АВТОМАТИЗИРОВАННАЯ СИСТЕМА ТРАНСФОРМАЦИИ ДИАГРАММ БИЗНЕС-ПРОЦЕССОВ В ДИАГРАММЫ КЛАССОВ

А. Ю. Атисков, В. И. Воробьев

Санкт-Петербургский институт информатики и автоматизации РАН

СПИИРАН, 14-я линия ВО, д. 39, Санкт-Петербург, 199178

<atiskov@mail.ru>

УДК 681.3

Атисков А. Ю., Воробьев В. И. **Автоматизированная система трансформации диаграмм бизнес-процессов в диаграммы классов** // Труды СПИИРАН. Вып. 3, т. 2. — СПб.: Наука, 2006.

Аннотация. *Объектом исследования является автоматизированная система трансформации компонент диаграмм языка структурного моделирования IDEF0 в диаграммы классов в нотации UML, повышающая эффективность проектирования на стадии смены функционального моделирования объектно-ориентированным, которая будет способна настраиваться под конкретные правила перевода, заданные пользователем.* — Библ. 5 назв.

UDC 681.3

Atiskov A. J., Vorobiev V. I. **Semi-automatic system of transforming business-process diagrams into class diagrams** // SPIIRAS Proceedings. Issue 3, vol. 2. — SPb.: Nauka, 2006.

Abstract. *Semi-automatic system transforming components of diagrams IDEF0 into class-diagrams in UML-notation is described. Enhancement of efficiency of design when functional design gives place to object-oriented one is discussed. The tuning of system according to concrete rules of transformation is provided for serviceability.* — Bibl. 5 items.

1. Постановка задачи

Процесс разработки информационных систем развивается по спирали, практически всегда возвращаясь к постановке задачи, но уже с новыми спецификациями. Таким путем бизнес непрерывно стремится улучшить свою эффективность за счет применения гибких, адаптируемых и перестраиваемых технологий. Долгосрочные прогнозы все сложнее сделать в изменяющейся окружающей среде. Поэтому предприятие (бизнес) должно не только быстро приспосабливаться к изменениям, но также и непрерывно развиваться. В этих условиях способность информационной системы перестраиваться становится постоянным внутренним процессом, а не просто внешним фактором, к которому предприятие должно приспособиться. Технологичное бизнес-моделирование представляет собой необходимый элемент создания адаптируемого программного обеспечения.

Материализация бизнес-идей (представленных диаграммами бизнес-процессов) в виде модели программного обеспечения является трудоемким процессом. Сложность обусловлена применением технологически разрозненных методов и средств описания бизнес-процессов; в частности, на практике применяются как функциональное моделирование бизнес-процессов, так и объектно-ориентированное. Не все классы и объекты, определенные в бизнес архитектуре, могут быть включены в модель программного обеспечения (например, диаграмму UML). Поэтому трансформация бизнес-модели в модель программного обеспечения не является строго формализованным процессом [1, 2].

Труды СПИИРАН. 2006. Вып. 3. Т. 2. ISSN 2078-9181 (печ.), ISSN 2078-9599 (онлайн)
SPIIRAS Proceedings. 2006. Issue 3. V. 2. ISSN 2078-9181 (print), ISSN 2078-9599 (online)
www.proceedings.spiiras.nw.ru

Не существует полностью универсальных инструментов, поэтому задача заключается в том, чтобы найти *соответствующий* инструмент для проектирования. Для заказчика важно, чтобы он предоставлял ясные описания бизнес-процессов. А для проектировщика важно, чтобы инструмент был удобным в работе и содержал максимум возможностей и их эффективную реализацию. Примерами инструментов являются средства построения бизнес-модели (AllFusion Process Modeler, AI0Win, Rational Rose) [3].

Бизнес-модель может использоваться в процессе разработки программного обеспечения для того, чтобы:

- принимать решение при выборе вида информационной системы (инновационная, стандартная или соблюдающая преемственность);
- определять *функциональные* требования: набор функций использования (Use Cases), которые должна поддерживать информационная система;
- определять *нефункциональные* требования (затрагивающие всю систему);
- идентифицировать подходящие *бизнес*-компоненты, которые инкапсулируют определенную область бизнес-функциональности.

На стадии перехода к модели программы возникает проблема трансформаций диаграмм бизнес-процессов в диаграммы описания программного кода. Здесь полезны автоматизированные средства представления требований на изменение, которые были сделаны в одном средстве, в терминах и моделях другого средства. В частности, при повторном использовании данных (CASE-средства зачастую оперируют одними понятиями и терминами), удобнее не ручной перенос их из одной модели в другую, а чтобы это выполнялось автоматически или с минимальным вмешательством в особых случаях. На этапе проектировании программного продукта перед его имплементацией разрабатываются диаграммы на основе моделей «Сущность–Связь» (ERD). Это может являться и диаграммой классов UML, и диаграммой IDEF1X — в зависимости от архитектуры системы (база данных, система управления, локальное приложение и так далее). На основе таких объектных диаграмм дальше работают архитекторы программного кода, оптимизаторы, и проводится рефакторинг [4]. Примерами инструментов, поддерживающих построение ER-моделей, являются Together, Rational Rose, ERwin, Paradigm+.

2. Программные средства для трансформации диаграмм IDEF0

Ряд инструментов, поддерживающих данные технологии, позволяет проводить трансформацию диаграмм бизнес-процессов в диаграммы классов (BPwin, ERwin, Paradigm+, Rational Rose, BPLink). Был проведен их анализ. В качестве критериев оценки возможностей, предоставляемых данными средствами, были использованы:

1) время на выполнение;
2) сложность освоения и проведения;
3) потери данных при трансформации;
4) время, с которого возможно проводить трансформацию (на какой стадии проектирования бизнес-процессов можно проводить пробную трансформацию без существенных затрат времени и средств);
5) возможность настройки процесса трансформации (степень автоматичности);
6) стоимость подхода (дополнительного программного обеспечения).

Труды СПИИРАН. 2006. Вып. 3. Т. 2. ISSN 2078-9181 (печ.), ISSN 2078-9599 (онлайн)
SPIIRAS Proceedings. 2006. Issue 3. V. 2. ISSN 2078-9181 (print), ISSN 2078-9599 (online)
www.proceedings.spiiras.nw.ru

Результаты исследований различных подходов для трансформации диаграмм совпадают с указанными авторами назначениями программ. А именно то, что они в большинстве своем служат:

- для связи моделей, а не для повторного использования предыдущих;
- для поддержания целостности представления всей системы в различных разрезах и с различных точек зрения;
- для мультипользовательской разработки и контроля версий.

С другой стороны, при сравнении линеек BPwin–ERwin–Rational Rose и BPwin–ERwin–Paradigm Plus было выявлено, что:

1) требуется время на заполнение словарей сущностей и атрибутов (и связывание их с процессами и стрелками), при этом, в случае их большого количества, возможно внесение ошибок (также требуется настройка как промежуточного, так и конечного средства проектирования);
2) требуются навыки моделирования на промежуточном средстве (ERwin, Paradigm Plus);
3) при трансформации теряется основное, что создается в IDEF0 — процессы;
4) проведение трансформации возможно лишь при заполненных словарях сущностей и атрибутов, которые чаще всего заполняют только при связывании модели бизнес-процессов с ERwin (и выполняется это чаще другими людьми, например, администраторами баз данных);
5) трансформация полностью автоматична. Вносить исправления возможно, лишь открыв диаграмму классов в конечном средстве проектирования;
6) в стоимость проекта добавляется стоимость промежуточного средства проектирования.

Перечисленные особенности чаще всего не позволяют повторно использовать модель бизнес-процессов для автоматизированного построения диаграмм классов.

Использование же подхода BPwin–BPLink–Paradigm Plus напрямую не дает диаграммы классов. Требуется трансформация диаграмм Use Case в диаграмму классов, что представляет собой отдельную задачу. К тому же теряются связи входных и выходных параметров.

3. Формальное описание элементов нотаций

Для описания системы трансформации следует выделить основные элементы диаграмм. Обозначим множество названий элементов (собственных имен, например «Адаптация») как N, состоящего из элементов n_i.

По диаграммам IDEF0 выделим пятерку:

$$E = \langle F, A, EN, R, ATR \rangle,$$

где F — множество функциональных блоков на диаграмме; A — множество стрелок; EN — множество сущностей; R — множество ролей стрелок; ATR — множество атрибутов.

Труды СПИИРАН. 2006. Вып. 3. Т. 2. ISSN 2078-9181 (печ.), ISSN 2078-9599 (онлайн)
SPIIRAS Proceedings. 2006. Issue 3. V. 2. ISSN 2078-9181 (print), ISSN 2078-9599 (online)
www.proceedings.spiiras.nw.ru

Множество F состоит из элементов

$$f_i = <n_i, p_i>,$$

где n_i — элемент из множества названий N; p_i — ссылка на родительский блок для данного блока из множества F.

Множество R состоит из элементов

$$R = <I, C, O, M>,$$

где I — обозначение входной стрелки; C — обозначение стрелки управления; O — обозначение выходной стрелки; M — обозначение стрелки механизма.

Множество A состоит из элементов

$$a_j = <n_j, r_j, p_j>,$$

где n_j — элемент из множества названий N; r_j — элемент из множества ролей стрелок; p_j — ссылка на блок, который связан с данной стрелкой

Множество ATR состоит из элементов

$$atr_m = <n_m, p_m, pEn_m>,$$

где n_m — элемент из множества названий N; p_m — ссылка на блок или стрелку, связанных с данным атрибутом; pEn_m — ссылка на сущность, агрегирующей данный атрибут.

Множество EN состоит из элементов

$$en_k = <n_k, p_k>,$$

где n_k — элемент из множества названий N; p_k — ссылка на блок или стрелку, связанных с данной сущностью.

Для диаграмм классов выделим такую совокупность:

$$U = <CL, AtrCl, MT, PM, RL, T, OB, TREL>,$$

где CL — множество классов; AtrCl — множество атрибутов классов; MT — множество методов классов; PM — множество параметров методов классов; RL — множество отношений между классами (наследование, ассоциация и т.д.); T — множество типов; TREL — множество типов отношений; OB — множество объектов.

Множество T состоит из элементов

$$t_i = <n_i, pCl_i>,$$

где n_i — элемент из множества названий N; pCl_i — ссылка на класс, который реализует тип.

Множество OB состоит из элементов

$$ob_j = <n_j, pT_j>,$$

где n_j — элемент из множества названий N; pT_j — ссылка на тип данного объекта.

Множество AtrCL состоит из элементов

$$atrCl_k = <n_k, pT_k, pCl_k>,$$

где n_k — элемент из множества названий N; pT_k — ссылка на тип данного атрибута; pCl_k — ссылка на класс, агрегирующий данный атрибут.

Множество PM состоит из элементов

$$pm_n = <n_n, pT_n, pMt_n>,$$

где n_n — элемент из множества названий N; pT_n — ссылка на тип данного параметра метода; pMt_n — ссылка на метод, которому принадлежит параметр.

Множество RL состоит из элементов

$$rl_r = <pClParent_r, pClChild_r, pTrel_r>,$$

где $pClParent_r$ — ссылка на класс-родитель отношения; $pClChild_r$ — ссылка на класс-приемник отношения; $pTrel_r$ — вид отношения.

Труды СПИИРАН. 2006. Вып. 3. Т. 2. ISSN 2078-9181 (печ.), ISSN 2078-9599 (онлайн)
SPIIRAS Proceedings. 2006. Issue 3. V. 2. ISSN 2078-9181 (print), ISSN 2078-9599 (online)
www.proceedings.spiiras.nw.ru

Множество MT состоит из элементов

$$mt_m = <n_m, pT_m, pCl_m>,$$

где n_m — элемент из множества названий N; pT_m — ссылка на тип данного метода (возвращаемое значение); pCl_m — ссылка на класс, агрегирующий данный метод.

Множество TREL состоит из элементов

$$Trel = <As, Ag, Comp, Gen>,$$

где As — ассоциация; Ag — агрегация; Comp — композиция; Gen — наследование;

Множество CL состоит из элементов

$$cl_p = <n_p>,$$

где n_p — элемент из множества названий N.

Итак, получаем, что требуется сформировать такое множество правил перевода RULES, которое будет отображать из E в U.

$$RULES: E \rightarrow U.$$

На основе приведенных описаний элементов можно перейти к формальному построению правил трансформаций, например, с помощью применения технологии онтологического анализа. Но перед этим шагом, следует рассмотреть технику неформальных методов правил трансформаций — для этого был проведен анализ приемов ряда специалистов, используемых ими при переходе от функциональных диаграмм к объектно-ориентированным. В результате были выявлены как наиболее простые (и наиболее часто встречаемые) правила, так и правила, сильно зависящие от контекста данных на IDEF0-диаграммах.

Обобщенные результаты представлены в табл. 1.

Таблица 1

Обобщенное представление трансформаций для автоматизированного перевода диаграмм

	Блок	**Вход**	**Выход**	**Механизм**	**Управление**
Класс	B	B (1)	B (1)	B (A, 1)	B
Объект типа класса	-	B (1)	B (1)	B (1)	B (A 1)
Тип	-	B (H)	B (H)	B (H)	-
Атрибут	-	B (H)	B (H)	-	B (H 1)
Метод	B (A, 1)	-	-	-	-
Параметр метода	-	B (A, 1)	B	-	-
Наследование	-	B (H)	B (H)	B (H)	B (H)
Тип метода	-	-	B (A, 1)	-	-
Роль ассоциации	B (H)	-	-	-	-
Зависимость	-	-	B	-	B

B — такая трансформация возможна;

H — трансформация сильно зависит от настроек правил;

A — трансформация является автоматической (выполняется, если иное не определено проектировщиком);

1 — трансформация, которая встречается чаще, нежели другие в данном столбце.

Для формализации правил трансформаций необходимо сравнить семантику понятий, используемых при объектно-ориентированном и функциональном

Труды СПИИРАН. 2006. Вып. 3. Т. 2. ISSN 2078-9181 (печ.), ISSN 2078-9599 (онлайн)
SPIIRAS Proceedings. 2006. Issue 3. V. 2. ISSN 2078-9181 (print), ISSN 2078-9599 (online)
www.proceedings.spiiras.nw.ru

проектировании. Далее выделены основные моменты, которые следует учитывать при их определении (табл. 2).

Таблица 2

Сравнение семантик при объектно-ориентированном и функциональном проектировании (полужирным шрифтом выделены элементы определенной нотации)

Объект трансформации	Описание возможных трансформаций
1	2
Диаграммы функциональной модели на нижнем уровне детализируют верхние.	Порядок создания элементов диаграммы классов должен быть в целом таков: 1) определение **типов**; 2) определение **классов**; 3) определение **наследования**; 4) определение **объектов типа класса**; 5) определение **атрибутов класса**; 6) определение **методов**, **параметров метода, типа метода, ролей ассоциации**.
По стандарту ИСО каждый **блок** должен иметь хотя бы одну **управляющую** и одну **исходящую дугу**.	Исполнитель **метода** не всегда определен. В таких случаях в качестве исполнителя следует брать ближайший **механизм** на **родительской диаграмме** или преобразовывать название **блока** в **класс** с одним **методом**, причем, возможно, имя **класса** будет либо именем **блока**, либо именем **выхода**. **Выход** при этом может быть **атрибутом механизма**.
Управление — это часть **блока**.	**Управление** может быть одним из **атрибутов класса**.
Дуга должна быть помечена до разветвления. А после разветвления, если есть **пометка** — **дуга** содержит все или часть объектов, указанных до разветвления, если нет пометки — **дуга** содержит все объекты, указанные до разветвления. **Дуга** должна быть помечена после слияния. А до слияния, если **дуга** не помечена — она содержит все объекты результата, если помечена — содержит все или часть объектов результата. Разветвляющиеся/соединяющиеся **дуги** отражают иерархию объектов, представленных этими **дугами**.	Эти правила должны определять возможность **наследования** (либо указано пользователем, либо употреблены однокоренные слова в названиях **стрелок**).
Дуги представляют множество объектов: планы, данные в компьютерах, машины и информация, бумажные материалы, физические материалы, инструменты, чертежи, рабочая среда, управленческая информация. **Дуги** описываются существительными или существительными с определениями.	Чаще всего **дуги** относятся к **именам классов**, **атрибутам, типам** и **объектам**.
Дуги механизмов — отражают, как функции реализуются. Показывают физические аспекты функций (склады, людей, приборы).	Дуги **механизмов** чаще всего являются **классами**, **объектами** класса, для которых **блок** — это их **метод**, а **входы** и **выходы** могут быть **атрибутами**.
Класс — описание объектов со свойствами. **Атрибут** — свойство **класса** — принимает множество значений. **Блок** может быть меткой **роли ассоциации**.	Понятие **атрибут** больше относится к стрелкам **вход**–**выход**, а **класс** — к стрелкам **механизм**–**управление**. При существовании лишь одиночных **стрелок**, входящих в **блок**, можно интерпретировать его как **роль ассоциации**.

Труды СПИИРАН. 2006. Вып. 3. Т. 2. ISSN 2078-9181 (печ.), ISSN 2078-9599 (онлайн)
SPIIRAS Proceedings. 2006. Issue 3. V. 2. ISSN 2078-9181 (print), ISSN 2078-9599 (online)
www.proceedings.spiiras.nw.ru

Таблица 2 (продолжение)

1	2
Как стрелка **управления** часто берется набор отдельных положений и должностных инструкций. **Управленческие дуги** — информация, управляющая действиями функций.	**Управление** — это чаще всего внешняя сущность, уже определенная вне рамок рассматриваемой системы, а, следовательно, это **объект типа класса**.
Блоки на диаграмме представляют функции. Их названиями служат глаголы и глагольные обороты.	Название **блока** чаще всего является именем **метода класса**.
Выходные дуги — объекты, используемые и преобразуемые функцией. **Выходные дуги** — объекты, в которые преобразуются входы.	Чаще всего **выходные дуги** — это **атрибуты** какого-то класса или **имена классов**. При явном указании (пользователя) на то, что **выходы** являются преобразованными **входами**, — на **вход** и **выход** создается лишь один **класс** или **атрибут** класса.
Выход может стать средством (**механизмом**) или **управлением**.	Все правила, относящиеся к **механизмам**/**управлениям**, должны применяться для такой дуги **выхода** (при этом должно быть согласовано название сущности, в которую трансформируется **механизм**/**управление**).
Выход — произведенные функцией данные или объекты.	**Выход** — это или **тип метода**, или **объект (класс)**.
Стрелки — не поток событий. **Управления** — определяют условия, необходимые функции. **Механизм** — поддерживает выполнение функции.	Нельзя однозначно отнести все **стрелки** к типу Boolean. Стрелки **управления** не зависят от других сущностей (**классов**, **объектов**, **методов**). **Метод** должен быть явно связан с сущностью, в которую трансформируется **механизм**.
Вход — преобразуется функцией (или расходуется).	**Вход** — может являться как **параметром метода**, так и **атрибутом** или **объектом класса**.
Входы — условия, при одновременном выполнении которых осуществляется функция.	Возможна трансформация имени **входа** в имя **объекта** (**атрибута**), а **типом** будет являться Boolean.
Несколько **механизмов** для одного **блока** могут показывать скоординированную деятельность.	Часть **механизмов** может входить друг в друга как **атрибуты** с помощью **агрегации**.
Управления — правила выполнения, **механизмы** — правила, как должна выполняться функция.	У **механизма** больший приоритет при определении принадлежности **метода** определенному **классу**.
В **словаре данных** описывается полная иерархия объектов системы.	**Словарь данных** — основная сущность при проведении трансформации (построение **наследований** и проведение **ассоциаций**).
Вход часто отображает финансы, а **выход** и **механизм** — оборудование.	Такие **входы** скорее будут **атрибутами** класса с именем Финансы. А **выходы** — **объектами** типа **подклассов** класса «Оборудование».

Использование приведенных в таблице правил позволит сформулировать алгоритм автоматической трансформации. В программной реализации используются следующие правила:

1) все стрелки механизмов на диаграмме IDEF0 трансформируются в имена классов — владельцев методов, в которые трансформируются блоки;
2) все блоки — в имена методов классов (при этом они дублируются в каждом механизме-классе, который входит в блок);

Труды СПИИРАН. 2006. Вып. 3. Т. 2. ISSN 2078-9181 (печ.), ISSN 2078-9599 (онлайн)
SPIIRAS Proceedings. 2006. Issue 3. V. 2. ISSN 2078-9181 (print), ISSN 2078-9599 (online)
www.proceedings.spiiras.nw.ru

3) внешние входы (не образованные от выходов) — в классы, и являются атрибутами класса-механизма, входящего в блок;
4) внутренние входы (образованы от выходов) — в классы, типы параметров метода (блока, в который входят), связаны ассоциацией с классом-механизмом, входящим в блок;
5) все управления — в классы, связаны зависимостью с классом-механизмом, входящим в блок;
6) все выходы — в типы возвращаемого значения метода (на каждый выход создается свой метод в данном классе-механизме);
7) диаграмма A–0 не участвует в трансформации (слишком много лишних зависимостей).

При проведении программной реализации был использован ряд паттернов проектирования («компоновщик», «посетитель», «итератор», «фабричный метод»), который позволил построить гибкий каркас приложения, легко адаптируемый под разные требования к организации системы трансформации [5].

4. Пример

В качестве примера были взяты IDEF0 диаграммы самой системы трансформации (рис. 1). После проведения трансформации с помощью программы была получена диаграмма классов (рис. 2), которая сразу же доступна для редактирования в среде Together (далее можно сгенерировать программный код для последующей имплементации).

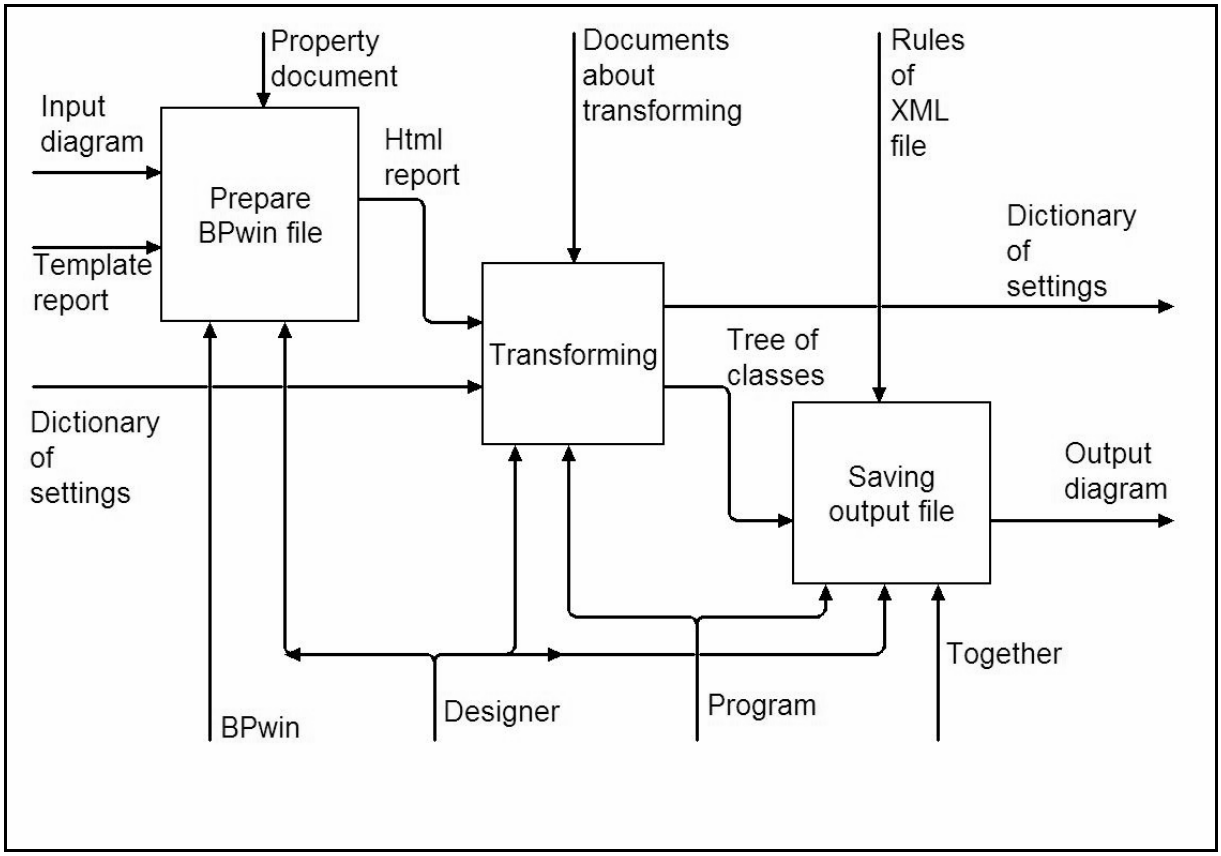

Рис. 1. Одна из исходных диаграмм IDEF0.

Труды СПИИРАН. 2006. Вып. 3. Т. 2. ISSN 2078-9181 (печ.), ISSN 2078-9599 (онлайн)
SPIIRAS Proceedings. 2006. Issue 3. V. 2. ISSN 2078-9181 (print), ISSN 2078-9599 (online)
www.proceedings.spiiras.nw.ru

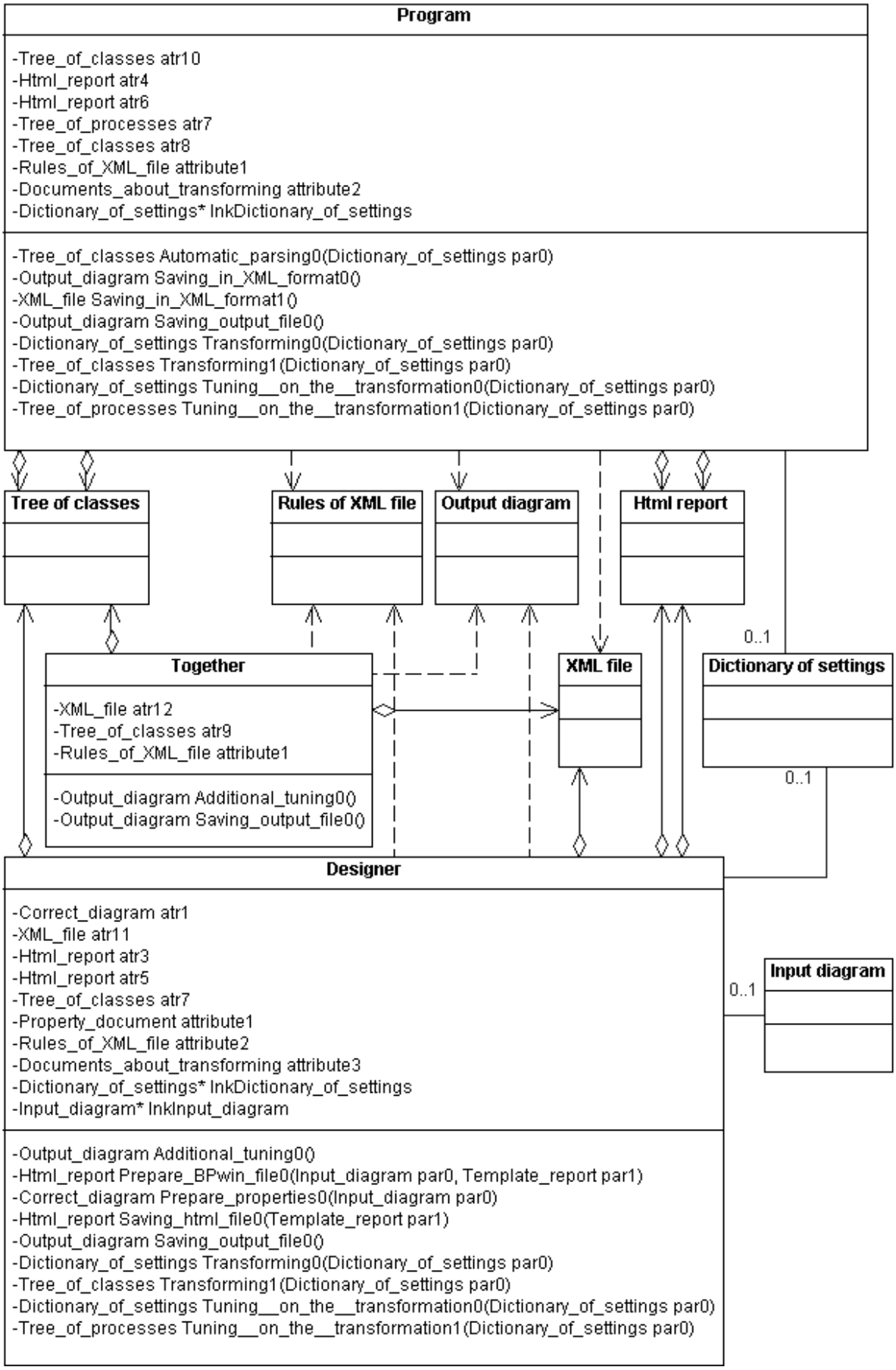

Рис. 2. Часть диаграмм классов системы трансформации.

Труды СПИИРАН. 2006. Вып. 3. Т. 2. ISSN 2078-9181 (печ.), ISSN 2078-9599 (онлайн)
SPIIRAS Proceedings. 2006. Issue 3. V. 2. ISSN 2078-9181 (print), ISSN 2078-9599 (online)
www.proceedings.spiiras.nw.ru

5. Заключение

Предложенная система трансформации позволяет решать задачи структурного проектирования и непосредственного получения программного кода. Особенность программной реализации состоит в том, что имеется возможность работы с моделями на любой стадии разработки (независимо от количества блоков на диаграмме), использовать данные о названиях и роли стрелок и названиях блоков, а также проводить автоматическую трансформацию. При этом можно открыть полученную модель классов непосредственно в окне Together, не используя промежуточные программные средства.

Для дальнейшего развития системы трансформации планируется применение онтологий в качестве описания семантики правил перевода диаграмм IDEF0 в диаграммы классов UML.

Литература

1. *Noran O.* UML vs IDEF: An ontology-oriented comparative study in view of business modelling [Электронный ресурс]: Paper accepted at the 6th International Conference on Enterprise Information Systems. 2003. 53 p. // <http: // www.cit.gu.edu.au/~noran> (по состоянию на 29.03.2006).
2. *Якобсон А., Буч Г., Рамбо Дж.* Унифицированный процесс разработки программного обеспечения. СПб.: Питер, 2002. 496 с.
3. *Маклаков С. В.* Моделирование бизнес-процессов с AllFusion. М.: Диалог-Мифи, 2003. 224 с.
4. *Орлов С. А*. Технологии разработки программного обеспечения. СПб.: Питер, 2003. 528 с.
5. *Гамма Э., Хелм Р., Джонсон Р., Влиссидес Дж.* Приемы объектно-ориентированного проектирования. Паттерны проектирования. СПб.: Питер, 2001. 368 с.