СИНТАКСИЧЕСКИЕ И СЕМАНТИЧЕСКИЕ ОСОБЕННОСТИ МЕТАСИСТЕМ

Колодин М.Ю.

УДК 004.91

Колодин М.Ю. **Синтаксические и семантические особенности метасистем.**
Аннотация. Метаподход позволяет рассматривать информационные системы, в т.ч. языки и системы программирования, а также данные в различных форматах как многоуровневые развиваемые системы и строить для них соответствующие преобразователи. В статье рассматриваются традиционные и специальные языки программирования и представления данных с точки зрения метаподхода, языки делятся на классы с внутренней либо с внешней метафункциональностью, показываются примеры записи и преобразований между языками, указываются некоторые сложности, возникающие при этом, и пути их преодоления, даются оценки эффективности применения различных языков, делаются выводы о полезности использования рассмотренных языков и подхода в целом для решения типовых задач и представления информации.
Ключевые слова: системы, метасистемы, преобразования, метапрограммы, метаданные, языки программирования, представление данных.

Kolodin M.Y. **Syntactic and Semantic Particularities of Metasystems.**
Abstract. Metaapproach allows to study information systems, including programming languages and systems, as well as data in different formats, as multilevel developed systems and to construct corresponding converters for them. The paper studies traditional and special programming languages and data representation languages viewed by metaapproach, where languages are divided to classes with internal and external metafunctionality. Examples of code and conversions between languages are shown, some conversion problems and ways of their solving are noted, estimations of efficiency of application of different languages are given, and conclusions on usefulness of discussed languages and of the entire approach for solving typical problems and information presentation are stated.
Keywords: systems, metasystems, conversions, metaprograms, metadata, programming languages, data representation.

1. Введение. При метаподходе информационные системы рассматриваются как многоуровневые развиваемые системы; при этом эффективность системы тем выше, чем больше подсистем удаётся свернуть, т. е. выразить в терминах порождающей системы с необходимой параметризацией [1]. Различные языки программирования и языки представления данных предоставляют различные возможности для реализации метасистем. Необходимо рассмотреть наиболее важные типичные языки из множества ныне существующих с тем, чтобы выбрать из них наиболее подходящие для практического применения, а также определить требования к таким инструментальным средствам для конструирования новых языковых средств. Прежде всего, целый ряд языков изначально имеют встроенные метасредства, например, Forth [2], Lisp [3], XML [4], макроязыки [5]; будем считать,

Труды СПИИРАН. 2009. Вып. 9. ISSN 2078-9181 (печ.), ISSN 2078-9599 (онлайн)
SPIIRAS Proceedings. 2009. Issue 9. ISSN 2078-9181 (print), ISSN 2078-9599 (online)
www.proceedings.spiiras.nw.ru

что у них внутренняя метафункциональность. В программах на этих языках можно динамически построить новую программу (или текст) и выполнить (обработать) её (его), при этом использовать результаты работы такой программы далее в главной программе либо иным образом распорядиться полученным программным кодом, например, для выполнения оценок кода по какому-либо критерию. Для большинства компилируемых языков таких удобных качеств не предусмотрено, но их можно рассматривать в контексте полной системы или технологической цепочки программирования, в которую мы можем сами внести необходимые элементы обратной связи, таким образом достроив исходный набор инструментальных элементов до метасистемы; в таком случае имеем внешнюю метафункциональность. Будем рассматривать только языки с линейной записью текста и обработкой считываемого текста построчно.

2. Языки и системы с внутренней метафункциональностью. Для полноценной реализации метафункциональности языковое средство должно содержать возможности динамического порождения выполняемого программного кода либо (для языков описания данных) — возможности описания новых языков на основе данного.

В метасистемах соответствие синтаксиса и семантики не является постоянным; более того, наиболее важные результаты получаются именно при смене контекста, когда тот же самый текст имеет другой значение и при исполнении даёт иной результат. Например, в определённом контексте вместо типичного исполнения программы получаем её версию, скомпилированную для другой целевой платформу, либо качественную или количественную оценку программы, полученную на основании некоторых правил-метрик. Это заставляет выделить понятие контекста в отдельную важную сущность при рассмотрении метасистем.

Прежде всего следует рассмотреть интерпретируемые языки. Наиболее богатые возможности предоставляют языки Forth (форт), Lisp (лисп) и аналогичные им [6].

Важно, что возможность простого порождения нового программного кода является необходимой, но не достаточной для наличия метафункциональности в языке. Требуется также возможность проведения некоторого набора операций над полученной программой не только как текстом, но и как над активным объектом. В частности, нужно иметь возможность получения различных оценок, в т. ч. метрик, для полученной программы, причём как статических, на основе внешнего анализа текста, так и динамических, получаемых при управ-

Труды СПИИРАН. 2009. Вып. 9. ISSN 2078-9181 (печ.), ISSN 2078-9599 (онлайн)
SPIIRAS Proceedings. 2009. Issue 9. ISSN 2078-9181 (print), ISSN 2078-9599 (online)
www.proceedings.spiiras.nw.ru

ляемом выполнении программы в определённом окружении и контексте. Для рассматриваемых языков это можно выразить следующим образом: пусть есть программа («слово») на форте `FOO`, тогда

```
FOO
```

даст исполнение этого слова,

```
' FOO
```

даст адрес этого слова для последующей обработки,

```
' FOO EXECUTE
```

снова исполнение слова по его адресу,

```
' FOO ANY-TEST
```

применит некое иное действие к слову как к отдельной сущности, например, какой-либо тест, оценивание, использует его как параметр для других вложенных операций и т. п.

Аналогичных результатов можно добиться в лиспе через аппарат «`'`», «`QUOTE`» и «`EVAL`»; но полностью реализовать там метавозможности удобно только если разрешен вызов функций в позиции первого параметра списка, т. е. не только как

```
(DOFUNC params)
```

но и как

```
((SELECT-FUNC params1) params2)
```

Важное направление в форте — целевая компиляция [1], т. е. трансляция программы в специальном контексте с получением целевой системы для платформы, отличной от инструментальной, либо с иными свойствами. Типичный пример — сборка форт-системы в ней же самой с получением новой системы с новыми заданными свойствами.

Другие интерпретируемые языки, например, Python [7], Tcl, Lua, Haskell, также интересны, хотя и менее проработаны пока в отношении метафункциональности по сравнению с «классическими» фортом и лиспом. Для языка Python явно указываются «метасвойства» как возможность динамически определить тип объекта, получить о нём сведения и т. п. Это свойственно не только интерпретируемым языкам, но и некоторым языкам со статической компиляцией, но динамическим присваиванием значений из некоторого набора классов (C++).

Однако и для языков менее высокого уровня возможно рассмотрение их как метаязыков. Важным классом полезных для нас классических инструментальных средств являются макроязыки [5]. Действительно, при возможности многократного прохода по тексту макропрограммы можно получать существенные преобразования её с получением необходимых свойств. В таких языковых системах макропроцессор многократно обрабатывает текстовую цепочку до тех пор,

Труды СПИИРАН. 2009. Вып. 9. ISSN 2078-9181 (печ.), ISSN 2078-9599 (онлайн)
SPIIRAS Proceedings. 2009. Issue 9. ISSN 2078-9181 (print), ISSN 2078-9599 (online)
www.proceedings.spiiras.nw.ru

пока не будет выполнено некоторое заданное наперёд условие, например, до тех пор, пока текст не перестанет изменяться, или пока в нём не будут устранены все возможные точки макроподстановки, или пока не возникнет цикл макрозамен (например, если получены два текста, при макроподстановках переходящие друг в друга, т. е. косвенная рекурсия), или пока не возникнет бесконечная прямая рекурсия (проход макропроцессора порождает код, который снова включает вызов того же самого макрорасширения с теми же параметрами); последние два случая весьма трудны и в типичных макропроцессорах обычно не реализуются, что может приводить к ошибкам выполнения, зацикливанию, переполнению стека.

Интересные результаты можно получить для формальных языков, например, грамматик, регулярных выражений [8]. Можно ставить вопрос об автоматическом построении и последующей полуавтоматической или даже полностью автоматической (для некоторых классов языков, которые ещё предстоит формально исследовать) реализации языков с заданными свойствами на основе определённых требований. Полученные результаты можно применить для выполнения оптимизирующих преобразований программ [9] и упрощения записи грамматик, приведения их к определённым формам, обладающим требуемыми свойствами [8].

3. Языки и системы с внешней метафункциональностью. Для реализации метафункциональности в таких инструментах необходимо построить вокруг них систему более высокого порядка, управляющую работой данной.

Например, исходно языки C и C++ не обладают нужной метафункциональностью; наличие в них простейшего макропроцессора (`#define`) даёт только управление текстовыми подстановками (возможно, с параметрами) и управления порядком сборки и выбором частей программы для неё. Но нетрудно написать C-программу, при своём исполнении печатающую другую C-программу (или программу на другом языке); однако (а) для этого нужно выполнить несколько шагов обработки, и (б) при этом в процесс добавляются C-компилятор и операционная система, каковые таким образом тоже нужно включать в состав метасистемы.

Дополнительный пример внешней метафункциональности — системы сборки программных продуктов [10] и документации, а также системы управления версиями; они различны по форматам и протоколам, но близки по основным технологическим принципам, что позволяет организовать метаобработку.

Труды СПИИРАН. 2009. Вып. 9. ISSN 2078-9181 (печ.), ISSN 2078-9599 (онлайн)
SPIIRAS Proceedings. 2009. Issue 9. ISSN 2078-9181 (print), ISSN 2078-9599 (online)
www.proceedings.spiiras.nw.ru

В более общем случае нужно предусмотреть возможность управления транслирующими и управляющими системами, которые обычно являются «чёрными ящиками».

После выделения всех необходимых составляющих процесса, пусть даже как «чёрных ящиков», система в целом может рассматриваться как метасистема с применением многих указанных ранее инструментов.

4. Преобразования между языками по отношению к метафункциональности. Прежде всего, отметим, что практически все языки программирования универсальны в смысле эквивалентности машине Тьюринга (или близкого ей соответствия), и таким образом теоретически программы на них могут самостоятельно построить все необходимые метавозможности самостоятельно. В реальности это, как правило, не так, и особенности языков и ограничения их реализаций ограничивают такую возможность. Кроме того, в большинстве случаев строить такую реализацию одного средства на другом практически неэффективно. Тем не менее, строго говоря, почти не существует реализаций чисто компилируемых и чисто интерпретируемых языков, поскольку даже классические компиляторы (например, фортрана) оставляют часть исходного кода нераскрытой до стадии выполнения (это так называемая остаточная интерпретация; например, для форматных строк в операторах печати), а большинство практически используемых интерпретаторов предварительно компилируют исходный текст в промежуточный код (при этом происходит проверка синтаксической правильности программы). Замечательным образцом в этом смысле является Java [11], где стандартизован не только внешний синтаксис, но и промежуточное представление (байт-код), что и способствует высокой переносимости программ между платформами; это, а также jit-компиляция («just-in-time») являются примерами несложной, малоуровневой, но практически очень полезной метафункциональности с генерацией кода для целевой платформы.

Многие текстовые преобразования удобно выполнять на языках, специально ориентированных на это, например, на языке Perl [12]. При этом сам Perl, конечно, к рекомендуемым метаязыкам не относится в силу сложности реализации и нерегулярности синтаксиса. Впрочем, возможность частичного изменения синтаксиса языка при подключении пакетов расширений (например, для Tcl при подключении Tk) является полезной.

Классической задачей для языков преобразований является задача «интроспекции», т. е. печати программой самой себя [13]. Помимо

Труды СПИИРАН. 2009. Вып. 9. ISSN 2078-9181 (печ.), ISSN 2078-9599 (онлайн)
SPIIRAS Proceedings. 2009. Issue 9. ISSN 2078-9181 (print), ISSN 2078-9599 (online)
www.proceedings.spiiras.nw.ru

игрового характера задачи, она хорошо показывает некоторые метасвойства языка. Написать такую программу можно практически на любом языке с синтаксисом выше ассемблерного, но удобство результата сильно различается: от простого

```
SOURCE TYPE
```

на форте до многостраничных выкладок на фортране. Одна из проблем, хорошо иллюстрируемая этим примером, это задача воспроизведения спецсимволов по именам или кодам в генерируемом тексте, особенно в случае нескольких шагов генерации. Пусть нужно напечатать символ с кодом «1234» в уникоде; имеем обозначение типа «&u1224;» (без внешних кавычек), при этом символы «&» и «;» становятся выделенными, со смыслом, отличным от обычного, и для их выдачи в выходной поток нужно опять же применять их кодовые или именные обозначения. Далее, пусть нужно написать программу, печатающую программу, в свою очередь печатающую эти символы; тогда непонятно в общем случае, как записать такие коды, поскольку они могут быть раскрыты при первом же проходе. Есть несколько неуниверсальных решений, применяемых в разных системах, от «&uu1234;» для обозначения «отложенного» раскрытия до сборки таких кодов посимвольно (многократными операциями конкатенации, что зачастую делает текст программы почти нечитаемым). Эти задачи, увы, типичны для всех известных многопроходных текстогенерирующих систем.

5. Построение общих и специальных метаязыков. Исторически очевидно, что процесс поиска и порождения новых языковых средств в т. ч. с метавозможностями идёт постоянно, что означает как то, что имеется объективная необходимость в них, так и то, что до сих пор универсальное средство не найдено.

Прежде всего, постоянно повышается уровень языков программирования; при этом пользователь (программист) ориентируется на главную задачу, а адаптация системы к конкретному окружению происходит автоматически, в большинстве случаев программист даже не видит полностью весь сгенерированный код (имеющий объём в многие десятки тысяч строк). При запуске программы в реальном окружении также происходит её настройка, в т. ч. на используемое оборудование; это тоже содержит элементы метафункциональности, хотя и неявно.

Кроме того, постоянно повышается уровень всемирно используемых инструментариев для построения сетевых служб, как в формате социальных сетей, так и в формате предоставления программного

Труды СПИИРАН. 2009. Вып. 9. ISSN 2078-9181 (печ.), ISSN 2078-9599 (онлайн)
SPIIRAS Proceedings. 2009. Issue 9. ISSN 2078-9181 (print), ISSN 2078-9599 (online)
www.proceedings.spiiras.nw.ru

обеспечения как сервисов (концепция «SaaS», «Software as a Service»). С одной стороны, они всё больше скрывают от рядовых пользователей подробности реализации, но в то же время они дают возможность порождения нужных сайтов с весьма большими возможностями (от простых наборов страничек на www.narod.ru до богатого встроенным функционалом www.ucoz.ru). Здесь важно, что пользователю предоставляется доступ к коду (на HTML, в bbcode или как-либо иначе) сайта и к его API, а значит, недалеко время, когда можно будет породить такими системами им равнофункциональные, что может привести к созданию сетевых метасистем планетарного масштаба. Появление систем типа «Google wave» — ещё один важный пример такого движения.

Языки представления данных и знаний постоянно развиваются, в основном, в направлении повышения удобства набора человеком, универсализации (в т. ч. объектной ориентированности, организованности). Например, выполнялось переписывание издательской системы TeX [14] в терминах ООП (на Java); причиной этого является то, что сам TeX Д. Кнута написан сложно, неудобен для анализа и расширения, и необходимо иметь его регулярную версию; тогда его можно будет более активно использовать в системах с многократным многоцелевым использованием текста, что сейчас крайне актуально; такие системы также движутся в своём развитии в сторону наращивания метасвойств.

Ещё один пример — современные системы сетевого и печатного набора. Стремление одновременно к упрощению набора и к повышению выразительных и структурных возможностей текста ведёт к появлению дополнительных преобразователей, появляются различные версии языков набора, примерно равные по возможностям, но заметно различающиеся по синтаксису. Предпочтение теоретически нужно отдавать более структурно совершенным системам, но они не всегда удобны и компактны в наборе; поэтому нужен компромисс. Как правило, наиболее удобны (особенно для автоматической обработки) системы с простым текстовым представлением как для текста, так и для разметки. Например,

простой *курсивный* ***полужирный*** текст

может быть представлен в TeXe как

```
простой {\it курсивный {\bf полу}}{\bf жирный} текст
```

или

```
простой \textit{курсивный
\textbf{полу}}\textbf{жирный} текст
```

в вики-формате это будет выглядеть примерно как

Труды СПИИРАН. 2009. Вып. 9. ISSN 2078-9181 (печ.), ISSN 2078-9599 (онлайн)
SPIIRAS Proceedings. 2009. Issue 9. ISSN 2078-9181 (print), ISSN 2078-9599 (online)
www.proceedings.spiiras.nw.ru

```
простой ''курсивный '''полу''жирный''' текст
```

в некотором XML как

```
<text>простой <font type='italics'>курсивный <font
type='bold'>полу</font></font><font
type='bold'>жирный</font> текст</text>
```

а в некоторой другой системе набора как

```
простой {font type=it: курсивный {font type=bf:
полу}}{font type=bf: жирный} текст
```

или как

```
просто {it: курсивный {bf: полу}}{bf: жирный} текст
```

Ясно, что у каждого варианта есть свои достоинства и недостатки по наличию зарезервированных (специальным образом обрабатываемых) и ограничительных символов, по читаемости, по компактности, по удобству автоматической обработки (в т. ч. по наличию универсальных обработчиков, как это есть в случае XML), по наличию или возможности настройки программ-редакторов для удобного набора текстов в соответствующих форматах. Эти форматы функционально близки, преобразователи текстов между ними могут быть созданы при необходимости; но для слабоструктурированных форматов сложность написания преобразователя приближается к сложности полной обработки исходного текста (например, в случае TeXa, основанного на макроопределениях с параметрами и сложных вложенных подстановках).

Ещё большие сложности возникают при наборе математических формул: набор в стиле MathML универсален, строг, имеет хорошую поддержку автоматическими обработчиками, но чрезмерно многословен и неудобочитаем; с другой стороны, набор в TeXe недостаточно структурен, в общем случае для его анализа необходимо полностью выполнить интерпретатором TeXa программу-формулу. Здесь возможен компромисс, по крайней мере на этапе ввода текста человеком: среди хорошо структурированного текста явно выделяются «островки» неструктурированного (но с явными границами вложения), например,

```
Формула для вычисления корней {it: квадратного
уравнения}:
\[
x_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}
\]
```

где блок между «`\[`» и «`\]`» (обязательно на отдельных строках) обрабатывается специальным образом (последующим вызовом внешнего интерпертатора TeXa); однако вложения таких блоков друг в друга по-прежнему сложны.

Труды СПИИРАН. 2009. Вып. 9. ISSN 2078-9181 (печ.), ISSN 2078-9599 (онлайн)
SPIIRAS Proceedings. 2009. Issue 9. ISSN 2078-9181 (print), ISSN 2078-9599 (online)
www.proceedings.spiiras.nw.ru

6. Выводы. Новые языки с внутренней метафункциональностью появляются, в основном, в академической среде; в промышленном программировании преобладают реализации элементов внешней метафункциональности. Но существенного изменения возможностей пока не отмечено, изменения в большей степени касаются синтаксиса, нежели семантики, и классический стандартный форт полностью охватывает все наиболее существенные метавозможности. В то же время это направление остаётся важным для полных метасистем, где построение новых языков различного уровня является типовым действием.

Весьма эффективно применение стандартных обработчиков для полностью формализованных стандартизованных языков; однако на этапе ввода текста человеком нужно стараться упростить форматы и облегчить правила ввода информации; при последующем преобразовании во внутренний формат нужно обязательно проверять правильность исходного текста.

В силу очевидной тенденции к метаструктуризации информации, усилению её многоуровневости изучение и применение данного подхода весьма актуально.

Литература

1. *Колодин М.Ю.* Мета-технология: Назначение и реализация. // Информационные технологии и интеллектуальные методы. СПб: СПИИРАН, 1995. С.83-86.
2. *Баранов С.Н., Ноздрунов Н.Р.* Язык форт и его реализации. Л.: Машиностроение, ЛО, 1988. 157 с.
3. *Хювёнен Э., Сеппянен Й.* Мир лиспа. В 2 т. М.: Мир, 1990. 447 с. и 319 с.
4. *Валиков А.Н.* Технология XSLT. СПб: БХВ-Петербург, 2002. 544 с.
5. *Браун П.* Макропроцессоры и мобильность программного обеспечения. М.: Мир, 1977. 256 с.
6. *Колодин М.Ю.* Инструментальные средства разработки, реализации и сопровождения гипертекстовых преобразователей. // Труды СПИИРАН. Вып. 7. СПб: Наука, 2008. С.64-69.
7. *Сузи Р.А.* Язык программирования Python. Курс лекций. URL: http://www.hunger.ru, 2005. 206 с.
8. *Рейуорд-Смит В.Дж.* Теория формальных языков. Вводный курс. М.: Радио и связь, 1988. 128 с.
9. *Касьянов В.П.* Оптимизирующие преобразования программ. М.: Наука, физматлит, 1988. 336 с.
10. *Колодин М.Ю.* Метауправление сборкой программных продуктов. // Научно-практическая конференция студентов, аспирантов, молодых учёных и специалистов «Интегрированные модели, мягкие вычисления, вероятностные системы и комплексы программ в искусственном интеллекте» («ИММВИИ-2009»). Научные доклады, том 1, с. 140-146. Г.Коломна, 26-27.05.2009. М.: Физматлит, 2009. С. 140-146.

Труды СПИИРАН. 2009. Вып. 9. ISSN 2078-9181 (печ.), ISSN 2078-9599 (онлайн)
SPIIRAS Proceedings. 2009. Issue 9. ISSN 2078-9181 (print), ISSN 2078-9599 (online)
www.proceedings.spiiras.nw.ru

11. *Вязовик Н.А.* Программирование на Java. М.: Интуит.ру, 2003. 592 с.
12. *Кристиансен Т., Торкингтон Н.* Perl: библиотека программиста. СПб: Питер, 2001. 736 с.
13. *Уэзерелл Ч.* Этюды для программистов. М.: Мир, 1982. 288 с.
14. *Гуссенс М., Миттельбах Ф., Самарин А.* Путеводитель по пакету LaTeX и его расширению LaTeX2e. М.: Мир, 1999. 606 с.

Колодин Михаил Юрьевич — научный сотрудник исследовательской группы информационных технологий в образовании (ИГИТО) Учреждения Российской академии наук Санкт-Петербургский институт информатики и автоматизации РАН (СПИИРАН). Область научных интересов: метасистемы, свободное программное обеспечение, интернет, образовательные технологии. Число научных публикаций — 72. myke@iias.spb.su, www.myke.spb.ru; СПИИРАН, 14 линия, 39, Санкт-Петербург, 199178, Россия; раб. тел. +7(812)328-0382. Научный руководитель — к. ф.-м. н., доцент, в.н.с. А.Л. Тулупьев.

Kolodin Mikhail Yurievich — researcher, Research Group for Information Technologies in Education (RGITE) of St.Petersburg Institute for Informatics and Automation of the Russian Academy of Sciences (SPIIRAS). Research interests: metasystems, free and open source software, internet, educational technologies. The number of publications — 72. myke@iias.spb.su, www.myke.spb.ru; SPIIRAS, 39, 14 Line, St.Petersburg, 199178, Russia; office phone +7(812)328-0382. Scientific supervisor — Ph.D., associate professor, leading researcher A.L. Tulupiev.

Рекомендовано группой междисциплинарных проблем информатики СПИИРАН; науч. рук. к. ф.-м. н., доцент, в.н.с. А.Л. Тулупьев.
Статья поступила в редакцию 24.06.2009.

Труды СПИИРАН. 2009. Вып. 9. ISSN 2078-9181 (печ.), ISSN 2078-9599 (онлайн)
SPIIRAS Proceedings. 2009. Issue 9. ISSN 2078-9181 (print), ISSN 2078-9599 (online)
www.proceedings.spiiras.nw.ru