ИНСТРУМЕНТАЛЬНЫЕ СРЕДСТВА РАЗРАБОТКИ, РЕАЛИЗАЦИИ И СОПРОВОЖДЕНИЯ ГИПЕРТЕКСТОВЫХ ПРЕОБРАЗОВАТЕЛЕЙ

М. Ю. Колодин

Санкт-Петербургский институт информатики и автоматизации РАН

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

<myke@mail.ru>

УДК 004.91

Колодин М. Ю. **Инструментальные средства разработки, реализации и сопровождения гипертекстовых преобразователей** // Труды СПИИРАН. Вып. 7. — СПб.: Наука, 2008.

Аннотация. *Рассмотрены программные средства для преобразования информации между гипертекстовыми форматами на основе метаподхода применительно ко всем фазам жизненного цикла документов в указанных форматах.* — Библ. 9 назв.

UDC 004.91

Kolodin M. Y. **Instrumentation for design, implementation and maintenance of hypertext transformers** // SPIIRAS Proceedings. Issue 7. — SPb.: Nauka, 2008.

Abstract. *Software means for transformation between most hypertext formats are studied, basing on meta-approach and with application to all phases of life-cycle of documents in such formats.* — Bibl. 9 items.

При рассмотрении преобразований документов, прежде всего в гипертекстовых форматах [1], была отмечена перспективность *метаподхода* к таким преобразованиям [2, 3] и выбраны оптимальные *форматы* для хранения информации [1, 4]. Необходимо исследовать также и активную составляющую — *преобразователи* — и выбрать соответствующие ей форматы управляющих данных.

Прежде всего нужно отказаться от всех закрытых (и тем более коммерческих) форматов, если задачей является получение общеполезного инструмента или подхода. Основными исходными и промежуточными (рабочими) форматами следует считать XML [5] (включая производные и родственные, т. е. для которых существуют взаимно-однозначные преобразования), при необходимости — SGML и wiki-форматы, а выходными (конечными) — ещё и LaTeX [6, 7], PDF, PS и т. п. специализированные, но чётко стандартизованные. Во всех случаях чётко специфицированный формат важнее конкретной программной реализации: такие реализации могут выполняться разными авторами (фирмами) под различные платформы, а пользователь выберет для себя наиболее удобный для себя вариант, однако важно, чтобы форматы, в которых работают такие программы, полностью совпадали; только в таком случае будет обеспечена совместимость и переносимость. Осмысленные примеры текстов на некоторых рассматриваемых языках приведены в листинге 1.

Преобразования документов могут как выполняться непосредственно программами (как скомпилированными, так и интерпретируемыми), так и описываться текстами на специализированных языках (впоследствии, возможно, интерпретируемыми).

Для *компилируемых* вариантов в принципе некритичен исходный язык (напр., c, c++, java) и система программирования, на которых подготовлена программа; однако нужно стремиться к межплатформенной переносимости программ, а также к уменьшению разнообразия используемых одновременно в од-

ном проекте выразительных средств, поскольку это повышает степень повторного использования программного кода и инструментария, что позволяет извлечь больше пользы от метаподхода к реализации системы. В значительной степени возможность использования систем программирования определяется не собственно языком, а наличием специализированных библиотек для обработки документов в соответствующих форматах.

```
а.
Простой текст не содержит никакой разметки для визуального или смыслового
выделения частей.

б.
<html>
Для выполнения <u>HTML-разметки</u> нужно знать язык <code>HTML</code>,
предоставляющий <strike>удобные</strike> возможности
по <i>несложной</i> <b>визуальной</b> разметке текста,
а также для формирования <a href=http://www.anywhere.else
target=etc>гипертекстовых ссылок</a>.
</html>

в.
'''Wiki'''-''разметка'' позволяет быстро и просто выделить
* ''визуально'' отличающиеся [[части|часть]] текста,
* а также некоторые ''смысловые'' единицы (напр., [[ссылки]])

г.
Документ в~формате \LaTeX~--- простой текст с~соответствующей разметкой;
представляет собой программу для~интерпретатора \TeX а;
изначально созданный для~поддержки набора сложной математики, напр.,
\(\sin^2 x + \cos^2 x = 1\),
сейчас широко используется и за~пределами этой области.

д.
<text><a href="voc/XML">XML</a>-<a href="voc/Document">документ</a>
позволяет выделить все <a href="voc/Sense">смысловые</a> единицы, однако,
как правило, не занимается визуальным форматированием текста;
для такового применяются <a href="voc/XSLT">XSLT</a>-преобразователи.
</text>
```

Листинг 1. Примеры форматирования текстов.

Для *интерпретирующих* систем возможности шире, поскольку можно динамически порождать нужные программные фрагменты и исполнять их в нужных контекстах. При этом языками могут быть как языки общего назначе-

Труды СПИИРАН. 2008. Вып. 7. ISSN 2078-9181 (печ.), ISSN 2078-9599 (онлайн)
SPIIRAS Proceedings. 2008. Issue 7. ISSN 2078-9181 (print), ISSN 2078-9599 (online)
www.proceedings.spiiras.nw.ru

ния (perl, tcl, python, lisp [8], forth [9] и т. п., вплоть до языков веб-систем, программы на которых исполняются как на стороне сервера (напр., php, parser), так и на стороне клиента (напр., javascript); уже есть технологии, построенные именно на динамическом создании как текстов, так и программ (например, web 2.0, ajax)), так и специализированные языки; более того, как в TeXe, сам текст документа может быть программой, равно как и правила его обработки. Роль библиотек здесь тоже велика, но дополнительную мощь даёт многоуровневость представления текстов и их преобразователей, особенно при правильной реализации свёрток.

С точки зрения метаподхода схема с компиляцией является частным случаем схемы с интерпретацией, поскольку ограничивает возможность варьирования составляющих процесса; преимуществом её является более высокая производительность получаемого продукта и отчуждаемость его от среды разработки. Желательно иметь возможность непосредственного применения гибких средств выбора всех элементов (в т. ч. и исполняемой функции); здесь наиболее удобен язык forth; в языке lisp не все диалекты поддерживают такую возможность непосредственно (например, такое ограничение действует в common lisp), однако всегда есть возможность обойти это ограничение (например, сформировать функцию как список и затем выполнить её по eval). Остальные упомянутые языки такой возможности напрямую не имеют, но её можно смоделировать различными способами. Примеры преобразователей на языках forth и lisp приведены в листинге 2.

Частным случаем целенаправленной разработки является гибкое управление сборкой системы. Классическими способами сборки являются командные файлы и схемы удовлетворения статических зависимостей компонентов (типа make). Теперь их можно объединить с логическими схемами (от таблиц решений до полностью динамического управления сборками в соответствии с меняющимися условиями в системе); соответственно значительно больше внимания нужно уделять контролю за процессом и обеспечению его сходимости и непротиворечивости; эти задачи становятся самостоятельными, их также надо планировать, описывать, управлять ими.

В реальных системах нет необходимости иметь все варианты представления документов; достаточно *возможности* их оперативного получения. Для этого нужно иметь заранее определённую или формируемую динамически *схему* получения документа в нужном формате, что включает указание исходного и целевого форматов, контексты и параметры преобразований, причём собственно преобразователи могут быть часто выбраны (или сформированы) по мере необходимости. При наличии *метрик* для преобразователей, в т. ч. информации о затратах на получение преобразователя и выполнение преобразований, можно выбрать оптимальную в конкретном случае схему преобразований и требуемый преобразователь, возможно, с их построением. Языки для описания таких схем также подчиняются указанным требованиям; чем более они унифицированы (возможно, с использованием контекстов и параметров), тем выше степень свёртки и отдача метаподхода.

В системах, выполняющих указанные преобразования, могут использоваться различные оценки и метрики. Каждая из них численно является результатом преобразования исходной системы специальным преобразователем при определённых условиях. Программно такие оценки и метрики могут быть как изначально заданными, так и получаемыми динамически в процессе развития системы. Метрики здесь рассматриваются как класс оценок с наложенными до-

полнительными требованиями, в частности соблюдения «неравенства треугольника» применительно к соответствующей области. Важно, что оценки включены в общую систему преобразований и ничем не отличаются от прочих программных подсистем. Оценки могут быть самого разного рода, в частности, это оценки сложности системы (по количеству элементов вообще, по количеству различных элементов, по количеству типов подсистем, по связности подсистем и элементов, оценки размера, и т. п.), а также оценки, специфические для данной конкретной системы. Оценки могут быть как статическими (типа вышеуказанных), так и динамическими (например, оценки затрат ресурсов (время процессора, занятая память (в среднем и максимально)), полученными при работе системы и т. п.; при этом статичность либо динамичность оценок является относительной. Важно иметь возможность выбирать и применять нужные оценки или наборы оценок в каждом конкретном случае, в т. ч. формировать их динамически, в т. ч. и на основе выполнения других предварительных оценок, что формирует в процессе работы системы соответствующее пространство оценок с параметрами.

```
а. простой преобразователь

FORTH
PREPARE-DATA SET-TRANSFORMER EXECUTE

LISP
(TRANSFORM (PREPARE-DATA))

б. сложный (обобщённый) преобразователь

FORTH
SET-DATA-CONTEXT PREPARE-DATA
      PREPARE-CONTEXT
            PREPARE-TRANSFORMER
                  EXECUTE

или
PREPARE-CONTEXT
      PREPARE-DATA
            PREPARE-TRANSFORMER
                  EXECUTE

LISP
((PREPARE-TRANSFORMER) (PREPARE-CONTEXT) (PREPARE-DATA))
```

Листинг 2. Реализации преобразователей на псевдокоде.

Полезно ввести некоторые термины. Преобразователи могут быть *внешними* по отношению к изучаемой системе, если они реализованы отдельно от неё, имеют более общее применение и собственный язык их реализации отли-

Труды СПИИРАН. 2008. Вып. 7. ISSN 2078-9181 (печ.), ISSN 2078-9599 (онлайн)
SPIIRAS Proceedings. 2008. Issue 7. ISSN 2078-9181 (print), ISSN 2078-9599 (online)
www.proceedings.spiiras.nw.ru

чается от используемого в данной системе (например, msxml или saxon для преобразований между XML или из XML в иные форматы), либо *внутренними*, если они входят в систему, языки реализации совпадают с системными или поддерживаются системой (например, wiki-текст, реализуемый собственным интерпретатором; задание на обработку документов, записанное в виде правил, интерпретируемых данной системой; набор правил для таблицы решений со своим интерпретатором). Сложность (в любом конструктивном смысле) программной системы, реализующей внешний преобразователь, для собственно исследуемой системы преобразования документов в общем случае не важна; внешний обработчик рассматривается атомарно. По возможности и внутренние обработчики нужно делать максимально сворачиваемыми, желательно — атомарными; это повышает эффективность метаподхода. В каждый момент времени любой объект (в т. ч. преобразователь) *активен*, если он исполняется над каким-либо иным объектом, либо *пассивен*, если он подвергается обработке со стороны других преобразователей (формирование, преобразование, параметризация, задание контекста, уничтожение). Любое действие рассматривается как разовое преобразование входной информации (в частности, документа) в выходную (обе — пассивные сущности), выполняемое преобразователем (программой, т. е. активной сущностью) при заданных параметрах в определённом контексте (оба — пассивны); эта схема близка к используемой в классическом системном анализе и может быть выражена в его терминах.

Для каждого требуемого преобразования документов (или цепочки преобразований) определяются исходный и результирующий форматы документов. Если для них есть готовый преобразователь (внешний или внутренний), он выполняется; при этом может возникнуть многоуровневая схема выполнения преобразователя в зависимости от того, как он реализован (особенно для интерпретирующих систем). Если преобразователя нет, то по схемам (которые в данный момент являются программами) интерпретатором выбирается или формируется новый преобразователь, который затем исполняется. Если нет преобразователя непосредственно между данными форматами, определяем, можно ли сформировать цепочку преобразователей и какие при этом есть ограничения, требования, параметры; если проверки успешны, формируем цепочку и для каждого этапа выполняем указанные выше шаги; промежуточные документы и преобразователи, возможно, впоследствии удаляются; при этом учитываются «равносильные» форматы (с известными взаимно-однозначными преобразователями, например, XML—YAML в определённых приложениях); все внутренние форматы документов должны быть строгими (однозначно интерпретируемыми), исходные (пользовательские) могут быть нестрогими, их разбор происходит с применением эвристик, результирующие форматы зависят от вида дальнейшего использования документов. Чем больше шагов удаётся унифицировать, свернуть, выполнить по общим правилам (возможно, с различными параметрами или в различных контекстах), тем эффективнее реализация данного подхода. Кратко указанная схема преобразований может быть выражена на forth-псевдокоде так, как указано в листинге 3.

Данными для передачи и преобразования документов могут быть как собственно документы, так и указания на способ их получения; более того, каждая из указанных выше составляющих может быть и часто является вычислимой на предыдущем этапе преобразований. Соответственно в общем случае имеем набор преобразований между всеми допустимыми форматами документов, прежде всего для представимых в виде простого текста с гипертекстовой раз-

меткой (также в форме простого текста с использованием тегов, отступов и т. п. форматирующих конструкций). Система, включающая такой набор преобразователей, может иметь практическую ценность как самостоятельно, так и в составе объемлющей системы программирования, документооборота или иной системы, основанной на текстах и их обработке.

```
PREPARE-DATA ( -- x )
GET-SOURCE-FORMAT GET-TARGET-FORMAT ( -- 2format )
?CONVERTIBLE ( 2format -- prog TRUE / 2format FALSE ) IF
      DO-CONVERT ( x1 prog -- x2 )
      ELSE
            BUILD-CONVERTER ( 2format -- prog TRUE / FALSE ) IF
                  DO-CONVERT ( x1 prog -- x2 )
                  ELSE
                        ERROR:Impossible ( -- ; exception )
                  THEN
      THEN
```

Листинг 3. Схема преобразования на forth-псевдокоде.

Представляется важным исследовать в дальнейшем *метрики* для таких систем и *оценки эффективности* преобразований, особенно для *совместных* преобразований групп (систем) документов.

Литература

1. *Колодин М. Ю.* Сравнительный анализ гипертекстовых форматов и преобразований документов между ними [Электронный ресурс] // <http://ru.wikipedia.org> «Википедия:Википконференция 2007/Программа/Доклады/Колодин М.Ю. Сравнительный анализ гипертекстовых форматов и преобразований документов между ними» (по состоянию на 29.02.2008)
2. *Колодин М. Ю.* Мета-технология: назначение и реализация // Информационные технологии и интеллектуальные методы. СПб: СПИИРАН, 1995. С. 83–86.
3. *Колодин М. Ю.* Произвольно-уровневые гипертекстовые системы // Сб. трудов «Современные проблемы информатизации в системах моделирования, программирования и телекоммуникациях». Вып. 9 (по итогам IX Международной открытой научн. конф.). Воронеж: Научная книга, 2004. С. 358–359.
4. *Колодин М. Ю.* Межформатные мета-преобразования гипертекста // Труды СПИИРАН. Вып. 6. СПб.: Наука, 2008. С. 168–170.
5. *Валиков А. Н.* Технология XSLT. СПб: БХВ-Петербург, 2002. 544 с.
6. *Кнут Д. Е.* Всё про TeX. Пер. с англ. Протвино: АО RD TeX, 1993. 592 с.
7. *Львовский С. М.* Набор и вёрстка в пакете LaTeX. М.: Космосинформ, 1995. 374 с.
8. *Хювёнен Э., Сеппянен Й.* Мир лиспа. В 2-х томах (447 с., 319 с.). М.: Мир, 1990.
9. *Баранов С.Н., Ноздрунов Н.Р.* Язык Форт и его реализации. Л.: Машиностроение, 1988. 157 с.