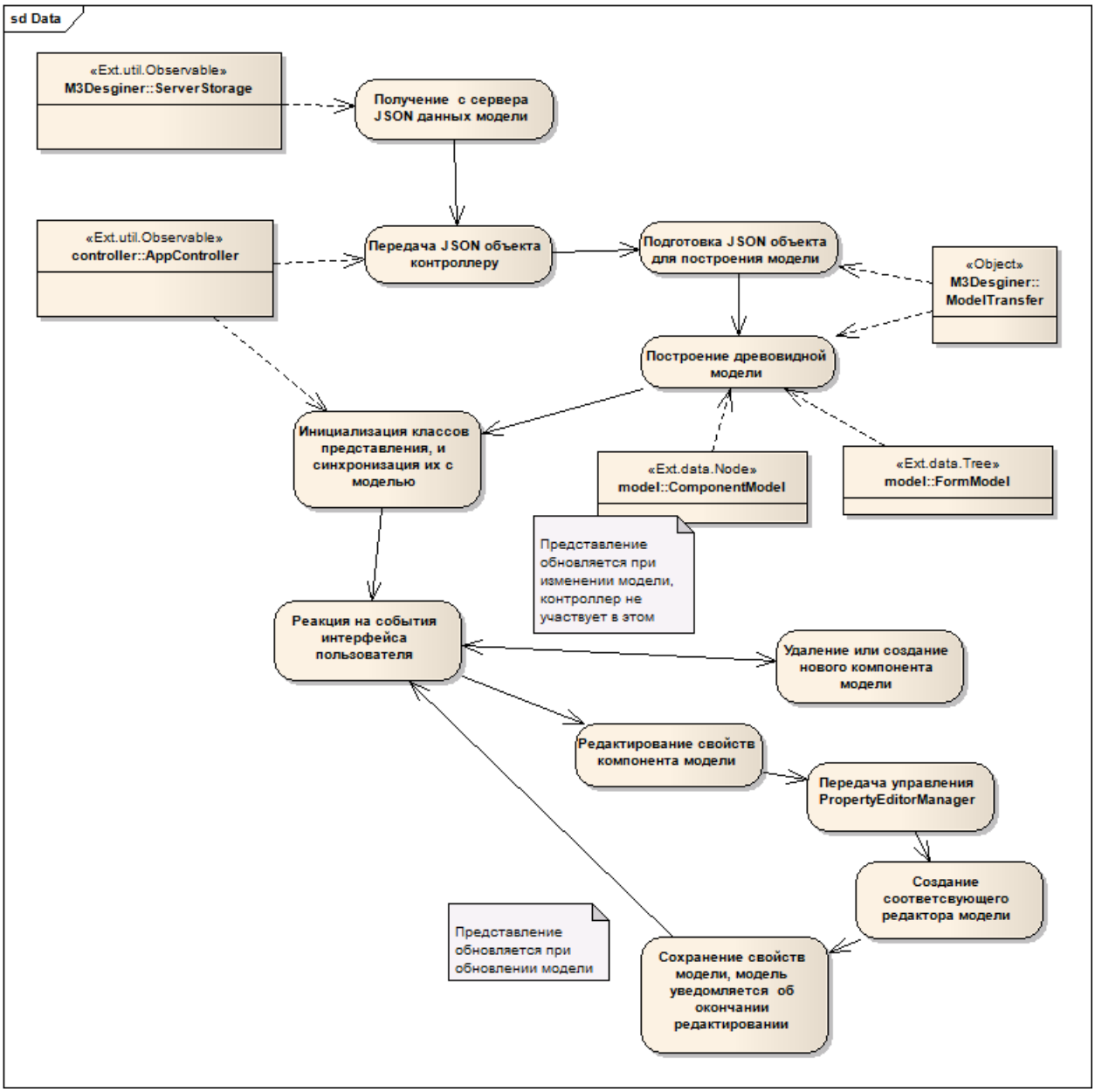
Общая диаграмма классов



Общая концепция строения это MVC. Модель представляет собой дерево с вершинами. Контроллер - объект реагирующий на пользовательские события и обновляющий модель. Представления - два класса, что синхронизированы с моделью. Под синхронизацией понимается обработка событий обновления модели(добавление, удаление, измнение узлов дерева)

Важно: Контроллер не обновляет экранное представление. Экранное представление обновляется само по себе при изменениях модели.

Структура поведения подсистемы:



Описание основных классов, подробнее - смотреть комментарии в коде:

AppController - контроллер обрабатывающий действия пользователя

FormModel и ComponentModel - древовидная модель и ее узлы

ModelTypeLibrary - содержит “бизнесс” логику по работе с моделями

BaseView - базовый класс представления, синхронизирующийся с моделью

ComponentView - структура формы - отображает модель в TreePanel на экране

DesignView - визуальный просмотр модели на экране. Принцип перерисовки заключается в очистке содержимого экстовой панели и пересоздание JavaScript компонентов

ModelUIPresentationBuilder - отвечает за формирование конфигов ExtJS из объектов модели, для работы DesignView

PropertyEditorManager - управляет редактированием моделей. Фактически это тоже контроллер, что создает окна и гриды для редактирования, и потом обновляет модель

QuickPropertyWindow - окошко быстрого редактирования

PropertyWindow - окошко обычного редактирования

InlinePropertyGrid - грид редактирования модели, встраиваемый в аккардеон панель

ModelTransfer - класс для сериализации и десериализации модели в транспортный JSON

ServerStorage - взаимодействие с сервером

Примечание: Более подробно о свойствах модели можно прочитать в соответсвующем разделе справки *Добавление и измененение компонентов в дизайнере*

1.5 Code editor

Наиболее очевидный блок системы

ExtendedCodeEditor - Экстовая панель с CodeMirror редактором

CodeAssistPlugin - Плагин в терминах ExtJS. Присоединяется к панели ExtendedCodeEditor, слушает нажатие клавиш с клавиатуры, передает на сервер текущий файл в текущем состоянии(целиком) и положение курсора. Если сервер возвращает предложения о дополнении кода, создает меню с дополнениями

CompletionMenu - меню дополнений

PyCodeWindow - Legacy код. Окошко с эдитором питоновского кода, используется для предпросмотра кода генерируемого UI-Designer'ом.

Глава 2

Добавление и измененение компонентов в дизайнере

2.1 Описание доступных типов и свойств компонентов

Все компоненты, используемые в дизайнере описаны в файле model-types-config.js. Внутри него находится объект typesConfig, полями которого являются доступные типы. Рассмотрим на примере ExtFormPanel:

```
formPanel: {
        parent:'panel',
        isContainer:true,
        properties : {
            id : {
                defaultValue:'frm_formpanel',
                isInitProperty:true,
                isQuickEditable: true
            },
            layout: {
                defaultValue:'form',
                isInitProperty:true,
                isQuickEditable: true
            },
            title: {
                defaultValue:'',
                isInitProperty:true,
                isQuickEditable: true
            },
            url: {
                defaultValue:''
            },
            fileUpload:{
                defaultValue:false
            },
            urlShortName:{
                defaultValue:'',
                isQuickEditable:true
            }
        },
        childTypesRestrictions:{
            disallowed:['arrayStore','gridColumn','jsonStore','pagingToolbar']
        },
        toolboxData: {
            category:'Containers',
```

```
            text:'Form panel'
        },
        treeIconCls:'designer-formpanel'
    }
```

parent:'panel' - Ссылка на тип, который является предком текущего типа. Типы компонентов наследуют свойства друг друга(под свойствами в данном контексте подразумеваются объекты в properties, ограничения и другие объекты первого уровня вложенности не наследуются) Раз formPanel является наследником Panel она получит свойства layout, region, height, width и тд

isContainer:true - В случае если поле указано и равно true, то в компонент можно добавлять дочерние компоненты. Нужно отметить что под добавление дочерних в данном случае подразумевается чисто абстрактное условие, регламентирующее возможность иметь дочерние компоненты во внутренней структуре данных используемых дизайнером, и никак не связана напрямую с объектной моделью ExtJs. Так ComboBox имеет isContainer:true тк в него можно добавить подчиненный DataStore.

properties: {} - Вложенные свойства это те поля что доступны для редактирования пользователю.

defaultValue:'form' - обязательное поле для каждого свойства типа. По типу(javascript типу) определеяется редактор. К примеру height и width у контенейров обязан быть числом, и поэтому defaultValue:0 и редактироваться будет NumberField'ом.

isInitProperty:true - Необязательное поле. Если установлено то при создании нового экземпляра поле будет проинициализировано. Типичный пример тайтлы у всего что можно

isQuickEditable:true - Необязательное поле. True - свойство появиться в редакторе быстрых свойств

propertyType:'enum' - Необязательное поле. Если не указано, то Javascript тип свойства будет определен по значению defaultValue. В данный момент обрабатываются значения 'enum' и 'object' Значения перечисления обрабатываются в файле property-editor.js Для значения 'object' - вводимый пользователем текст будет про'eval'ен, это нужно для вложенных объектов. Значения перечислений прописываются в model-types.js

isNotEditable:true - Необязательное поле. Используется когда нужно проинизализировать что-то в экземпляре объекта, но не давать пользователю рукам это что-то перебить. Пример TabPanel, дабы не унаследовать от Panel значение layout по умолчанию 'auto'

childTypesRestrictions - Ограничения для подчиненных объектов. Состоит из трех массивов со строковыми названиями типов. allowed - то что можно добавлять, disallowed - то что нельзя, single - одиночные типы Каждый из массивов может отсутствовать, в таком случае условие игнорируется при проверке. К примеру disallowed:undefined, allowed:['panel'], single:['panel'] - к текущему компоненту можно добавить ровно одну панель

toolboxData - Информация для панели инструментов. Может отсутствовать

treeIconCls - Очевидно иконка типа компонента. Необязательное поле

2.2 Визуальное отображение компонентов

Код рисующий штуки на экране уютно расположился в файле ui.js Для рисования использовался хитрый паттерн "Посетитель" Для рисования нужно добавить новую функцию объекту mapObject, имя которой совпадало бы с названием типа определенного в файле конфигов. Если функция не определена - ничего не случится. Просто компонент не нарисуется:

```
comboBox:function(model, cfg) {
    var store = undefined;
    //попробуем найти стор
    for (var i = 0; i < model.childNodes.length; i++) {
```

```
        if (model.childNodes[i].attributes.type == 'arrayStore') {
            store = new Ext.data.ArrayStore(
                        Ext.apply({
                            fields:['id',model.attributes.properties.displayField]
                        },model.childNodes[i].attributes.properties)
                    );
        }
    }
    //или создадим пустой
    if (!store) {
        store = new Ext.data.Store({
            autoDestroy:true
        });
    }
    return Ext.apply( cfg , {
                store:store,
                mode:'local',
                xtype:'combo'
            });
}
```

В аргументе model передается объект внутренней модели дизайнера, через него можно получить доступ к родительскому компоненту или к дочерним. cfg - готовый объект конфига со свойствами отредактированными пользователем. Для простых случаев, например, textField достаточно в объект конфига добавить xtype для того чтобы экст корректно создал визуальный компонент. Можно не использовать xtype, и создавать инстансы классов. В примере выше рассамтривается создание комбобокса. Чаще всего для дизайна нам не требуется сложное поведение компонентов, и поэтому можно создать ограниченую болванку, которой достаточно чтобы послужить отражением более сложного компонента(или закрыть от активации какое то поведение, как правило обмен данными с севрером)

2.3 Сериализация/десериализация

Код находиться в файле transfer.js В подавляющих случаях туда не нужно ничего добавлять. Но иногда, когда у компонентов есть компоненты не принадлежащие items, приходится добавлять всячески исключения. Пример тому свойство store, или массив columns у грида. Смотреть нужно на объект childPropertyObjects. В массивы добавляются свойства для особенной десериализации, в функции типы требующие сериалзиации как-то поособенному.

2.4 Серверный маппинг

Для того, чтобы код успешно серриализовался в python код и обратно нужно добавлять компоненты в серверный маппинг.

Серверный маппинг находится в файле m3_designer/ide/mapping.json. Рассматорим пример маппинга:

```
{
    "class": { "objectGrid": "ExtObjectGrid"
    }, "config": {
        "urlNew":"url_new" ,"urlEdit":"url_edit" ,"urlDelete":"url_delete" ,"urlData":"url_data"
        ,"rowIdName":"row_id_name" ,"columnParamName":"column_param_name"
        ,"allowPaging":"allow_paging" ,"localEdit": "local_edit"
```

,"urlDataShortName": { "type":"shortname" ,"value":"url_data"
} ,"urlEditShortName": {
"type":"shortname" ,"value":"url_edit"
} ,"urlDeleteShortName":{
"type":"shortname" ,"value":"url_delete"
} ,"urlNewShortName":{
"type":"shortname" ,"value":"url_new"
}
} ,"parent": "gridPanel"
}

Объект class говорит о том, что клиентский класс *objectGrid* будет маппиться в серверный класс *ExtObjectGrid*. Конфиг объектов данного класса так же маппится из клиенского кода *urlNew* в серверный код *url_ new* и наоборот. Могут быть сложные свойтсва, такие свойства помечаются свойством "type", например объект "shortname", имеющий свойство "create" будет отображен в серверный поиска шортнейма - *instanse.attr = urls.get_ url('create')*

Глава 3

TO DO

- Реализовать прозрачную конфигурацию дизайнера и его подсистем. Сейчас существует несколько классов с конфигурацией это не понятно и не интуитивно. В идеале должен быть какой-то один файл “config.js”, где указываются необходимые параметры(конфигурация ui дизайнера, адреса для запросов на сервер, команды над структурой проекта etc), и класс для запуска всего этого.
- Перенести код в отдельный проект. Дизайнер это самостоятельное JavaScript приложение(вернее даже два приложения), и должен жить отдельно. По правильному, код должен быть перенесен, и проекты m3_designer и m3_sandbox должны подключать только собранный, минифицированный скрипт и файлы с конфигурацией(см предыдущий пункт)
- Файл requests.js это плохой негодный legacy код. Из него должна быть вынесена логика с классы команд.
- Создание template global’ов сейчас происходит неправильно. Нужно добавить некий класс медиатор между подсистемой IDE и подсистемой UIDesigner’а, чтобы логика по созданию файла на сервера исполнялась в классе команды(соответсвенно это класс следует написать)
- Код в подсистеме IDE далек от идеала. Желателен рефакторинг, исправление варнингов, и проверка всех файлов JSLint’ом. И еще неплохо было написать побольше коментариев.
- Файл exntesions.js требует разнесение по корректным подсистемам

Indices and tables

- *genindex*
- *modindex*
- *search*