ВЕСТНИК ТОМСКОГО ГОСУДАРСТВЕННОГО УНИВЕРСИТЕТА

2014 Управление, вычислительная техника и информатика № 3 (28)

УДК 004.4'22

А.Н. Моисеев, А.М. Политов, В.В. Рахимов, М.О. Хомич

БАЗОВАЯ АРХИТЕКТУРА СИСТЕМЫ СОВМЕСТНОГО РЕДАКТИРОВАНИЯ ДИАГРАММ С СИНХРОНИЗАЦИЕЙ В РЕЖИМЕ РЕАЛЬНОГО ВРЕМЕНИ

Разрабатывается система совместного редактирования диаграмм с синхронизацией в режиме реального времени. Основной задачей является одновременная реализация свойств расширяемости и кроссплатформенности в условиях клиент-серверной архитектуры. Для достижения этих свойств выработана специальная модель описания представления и трехслойная архитектура клиент-серверного взаимодействия на ее основе. Система доработана до состояния архитектурного прототипа и реализует весь критический функционал на базовом уровне.

Ключевые слова: объектно-ориентированная разработка; клиент-серверная архитектура; синхронизация в режиме реального времени.

Диаграммы и различного рода схемы применяются в самых различных областях: при проектировании и разработке ПО, анализе требований, в радиофизике, математике. Кроме того, нередко над одной и той же диаграммой работают несколько людей одновременно (например, при проектировании ПО), а синхронизация с помощью пересылки файлов или с использованием системы контроля версий зачастую сводит преимущества от совместной работы к нулю. Подобное программное обеспечение для совместного редактирования с синхронизацией в режиме реального времени получает все большее распространение в последнее время, в связи с чем растут и требования к подобным системам. Для систем совместного редактирования диаграмм важным свойством является расширяемость – возможность добавлять функционал в систему без изменения ее исходного кода. Вторым немаловажным свойством является кроссплатформенность, т.е. возможность написания клиентского приложения (клиента) под любую платформу. Реализация системы, удовлетворяющей одновременно свойствам расширяемости и кроссплатформенности, связана с определенными техническими трудностями. В [1] описана концепция системы совместного редактирования, удовлетворяющая данным требованиям. В настоящей работе представлено более детальное описание самой концепции.

В области разработки систем совместного редактирования имеется достаточно большое число различных решений. Рассмотрим самые популярные и крупные системы совместного редактирования в аспекте соответствия перечисленным выше свойствам:

1. Документы Google [2] – это веб-ориентированный онлайн-сервис, включающий в себя текстовый и табличный процессор, инструмент для создания презентаций, а также облачное хранилище. Система поддерживает многопользовательское совместное редактирование, однако основным недостатком является очень скудная поддержка редактирования диаграмм без возможности расширения.

2. Google Wave – веб-ориентированный онлайн-сервис для взаимодействия и совместной работы. Система поддерживала совместную работу с форматированным текстом, фотографиями, видео, простыми графическими примитивами. С 30 апреля 2012 г. сервис прекратил свою работу [3].

3. Различные веб-ориентированные сервисы редактирования диаграмм, например LucidCharts [4] или Cacoo [5]. Системы предоставляют возможность многопользовательского совместного редактирования, централизованное хранилище, достаточно активно поддерживаются и развиваются (создаются новые типы диаграмм и элементов, совершенствуется пользовательский интерфейс). Основным недостатком является ориентированность на конкретную клиентскую платформу (веб) и отсутствие поддержки формальной модели редактируемых диаграмм, что можно было бы использовать для поддержки целостной модели проекта).

4. VisualParadigm [6] – мощный инструмент проектирования, обладающий следующими преимуществами: реализация клиентского приложения на языке Java, что позволяет работать на большом числе платформ, поддержка формальной модели редактируемых диаграмм, генерации кода, а также обратного проектирования, расширяемая модель. Основные недостатки: отсутствие совместного редактирования диаграмм с синхронизацией в режиме реального времени, отсутствие поддержки JVM на некоторых современных платформах (например, iOS).

Разрабатываемая авторами система Obordesus спроектирована таким образом, чтобы удовлетворять заявленным выше требованиям расширяемости и кроссплатформенности. В настоящей работе представлены архитектурно значимые решения, принятые при создании системы.

1. Свойства системы

Obordesus [1] – централизованная система совместного редактирования диаграмм с синхронизацией в режиме реального времени. Ее ключевые свойства:

1. Многопользовательская работа с диаграммами – возможность одновременного доступа нескольких участников проекта к одной диаграмме.

2. Синхронизация диаграмм в режиме реального времени – изменения, вносимые одним участником, сразу же отображаются у других участников, редактирующих тот же документ.

3. Расширяемость – возможность добавления функционала к системе без переработки самой системы.

4. Кроссплатформенность – возможность реализации клиентского приложения под любую платформу.

5. Устойчивость клиентского приложения к изменению функционала системы – расширение функционала не требует внесения изменений в клиентские приложения.

6. Централизованность – доступ к документам, созданным с помощью системы, осуществляется через единую глобальную точку, обеспечивая таким образом их повсеместную оперативную доступность.

2. Модель предметной области

Основным элементом предметной области (рис. 1) является диаграмма (объект Document). Диаграммы представляют собой набор визуальных элементов (объект ViewItem), которые являются образами некоторых моделируемых сущностей (объект ModelItem). Моделируемую сущность также будем называть моделью элемента, а его визуальный образ – представлением элемента. Стоит отметить, что несколько визуальных элементов могут представлять одну и ту же модель.

Элементы диаграммы могут быть связаны между собой (например, классы в языке UML [7] могут быть связаны между собой ассоциацией). Связи между элементами рассматриваются как отдельные элементы, которые также имеют свою модель и представление. Модель в данном случае определяет, какие элементы связаны, а представление – как эта связь отображается.

Диаграммы объединяются в проекты (объект Project). Совокупность всех моделей элементов проекта является моделью проекта, а совокупность диаграмм – представлением проекта.

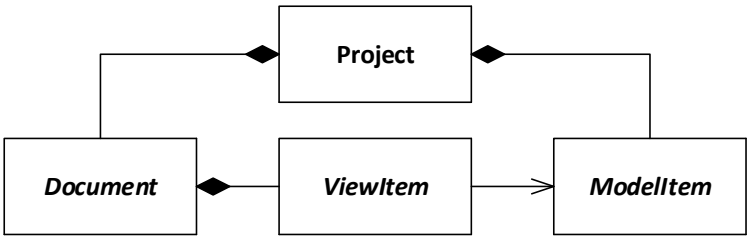

Рис. 1. Модель предметной области

3. Универсальное представление элементов

Одной из задач клиент-серверного взаимодействия [8] в разрабатываемой системе совместного редактирования диаграмм является обеспечение возможности расширения функционала без необходимости модификации клиентских приложений. С учетом требования кроссплатформенности, реализовывать модуль расширения с новым функционалом для каждого клиента слишком затратно. Кроме того, ограничением при выборе решения является также и тот факт, что взаимодействие происходит через сеть, а значит, присутствует некоторая задержка между совершением действия пользователем и реакцией сервера на это действие.

Таким образом, механизм клиент-серверного взаимодействия должен удовлетворять следующим требованиям:

1. Отображение произвольных элементов. Единственным ограничением является набор примитивов, из которых потом будет строиться представление элемента. Этот набор должен быть фиксирован при разработке и быть достаточно полным для представления любых графических элементов.

2. Синхронизация данных между клиентами и сервером. Интерфейс для обмена данными должен быть достаточно абстрактным, чтобы оставаться устойчивым к расширению функционала. Кроме того, особое внимание при проектировании механизма синхронизации необходимо уделять конфликтам синхронизации, вероятность которых очень высока при одновременном редактировании.

3. Взаимодействие пользователя с элементами и элементов между собой. Эта задача появляется из-за ограничения, накладываемого сетевым взаимодействием, так как постоянно присутствует некоторая задержка (до нескольких секунд), во время которой клиент работает самостоятельно, не получая отклика от сервера. В таком случае недостаточно просто уметь отображать элементы, необходимо дублировать взаимодействие между ними на стороне клиента.

Для удовлетворения этим требованиям предлагается следующее высокоуровневое решение (рис. 2): объектная модель, в которой работают клиентские приложения (Client Object Model), должна опираться не непосредственно на серверную объектную модель (Object Model), а на некоторую промежуточную модель описания представления (Presentation Description Model).

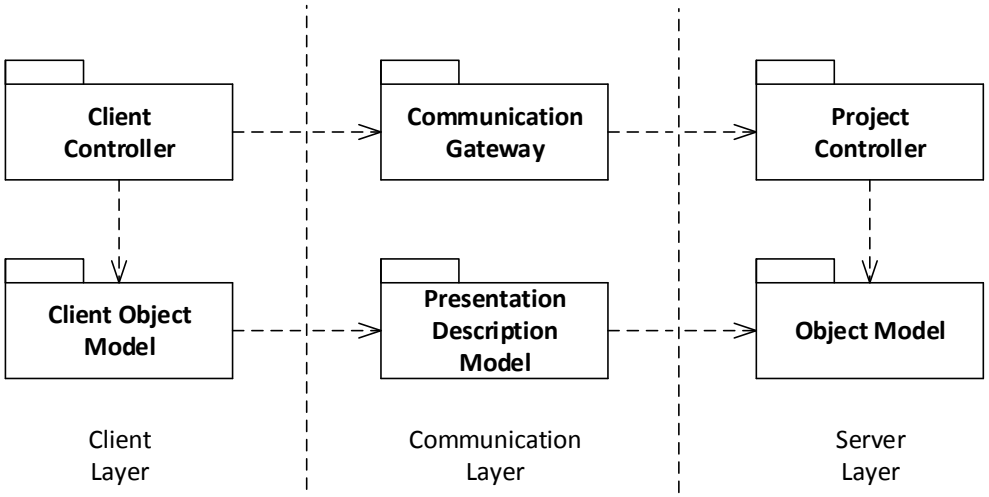

Рис. 2. Трехслойная архитектура системы

Обучение клиента новым типам элементов происходит следующим образом:

1. На сервере генерируется (либо берется готовое) описание представления нового типа элементов.

2. Описание представления доставляется на клиента.

3. Клиент по определенному заранее алгоритму строит свою объектную модель этого типа элементов.

Такое решение позволяет:

1) только один раз реализовать серверную модель;

2) только один раз специфицировать протокол взаимодействия и модель описания представления;

3) для каждой клиентской платформы реализовать модуль преобразования из модели описания представления в объектную модель конкретного клиентского приложения.

В рамках предложенного решения достигается выполнение всех поставленных требований:

1. Отображение произвольных элементов обеспечивается универсальностью модели описания представления.

2. Синхронизация между клиентом и севером (Client Controller и Project Controller через Communication Gateway) происходит по заранее определенному протоколу в терминах серверной объектной модели, на основе которой построены остальные модели, что позволяет реализовать универсальный механизм синхронизации данных безотносительно к особенностям конкретных элементов.

3. Взаимодействие пользователя с элементами и элементов между собой достигается с помощью особенностей модели описания представления (подробнее см. раздел 7).

5. Применение существующих технологий

В данном разделе приводится обзор существующих технологий, являющихся кандидатами для реализации предложенного решения. Во-первых, существует некоторое количество языков разметки визуальных графических элементов, например SVG [9], VML [10] и др. При использовании этих языков в чистом виде возникают следующие проблемы:

1. Невозможно задать связь представления, описанного с помощью разметки, с серверной моделью.

2. Отсутствие алгоритмической составляющей для организации взаимодействия пользователя с элементами и задания сложной логики представления элементов.

Среди языков разметки интерфейсов существуют решения, позволяющие решить первую проблему (XAML [11], механизм DataBinding [12]), однако в них тоже очень слабо развита алгоритмическая составляющая. Но несмотря на то, что ни один из языков нельзя использовать непосредственно, разумно выделить из них удачные решения и использовать их для задания представления элементов.

Другим вариантом является использование какого-либо скриптового языка (JavaScript, Python) либо языка, исполняемого на виртуальной машине (Java байт-код). Недостатки такого решения:

1. Существуют реализации интерпретаторов (Java-машины в случае с Java) не под все платформы, а сложность реализации интерпретатора любого из таких языков неоправданно высока.

2. Необходимость серьезно ограничивать возможности языка, так как все они являются языками программирования общего назначения, для системы же необходим узкоспециализированный язык. Трудоемкость реализации такого ограничения (особенно при использовании сторонних компонентов для интерпретации) может быть также неоправданно высока.

3. Неудобство использования языков программирования для описания представления. Этот недостаток можно было бы устранить, используя сочетание языка разметки и скриптового языка (пример: HTML+JavaScript), однако это не снимает остальных серьезных ограничений.

Таким образом, более простой в реализации является разработка специального языка описания представления, который будет включать в себя минимально необходимый набор возможностей. С одной стороны, это снимет необходимость реализации ограничения функциональности языка, а с другой – сделает более реальной задачу написания интерпретатора такого языка.

6. Базовые концепции строения элемента диаграммы

По результатам исследования предметной области и функциональных требований к механизму клиент-серверного взаимодействия приняты следующие базовые концепции описания элементов:

1. Информация об элементах содержится в их свойствах.

2. Представление элемента – это функция от свойств данного элемента и, возможно, других элементов.

3. Каждый элемент состоит из трех частей: модели данных, визуальной модели и представления.

4. Представление элементов создается с помощью набора примитивов.

Таким образом, имеется следующая структура элемента: элемент состоит из трех частей (рис. 3), две из которых (Model и VisualModel) являются, по сути, хранилищами значений свойств, инкапсулируя тем самым всю информацию об элементе. Таким образом, для синхронизации информации достаточно использовать только данные этих двух частей элемента. Третья составная часть элемента, Presentation, задается с помощью набора примитивов, значения параметров которых вычисляются на основе значений свойств Model и VisualModel.

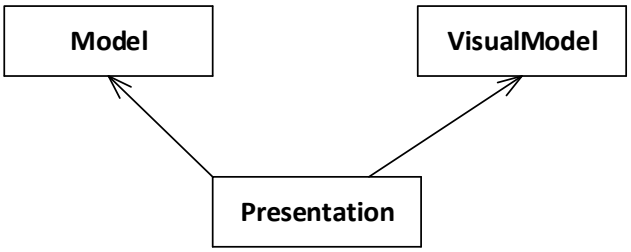

Рис. 3. Структура элемента диаграммы

Такое строение элемента идеально согласуется с моделью предметной области (для этого достаточно объединить VisualModel и Presentation в один блок – View). Кроме того, описанные выше концепции позволяют достичь решения всех трех задач, поставленных перед механизмом клиент-серверного взаимодействия:

1. Отображение произвольных элементов – концептуально нет никаких ограничений на задание набора свойств и набора примитивов, из которых будет строиться описание элемента, т.е. с помощью такого механизма можно построить представление любого элемента.

2. Синхронизация данных между клиентами и сервером – механизм синхронизации сводится к синхронизации свойств Model и VisualModel.

3. Воздействие пользователя на элементы и взаимодействие элементов между собой – фиксированный, заранее определенный набор примитивов позволяет описать указанные сложные взаимодействия в виде обработки параметров этих примитивов с помощью некоторого алгоритмического языка.

7. Модель описания представления

Рассмотрим объектную модель описания представления на высоком уровне (рис. 4).

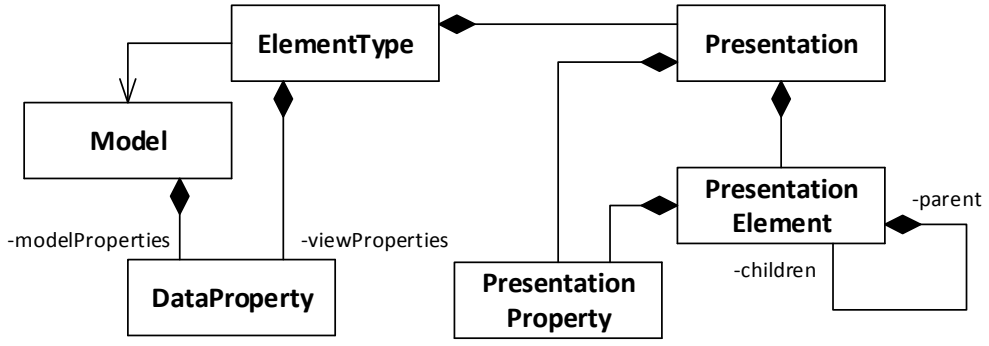

Рис. 4. Модель описания представления

Основным классом является ElementType, представляющий типы элементов, из которых в конечном итоге строится визуальное представление диаграммы. Для типа элемента определяется его название. Каждый элемент имеет два набора свойств (DataProperty), представляющих свойства модели данных и визуальной модели.

У каждого элемента есть его графическое представление (Presentation), которое имеет свой набор свойств (PresentationProperty) и строится как композиция атомарных графических элементов (линий, прямоугольников, текстовых меток и т.п.), представленных наследниками класса PresentationElement. Некоторые графические элементы могут содержать в себе другие графические элементы, причем из разных соображений: иногда исключительно для удобства (наследование родительских координат), а иногда по структурной необходимости (ячейки вложены в таблицу).

Каждый графический элемент в свою очередь состоит из набора свойств PresentationProperty. Представление элемента Presentation можно рассматривать как композитный графический элемент, что позволяет использовать PresentationProperty для задания свойств как самого представления Presentation, так и для задания свойств графических элементов.

Для каждого свойства представления (PresentationProperty) задаются 2 функции – get и set. Функция get не обладает побочными эффектами и должна в любой момент времени возвращать актуальное значение свойства. Функция set необходима для установки нового значения для свойства в результате действий пользователя и может быть опущена, если свойство не должно редактироваться через пользовательский интерфейс.

8. Серверная архитектура

В серверной части данной системы используется 5 логически разделённых типов серверов (рис. 5).

1. Root Server (корневой сервер) отвечает за управление серверами и их взаимодействие друг с другом.

2. Accounts Server (сервер пользователей) отвечает за централизованное управление учетными записями пользователей.

3. Repository Server (репозиторий) отвечает за централизованное управление проектами.

4. Login Server (сервер аутентификации) – пользовательский шлюз для аутентификации и управления проектами.

5. Project Server (сервер проектов) отвечает за работу с проектами.

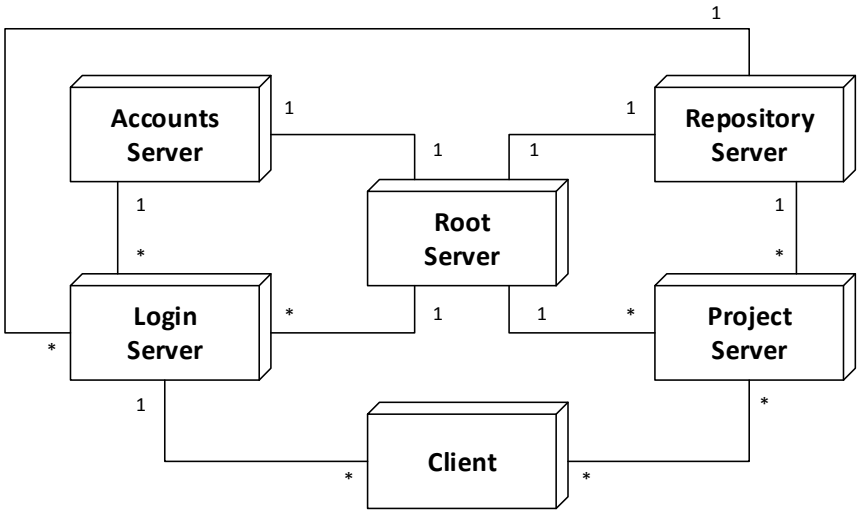

Рис. 5. Взаимодействие серверов и клиентов

Первые три типа серверов составляют управляющую основу серверной части системы и существуют в единственном экземпляре для каждого серверного экземпляра системы. Прямой доступ

пользователей к этим серверам не предусмотрен (за исключением администраторов системы). Последние два типа серверов составляют рабочую часть системы и будут горизонтально масштабироваться при необходимости (развертывание системы планируется на базе так называемых облачных провайдеров, например, Amazon AWS [14], Windows Azure [15]). Пользователи первоначально подключаются к серверу аутентификации, выбор которого происходит автоматически клиентским приложением, затем для работы над конкретными проектами они подключаются к проектному серверу, который выбирается репозиторием.

Сервер аутентификации позволяет пользователям:

1. Выполнять действия со своим аккаунтом (регистрация, аутентификация, смена учетных данных).

2. Управлять проектами (создание проектов, изменение информации о проектах, предоставление пользователям доступа к проекту, управление лицензиями и т.д.).

Проектный сервер – единственный тип серверов, который работает с серверной расширяемой объектной моделью предметной области. Расширение этой модели выполняется с помощью архитектурного решения «Plugin» [16] путем подключения модулей расширения (плагинов), в результате чего сервер должен удовлетворять следующим требованиям:

1. Возможность изоляции плагинов от проектного сервера и системы, на которой он запущен.

2. Возможность изоляции проектов друг от друга.

3. Возможность в рамках одного сервера иметь подключенными разные модули (в том числе одни и те же модули, но разных версий) для каждого запущенного проекта.

Для реализации этих требований используется изоляция на уровне доменов приложения .NET [17]. Для каждого проекта создается свой домен, в который загружаются только необходимые ему плагины с минимальными правами в системе. Вся обработка изменений выполняется в домене проекта, исключение составляет сетевое взаимодействие – им занимается сам сервер, передавая запросы домену с помощью технологии .NET Remoting [18].

10. Серверная объектная модель

Для серверной реализации расширяемой модели предметной области (рис. 6) необходимо реализовать следующие механизмы:

1. Механизм сохранения и загрузки объектов в данной модели.

2. Механизм транзакций для возможности выполнения сложных модификаций объектов данной модели в рамках транзакций с возможностью отката всех модификаций, если какие-то из них не могут быть применены.

3. Механизм контроля прав доступа к объектам модели.

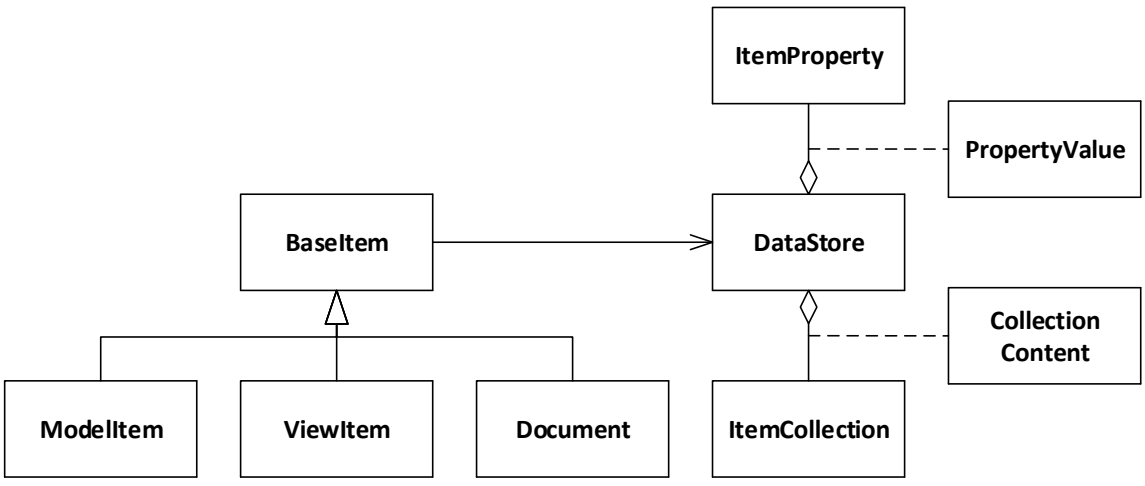

Рис. 6. Серверная объектная модель предметной области

Для реализации этих механизмов существует отдельное хранилище данных элемента (DataStore) (рис. 6), доступ к которому предоставляется через специальные объекты для свойств (ItemProperty) и коллекций (ItemCollection), которые полностью контролируются системой, что позволяет реализовать все указанные выше особенности.

Заключение

На данный момент система доработана до состояния архитектурного прототипа [19], т.е. реализует весь критический функционал на базовом уровне архитектуры. Помимо необходимой серверной инфраструктуры реализовано два клиентских приложения: на платформах Windows (Windows Store app) и Web (HTML + JavaScript). Дальнейшие исследования направлены в сторону интенсивного и экстенсивного расширения модели описания представления. Доступ к ознакомительной версии системы можно получить по адресу: http://beta.obordesus.com/

ЛИТЕРАТУРА

1. *Моисеев А.Н., Политов А.М., Хомич М.О.* Концепция системы поддержки командной разработки программного обеспечения с синхронизацией в режиме реального времени // Информационные технологии и математическое моделирование (ИТММ-2011) : материалы X Всерос. науч.-практ. конф. с междунар. участием (25–26 ноября 2011 г.). Томск, 2011. Ч. 1. С. 69–71.
2. *An overview* of Google Docs: Google Docs Help / Google. Service information. 2013. URL: http://support.google.com/drive/bin/answer.py?hl=en&answer=49008 (retrieval date: 13.03.2013).
3. *Status* of Google Wave: Google Wave Help / Google. Service information. 2013. URL: http://support.google.com/bin/answer.py?hl=en&answer=1083134 (retrieval date: 13.03.2013).
4. *Flow* Chart Maker & Online Diagram Software : Lucidchart / Lucid Software Inc. 2013. Service information. URL: http://www.lucidchart.com/ (retrieval date: 13.03.2013).
5. *Cacoo*: Create diagrams online Real time collaboration / Nulab inc. 2013. Service information. URL: https://cacoo.com/ (retrieval date: 13.03.2013).
6. UML, BPMN and Database Tool for Software Development / Visual Paradigm. 2011. Product information. URL: http://www.visual-paradigm.com/ (retrieval date: 13.03.2013).
7. ISO/IEC 19505-2:2012(E). Information technology – Object Management Group Unified Modeling Language (OMG UML). Pt. 2: Superstructure. April 2012. International Organization for Standardization, 2012. 740 p.
8. *Политов А.М., Рахимов В.В., Хомич М.О.* Клиент-серверное взаимодействие в системе поддержки командной разработки с синхронизацией в реальном времени // Материалы 50-й Международной научной студенческой конференции «Студент и научно-технический прогресс»: Информационные технологии. Новосибирск, 2012. С. 42.
9. W3C SVG Working Group / W3C. 2010. Electronic data. URL: http://www.w3.org/Graphics/SVG/ (retrieval date: 13.03.2013).
10. VML : the Vector Markup Launguage / J. Bowler, H. Cooperstein, B. Dister, A. Jindal, D. Lee, B. Mathews, T. Sandal, P. Wu. 1998. URL: http://www.w3.org/TR/NOTE-VML (retrieval date: 13.03.2013).
11. *Mamta Dalal, Ashish Ghoda.* XAML Developer Reference. Microsoft Press, December 2011. 342p.
12. *Raffaele Garofalo.* Building Enterprise Applications with Windows® Presentation Foundation and the Model View ViewModel Pattern. Microsoft Press, March 2011. 226 p.
13. *Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides.* Design Patterns : Elements of Reusable Object-Oriented Software. Addison-Wesley Professional, 1994. 416 p.
14. *Amazon* Elastic Compute Cloud (Amazon EC2), Cloud Computing Servers // Amazon. 2013. Service information. URL: http://aws.amazon.com/ec2/ (retrieval date: 13.03.2013).
15. *Cloud* Services on Windows Azure // Microsoft. 2013. Service information. URL: http://www.windowsazure.com/en-us/home/scenarios/cloud-services/ (retrieval date: 13.03.2013).
16. *Fowler M.* Patterns of Enterprise Application Architecture. Addison-Wesley Professional, 2002. 560 p.
17. *Programming* with Application Domains and Assemblies // Microsoft. 2013. Technology overview. URL: http://msdn.microsoft.com/en-us/library/dah4cwez.aspx (retrieval date: 13.03.2013).
18. .NET Remoting / Microsoft. 2013. Technology overview. URL: http://msdn.microsoft.com/en-us/library/vstudio/72x4h507(v=vs.100).aspx (retrieval date: 13.03.2013).
19. *Политов А.М., Рахимов В.В., Хомич М.О.* Прототип системы совместного редактирования диаграмм // Материалы XI Всероссийской научно-практической конференции с международным участием «Информационные технологии и математическое моделирование (ИТММ-2012)», Анжеро-Судженск, 23–24 нояб. 2012 г. Кемерово : Практика, 2012. Ч. 2. С. 60–62.

Моисеев Александр Николаевич, канд. техн. наук, доцент. E-mail: moiseev.tsu@gmail.com
Политов Арсентий Михайлович. E-mail: a.m.politov@gmail.com
Рахимов Валентин Валентинович. E-mail: rakhimovvv@gmail.com
Хомич Максим Олегович. E-mail: mohomich@gmail.com
Томский государственный университет
Поступила в редакцию 18 апреля 2014 г.

Moiseev Alexander N., Politov Arsenty M., Rakhimov Valentin V., Khomich Maxim O. (Tomsk State University, Tomsk, Russian Federation).
Real-time collaborative diagram editor "Obordesus": concept of unified presentation of elements.
Keywords: collaborative editing; diagram, graphic element; client-server architecture; real-time synchronization

The software for collaborative editing with the real-time synchronization is currently spreading, and therefore the requirements to the functionality of such systems are rising.

Obordesus is a centralized system for collaborative diagram editing with the real-time synchronization. Its main features are the following:

1. Real-time synchronization of currently active diagrams.
2. Extensibility, i.e. an ability to expand functionality without altering the system.
3. Stability of the client to changes in system functional (functional expansion should not affect client code).
4. Remote server, i.e. the main logic, project information and diagrams are stored remotely.

The basic domain area elements include the model entities, their visual images, diagrams (sets of visual images) and projects (sets of model entities and diagrams).

A goal of the system client-server interaction is to allow functional extension without the need to alter the clients. Targeting multiple platforms makes developing extensions for each client time-consuming. Moreover, the communication is held through network, which creates a delay between user's action and server's reaction.

To sum up, the client-server interaction mechanism should allow:

1. Drawing of any arbitrary element.
2. Data synchronization between clients and server.
3. User-element and element-element interaction.

The following higher-level solution meeting the requirements is suggested: client object model is based not on the server object model directly, rather on some kind of intermediate presentation description model.

The presentation description model is based on the following principles:

1. All the information about elements is contained in their properties.
2. Presentation of the element is derived from the properties (of the element itself or of any other elements).
3. Any element consists of three parts: model, visual model and presentation.
4. Presentation of the element is described as a composition of primitive elements.

New presentation description language based on the XML has been developed as the implementation of the listed principles.

Server-side implementation of the extensible domain area model needs the following mechanisms to be implemented:

1. Saving and loading of any specific model objects.
2. Transactions for composite modification of the objects (including rollback support).
3. Access rights control for model objects.

Model extensibility is provided by the extension module (plugin) subsystem on project server. The following requirements apply:

1. Plugins should be isolated from the project server and its hosting system.
2. Plugins should be isolated from each other.
3. The server should allow loading of different plugins (including different plugins versions) for each loaded project.

To meet these requirements the application domain isolation technique is used. Each project is loaded into separate application domain along with the required plugins and with minimal permissions. Model objects operations are executed in the project application domain, while network communication operations are processed by the project server itself.

As of this paper, the basic prototype of the system has been implemented. In addition to critical server-side infrastructure, clients for Windows (Windows Store app) and Web (HTML + JavaScript) have been implemented. Further research is focused on intensive and extensive development of the presentation description model.

REFERENCES

1. Moiseev A.N., Politov A.M., Khomich M.O. [Collaborative real-time software development supply system concept]. *Informatsionnye tekhnologii i matematicheskoe modelirovanie (ITMM-2011): materialy X Vseros. nauch.-prakt. konf. s mezhdunar. uchastiem* [Information technologies and mathematical modelling (ITMM-2011): Proc. of the 10th Russian national research-to-practice conference with international participation]. Tomsk, 2011. Part 1, pp. 69-71. (In Russian).
2. *An overview of Google Docs*: Google Docs Help. Google. Service information, 2013. Available at: http://support.google.com/drive/bin/answer.py?hl=en&answer=49008. (Accessed: 13th March 2013).
3. *Status of Google Wave*: *Google Wave Help*. Google. Service information. 2013. Available at: http://support.google.com/bin/answer.py?hl=en&answer=1083134. (Accessed: 13th March 2013).

4. *Flow Chart Maker & Online Diagram Software: Lucidchart*. Lucid Software Inc, 2013. Service information. Available at: http://www.lucidchart.com/. (Accessed: 13th March 2013).
5. *Cacoo: Create diagrams online Real time collaboration*. Nulab inc, 2013. Service information. Available at: https://cacoo.com/. (Accessed: 13th March 2013).
6. *UML, BPMN and Database Tool for Software Development*. Visual Paradigm, 2011. Product information. Available at: http://www.visual-paradigm.com/. (Accessed: 13th March 2013).
7. *ISO/IEC 19505-2:2012(E)*. Information technology – Object Management Group Unified Modeling Language (OMG UML). Part 2: Superstructure. April 2012. International Organization for Standardization, 2012. 740 p.
8. Politov A.M., Rakhimov V.V., Khomich M.O. [Client-server interaction in collaborative real-time software development supply system]. *Materialy 50-y Mezhdunarodnoy nauchnoy studencheskoy konferentsii "Student i nauchno-tekhnicheskiy progress": Informatsionnye tekhnologii* [Proc. of the 50th International Students Scientific Conference "Students and Progress in Science and Technology". Information Technologies]. Novosibirsk: Novosibirsk State University Publ., 2012, p. 42. (In Russian).
9. *W3C SVG Working Group*. W3C. 2010. Available at: http://www.w3.org/Graphics/SVG/. (Accessed: 13th March 2013).
10. Bowler J., Cooperstein H., Dister B., Jindal A., Lee D., Mathews B., Sandal T., Wu P. *VML: the Vector Markup Launguage*. 1998. Available at: http://www.w3.org/TR/NOTE-VML. (Accessed: 13th March 2013).
11. Mamta Dalal, Ashish Ghoda. *XAML Developer Reference*. Microsoft Press, December 2011. 342p.
12. Garofalo R. *Building Enterprise Applications with Windows® Presentation Foundation and the Model View ViewModel Pattern*. Microsoft Press, 2011. 226 p.
13. Gamma E., Helm R., Johnson R., Vlissides J. *Design Patterns: Elements of reusable object-oriented software*. Addison-Wesley Professional, 1994. 416 p.
14. *Amazon Elastic Compute Cloud (Amazon EC2), Cloud Computing Servers*. Amazon, 2013. Available at: http://aws.amazon.com/ec2/. (Accessed: 13th March 2013).
15. *Cloud Services on Windows Azure*. Microsoft, 2013. Available at: http://www.windowsazure.com/en-us/home/scenarios/cloud-services. (Accessed: 13th March 2013).
16. Fowler M. *Patterns of enterprise application architecture*. Addison-Wesley Professional, 2002. 560 p.
17. *Programming with Application Domains and Assemblies*. Microsoft, 2013. Available at: http://msdn.microsoft.com/en-us/library/dah4cwez.aspx. (Accessed: 13th March 2013).
18. *.NET Remoting*. Microsoft, 2013. Available at: http://msdn.microsoft.com/en-us/library/vstudio/72x4h507(v=vs.100).aspx. (Accessed: 13th March 2013).
19. Politov A.M., Rakhimov V.V., Khomich M.O. [Collaborative diagram editing system prototype]. *Materialy XI Vserossiyskoy nauchno-prakticheskoy konferentsii s mezhdunarodnym uchastiem "Informatsionnye tekhnologii i matematicheskoe modelirovanie (ITMM-2012)"* [Information technologies and mathematical modelling (ITMM-2012): Proc. of the 11th Russian national research-to-practice conference with international participation]. Kemerovo: Praktika Publ., 2012. Pt 2, pp. 60-62. (In Russian).