А. П. Стасенко

СИСТЕМА ИНТЕРФЕЙСОВ ТРАНСЛЯТОРА ВО ВНУТРЕННЕЕ ПРЕДСТАВЛЕНИЕ IR1*

ВВЕДЕНИЕ

Разработанный в виде COM-библиотеки транслятор из языка Sisal [1] во внутреннее представление IR1 [2] требует описания системы интерфейсов, с помощью которых происходит взаимодействие с клиентом библиотеки.

1. ТРЕБОВАНИЯ К СИСТЕМЕ ИНТЕРФЕЙСОВ

При разработке системы COM-интерфейсов библиотеки txt2ir1.dll, описывающих трансляцию из некого текстового представления программы во внутреннее представление IR1, требовалось обеспечить следующие свойства.

1. Отсутствие жесткой привязки к языку текстового представления. Это требование закладывает возможность использования данной системы интерфейсов для трансляции во внутреннее представление IR1 отличных от Sisal языков программирования.
2. Полная независимость от реализации транслятора, за исключением явного выделения хорошо формализуемого перевода потока символов в поток лексем (лексического анализа). Скрытие деталей реализации в данном случае повышает наглядность интерфейсов и не является существенным недостатком, так как результат трансляции обычно более важен, чем её процесс.
3. Исключение явного выделения сущности (интерфейса) синтаксического дерева, как результата работы синтаксического анализатора и

* stasenko@nm.ru

Работа выполнена при финансовой поддержке Министерства образования РФ (научная программа «Университеты России», грант № УР.04.01.201).

вытекающее из этого совмещение этапов синтаксического и семантического анализов. Данная особенность следует из графовой природы внутреннего преставления IR1, которое можно рассматривать в качестве обобщения понятия синтаксического дерева.

4. Поддержка UTF-8 кодировки уникода (Unicode-16) во входном потоке символов лексического анализатора и дальнейшее оперирование только уникод-строками. Уникод позволяет избежать трудностей с поддержкой иностранных алфавитов, а UTF-8 кодировка была выбрана из-за компактного текстового (ASCII) представления синтаксических конструкций большинства языков программирования (включая Sisal).
5. Ориентация на анализ потоков символов и лексем. Такое требование обеспечивает на уровне интерфейсов однопроходный, последовательный лексический и синтаксический анализы.
6. Возможность рассматривать исходное представление в качестве части (модуля) полного исходного представления программы (проекта), связанной с другими частями. Необходимость этого условия вытекает из модульной структуры программ на языке Sisal 3.0 [3].
7. Поддержка реализации лексического анализатора в виде самостоятельной фазы трансляции и в виде подпрограммы синтаксического анализатора, возвращающей очередную лексему. Первый вариант удобен при дополнительной предварительной обработке препроцессором списка разбираемых лексем, а второй — снижает емкостную сложность лексического анализа до константной, в случае, если на входе используется поток символов файла, считываемых с диска по мере необходимости.
8. Наличие развитого механизма сообщения об ошибках и предупреждениях на этапах лексического, синтаксического и семантического анализов. Под «развитым механизмом» подразумевается возможность получить описания и места возникновений произошедших событий и достаточно оперативно (до следующего разбираемого символа или лексемы потока) на них отреагировать.
9. Использование множества строк внутреннего представления в качестве таблицы имен лексического анализатора. При выполнении

данного условия отпадает необходимость дублировать строки на этапе разбора лексем.

10. Упрощение вспомогательных интерфейсов, указатели на которые требуются при трансляции. Более простые интерфейсы легче реализовать в случае, если требуется реализация, отличная от стандартной. Также ограничение интерфейсов уменьшает количество побочных эффектов реализации, их использующей.
11. Полная совместимость со скриптовыми клиентами библиотеки, наподобие VBScript и JavaScript. В этом случае диапазон и удобство применения разрабатываемой библиотеки существенно увеличиваются за счёт широкого распространения интернет-технологий.

2. АНАЛОГИЧНЫЕ РАЗРАБОТКИ

К сожалению, в открытых источниках практически не встречаются хорошо разработанные системы интерфейсов трансляторов. Это связано, прежде всего, с тем, что для разработки трансляторов широко применяются компиляторы компиляторов [4, 5], создающие встраиваемый исходный код, который не нуждается в хорошо проработанном внешнем интерфейсе. К тому же проблемой многих генераторов лексических анализаторов является отсутствие встроенной поддержки уникода. Положение также несколько усугубляется нетипичной для императивных языков природой цели (back-end) трансляции, представляющей собой поток вычислений, а не абстрактное синтаксическое дерево.

Тем не менее, несмотря на неприменимость существующих разработок в непосредственном виде, можно выделить следующие существующие системы интерфейсов транслятора.

1. GOLD Parser ActiveX Engine, ClearParse ActiveX/COM Interface и ProGrammar ActiveX Control. Обладают сходной функциональностью и позволяют аккуратно контролировать процесс последовательного выделения нетерминалов, заданной грамматики, из входного потока символов. Данная функциональность является достаточно низкоуровневой и не может быть обобщена на уровне интерфейсов с реализацией транслятора, которая не имеет в основании автоматной модели, построенной по некоторой грамматике. К

недостаткам указанных интерфейсов также относится отсутствие концепции потоков символов и лексем. Входные данные лексического анализатора непосредственно задаются с помощью имени файла или строкой символов, что в некоторых случаях может быть неоптимальным.

2. JavaCC [6]. Генерирует транслятор для языка Java. Использует стратегию нисходящего разбора [7], в отличие от стандартной для генераторов типа *yacc* стратегии восходящего разбора. Данный подход является более интуитивным и упрощает отладку транслятора. Рекурсивная природа алгоритма трансляции допускает естественное обобщение на уровне системы вложенных интерфейсов (понятий). Также такой подход позволяет использовать более развитую и точную систему сообщений об ошибках трансляции.
3. SabreCC [8]. Является в некотором роде развитием системы JavaCC в области использования объектно-ориентированного подхода для лучшего разделения автоматически генерируемого и пользовательского кода транслятора. SableCC генерирует классы прохода дерева, используя расширенную версию шаблона «посетитель» («visitor design pattern») [9], позволяющего выполнять действия над узлами абстрактного синтаксического дерева, используя наследование.
4. Система TeachLisp. Представляет собой интерпретатор языка Lisp, встроенный в Internet-учебный курс для обучения языку Лисп (в настоящий момент не функционирует).

3. ОПИСАНИЕ СИСТЕМЫ ИНТЕРФЕЙСОВ

На рис. 1 приведена UML [10] диаграмма всех интерфейсов библиотеки за исключением интерфейсов, связанных с представлением текста и лексем. На этой диаграмме у каждого интерфейса в алфавитном порядке указаны его атрибуты и операции, причём те или другие могут быть опущены. Атрибуты, помеченные символом @ и словом {frozen}, доступны только для чтения.

Основной интерфейс библиотеки *ITXT2IR1*, отвечающий за трансляцию, в ней не реализован, так как его реализация является языково-специфичной и вынесена в отдельную библиотеку. Метод *TextToLexemes* осуществляет

лексический разбор, а метод *ModuleLexemesToIR1* отвечает за синтаксический и семантический анализ. Аргумент *type* метода *ModuleLexemesToIR1* является языково-зависимым идентификатором понятия, задаваемого потоком лексем (например, интерфейс или реализация модуля).

Интерфейс *IModuleSearch* служит для обеспечения поиска интерфейса модуля по его имени, что может потребоваться для семантического анализа имён текущего модуля. Интерфейс модуля возвращается как указатель на интерфейс *IUnknown*, и его дальнейшая идентификация зависит от реализации метода *ITXT2IR1* → *ModuleLexemesToIR1*. Обычно интерфейс модуля можно извлечь из потока лексем (*ILexemeStream*), его задающих, или внутреннего представления IR1 (интерфейс *IIR1*).

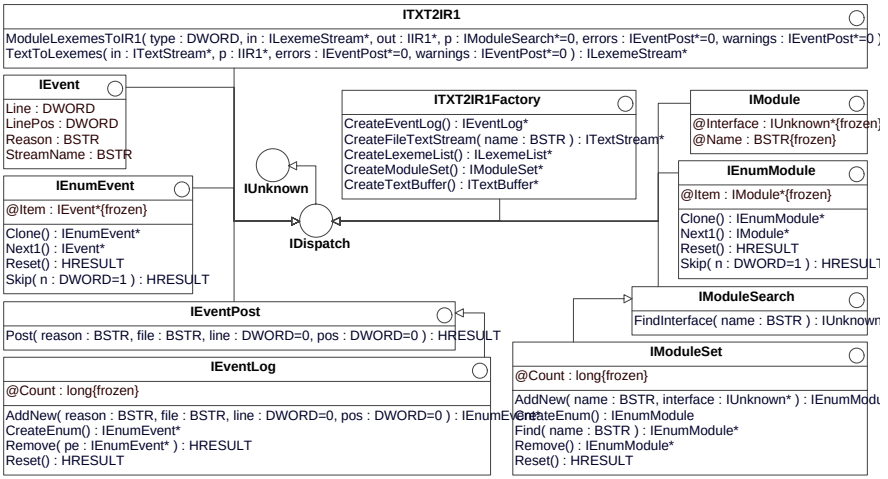

Рис. 1. Диаграмма основных интерфейсов

Простота интерфейса *ITXT2IR1* повышает его универсальность, так как этот интерфейс не зависит ни от входного языка, ни от реализации транслятора. Допускается использование утилиты типа *lex* для порождения лексического анализатора, а утилиты типа *yacc* для порождения синтаксического и семантического транслятора. Для реализации библиотеки sis2ir1.dll

транслятора из языка Sisal 3.0 был использован автомат, описанный с помощью графического метаязыка [11].

В библиотеке реализован интерфейс *ITXT2IR1Factory*, служащий в качестве фабрики классов для остальных интерфейсов библиотеки. На рис. 2 приведена UML диаграмма всех остальных интерфейсов библиотеки, не вошедших на рис. 1.

К интерфейсам, непосредственно участвующим в процессе трансляции, относятся *ITextStream*, *ILexemeStream*, *IEventPost*, *IModuleSearch*, *ILexeme*, *IEnumLexeme* и *IEnumText*. Все остальные интерфейсы являются вспомогательными и служат для обеспечения полноценной работы скриптовых клиентов с библиотекой.

К необходимым для эффективной трансляции методам относится метод *ITXT2IR1Factory → CreateFileTextStream*, который возвращает поток символов (*ITextStream*), связанный с файлом, с указанным именем. Непосредственное чтение байтов файла происходит по мере необходимости в момент перечисления элементов потока символов (*IEnumText*). Файл рассматривается как последовательность UNICODE символов в UTF-8 кодировке, поэтому метод *ITextStream→Count*, возвращающий количество элементов коллекции, в этом случае не реализован.

Предполагается, что при реализации метода *ITXT2IR1 → TextToLexemes* лексический анализ производится только при перечислении (*IEnumLexeme*) элементов возвращаемого потока лексем (*ILexemeStream*). Соответственно, тогда же и происходит возникновение возможных событий (ошибок и предупреждений). Поэтому временная и емкостная сложности работы самого метода константны.

Интерфейс *IEventPost* дает транслятору возможность сообщить о произошедшем событии. Стандартная реализация этого интерфейса (*IEventLog*) позволяет накапливать произошедшие события. Их просмотр возможен после получения следующей лексемы потока *ITXT2IR1 → TextToLexemes* или завершения процесса трансляции *ITXT2IR1 → ModuleLexemesToIR1*. Во втором случае для более тонкой реакции можно переопределить стандартную реализацию интерфейса входного потока лексем для проверки произошедших событий после каждой лексемы. Также возможно расширение реализации самого интерфейса *IEventPost* для ещё более оперативной реакции на событие.

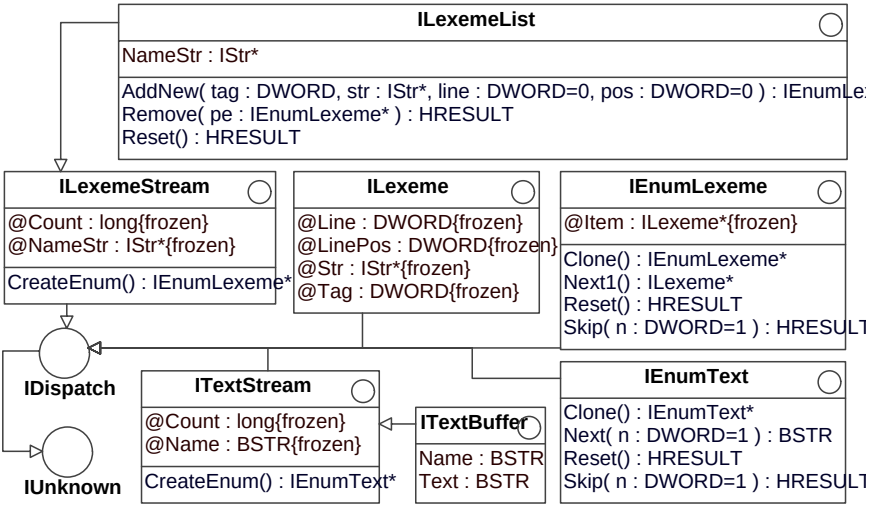

Рис. 2. Диаграмма интерфейсов потоков

К поддержке совместимости со скриптовыми клиентами относятся:

1) наследование всех интерфейсов от интерфейса *IDispatch*;
2) использование только совместимых типов данных: указатели на интерфейс, *long*, *HRESULT = long*, *DWORD = unsigned long*, *BSTR*;
3) отсутствие передачи параметров по ссылке (исключая возвращаемое значение);
4) наличие стандартной реализации всех вспомогательных интерфейсов;
5) все коллекции поддерживают скрытое свойство *_NewEnum* для стандартных средств их итерирования;
6) возможность регистрации библиотеки в качестве сервера автоматизации.

4. ПУТИ ДАЛЬНЕЙШЕГО РАЗВИТИЯ

Дальнейшее развитие системы интерфейсов планируется в основном за счёт увеличения функциональности интерфейса *ITXT2IR1*. Разумным выглядит добавление методов, ответственных за анализ сущностей на уровне модуля программы: типов, функций и выражений. Имеет смысл в качестве расширения концепции аргумента *type* метода *ModuleLexemesToIR1* произвести добавление интерфейса, задающего параметры трансляции, которые зависят от её реализации и входного языка.

Вероятным выглядит появление дополнительного интерфейса, реализующего сущность транс-литератора из потока байтов в последовательность уникод-символов. Также по мере развития других частей системы функционального программирования возможно расширение интерфейсов *ILexeme* и *IEvent*. В частности, планируется расширить интерфейс лексемы следующими свойствами.

1. Номер позиции начала текстового представления лексемы в потоке символов. Это поможет обнаруживать её местоположение в потоке за константное время без предварительного разделения потока на строки символов. Такое свойство также важно и для интерфейса *IEvent*.
2. Окончание (номер позиции, строка и столбец) текстового представления лексемы в потоке символов. Данная информация необходима для указания точных границ синтаксических конструкций во внутреннем представлении, что, возможно, потребуется при их «подсвечивании» в процессе отладки.
3. Строка с текстовым представлением лексемы в потоке символов. Может оказаться полезным в качестве дополнительной отладочной информации.

Интерфейс *IEvent* разумно расширить свойствами, дополнительно классифицирующими и уникально идентифицирующими произошедшее событие. В качестве классификации можно использовать:

1) более тонкую степень серьезности события: совет, предупреждение, ошибка, «фатальная» ошибка;
2) уровень важности события;

3) лексическую, синтаксическую или семантическую природу события.

С введением подобной классификации событий, станет ненужным существующее в интерфейсе *ITXT2IR1* жесткое деление событий на ошибки и предупреждения. Для этого в интерфейс *IEventLog* необходимо внедрить возможность перечисления событий, принадлежащих одному классу.

ЗАКЛЮЧЕНИЕ

В рамках описанной системы интерфейсов реализован транслятор из языка Sisal 3.0 во внутреннее представление IR1. С точки зрения разработки этого конкретного транслятора предложенная система интерфейсов оказалась достаточно удобной. Отсутствие привязки интерфейсов к языку Sisal позволяет надеяться, что рассмотренная система интерфейсов будет приемлемой и для других языков программирования.

СПИСОК ЛИТЕРАТУРЫ

1. **Бирюкова Ю. В.** Sisal-90: руководство пользователя. — Новосибирск, 2000. — 84 с. — (Препр. / РАН. Сиб. Отд-ние. ИСИ; № 72).
2. **Стасенко А. П.** Внутреннее представление системы функционального программирования Sisal 3.0. — Новосибирск, 2004. — 54 с. — (Препр. / РАН. Сиб. Отдние. ИСИ; № 110).
3. **Касьянов В. Н., Бирюкова Ю. В., Евстигнеев В. А.** Функциональный язык Sisal 3.0 // Поддержка супервычислений и интернет-ориентированные технологии. — Новосибирск, 2001. — С. 54-67.
4. **Lesk M. E.** Lex — a lexical analyzer generator. — Computing Science Technical Report 39 — AT&T Bell Laboratories, Murray Hill, N.J., 1975.
5. **Johnson S. C.** YACC — yet another compiler compiler. — Computing Science Technical Report 32 — AT&T Bell Laboratories, Murray Hill, N.J., 1975.
6. **Galles D.** Modern Compiler Design. — Scott Jones Publishers, 2005.
7. **Евстигнеев В. А., Касьянов В. Н.** Теория графов: алгоритмы обработки деревьев. — Новосибирск: Наука, 1994.
8. **Gagnon E.** SableCC, An Object Oriented Compiler Framework. — M.S. Thesis — McGill University, 1998.

9. **Гамма Э., Хелм Р., Джонсон Р., Влиссидес Дж.** Приемы объектно-ориентированного проектирования. Паттерны проектирования. — СПб.: Питер, 2001 — 368 с. — ISBN: 5-272-00355-1.
10. **Stevens P., Pooley R.** Using UML: software engineering with objects and components. — Object Technology Series. — Addison-Wesley Longman, 1999. — 280 p. — ISBN: 0-201-64860-1.
11. **Стасенко А.П.** Графический метаязык для описания транслятора // V Всерос. конф. молодых ученых по математическому моделированию и информационным технологиям / Программа и тезисы докладов. — Новосибирск, 2004. — С. 53.