Разработка модуля десериализации и визуализации бизнес-моделей нотации *IDEF0*

Широбокова С.Н., к.э.н., доцент, Южно-Российский государственный политехнический университет (НПИ) имени М.И. Платова, Новочеркасск, Россия

Холодков В.С., магистрант, Южно-Российский государственный политехнический университет (НПИ) имени М.И. Платова, Новочеркасск, Россия

Бейбалаев А.М., магистрант, Южно-Российский государственный политехнический университет (НПИ) имени М.И. Платова, Новочеркасск, Россия

Аннотация. Для формализованного описания логики бизнес-процессов сегодня используются программные продукты, поддерживающие работу с нотацией *IDEF0*. Её активное использование обусловлено простотой описания логических отношений между работами без учёта их временной последовательности. Однако использование нотации *IDEF0* для создания абстрактной модели информационной системы не представляется возможным. Для этих целей наиболее подходящим решением будет являться применение нотации *UML*. С целью повышения эффективности бизнес-инжиниринговых подходов, основанных на применении упомянутых выше методологий, возникает задача реализации такого инструментария, который позволит преобразовывать *IDEF0*-модели в модели *UML*. В данной статье описана разработка модуля десериализации и визуализации *IDEF0*-моделей средствами языка программирования *C#*. Разработанный алгоритм позволяет получить визуальное представление любой модели. Реализованные алгоритмы станут основой для графического представления моделей других форматов. В дальнейшем полученные результаты будут служить основой для разработки

автоматизированного конвертера диаграмм *IDEF0/IDEF3* в *UML* диаграммы деятельности.

Ключевые слова: *IDEF0*, технология *.NET*, *UML*, *C#*, графический интерфейс *GDI+*.

Development of module deserialization and visualization of business models IDEF0 notation

Shirobokova S.N., Associate Professor, candidate of economical sciences, Platov South-Russian State Polytechnic University (NPI), Novocherkassk, Russia

Kholodkov V.S., master's degree student of Platov South-Russian State Polytechnic University (NPI), Novocherkassk, Russia

Beybalaev, A.M., master's degree student of Platov South-Russian State Polytechnic University (NPI), Novocherkassk, Russia

Annotation. For the formalized description of the logic of the business processes currently used software products that supports the IDEF0 notation. Its active use is due to the simplicity of descriptions, logical relationships between jobs without regard to their temporal sequence. However, the use of IDEF0 notation for creating abstract models of the information system is not possible. For these purposes, the most appropriate solution will be the use of UML notation. With the aim of increasing the effectiveness of business engineering approaches based on the use of the above mentioned methodologies, the challenge of implementing such a Toolkit, which will allow conversion of IDEF0 models into UML models. This article describes the development of the module, deserializing and visualization of IDEF0 models by means of the programming language C#. The developed algorithm allows to obtain a visual representation of any model. Algorithms will be the basis for the graphic representation of models in other formats. In the future the obtained results will serve as a basis for the development of automated Converter diagrams, IDEF0/IDEF3 UML activity diagrams.

Keywords: IDEF0, technology .NET, UML, C#, graphics, GDI+.

В работах [1-2] была предложена концепция и правила преобразований для моделей *IDEF0* в *UML*-диаграммы, на основе которых был реализован инструментарий «*ToADConverter*» [3]. Более поздние версии программных продуктов для моделирования в стандартах *IDEFx* (например, *AllFusion Process Modeler 7*) позволяют выгружать для обмена с другими приложениями смешанные модели *IDEF0* и *IDEF3* в формате *XML*-файла. Поэтому актуальна алгоритмизация задачи преобразования смешанных *IDEF0/IDEF3* моделей бизнес-процессов в *UML*-диаграммы и ее программная реализация [4]. Рассматриваемые в статье задачи получения и обработки данных в формате *XML*, а также дальнейшей визуализации являются основой для разработки автоматизированного конвертера.

На начальном этапе работы пользователь должен подготовить файл формата *XML*, генерируемый пакетом визуального моделирования бизнес-процессов *AllFusion Process Modeler*. После того, как этот документ был загружен в систему, начинается его первичная обработка, в основе которой лежит так называемый процесс десериализации.

Для хранения массива файлов был создан специальный класс, именуемый как *ProcessModeler*. Заполнение переменной данного типа производится с помощью функции *Deserialize*(), принимающей на вход считанный поток данных.

После успешной десериализации необходимо получить набор универсальных объектов, которые будут использованы в дальнейшей работе по визуализации модели *IDEF0*.

Описание общей структуры модели содержится в классе *GeneralDiagram*. Он содержит в себе следующие свойства:

- *DiagramName* – строковое свойство, хранящее информацию о наименовании модели;
- *Autor* – строковое свойство, содержащее данные об авторе модели;

- *AllDiagrams* – свойство-коллекция типа *DiagramGroups*, содержащее информацию обо всех диаграммах, имеющихся в данной модели, и их характеристиках;
- *AllActivityes* – свойство-коллекция типа ActivityGroups, содержащее общую информацию (идентификатор, имя, тип и т.п.) о всех блоках модели;
- *AllArrows* – свойство-коллекция типа *ArrowGroups*, содержащее общую информацию (идентификатор, имя, тип и т.п.) о всех стрелках модели.

Рассмотрим более подробно дочерние классы для описанного выше класса. Класс *ActivityGroups* содержит следующие свойства:

- *Name* – строковое свойство, описывающее наименование блока, присвоенное ему автором модели;
- *Id* – строковое свойство, описывающее идентификатор блока;
- *BoxType* – свойство типа *byte*, описывающее тип блока (его принадлежность к той или иной нотации).

Класс *ArrowGroups* содержит в своём составе следующие свойства:

- *Id* – строковое свойство, описывающее идентификатор стрелки;
- *Name* – строковое свойство, описывающее наименование стрелки, присвоенное автором модели.

Класс *DiagramGroups* содержит в своём составе следующие элементы:

- *Name* – строковое свойство, описывающее наименование диаграммы;
- *Id* – строковое свойство, описывающее идентификатор диаграммы;
- *BoxGroupsForEachDiag* – свойство-коллекция типа *BoxGroups*, описывающее свойства блоков на конкретной диаграмме;
- ArrowLabelForEachDiag – переменная типа *ArrowLabelGroups*, описывающая свойства стрелок на конкретной диаграмме.

Рассмотрим более подробно класс *BoxGroups*, предназначенный для хранения свойств блоков, входящих в состав той или иной диаграммы декомпозиции. Он содержит следующие структурные элементы:

- *Id* – строковое свойство, хранящее информацию об идентификаторе конкретного блока;
- *ActivityRef* – строковое свойство, хранящее идентификатор блока, по которому осуществляется получение данных о первичных свойствах блока, описанных в классе *ActivityGroups*;
- *Name* – строковое свойство, хранящее наименование блока диаграммы;
- *CordLeft* – целочисленное свойство, описывающее значение левой координаты блока;
- *CordRight* – целочисленное свойство, описывающее значение правой координаты блока;
- *CordTop* – целочисленное свойство, описывающее значение верхней координаты блока;
- *CordBottom* – целочисленное свойство, описывающее значение нижней координаты блока.

Перейдём к описанию класса *ArrowLabelGroups*, который содержит информацию о свойствах всех стрелок, принадлежащих конкретной диаграмме модели. Он содержит свойство типа *ArrowLabel*. Данный класс состоит из следующих элементов:

- *Id* – строковое свойство, описывающее идентификатор конкретной стрелки в конкретной диаграмме;
- *CordBoxLabLeft* – целочисленное свойство, описывающее левую координату прямоугольника, в который будет вписано наименование стрелки;
- *CordBoxLabTop* – целочисленное свойство, описывающее верхнюю координату прямоугольника;
- *CordBoxLabRight* – целочисленное свойство, описывающее правую координату прямоугольника;
- *CordBoxLabBottom* – целочисленное свойство, описывающее нижнюю координату прямоугольника;

- *ArrowRef* – строковое свойство, указывающее идентификатор стрелки, позволяющий получить доступ к первичным свойствам каждой из них, описанных в классе *ArrowGroups*;
- *SegmentsProps* – свойство-коллекция типа *ArrowSegment*, содержащая информацию обо всех сегментах, из которых состоит конкретная стрелка.

Класс *ArrowSegment* содержит в своём составе следующие структурные элементы:

- *Id* – строковое свойство, хранящее идентификатор каждого конкурентного сегмента;
- *CordSegments* – свойство-коллекция типа *Vertices*, хранящая координаты каждого сегмента стрелки.

Класс *Vertices*, описывающий координаты каждого сегмента, содержит следующие элементы:

- *X* – координата точки сегмента по оси абсцисс;
- *Y* – координата точки сегмента по оси ординат.

Заполнение объекта, обладающего описанной выше структурой, происходит непосредственно в процессе его создания с помощью соответствующего конструктора, принимающего на вход десериализованный объект с данными из документа *XML*. Первоначально происходит заполнение данных, являющихся общими для всей модели (наименование и сведения об авторах). Затем с помощью соответствующих циклов перебираются первичные свойства стрелок и блоков, имеющихся в модели. Полученные данные записываются в соответствующие универсальные объекты.

Далее с помощью цикла перебираются все имеющиеся в *XML*-документе сведения о диаграммах модели, а также о стрелках и блоках, входящих в диаграммы. На основе полученных универсальных объектов (в частности свойства *GeneralDiagramIDEF0.BoxGroupsForEachDiag*) происходит заполнение так называемого «дерева диаграмм», описывающего структуру модели в главном окне приложения.

Программная реализация описанной выше функциональности приведена на листинге 1.

```
void FillTree (List<DiagramGroups> diagrams, TreeView treeV, StatusStrip status, ToolStripStatusLabel labelX)
{
treeV.Nodes.Add(new TreeNode(this.DiagrammName)); /// Добавляем самый верхний узел дерева с именем диаграммы
treeV.Nodes[1].Nodes.Add(diagrams[1].Id, diagrams[1].Name);
///добавляем дочерний узел с именем контекстной диаграммы treeV.Nodes[1]
try
{
///Цикл, добавляющий группу дочерних узлов с именами диаграмм, декомпозирующих контекстную диаграмму
for (int i = 0; i < diagrams[1].BoxGroupsForEachDiag.Count; i++)
{
treeV.Nodes[1].Nodes[0].Nodes.Add(diagrams[1].BoxGroupsForEachDiag[i].Id, diagrams[1].BoxGroupsForEachDiag[i].Name);
}
///Цикл, загружающий дочерние узлы с именами диаграмм, декомпозирующих блоки декомпозиции контекстной диаграммы
int j = 2;
for (int i = 0; i < treeV.Nodes[1].Nodes[0].Nodes.Count; i++)
/// Перебираем все узлы, декомпозирующие контекстную диаграмму
{
for (; j < diagrams.Count;)
/// Перебираем все диаграммы начиная 2-й
{for (int k = 0; k < diagrams[j].BoxGroupsForEachDiag.Count; k++)
/// Перебираем все блоки, входящие в состав текущей диаграммы.
{
/// Добавляем новый дочерний узел для каждой диаграммы, входящей в декомпозицию блоков декомпозиции контекстной диаграммы
treeV.Nodes[1].Nodes[0].Nodes[i].Nodes.Add(diagrams[j].BoxGroupsForEachDiag[k].Id,
diagrams[j].BoxGroupsForEachDiag[k].Name); }
j++;
break; }}}
catch (Exception ex)
{status.BackColor = System.Drawing.Color.Red;
labelX.Text = ex.Message;}}}
```

Листинг 1 – Описание процесса загрузки информации о структуре модели IDEF0

Разработанная структура универсальных объектов и алгоритмы их наполнения данными позволили составить удобный для работы класс, позволяющий избегать глубокой вложенности элементов, присущей исходному объекту, полученному в результате десериализации, и, как следствие, облегчить дальнейший процесс разработки алгоритмов рисования и визуализации.

Главным универсальным объектом разрабатываемого конвертера станет коллекция, получившая название *GeneralDiagram*. Она содержит в себе следующие объекты: *AllDiagrams* – свойство-коллекция типа *DiagramGroups*, содержащее информацию обо всех диаграммах, имеющихся в данной модели, и их характеристиках; *AllActivityes* – свойство-коллекция типа *ActivityGroups*, содержащее общую информацию (идентификатор, имя, тип и т.п.) о всех блоках модели; *AllArrows* – свойство-коллекция типа *ArrowGroups*, содержащее общую информацию (идентификатор, имя, тип и т.п.) обо всех стрелках модели [5].

Исходя из описанной выше структуры универсальных объектов, в одном структурном элементе приложения будет собран весь перечень данных, необходимых и достаточных для построения визуальной модели и дальнейшего её преобразования.

Что касается визуализации модели, то для реализации данной цели в рамках проекта был создан отдельный класс, получивший название *DrawingElement*. Его функциональность целиком и полностью основана на использовании стандартной библиотеки *System.Drawing*.

Перед началом описания методов необходимо отметить, что для корректной прорисовки необходимо создать переменную типа *Graphics*, инкапсулирующую поверхность рисования *GDI+*. Также для работы необходимо выполнить первоначальную инициализацию переменных класса *Pen*, позволяющих задать стиль прорисовки границ стрелок и блоков, а также их выделения.

Процедуры рисования должны быть вызваны тогда, когда пользователь осуществляет перемещение по узлам и элементам «дерева объектов» – главного инструмента навигации в приложении. Для этих целей в описании логики главной формы приложения для элемента *TreeView* было реализовано событие *AfterSelect*, происходящее тогда, когда пользователь выбрал новый элемент в дереве. Данное событие позволяет получить доступ к таким параметрам, как уровень узла, его индекс, а также имя, которые в дальнейшем будут

использованы для однозначной идентификации того или иного блока диаграммы.

Также стоит сказать несколько слов об организации рабочей области для рисования. Её основу составляют два элемента: *Panel* и *PictureBox*. Для первого элемента обязательно необходимо установить свойство *AutoSize* в значение *false*, чтобы его размер был строго фиксирован. Размеры элемента *PictureBox* будут изменяться в рамках алгоритма в зависимости от размеров и положения входящих в состав диаграммы блоков.

При появлении полос прокрутки для элемента *Panel*, корректная прорисовка может быть нарушена. Во избежание возникновения подобных ситуаций было настроено событие *Scroll*, возникающее при перемещении ползунка прокрутки.

Для рисования блоков была создана процедура *DrawingActivity*(), принимающая на вход переменную, обозначающую панель для рисования, переменную, описывающую совокупность универсальных объектов, индекс и уровень узла дерева, а также его наименование.

Изначально происходит очистка области рисования и в зависимости от выбранного узла или уровня вызывается процедура *SelectActivity*(), принимающая аналогичные предыдущей функции параметры. Она предназначена для непосредственной визуализации блоков модели и их выделения.

На первом этапе инициализируются массивы типов *Rectangle*, *StringFormat* и *string*. В первом будет храниться информация о конфигурации прямоугольников, соотнесённых с блоками диаграммы, второй необходим для хранения характеристик форматирования, а третий будет содержать информацию о наименованиях каждого из блоков.

После этого с помощью цикла перебираются все диаграммы и в соответствующие переменные записываются их координаты. Также вычисляются параметры ширины и высоты текущего блока, после чего происходит создание нового элемента массива *Rectangle* путём передачи ему

значений координат левого верхнего угла блока и текущей длины и ширины. В рамках алгоритма также присутствует возможность проверки на предмет того, все ли блоки диаграммы вместились в область для рисования. В случае, если это не так, автоматически происходит изменение размера элемента *PictureBox*. В массив типа *StringFormat* помещается элемент, хранящий свойства горизонтального и вертикального выравниваний.

Затем циклично перебираются все элементы массива *Rectangle*, для каждого из которых проверяется условие совпадения имени узла. В случае если оно совпало – происходит рисование прямоугольника с помощью функции *DrawRectanle*(), которой передаётся тип пера (в данном случае обладающим зелёным цветом и жирным вариантом начертания) и параметры самого прямоугольника. После выполнения данной операции вызывается функция *DrawString*().

После прорисовки наименований всех блоков этап визуализации блоков диаграммы подходит к концу. Следующий этап обеспечивает рисование стрелок, относящихся к текущей диаграмме модели. Для этого была реализована соответствующая процедура, получившая название *DrawArrows*(). В качестве параметров ей передаются текущая модель и графический объект типа *Graphics*.

В самом начале работы процедуры объявляется коллекция координат точек типа *Point*. Затем перебираются сначала все стрелки, входящие в модель, после этого все сегменты, входящие в состав текущей стрелки. При каждой новой итерации цикла перебора сегментов происходит новая инициализация коллекции точек. Перед заполнением осуществляется проверка на предмет чётности или нечётности индекса группы координат. Если индекс нечётный или равен нулю – запись в коллекцию производится, в противном случае – нет. Это обусловлено тем, что все чётные индексы в файле *XML* хранят информацию о направлении сегментов и не несут практической значимости при решении задачи визуализации.

После заполнения массива точек, осуществляется перебор его элементов. На каждой итерации цикла происходит проверка текущего индекса, смещённого на единицу, на предмет того, превышает ли он количество элементов в коллекции. Если условие ложно, то происходит прорисовка линии с помощью функции *DrawLine*(), которая принимает на вход тип пера, текущую координату (координату начала) и координату, следующую за текущей (координату конца). В случае, если условие выполнилось и значение индекса вышло за пределы диапазона коллекции, циклы прерывают свою работу. На этом процесс визуализации заканчивается.

На рисунке 1 представлен внешний вид работы приложения, визуализирующего исходную бизнес-модель.

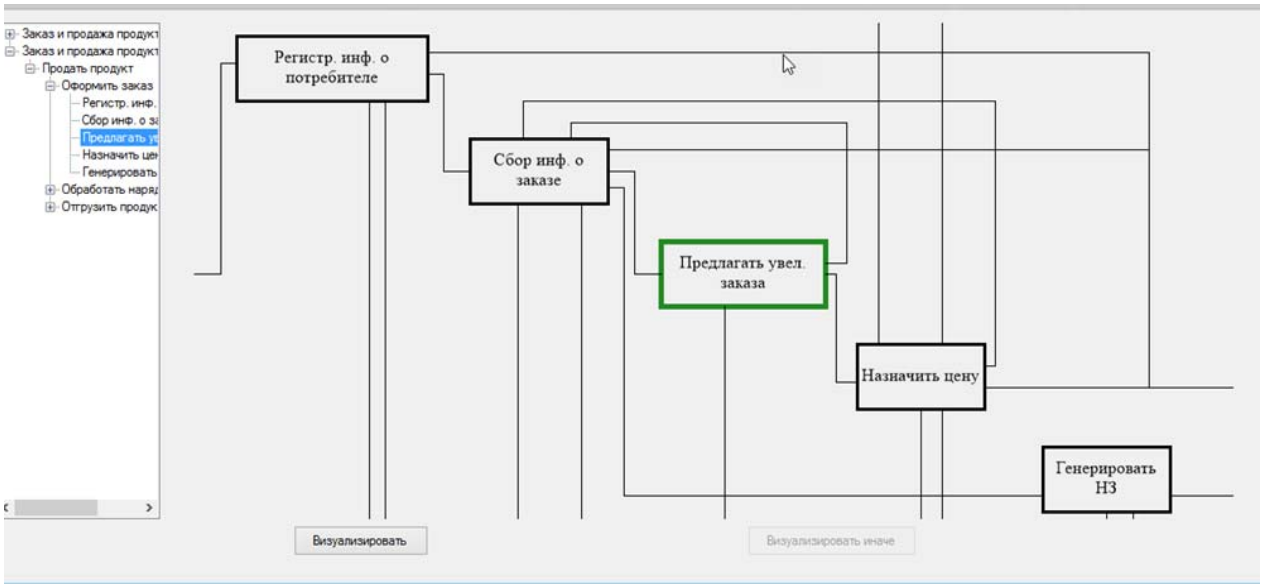

Рис. 1 – Главное окно приложения, визуализирующего модель IDEF0

Таким образом, был разработан алгоритм десериализации и визуализации бизнес-моделей нотации *IDEF0*, исходными данными для которого является *XML*-документ, выгруженный из программного пакета *AllFusion Process Modeler*. Результаты проведенной работы будут служить основой для разработки автоматизированного конвертера. Интеграция разрабатываемого инструментария с системой автоматизированного синтеза имитационных моделей СИМ-*UML* [6] дает эффективный инструмент решения различных экономических и технико-технологических задач, связанных с экспресс-оценкой и оптимизацией ресурсоемкости товаров и услуг.

Библиографический список

1. Хубаев Г.Н., Широбокова С.Н. Конвертирование диаграмм *IDEF0* в *UML*-диаграммы: концепция и правила преобразования // Проблемы экономики.– 2008.– № 6.– С. 139-152.

2. Хубаев Г.Н., Широбокова С.Н. Автоматизированный конвертор моделей *IDEF0* в *UML*-диаграммы: концептуальная идея // Проблемы информатики в образовании, управлении, экономике и технике: сб. ст. VIII Всерос. науч.-техн. конф., 19-20 нояб. 2008 г.– Пенза: Приволжский Дом знаний, 2008.– С. 102-108.

3. Хубаев Г.Н., Широбокова С.Н., Титаренко Е.В., Ткаченко Ю.В. Конвертирование моделей *IDEF0* в *UML*-диаграммы: формализованная постановка задачи // Математика. Компьютер. Образование: тез. докл. XVI Междунар. конф., г.Пущино, 19-24 янв. 2009 г.– М. ; Ижевск, 2009.– Вып. 16. Ч. 2.– С. 402.

4. Широбокова С.Н., Титаренко Е.В., Ткаченко Ю.В. Формализованная постановка задачи конвертирования диаграмм бизнес-процессов в стандарте *IDEF0* в диаграммы деятельности *UML* // Математические методы и информационные технологии в экономике, социологии и образовании : сб. ст. XXII Междунар. науч.-техн. конф. (зимняя сессия), дек. 2008 г. / Пенз. гос. технол. акад.– Пенза : Приволжский Дом знаний, 2008.– С. 19-22.

5. Хубаев Г.Н., Широбокова С.Н. Формирование универсальных объектов конвертера «*TOADCONVERTER*» из *XML*-файла экспорта *IDEF3*-диаграмм // Вестник Ростовского государственного экономического университета (РИНХ). – №1(53).– 2016.– С.178-185.

6. Хубаев Г.Н., Широбокова С.Н., Щербаков С.М. Автоматизированный синтез имитационных моделей деловых процессов // Изв. вузов. Сев.-Кавк. регион. Техн. науки.– 2008.– № 4.– С. 73-79.

References

1. Khubaev G.N., Shirobokova S.N. The conversion of the IDEF0 diagrams in UML diagrams: concept and conversion rules // Problemy ekonomiky.– 2008.– № 6.– pp. 139-152.

2. Khubaev G.N., Shirobokova S.N. Automated conversion of IDEF0 models into UML-diagrams: the conceptual idea // Problemy informatiki obrazovanii, upravlinii ekonomike i technike: sb. st. VIII Vseros. nauch.-tekhn. konf., 19-20 noyab. 2008 g.– Penza: Privolzhskij Dom znanij, 2008.– pp. 102-108.

3. Khubaev G.N., Shirobokova S.N., Titarenko E.V., Tkachenko U.V. The conversion of IDEF0 models into UML-diagrams: the formalized statement of the problem // Matematika. Computer. Obrazovanie: tez. dokl. XVI Mezhdunar. konf., g.Pushchino, 19-24 yanv. 2009 g. – M. ; Izhevsk, 2009.– Vyp. 16. CH. 2.– p. 402

4. Shirobokova S.N., Titarenko E.V., Tkachenko U.V. A formalized statement of the problem conversion charts business processes in IDEF0 standard in activity diagrams UML // Matematicheskie metody i informacionnye tekhnologii v ehkonomike, sociologii i obrazovanii : sb. st. XXII Mezhdunar. nauch.-tekhn. konf. (zimnyaya sessiya), dek. 2008 g. / Penz. gos. tekhnol. akad.– Penza : Privolzhskij Dom znanij, 2008.– pp. 19-22.

5. Khubaev G.N., Shirobokova S.N. The formation of a universal object Converter «TOADCONVERTER» from the XML export file IDEF3 diagrams // Vestnik Rostovskogo gosudarstvennogo ehkonomicheskogo universiteta (RINH).- №1(53). – 2016. – pp.178-185.

6. Khubaev G.N., Shirobokova S.N., Sherbakov S.M. Automated synthesis of simulation models of business processes // Izv. universities. SEV.-Kavko. region. Tech. science. – 2008.– No. 4.– pp. 73-79.