РОССИЙСКИЙ ФЕДЕРАЛЬНЫЙ ЯДЕРНЫЙ ЦЕНТР
ВСЕРОССИЙСКИЙ НАУЧНО-ИССЛЕДОВАТЕЛЬСКИЙ ИНСТИТУТ
ЭКСПЕРИМЕНТАЛЬНОЙ ФИЗИКИ
(ВНИИЭФ)

На правах рукописи

РАТКЕВИЧ ИРИНА СЕРГЕЕВНА

РАСШИРЕННЫЙ ЯЗЫКОВОЙ СЕРВИС FRIS ДЛЯ ПРОГРАММИРОВАНИЯ НА ЯЗЫКЕ FORTRAN В MICROSOFT VISUAL STUDIO

Специальность 05.13.11 – «Математическое и программное обеспечение вычислительных машин, комплексов и компьютерных сетей»

Диссертация на соискание учёной степени
кандидата физико-математических наук

Научный руководитель:
доктор физико-математических наук

Бартенев Юрий Германович

Саров - 2016

Оглавление

Введение

Актуальность темы исследования

Язык программирования Fortran [1-7] занимает одно из лидирующих положений в мире в среде разработки приложений для наукоёмких вычислений, включая параллельные высокопроизводительные вычисления [8-9], хотя в целом и уступает лидирующим языкам (Java, C/C++) [10-11]. Это связано с рядом факторов. Во-первых, Fortran является одним из первых высокоуровневых языков программирования (появился в 1950-х годах). Во-вторых, с тем, что он изначально разрабатывался как язык для математических вычислений. В-третьих, Fortran является одним из первых языков программирования, который был стандартизирован: сначала в рамках ANSI [2-3], а затем уже на мировом уровне в рамках ISO [3-7]. Благодаря этому к настоящему времени накоплен многочисленный пакет высококачественных программных продуктов различной направленности, реализованных на данном языке и используемых во всём мире: NAG [12] и LAPACK [13] – математические пакеты с обширным функционалом для высокопроизводительных параллельных вычислений, NJOY [14] – физический пакет для обработки ядерно-физических данных, и ряд других.

Язык программирования Fortran используется в ведущих научно-исследовательских организациях, таких как LANL [15], BNL [16], CERN [17], РФЯЦ-ВНИИЭФ [18], РФЯЦ-ВНИИТФ [19] и других. В этих организациях ведётся не только поддержка уже разработанных программных пакетов, но и их развитие с учётом современных особенностей технологий программирования и аппаратных компонентов, на которых необходимо обеспечивать эффективную работу указанных продуктов.

Необходимо отметить, что в последнее время произошло качественное изменение в технологии создания СуперЭВМ [20], связанное с новой концепцией достижения максимальной производительности при минимизируемых энергозатратах. А именно — на смену СуперЭВМ, построенным на универсальных процессорах, пришли гетерогенные СуперЭВМ, в которых в каждом вычислительном узле присутствуют одновременно многоядерные универсальный процессор и специализированный арифметический сопроцессор [21-22], например фирмы Intel. Соответственно, для эффективного использования такой СуперЭВМ необходимо применять смешанную схему распараллеливания: MPI [23] — для распараллеливания вычислений на распределённой памяти множества

процессоров и сопроцессоров, OpenMP [24] — для распараллеливания на общей памяти множества вычислительных ядер процессора и сопроцессора, SIMD-операции — для эффективного использования векторных операций процессоров и сопроцессоров. В связи с этим необходима переработка уже существующих программ вплоть до использования иных математических схем и программных алгоритмов их реализующих. Подобные работы были широко развёрнуты, например, в РФЯЦ-ВНИИЭФ [25-32], начиная с ЭВМ, оснащённых графическими ускорителями.

Рассмотрим некоторые особенности, присущие написанию сложных программ, в частности в РФЯЦ-ВНИИЭФ. Их написание ведётся в том числе с использованием современной интегрированной среды разработки (IDE) – Microsoft Visual Studio (VS) [33] и компиляторов от фирм Intel [34] и PGI [35].

К специфике сложных программ, например, РФЯЦ-ВНИИЭФ относятся:

- разработка и усовершенствование в ходе эксплуатации коллективом специалистов;
- использование специализированных библиотек программ, написанных на различных языках программирования;
- создание специализированных связанных программ, включающих в себя в качестве отдельных компонентов блоки, являющиеся самостоятельными программами, для учёта большего числа моделируемых процессов;
- возможность выполнения расчёта в условиях сбоев ЭВМ;
- возможность исполнения программ на различных классах ЭВМ;
- эффективное использование вычислительных ресурсов ЭВМ в широком диапазоне производительности и вариаций архитектуры.

Таким образом, появившаяся **новая практическая задача** по адаптации существующих и созданию новых программ, учитывающих современные тенденции развития и эффективного использования вычислительной техники, в особенности суперкомпьютеров, актуализирует создание специализированных инструментов — помощников по созданию параллельных программ.

В этой связи проводятся многочисленные исследования по предоставлению инструментов - помощников по распараллеливанию [36-40] существующих программ — для эффективного использования ими современных ЭВМ и СуперЭВМ. Однако такие инструменты, как правило, позволяют лишь выявить потенциальные места для распараллеливания, а окончательное решение принимает программист. Более того, программная

реализация математического алгоритма, поддающегося распараллеливанию, может быть такой, что инструмент просто не сможет обнаружить саму возможность его распараллеливания [8-9]. В любом случае подобные инструменты работают с уже реализованными на языке программирования алгоритмами.

При этом упускается из виду другой процесс, слабо автоматизированный в части языка программирования Fortran, – это процесс написания самого текста программы. Он отличается от уже указанных процессов тем, что необходимо анализировать динамически меняющуюся по мере ввода пользователем текста структуру программы и предоставлять различную информацию о ней непосредственно в процессе ввода текста на целевом языке программирования. Ключевой особенностью здесь является априорное знание о некорректности с точки зрения синтаксиса и семантики языка программирования редактируемой пользователем программы. Поэтому инструмент анализа должен делать ряд предположений о том, что имеет в виду пользователь, набирая текущую синтаксическую конструкцию.

Вторым важным моментом в автоматизации процесса написания текста программы является возможность встроенного предоставления справки по таким общеиспользуемым средствам распараллеливания как MPI [23] и OpenMP [24]. Хотя их спецификация описана в соответствующих документах, обращение к ней или даже к отдельно стоящей справочной системе снижают скорость написания программы, в то время как предоставление подсказок с описанием спецификации и назначения тех или иных элементов указанных средств распараллеливания непосредственно в процессе написания текста программы позволит ускорить работу и существенно повысит удобство использования средств распараллеливания. В общем случае возможность предоставления справочной информации и спецификации интерфейсов полезна не только для значимых внешних программных библиотек, возможно, реализованных на иных языках программирования, но и для элементов, описанных и использующихся в текущей программе.

Подобная, условно назовём её «низкоуровневой», поддержка и инструменты для её обеспечения разрабатываются в основном за рубежом [34-35, 41-45] и для Fortran недостаточно развиты. Отметим, что указанные инструменты работают в соответствующих интегрированных средах разработки (IDE) и задействуются при работе пользователя с текстовым редактором для набора текста программы.

Таким образом, возникает новая **актуальная научная задача** по разработке абстрактной (общей) модели анализатора динамически меняющейся во времени программы, выполняющего построение её внутренней структуры и предоставляющего затем эти сведения пользователю в процессе написания текста программы для автоматизации этого процесса.

Для реализации указанной модели были выбраны интегрированная среда разработки Visual Studio и спецификация языка Fortran – Fortran 2003. Данный выбор обусловлен соображениями их широкого практического использования в РФЯЦ-ВНИИЭФ, где для написания программ используются как стандарт Fortran 90 [28-29], так и стандарт Fortran 2003 [46]. Поэтому для реализации был выбран стандарт Fortran 2003, полностью включающий в себя конструкции стандарта Fortran 90.

Степень разработанности

Будем называть *языковым сервисом*[1] [47] инструмент, отвечающий за предоставление ориентированной на конкретный язык программирования поддержки программиста при редактировании текста программ в редакторе интегрированной среды разработки. При этом, по определению Microsoft, *базовый языковой сервис* [48] должен предоставлять подсветку синтаксических конструкций (далее синтаксиса) программы. *Расширенный языковой сервис* – должен, помимо подсветки синтаксиса, предоставлять поддержку технологии IntelliSense [49] и, возможно, дополнительные функции.

Технология IntelliSense [49], согласно определению Microsoft, это обобщённый термин для обозначения следующих возможностей:

- построение списка элементов сложного объекта (List Members);
- отображение сведений о параметрах процедур (Parameter Info);
- отображение кратких сведений об элементе языка программирования (Quick Info);
- завершение слова или автодополнение (Complete Word).

Отметим, что все современные интегрированные среды разработки поддерживают указанные возможности, названия которых могут варьироваться от одной среды разра-

[1] Здесь определение языкового сервиса, данное Microsoft только для среды разработки Visual Studio, распространено на все интегрированные среды разработки, т.е. обобщено для всех интегрированных сред разработки.

ботки к другой. В целях общности изложения будем использовать терминологию Microsoft, понимая под ней функциональное назначение той или иной возможности.

Используемые в РФЯЦ-ВНИИЭФ компиляторы Fortran и соответствующие им языковые сервисы от фирм Intel [34] и PGI [35] реализуют не все возможности IntelliSense (таблица 1). Подчеркнём, что они не поддерживают возможности работы с внешними библиотеками программ и не поддерживают возможность по предоставлению смыслового описания для элементов языка программирования.

Таблица 1. Сравнение языковых сервисов Intel и PGI.

Возможность	Intel	PGI
Возможности технологии IntelliSense		
Построение списка элементов сложного объекта	нет	нет
Отображение сведений о параметрах процедур	есть, кроме перегруженных процедур и процедур, связанных с типом данных	есть, только для встроенных процедур
Отображение кратких сведений об элементе языка программирования	есть, за исключением полей и процедур производных типов данных	есть, только для встроенных процедур
Автодополнение	есть, только для имён модулей и подпрограмм и функций	есть, только для ключевых слов
Дополнительные возможности		
Поддержка комментариев документирования	нет	нет
Поддержка внешних библиотек	нет	нет

Основной частью любого языкового сервиса является блок анализа текста на целевом языке программирования. Данная область – область построения анализаторов (лексических, синтаксических и семантических) – довольно хорошо разработана [50-56]. Классическими трудами в этой области являются, например, книги «Компиляторы: принципы, технологии и инструменты» Ахо А., Сети Р. и Ульмана Д. [50-53]. Они описывают технологию построения компиляторов [57] и трансляторов [57].

Что касается современных разработок в этой области, то в данной работе использовались принципы построения так называемых *языковых приложений*, изложенные в работах Т. Пэрра [55-56]. *Языковыми приложениями* Т. Пэрр называет любые программы, тем или иным образом анализирующие некоторые данные, собирающие необходимую им информацию и/или генерирующие некоторые данные. Таким образом, термин языковые приложения является наиболее общим и включает в себя трансляторы, компиляторы, языковые сервисы и другие программы по анализу и/или генерации некоторых данных в качестве частных видов языковых приложений.

Отметим, что, несмотря на наличие широкой теоретической базы по созданию языковых приложений, дополнительные сложности при их разработке могут вызывать особенности языка программирования. В части Fortran примером может служить отсутствие в нём зарезервированных ключевых слов. Поэтому определение того, что обрабатывается ключевое слово, полностью зависит от контекста, а не наоборот — определение контекста основывается на присутствии ключевого слова. Это и многое другое существенно усложняет разработку лексического и синтаксического анализаторов. Подавляющее большинство существующих анализаторов разработаны для стандартов Fortran 77/90/95 [3-5], которые уже устарели. Основной причиной является то, что новые стандарты [6-7], начиная со стандарта Fortran 2003, вводят дополнительные языковые конструкции для поддержки объектно-ориентированного программирования.

Так, среди отечественных программ выделим две разработки ИПМ им. Келдыша: интерактивную программу — анализатор Fortran программ [58-59] на основе пакета Sage [60], и систему автоматизации распараллеливания САПФОР [38-39]. Обе программы используют для анализа пакет Sage, при этом первая работает с языком Fortran 77, а вторая с языком Fortran 95. Необходимо отметить, что оба продукта реализованы в виде приложений с графическим пользовательским интерфейсом. Отметим здесь, что пакет Sage имеет ограничение на длину обрабатываемого файла, составляющую 5000 строк [58-59]. Если обрабатываемый файл имеет большую длину, то его необходимо разбивать на блоки, удовлетворяющие требованиям пакета. Второй важный момент - необходимость явного запуска инструментов анализа, т.е. анализ выполняется не во время непосредственного набора текста, поэтому не является в строгом смысле интерактивным.

Среди зарубежных анализаторов для Fortran необходимо отметить проект Open Fortran Parser (OFP) [61], разрабатываемый LANL (Лос-Аламосская национальная лаборатория, США). Он свободно распространяется в исходных кодах, поддерживает стандарт Fortran 2003/2008, реализован на языке программирования Java. Основная особенность OFP как синтаксического анализатора – это возможность встраивания в другие продукты, требующие механизма анализа текстов программ на языке Fortran. Он работает по принципу обратных вызовов, сходных с технологией Simple API for XML (SAX) [62]. Т.е. внешнее приложение регистрирует функции, которые OFP вызывает при обнаружении соответствующей языковой конструкции, а именно – инструкции

(statement). Кроме того, подход, применяемый для разбора файлов в OFP, основан на предварительной и весьма серьёзной модификации исходного текста программы, что приводит к потере части информации о местоположении ряда программных элементов, например, определений переменных, что усложняет его использование в интегрированных средах разработки. Несмотря на это, он используется как компонент языкового сервиса Fortran [45] в свободно распространяемой среде разработки NetBeans [63], реализованной на Java. Необходимо также отметить, что OFP работает только с синтаксическими определениями элементов стандарта языка Fortran, то есть не предоставляет механизмов по извлечению из текста программы каких-либо комментариев и, соответственно, не предоставляет возможности сопровождать определения элементов их смысловым описанием.

Таким образом, существующие языковые сервисы для языка Fortran в Visual Studio от Intel и PGI, как было отмечено ранее, не реализуют в полном объёме необходимой поддержки для эффективной работы Fortran-программистов. При этом наиболее развитый анализатор OFP не пригоден для использования в Visual Studio ввиду того, что, во-первых, реализован на Java (для которой отсутствуют инструменты интеграции с Visual Studio) и, во-вторых, он существенно изменяет исходный текст программы, что может исказить часть информации, например касающейся местоположения программных элементов в тексте программы.

Всё это позволяет сделать вывод, что для Visual Studio существующие реализации языкового сервиса Fortran не обеспечивают в полной мере автоматизацию процесса написания текста программ.

Цели и задачи

Целью данной работы является автоматизация процесса написания текста программ на языке программирования Fortran 2003 в Microsoft Visual Studio:

- учитывающая специфику написания сложных программ;
- учитывающая использование средств параллельного программирования;
- направленная на повышение эффективности труда программистов и, как следствие, сокращение сроков и снижение количества допускаемых ошибок при программировании, а также на повышение качества программных продуктов.

Для достижения поставленной цели нужно решить следующие *задачи* по разработке и реализации:

1) языкового сервиса с полной поддержкой технологии IntelliSense для языка Fortran 2003;
2) подсистемы анализа текстов на языке Fortran 2003, учитывающей его особенности и позволяющей, помимо основного определения элемента, извлекать из программного кода его смысловое описание;
3) системы хранения информации о значимых (для языкового сервиса) элементах языка программирования для обеспечения оперативной работы языкового сервиса;
4) механизма документирования текста программ, позволяющего использовать в языковом сервисе комментарии в виде подсказок;
5) механизма поддержки[2] языковым сервисом внешних[3] программных средств:
   - технологий параллельного программирования MPI, OpenMP, SIMD-операций;
   - внешних библиотек программ. В РФЯЦ-ВНИИЭФ такими общеиспользуемыми библиотеками являются: УРС-ОФ [31,64] – для расчёта уравнения состояния вещества; ЕФР [32,65-67] – для параллельного чтения/записи сеточных массивов.

Научная новизна

Научной новизной данной работы обладают:

1. Абстрактная модель языкового сервиса, обеспечивающая, в отличие от аналогов, расширяемую интеллектуальную поддержку использования в программе внешних программ не обязательно на том же языке.
2. Модель значимых для языкового сервиса элементов языка Fortran 2003, позволяющая, в отличие от аналогов, строить в оперативной памяти эквивалентное представление программы в виде дерева значимых элементов, снабжённых смысловым описанием своего предназначения.
3. Концепция расширенной поддержки внешних программных библиотек Fortran-программы, основанная на использовании языка XML и позволяющая, в отличие от

[2] Выделение цветом, включение в функции технологии IntelliSense

[3] Под сторонней или внешней библиотекой программ понимается библиотека программ, не являющаяся частью текущего решения Visual Studio, и, возможно, реализованная на языке, отличном от Fortran

аналогов, выполнять анализ программы с использованием любых средств обработки структурированных XML-данных. Концепция включает:

- модель прикладных программных интерфейсов (API) Fortran, служащую как для описания Fortran API внешней библиотеки программ, так и для описания Fortran API программного проекта;
- модель комментариев документирования, позволяющую предоставлять смысловые описания для элементов модели Fortran API.

4. Алгоритм подсветки синтаксических конструкций Fortran-программ в текстовом редакторе интегрированной среды разработки, который, в отличие от аналогов, поддерживает выделение элементов внешних библиотек программ и средств параллельного программирования MPI, OpenMP.

Теоретическая и практическая значимость работы

Результаты диссертационной работы носят как теоретический, так и практический характер. Предложена модель абстрактного языкового сервиса, обеспечивающая расширенную поддержку языка программирования и учитывающая работу с внешними программными библиотеками, реализованными на языках, отличных от целевого.

Практическим результатом выполнения данной работы является программная реализация специализированного программного обеспечения – языкового сервиса FRIS [68-78] — для языка программирования Fortran 2003 и интегрированной среды разработки Microsoft Visual Studio, использующегося [76-78] для поддержки написания текстов программ на языке Fortran. Данный сервис:

- учитывает специфику разработки современных сложноструктурированных прикладных программ;
- учитывает использование современных средств параллельного программирования: MPI, OpenMP, SIMD-операции;
- предоставляет контекстную помощь, включающую информацию по предназначению использования тех или иных элементов программы;
- обеспечивает встраивание поддержки внешних библиотек программ на примере встроенной поддержки общеиспользуемых в РФЯЦ-ВНИИЭФ библиотек УРС-ОФ и ЕФР.

Применение созданного в рамках данной работы специализированного программного обеспечения (языкового сервиса) состоялось при написании ряда программ РФЯЦ-ВНИИЭФ (см. пункт 3.3), а началось в библиотеке MAGIK [46] с приспособления к: 1) применению в «многомерной» программе САТУРН [79-80]; 2) современным особенностям использования СуперЭВМ.

Разработанная модель прикладных программных интерфейсов (API) Fortran позволяет строить различные анализаторы значимых элементов в проектах и их взаимосвязи. Совместно с моделью комментариев документирования модель прикладных программных интерфейсов может использоваться для автоматического создания документации программиста и/или пользователя. Разработанные и реализованные алгоритмы анализа Fortran-программ могут использоваться для создания статических анализаторов, а также препроцессоров.

FRIS используется для написания прикладных Fortran-программ на предприятиях: РФЯЦ-ВНИИЭФ [76], ОАО «НПК КБМ» [77] и АО «КБП им. академика А.Г. Шипунова» [78].

FRIS повышает удобство программирования, позволяет ускорить написание текстов программ [81], скоординировать «командную» работу и снизить количество ошибок.

Методология и методы исследования

При выполнении данной работы использовались: деятельностный и системный подход (философские методы), методы теории построения компиляторов, трансляторов и языковых приложений, методы анализа текстов на искусственных языках, методы отображения конкретного синтаксиса в абстрактный, методы построения таблиц символов. При реализации языкового сервиса использовался объектно-ориентированный подход к программированию.

Положения, выносимые на защиту

1. Абстрактная модель языкового сервиса, предназначенная для построения языковых сервисов для языков программирования и интегрированных сред разработки, обеспечивающая поддержку использования внешних библиотек программ не обязательно на том же языке.

2. Модель значимых элементов языка программирования Fortran 2003, позволяющая строить в оперативной памяти эквивалентное представление программы в виде дерева элементов, снабжённых смысловым описанием своего предназначения, и связанные с ней:
   - модель описания прикладных программных интерфейсов Fortran 2003 для внешних программных библиотек;
   - модель XML комментариев документирования для Fortran 2003, позволяющая предоставлять смысловое описание для значимых элементов.
3. Алгоритмы анализа программ на языке Fortran 2003, обеспечивающие при редактировании программы:
   - обработку некорректных конструкций программы;
   - построение дерева значимых элементов с учётом наличия в анализируемом тексте их смыслового описания;
   - подсветку синтаксиса в режиме построчного инкрементального разбора;
   - поддержку выделения элементов внешних библиотек.
4. Программная реализация моделей, алгоритмов и структур данных в языковом сервисе FRIS, предназначенном для ускоренного написания программ на языке Fortran 2003.

Соответствие паспорту специальности

Область исследования соответствует паспорту специальности 05.13.11 — «Математическое и программное обеспечение вычислительных машин, комплексов и компьютерных сетей».

Пункту 1 «Модели, методы и алгоритмы проектирования и анализа программ и программных систем, их эквивалентных преобразований, верификации и тестирования» соответствуют главы 1, 2 диссертационной работы.

Пункту 2 «Языки программирования и системы программирования, семантика программ» соответствуют главы 1, 2, 3 диссертационной работы.

Степень достоверности и апробации результатов

Достоверность результатов подтверждается реализацией и тестированием разработанных моделей, алгоритмов, программных модулей и языкового сервиса в целом.

Изложенные в диссертации результаты обсуждались на следующих международных и российских научно-технических конференциях:

- математическая конференция РФЯЦ-ВНИИТФ (2014 г., Снежинск);
- IX Отраслевая научно-техническая конференция молодых специалистов Росатома «Высокие технологии атомной отрасли. Молодёжь в инновационном процессе» в рамках Третьего Международного бизнес-саммита (2014 г., Нижний-Новгород);
- XV Международная конференция «Супервычисления и математическое моделирование» (2014 г., Саров);
- XIII научно-техническая конференция молодых работников и специалистов «Молодёжь в науке» (2014 г., Саров);
- 9-й Весенне-летний коллоквиум молодых учёных по программному обеспечению (ИСП РАН) SYRCoSE 2015 (2015 г., Самара);
- VI научно-техническая конференция молодых учёных и специалистов по тематике «Актуальные вопросы развития систем и средств ВКО» (2015 г., Москва);
- VII Всероссийский конкурс молодых учёных, посвящённый 70-летию победы (2015 г., Миасс).

Внедрение результатов работы

Результатом данной работы является программно реализованный языковой сервис FRIS для языка программирования Fortran стандарта Fortran 2003 и интегрированной среды разработки Microsoft Visual Studio, разработанный для автоматизации процесса написания Fortran-программ. Предложенное техническое решение защищено свидетельствами об официальной регистрации программы для ЭВМ [74-75].

Языковой сервис FRIS используется при написании и модификации Fortran-программ в РФЯЦ-ВНИИЭФ [76], ОАО «НПК КБМ» [77] и АО «НПК КБП им. академика А.Г. Шипунова» [78], откуда получены акты о внедрении FRIS.

Личный вклад

Все описанные модели, алгоритмы разработаны теоретически и реализованы программно лично автором в языковом сервисе FRIS для языка программирования Fortran и интегрированной среды разработки Microsoft Visual Studio. А именно:

- абстрактная модель языкового сервиса;

- модель прикладных программных интерфейсов (API) Fortran;
- модель XML комментариев документирования;
- схема работы анализаторов некорректных (с точки зрения стандарта Fortran) программ при редактировании программ пользователем.

Автор также реализовал поддержку:

- библиотек УРС-ОФ, ЕФР, MPI и директив OpenMP (предложил Ю.Г. Бартенев);
- директив и функций работы с SIMD-операциями (предложил С.С. Касаткин).

Документирование программных интерфейсов некоторых внешних библиотек для предоставления смыслового описания в языковом сервисе FRIS выполнили:

- для библиотеки УРС-ОФ – Жильникова Н.Н.;
- для библиотеки ЕФР – Олесницкая К.К., Петрова М.А.

В сотрудничестве с разработчиками параллельных программ РФЯЦ-ВНИИЭФ отработаны технические решения и исследованы эксплуатационные характеристики FRIS [81].

Публикации

Результаты диссертации опубликованы в 8 работах, 4 из них статьи в журналах перечня ВАК [68-71], 2 публикации в сборниках трудов и тезисов всероссийских и международных конференций [72-73], получено 2 свидетельства на регистрацию программы для ЭВМ. В совместной работе [72] личный вклад автора состоит в разработке описанного в диссертации языкового сервиса, его моделей и алгоритмов.

Структура диссертации

Диссертация состоит из введения, трёх глав, заключения, списка сокращений и условных обозначений, списка литературы и восьми приложений.

В первой главе рассмотрены основные модели языкового сервиса FRIS, учитывающие специфику написания современных сложноструктурированных программ. Приведены требования к языковому сервису для учёта указанной специфики. Рассмотрена разработанная автором абстрактная модель языкового сервиса, которая может применяться для разработки языковых сервисов для различных языков программирования и интегрированных сред разработки, перечислены структурные блоки модели:

- блок интеграции со средой разработки (целевой IDE), который абстрагирует остальные части языкового сервиса от конкретных деталей работы IDE;
- блок анализа текстов на целевом языке программирования и сбора информации для построения эквивалентной структуры программы и подсветки синтаксиса;
- блок хранения распознанных элементов программы, полученных при анализе текстов программ и обработке XML-описаний API и документации внешних библиотек программ;
- блок сериализации (сохранения) и десериализации (восстановления) элементов, сохраняющий элементы блока хранения в виде XML-представления, восстанавливающий их из XML-представления и помещающий в блок хранения распознанных элементов;
- блок модели представления распознанных элементов блока хранения в подходящем виде для работы компонентов технологии IntelliSense.

Подробно изложены вопросы построения модели значимых элементов Fortran, которые необходимо распознавать и хранить для предоставления соответствующей информации возможностям технологии IntelliSense. Рассмотрена модель комментариев документирования, которая позволяет не только документировать различные структурные элементы программы как непосредственно в теле программы, так и во внешнем файле специальной структуры, но и автоматически отображать данную информацию программисту в дополнение к синтаксическому описанию значимых элементов. Изложена разработанная модель описания прикладных программных интерфейсов (API) Fortran для программ и программных библиотек. Данная модель позволяет обеспечивать языковой сервис всеми необходимыми данными, в том числе, если тексты подключаемой к основному программному проекту библиотеки недоступны или если библиотека реализована на языке программирования, отличном от Fortran, но способна взаимодействовать с программой на этом языке. Модель комментариев документирования и модель прикладных программных интерфейсов составляют основу концепции расширенной поддержки внешних библиотек в языковом сервисе FRIS.

Во второй главе изложены основные вопросы программной реализации языкового сервиса FRIS. Вначале рассмотрена общая структура языкового сервиса. Затем поэтапно рассмотрено устройство и алгоритмы работы основных её структурных блоков. При рассмотрении блока интеграции с IDE дано общее представление о расширении функ-

циональных возможностей Visual Studio. Обсуждается место языкового сервиса в модели расширения Visual Studio при интеграции в неё нового языка программирования. Подробно рассмотрен блок анализа тестов программ на языке Fortran 2003, учитывающий особенности выполнения анализа программы непосредственно во время написания её текста пользователем в текстовом редакторе. Излагаются вопросы построения анализаторов, использующихся для подсветки синтаксиса программы в редакторе исходного кода Visual Studio. Данный режим работы также налагает свои ограничения, например, необходимость работы с текстом программы в построчном инкрементальном режиме с сохранением и восстановлением состояния анализатора в любой момент времени. Рассматриваются вопросы реализации модели значимых элементов в блоке хранения распознанных элементов. Отдельное внимание уделяется блоку модели представления значимых элементов — связующему звену между блоком хранения распознанных элементов и блоком интеграции с IDE, который запрашивает данные для использования в технологии IntelliSense.

Третья глава посвящена вопросам подтверждения работоспособности языкового сервиса FRIS. Рассматриваются вопросы использования FRIS при написании программ в РФЯЦ-ВНИИЭФ. Особое внимание уделяется встроенным возможностям поддержки внешних проблемно-ориентированных и общеиспользуемых в РФЯЦ-ВНИИЭФ библиотек программ УРС-ОФ и ЕФР, технологий параллельного программирования MPI, OpenMP, SIMD-операций. Представлено сравнение разработанного языкового сервиса FRIS с аналогами.

В заключении излагаются основные полученные результаты.

Приложение А содержит определение разработанной автором XML схемы описания прикладных программных интерфейсов языка Fortran, используемой для сохранения и восстановления значимых элементов в языковом сервисе. Приложение Б является справочным и содержит описание основных классов и программных интерфейсов, реализованных и используемых во FRIS. Приложения В, Г содержат копии свидетельств о государственной регистрации программы – языкового сервиса FRIS. Приложения Д-Ж содержат копии актов внедрения программы в РФЯЦ-ВНИИЭФ, ОАО НПК КБМ и АО КБП им. академика А.Г. Шипунова. Приложение З содержит экспертное заключение с оценкой качественных и количественных характеристик от использования FRIS в РФЯЦ-ВНИИЭФ.

Глава 1. Разработка моделей языкового сервиса FRIS

Перечислим задачи, которые должен решить языковой сервис FRIS для учёта специфики написания современных сложноструктурированных программ:

- предоставлять контекстно-зависимую помощь, в которой, помимо определения элемента языка программирования, должно присутствовать его смысловое описание;
- иметь встроенную поддержку внешних общеиспользуемых библиотек (для РФЯЦ-ВНИИЭФ это УРС-ОФ и ЕФР);
- иметь встроенную поддержку современных средств параллельного программирования: MPI, OpenMP и SIMD-операций;
- обеспечивать визуальное выделение элементов указанных библиотек и средств распараллеливания;
- работать в процессе написания текстов программ.

Рассмотрим разработанные модели, которые обеспечивают реализацию указанных требований.

Вначале определим место, которое занимает языковой сервис в процессе создания программного обеспечения. Классической моделью такого процесса является модель водопада, или каскадная модель [82]. Она состоит следующих процессов, или компонент, следующих друг за другом (рисунок 1.1).

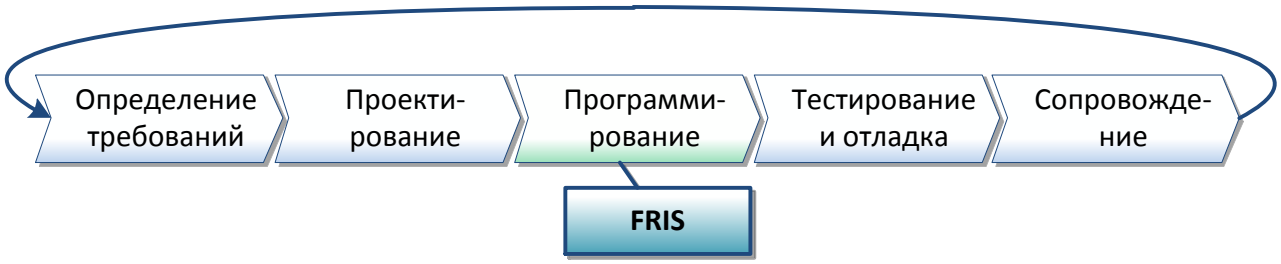

Рисунок 1.1 - Место FRIS в процессе создания программного обеспечения

Языковой сервис используется в процессе «программирование» жизненного цикла программного обеспечения.

1.1 Абстрактная модель языкового сервиса для расширенной поддержки языка программирования

Реализовать указанные требования возможно с использованием абстрактной (общей) модели языкового сервиса [69,71], обеспечивающей расширенную поддержку языка программирования. Модель не привязана к какому-либо конкретному языку программирования и интегрированной среде разработки. Языковой сервис может быть представлен в виде 5-ти основных блоков (рисунок 1.2). Стрелками обозначены обмены данными между блоками.

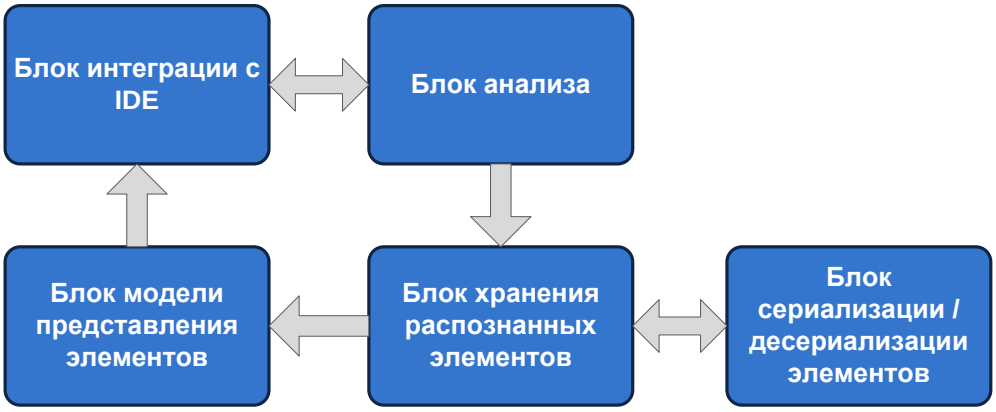

Рисунок 1.2 – Абстрактная модель языкового сервиса для расширенной поддержки языка программирования

Блок интеграции с IDE содержит реализацию программных интерфейсов, необходимых для взаимодействия с IDE. Он отвечает за подписку языкового сервиса на события редактирования текста пользователем в редакторе и за соответствующие отклики, например, за подсветку синтаксиса и предоставление информации для работы возможностей технологии IntelliSense.

Блок анализа отвечает за проведение лексического, синтаксического, семантического анализа. Получая от блока интеграции с IDE те или иные события, он выполняет соответствующие действия.

Блок хранения распознанных элементов является центральным хранилищем данных обо всех элементах, необходимых для работы языкового сервиса. В общем случае он является разновидностью таблицы символов. Наполнение блока хранения может вестись из двух источников: из блока анализа как результат разбора файлов с текстами про-

грамм и из блока сериализации/десериализации элементов в случае использования описания API для сторонних библиотек в виде XML-представления.

Блок сериализации/десериализации элементов выполняет две функции. Во-первых, он позволяет сохранять содержимое программных проектов в виде XML-файлов описания API и комментариев документирования к ним. Во-вторых, он позволяет восстанавливать содержимое программных проектов из их XML моделей.

Блок модели представления элементов является связующим звеном, своеобразным адаптером элементов блока хранения к тому виду, который необходим для использования в блоке интеграции с IDE. Так, распознанные элементы могут содержать некоторую информацию, которая не требуется функциям технологии IntelliSense, или наоборот, не содержать нужной информации. В модели представления элементов содержатся типы данных – адаптеры для элементов блока хранения, соответствующие требованиям блока интеграции с IDE. Также здесь реализуются всевозможные функции выборки и поиска необходимой информации. Можно сравнить блок хранения с базой данных, а блок модели представления с процедурами выборки необходимой информации.

Дадим формализованное[4] описание абстрактной модели языкового сервиса. Здесь и далее при обсуждении грамматики языка будем пользоваться устоявшимися определениями, данными в работах [50-54].

Модель состоит из следующих (множеств) элементов:

- множества программных проектов – P (*project*);
- множества файлов исходного кода – F (*file*);
- множества событий – E (*event*);
- множества действий – A (*action*);
- грамматики – G (*grammar*)$=(T,N,R,S)$, где:
  - T – конечное непустое множество *терминальных символов*, или *токенов*[5];
  - N – конечное непустое множество *нетерминальных символов*[6], или синтаксических переменных;

[4] Формализованное описание дано в терминах теории множеств. Указаны множества и их взаимные отображения

[5] Терминальный символ, токен или лексема – элементарные символы языка программирования.

[6] Нетерминал или нетерминальный символ – множество строк [54] терминалов.

  - *R* – множество *продукций,* или *правил вывода*, каждая из которых состоит из нетерминала, называемого заголовком, или левой частью, стрелки и последовательности терминалов и/или нетерминалов, называемых телом, или правой частью продукции;
  - *S* – нетерминальный символ, указываемый как стартовый или начальный;
- множества значимых элементов – *I* (*important*);
- множества элементов представления – *V* (*view*);
- множества цветов для визуального выделения текста – *C* (*color*);
- множества унифицированных элементов описания API – *U* (*unified*);
- множества смысловых описаний элементов – *D* (*definition*);
- множества шаблонов кода – *B* (*boilerplate text*).

Определение. *Программный проект* – совокупность файлов метаданных, файлов исходного кода, а также правил для их сборки и отладки.

Определение. *Файл исходного кода* – текстовый файл, расположенный на внешнем запоминающем устройстве и содержащий текст программы на целевом языке программирования.

Определение. *Событие* – некое свершившееся действие: нажатие клавиши, выполнение команды и тому подобное.

Определение. *Действие* – последовательность исполняемых операций, предназначенных для достижения какой-либо (определённой) цели.

Определение. *Значимый элемент* – именованный элемент языка программирования, определённый в соответствующем стандарте, который можно использовать в программе по его имени и/или псевдониму.

Определение. *Элемент представления* – элемент, задающий конкретный шаблон отображения пользователю каких-либо значимых элементов.

Определение. *Цвет для визуального выделения текста* – код цвета текста, начертания и цвета фона в доступной палитре цветов ЭВМ, которым возможно выделять текст в редакторе.

Определение. *Унифицированный элемент описания API* – элемент, позволяющий задать синтаксическое определение значимого элемента в терминах специального унифицированного расширяемого метаязыка, не являющегося языком программирования.

Определение. *Смысловое описание элемента* – элемент, позволяющий задать для значимого элемента описание его предназначения, использования и другой важной информации в терминах специального унифицированного расширяемого метаязыка, не являющегося языком программирования.

Определение. *Шаблон кода* – стандартный текст с выделенными позициями для заполнения изменяемым текстом. Множество позиций для заполнения изменяемым текстом в шаблоне будем обозначать *H* (*holes*).

Введём формальные описания основных свойств и отношений элементов модели.

Определение. Каждый элемент $e \in \mathrm{Z} = \{W \cup P \cup F \cup E \cup A \cup G \cup T \cup N \cup I \cup V \cup U \cup D \cup B\}$ является аннотированным, то есть содержит множество пар $(имя, значение)$ атрибута.

Определение. Множество допустимых аннотаций $\mathrm{A} = \mathrm{Z} \times \mathrm{S} \mapsto \mathrm{D}$ зададим как отображение декартова произведения множества элементов $e \in \mathrm{Z}$ и строки из множества строк S, задающей имя атрибута, во множество допустимых значений атрибутов D. Для удобства записи получение значения $v \in \mathrm{D}$ атрибута с именем $n \in \mathrm{S}$ у элемента $e \in \mathrm{Z}$ будем обозначать $v = e.n$.

Определение. Введём оператор $\mathrm{P}(X)$, обозначающий получение множества всех подмножеств произвольного множества *X*.

Введём отношение зависимости проектов друг от друга.

Определение. Отображение $dep : P' = \mathrm{P}(P) \mapsto P'' = \mathrm{P}(P)$ задаёт зависимости проектов друг от друга, при этом $\forall p \in P' : p \notin P''$ и $\forall p \in P'' : p \notin P'$, то есть проекты не зависят сами от себя и не содержат циклических зависимостей. Т.е. для каждого отдельно взятого проекта *p* множества P', P'' не пересекающиеся.

Определение. Отображения $pf : \mathrm{P}(P) \mapsto \mathrm{P}(F)$ и $fp : \mathrm{P}(F) \mapsto \mathrm{P}(P)$ задают отношения принадлежности файлов программным проектам. При этом первое позволяет получить все файлы, относящиеся к заданному программному проекту, а второе - программный проект, относящийся к данному файлу.

Определение. Отображение $fi : \mathrm{P}(F) \mapsto \mathrm{P}(I)$ задаёт соответствие между файлом исходного кода и содержащимися в нём распознанными значимыми элементами.

Определение. Отображение $ft : \mathrm{P}(F) \mapsto \mathrm{P}(\mathrm{S})$ задаёт соответствие между файлом исходного кода и его текстом.

Определение. Отображение $analyze : ft(\mathrm{P}(F)) \times G \mapsto \mathrm{P}(I)$ задаёт соответствие между текстом файла исходного кода и распознаваемыми при помощи грамматики G значимыми элементами, содержащимися в файле.

Определение. Отображение $tokenyze : \mathrm{P}(\mathrm{S}) \times G \mapsto \mathrm{P}(T)$ задаёт соответствие между указанным текстом и распознаваемыми при помощи грамматики G токенами.

Определение. Отображение $tc : \mathrm{P}(T) \mapsto \mathrm{P}(C)$ задаёт соответствие токена и цвета для его визуального выделения или обозначения в редакторе среды разработки.

Определение. Отображение $vc : \mathrm{P}(V) \mapsto \mathrm{P}(C)$ задаёт соответствие элемента представления и цвета для его визуального обозначения во всплывающих списках контекстной помощи.

Определение. *Файловой позицией* будем называть кортеж $\langle line, column \rangle$ (строка в файле, номер символа в строке).

Определение. Отображение $fs : \mathrm{P}(\langle line, column \rangle) \mapsto \mathrm{P}(\mathrm{S})$ задаёт соответствие файловой позиции и текста строки, содержащей эту файловую позицию.

Определение. Отображение $ii : \mathrm{P}(I) \mapsto \mathrm{P}(I)$ задаёт соответствие между значимым элементом и всеми содержащимися в нём значимыми элементами. Иными словами, отображение определяет элементы, являющиеся областями видимости (т.е. контейнерами для других элементов).

Определение. Отображение $context : \mathrm{P}(\langle line, column \rangle) \mapsto \mathrm{P}(I)$ задаёт соответствие между указанной файловой позицией и всеми видимыми в ней значимыми элементами. Иными словами, это отображение позволяет определить текущий контекст.

Определение. Отображение $typecontext : \mathrm{P}(\langle line, column \rangle) \mapsto \mathrm{P}(I)$ задаёт соответствие между указанной файловой позицией, являющейся окончанием селектора сложного объекта, и всеми видимыми значимыми элементами, являющимися компонентами указанного объекта. Иными словами, это отображение позволяет определить контекст типа данных.

Определение. Отображение $procedurecontext : \mathrm{P}\left(\langle line, column\rangle\right) \mapsto \mathrm{P}\left(I\right)$ задаёт соответствие между указанной файловой позицией, являющейся вызовом процедуры, и всеми значимыми элементами, являющимися формальными аргументами данной процедуры. Иными словами, это отображение позволяет определить контекст процедуры.

Определение. Отображение $elementcontext : \mathrm{P}\left(\langle line, column\rangle\right) \mapsto \mathrm{P}\left(I\right)$ задаёт соответствие между указанной файловой позицией, являющейся селектором объекта, и всеми значимыми элементами, являющимися определениями данного объекта. Иными словами, это отображение позволяет найти определение значимого элемента или элементов (если язык поддерживает перегрузку объектов).

Определение. Отображение $view : \mathrm{P}\left(I\right) \mapsto \mathrm{P}\left(V\right)$ задаёт соответствие между значимыми элементам и элементами представления.

Определение. Отображение $hole : \mathrm{P}\left(B\right) \mapsto \mathrm{P}\left(H\right)$ задаёт соответствие шаблона кода и всех позиций с изменяемым текстом в нём.

Определение. Отображение $template : \mathrm{P}\left(\mathrm{S}\right) \mapsto \mathrm{P}\left(B\right)$ задаёт соответствие между именем шаблона и самим шаблоном кода.

Определение. Отображение $templatetext : \mathrm{P}\left(B\right) \mapsto \mathrm{P}\left(\mathrm{S}\right)$ задаёт соответствие между шаблоном кода и его текстом.

Определение. Отображение $cpiud : \mathrm{P}\left(P\right) \times \mathrm{P}\left(I\right) \mapsto \mathrm{P}\left(U\right) \times \mathrm{P}\left(D\right)$ задаёт соответствие между всеми значимыми элементами проекта и их эквивалентным представлением в виде множества унифицированных элементов описания API и смысловых описаний.

Определение. Отображение $cudpi : \mathrm{P}\left(U\right) \times \mathrm{P}\left(D\right) \mapsto \mathrm{P}\left(P\right) \times \mathrm{P}\left(I\right)$ задаёт соответствие между унифицированными элементам описания API и их смысловым описанием и соответствующим им значимыми элементами и проектом, которому они принадлежат.

Множество событий (таблица 1.1) можно разделить на четыре непересекающиеся множества, относящиеся к концептуальным типам операций пользователя:

1) однократные действия E^{single};
2) изменение структуры проекта $E_{change}^{project}$;
3) редактирование текста файла исходного кода E_{edit}^{text};

4) явный вызов команд (например, пункты меню) $E^{explicit}_{command}$.

Таблица 1.1. Перечень событий

Множество событий	Событие	Описание действий породивших событие
однократные действия E^{single}	**загрузка среды разработки**	пользователь запустил среду разработки
изменение структуры проекта $E^{project}_{change}$	**загрузка проекта**	проект загружен средой разработки
изменение структуры проекта $E^{project}_{change}$	**добавление нового проекта**	пользователь добавил в рабочую область новый проект
	удаление проекта	пользователь удалил из рабочей области существующий проект
	изменение имени проекта	пользователь переименовал проект
	добавление нового файла в проект	пользователь добавил в проект новый файл исходного кода
	удаление файла из проекта	пользователь удалил существовавший файл исходного кода из проекта
	изменение имени файла	пользователь переименовал файл
	добавление зависимости проекта от другого проекта	пользователь установил признак зависимости проектов
	удаление зависимости проекта от другого проекта	пользователь удалил признак зависимости проектов
редактирование текста файла исходного кода E^{text}_{edit}	**открытие файла**	файл исходного кода открыт в редакторе
	добавление символа	пользователь ввёл символ с клавиатуры
	удаление символа	пользователь удалил символ из текста
явный вызов команд $E^{explicit}_{command}$	**построение списка элементов сложного объекта**	вызов команды для построения списка элементов объекта, находящегося слева от каретки
	отображение сведений о параметрах процедур	вызов команды для отображения справки о параметрах вызываемой процедуры в процессе написания вызова процедуры
	отображение кратких сведений об элементе языка программирования	вызов команды отображения всплывающей подсказки о символе под курсором
	автодополнение	вызов команды для построения списка элементов, которые можно использовать в данном контексте (точки программы)
	переход к определению	вызов команды для перехода к определению элемента, находящегося под курсором
	вставка шаблона кода	вызов команды для вставки шаблона кода в указанное место программы
	сохранение проекта в виде унифицированного представления	вызов команды для сохранения указанного проекта в виде его эквивалентного представления

В приведённой таблице содержится перечень наиболее важных событий, однако этот список может быть дополнен как новыми событиями, так и новыми типами событий. Тем не менее для работы языкового сервиса данный список можно считать исчерпывающим.

Необходимо отметить, что каждому событию ставится в соответствие некое действие (реакция или отклик), которое будет проводиться в ответ на событие.

Определение. Отображение $ea : \mathrm{P}(E) \mapsto \mathrm{P}(A)$ задаёт соответствие действий порождающим их событиям.

Рассмотрим множество типовых действий абстрактной модели (таблица 1.2) и их соответствий множеству событий (таблица 1.3). Отметим, что все необходимые параметры получаются в действиях из атрибутов порождающих их событий.

Таблица 1.2. Множество типовых действий абстрактной модели языкового сервиса

Действие	Описание
загрузить список файлов проекта	получить подмножество файлов проекта; добавить их во множество файлов; построить зависимости между проектом и данными файлами
добавить проект во множество проектов	добавить проект во множество проектов; если проект содержит файлы – загрузить список файлов проекта
удалить проект из множества проектов	получить при помощи отображения pf все файлы проекта; получить при помощи отображения fi все значимые элементы для подмножества файлов; удалить полученные значимые элементы из множества значимых элементов; удалить полученное множество файлов из множества файлов; удалить проект из множества проектов; удалить все зависимости, в которых участвует данный проект
изменить имя проекта	изменить значение атрибута *name* у проекта
добавить файл в проект	добавить файл во множество файлов; построить зависимость между файлом и проектом
удалить файл из проекта	получить при помощи отображения fi все значимые элементы для файла; удалить полученные значимые элементы из множества значимых элементов; удалить файл из множества файлов; удалить связь между файлом и проектом
изменить имя файла	изменить значение атрибута *name* у файла
добавить зависимость проекта от указанного проекта	добавить зависимость проекта от указанного проекта
удалить зависимость проекта от указанного проекта	удалить зависимость проекта от указанного проекта
открыт файл	при помощи отображения *analyze* получить множество значимых элементов, для указанного файла; добавить значимые элементы в множество значимых элементов; установить связь между файлом и значимыми элементами; при помощи отображения ft получить текст файла исходного кода; при помощи отображения *tokenyze* получить множество токенов для указанного текста; при помощи отображения tc получить цвета для визуального выделения токена в редакторе

Таблица 1.2. Продолжение

изменён текст	при помощи отображения *fs* получить текст стоки, в которой произошли изменения, и текст всех оставшихся строк до конца файла; при помощи отображения *tokenyze* получить множество токенов для указанного текста; при помощи отображения *tc* получить цвета для визуального выделения токена в редакторе
приготовить элементы к отображению	при помощи отображения *view* привести указанные значимые элементы к виду, необходимому для отображения пользователю
построить список элементов сложного объекта	при помощи отображения *typecontext* получить элементы, являющиеся компонентами типа данных; **приготовить элементы к отображению**
отобразить параметры процедуры	при помощи отображения *procedurecontext* получить элементы, являющиеся формальными аргументами процедуры; **приготовить элементы к отображению**
найти элемент	при помощи отображения *elementcontext* получить определения всех элементов, чьё имя находится в указанной позиции; **приготовить элементы к отображению**
построить контекст	при помощи отображения *context* получить значимые элементы, видимые в указанной файловой позиции; **приготовить элементы к отображению**
перейти к определению	при помощи отображения *elementcontext* получить определения всех элементов, чьё имя находится в указанной позиции; выполнить переход (при необходимости, открыть нужный файл; спозиционироваться на начало имени элемента)
вставить шаблон кода	при помощи отображения *template* получить шаблон с заданным именем; при помощи отображения *templatetext* получить текст шаблона и вставить его в редактор; при помощи отображения *hole* получить все изменяемые части шаблона и произвести их редактирование
инициализация	при помощи отображения *cudpi* получить проекты и их значимые элементы, соответствующие внешним библиотекам; поместить значимые элементы во множество значимых элементов; поместить проекты во множество проектов
сохранить проект в унифицированном виде	при помощи отображения *cpiud* получить множество унифицированных элементов описания API и смысловых описаний для указанных проекта и его значимых элементов

Таблица 1.3. Соответствие действий событиям

Событие	Действие
загрузка среды разработки	**инициализация**
загрузка проекта	**загрузить список файлов проекта**
добавление нового проекта	**добавить проект во множество проектов**
удаление проекта	**удалить проект из множества проектов**
изменение имени проекта	**изменить имя проекта**
добавление нового файла в проект	**добавить файл в проект**
удаление файла из проекта	**удалить файл из проекта**
изменение имени файла	**изменить имя файла**
добавление зависимости проекта от другого проекта	**добавить зависимость проекта от указанного проекта**
удаление зависимости проекта от другого проекта	**удалить зависимость проекта от указанного проекта**
открытие файла	**открыт файл**

Таблица 1.3. Продолжение

добавление символа	**изменён текст**
удаление символа	**изменён текст**
построение списка элементов сложного объекта	**построить список элементов сложного объекта**
отображение сведений о параметрах процедур	**отобразить параметры процедуры**
отображение кратких сведений об элементе языка программирования	**найти элемент**
автодополнение	**построить контекст**
переход к определению	**перейти к определению**
вставка шаблона кода	**вставить шаблон кода**
сохранение проекта в виде унифицированного представления	**сохранить проект в унифицированном виде**

Отметим, что при описании действий в таблицах используются уже введённые определения и отображения (зависимости) между элементами модели.

Таким образом, рассмотрены все значимые элементы абстрактной модели языкового сервиса и взаимосвязи между ними.

Опишем (таблица 1.4) принадлежность элементов и отношений блокам языкового сервиса и связям между блоками абстрактной модели языкового сервиса (рисунок 1.2).

Таблица 1.4. Принадлежность элементов и отношений модели её структурным блокам

Структурный элемент абстрактной модели	Элементы и отношения абстрактной модели
Блок интеграции с IDE	множество программных проектов – *P*; множество файлов исходного кода – *F*; множество событий – *E*; множество действий – *A*; множество шаблонов кода – *B*; отображение *dep*; отображение *pf*; отображение *fp*; отображение *ft*; отображение *fs*; отображение *hole*; отображение *template*; отображение *templatetext*; отображение *ea*
Блок анализа	грамматика – *G*
Блок хранения распознанных элементов	множество значимых элементов – *I*; отображение *fi*; отображение *ii*
Блок модели представления элементов	множество элементов представления – *V*; множество цветов для визуального выделения текста – *C*
Блок сериализации/десериализации элементов	множество унифицированных элементов описания API – *U*; множество смысловых описаний элементов – *D*
Отношение между блоком интеграции с IDE и блоком анализа	отображение *analyze*; отображение *tokenyze*; отображение *tc*

Таблица 1.4. Продолжение

Отношение между блоком интеграции с IDE и блоком модели представления элементов	отображение *vc*
Отношение между блоком анализа и блоком хранения распознанных элементов	отображение *context*; отображение *typecontext*; отображение *procedurecontext*; отображение *elementcontext*
Отношение между блоком хранения распознанных элементов и блоком модели представления элементов	отображение *veiw*
Отношение между блоком хранения распознанных элементов и блоком сериализации/десериализации элементов	отображение *cpiud*; отображение *cudpi*

Поскольку абстрактная модель *оперирует абстрактными понятиями*, которые не привязаны к конкретному языку программирования и среде разработки, она без потери общности может быть использована для разработки языковых сервисов для различных языков программирования (Fortran, C/C++, C#, Java, Python, Ruby, Pascal и др.) и интегрированных сред разработки (Microsoft Visual Studio, Eclipse, NetBeans, CodeBlocks и др.). Абстрактная модель отражает общую схему построения языкового сервиса для расширенной поддержки языка программирования и те возможности, которыми будет обладать языковой сервис в случае реализации данной модели.

Действительно, многие интегрированные среды разработки разделяют понятия проект, файл, событие и действия над ними, а многие языки программирования разделяют концепцию абстрактных элементов — элемента языка программирования и области видимости, которыми оперирует абстрактная модель языкового сервиса. Поэтому, хотя для конкретного языка программирования и среды разработки реализация блоков модели будет своей, сами блоки и их взаимосвязи останутся неизменными, поскольку не накладывают ограничений на внутреннее устройство.

Блок интеграции со средой разработки абстрактной модели отвечает не только за взаимодействие со средой разработки в части реализации программных интерфейсов по её расширению, но и за абстрагирование других блоков модели от необходимости знать конкретные детали реализации такой интеграции. Таким образом, для интеграции с различными средами разработки в абстрактной модели языкового сервиса будет меняться только один или два блока – блок интеграции со средой разработки и, возможно, блок модели представления элементов (о чём будет сказано далее).

Блок анализа будет присутствовать, поскольку первичным источником информации об элементах любого языка программирования являются тексты программ на дан-

ном языке. Поэтому для получения этой первичной информации необходимо проводить лексический, синтаксический и семантический анализ, за что и отвечает блок анализа.

Блок хранения распознанных элементов также всегда будет присутствовать, поскольку он является тем местом, где производится хранение полученной информации. Было бы нелогично проводить анализ текстов программ и сразу же избавляться от результатов анализа, т.е. нигде их не запоминать.

Блок сериализации и десериализации элементов необходим для всех языковых сервисов, которым нужен дополнительный источник информации, например, для обеспечения помощи для библиотек, реализованных на других языках программирования. Понятно, что в данном случае, даже при наличии исходных текстов программ библиотеки, провести их анализ при помощи существующего блока анализа не представляется возможным, поскольку используются различные языки программирования. Так, невозможно провести анализ текста C++программы с использованием анализатора для языка программирования Fortran. Отметим ещё раз, что при сохранении (сериализации) элементов блок хранения используется в качестве источника данных, а при восстановлении элементов блок хранения используется в качестве приёмника данных.

Блок модели представления элементов также должен всегда присутствовать, поскольку каждая среда разработки может предъявлять свои специфические требования и обладать специфическими возможностями по предоставлению данных для отображения пользователю. Блок модели представления является связующим звеном между блоком интеграции со средой разработки и блоком хранения и отвечает за приведение данных, хранимых в блоке хранения, к виду, необходимому для работы блока интеграции со средой разработки и обусловленному, в свою очередь, требованиями самой среды разработки.

Всё вышеперечисленное доказывает принципиальную применимость абстрактной модели языкового сервиса для построения языковых сервисов для различных языков программирования и интегрированных сред разработки.

Рассмотрим конкретизацию данной модели применительно к языку программирования Fortran 2003, позволяющую удовлетворить требования по учёту специфики написания современных сложноструктурированных программ, изложенные в начале данной главы.

1.2 Оценка сложности реализации намеченных требований

В оценке сложности реализации намеченных требований необходимо выделить два основных этапа:

- выявление источников данных, которые необходимо использовать при реализации языкового сервиса;
- анализ сложности реализации намеченных функций.

1.2.1 Анализ источников данных

Прежде чем выявлять источники данных, нужно провести анализ тех данных, которые необходимы языковому сервису для предоставления всех требуемых пользователю возможностей. Это выполняется при разработке модели значимых элементов Fortran 2003 (см. пункт 1.3).

Наиболее очевидным способом получения определений элементов языка программирования является анализ текста программы. При этом синтаксическая форма такого определения фиксирована в стандарте языка программирования. Смысловое описание может быть получено, если дополнить тексты программ комментариями специального формата – комментариями документирования (см. пункт 1.4). Существует два наиболее распространённых вида комментариев документирования: XML [83] и Doxygen [84]. В Visual Studio стандартом считаются XML комментарии документирования. Таким образом, в тексте программы содержатся по меньшей мере два языка: основной – Fortran, и дочерний – язык комментариев документирования.

Будем называть *базовым языком* основной язык программирования, Fortran, а *дочерним* – специальный язык, который не входит в определение базового языка, но может использоваться совместно и непосредственно внутри базового языка. Помимо языка комментариев документирования, к дочерним языкам относятся, например, директивы OpenMP.

Отметим особенность, присущую Fortran при использовании внешних библиотек программ. Возможны три способа подключения библиотеки (для Windows это *.lib файл) к основному программному проекту (рисунок 1.3):

- с использованием файлов исходного кода, в которых описаны программные интерфейсы библиотеки, включая определения процедур, типов данных и т.д.;

- с использованием скомпилированных бинарных файлов Fortran-модулей. В этом случае используются файлы с закрытым форматом, понятным лишь компилятору, содержащие определения интерфейсов библиотеки;
- без использования описания программных интерфейсов библиотеки. В этом случае компилятор будет выводить внешние интерфейсы для используемых процедур и пытаться разрешить ссылки по их именам.

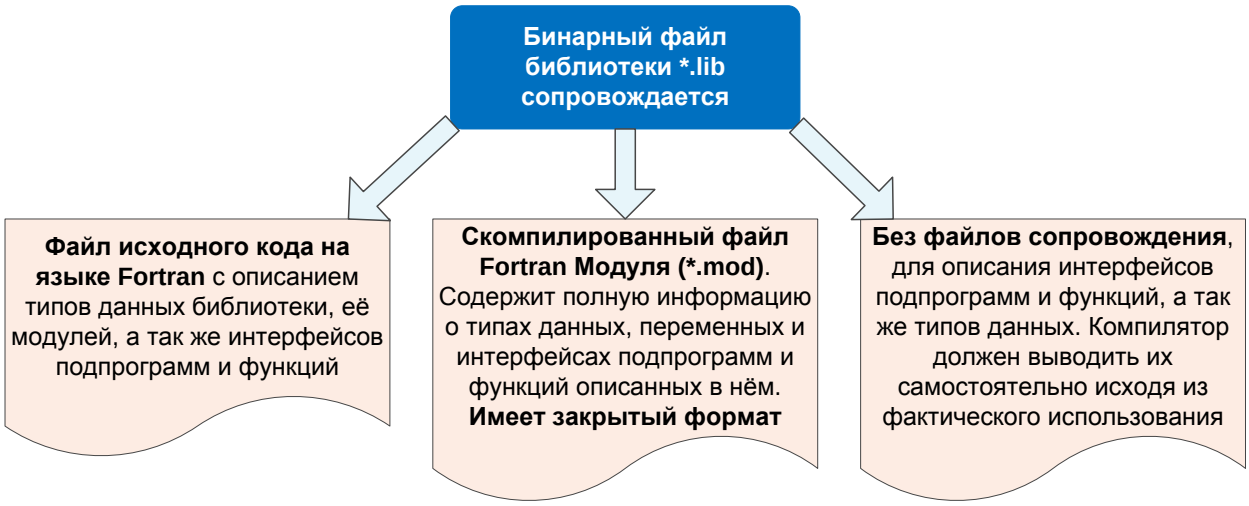

Рисунок 1.3 – Способы подключения библиотеки к Fortran программе

Если в первом случае можно провести анализ файла, содержащего описание интерфейсов библиотеки, и получить всю необходимую информацию из него, то в двух оставшихся это сделать уже невозможно и необходимо обеспечивать иные механизмы получения информации.

За основу при решении данной задачи была взята идея, используемая в программе автоматической генерации документации для так называемых управляемых приложений [85] – Sandcastle [83]. Для создания документации для программы используются два файла: один с описанием прикладных программных интерфейсов (Application Programming Interface – API), второй с описанием их документации.

Необходимо отметить, что существующий в Sandcastle формат описания API непригоден для языка Fortran, поскольку ориентирован на модели значимых элементов других языков. Поэтому во FRIS для этих целей была разработана модель описания Fortran API (см. пункт 1.5.1), представляющая собой XML-файл специального формата, в котором содержится описание основных элементов Fortran. Во FRIS имеется возможность как сохранения (сериализации) структуры элементов, полученной из анализа тек-

стов программ в XML-формат, так и восстановления (десериализации) элементов Fortran из их XML-представления.

Аналогично разработана XML модель комментариев документирования (см. пункт 1.5.2) для Fortran с возможностью её сериализации и десериализации. Это позволяет в дальнейшем реализовать специальный программный модуль расширения базовых возможностей Sandcastle и использовать файлы описания API и комментариев документирования Fortran для автоматической генерации документации пользователя и/или разработчика.

1.2.2 Анализ сложности реализации требуемых функций языкового сервиса

При реализации языкового сервиса необходимо учитывать, что анализ текстов программ необходимо проводить в режиме непосредственного его редактирования пользователем. Это означает, что в большинстве случаев анализируемый текст будет находиться в лексически, синтаксически или семантически некорректном состоянии с точки зрения спецификации языка программирования. Данную особенность необходимо учитывать при построении соответствующих анализаторов.

Вторая особенность заключается в том, что анализ для подсветки синтаксиса осуществляется в Visual Studio в построчном режиме. Анализатору, в терминах VS — колорайзеру, для анализа передаётся строка и состояние, в котором он находился в конце анализа предыдущей строки. Это означает, что соответствующий анализатор необходимо проектировать с учётом возможности сохранения своего состояния на произвольный момент времени и восстановления своей работы с любого такого состояния. Подобный подход позволяет проводить инкрементальный анализ, что особенно актуально для больших файлов с текстами программ (более 10000 строк). Тогда при изменении части строк необходимо провести анализ только этих строк, а не всего файла в целом.

Третья особенность, которую необходимо учитывать для построения эффективных полнотекстных анализаторов, заключается в учёте состояний файлов с текстами программ. С точки зрения использования файлов программного проекта в среде разработки файл может находиться в одном из 2-х состояний:

- открыт в редакторе;
- не открыт в редакторе.

В первом случае необходимо осуществлять полноценный разбор файлов, а во втором можно ограничиться упрощённым анализом для сбора информации только о ви-

димых извне программных элементах. Так, например, совершенно не обязательно проводить анализ всего тела процедур, поскольку информация, скажем, об их локальных переменных может понадобиться пользователю только в момент редактирования тела процедур. Это автоматически переведёт файл в состояние «открыт для редактирования», следовательно, к нему будут применены другие правила разбора. Таким образом, возможность функционирования анализаторов в двух режимах, условно «полного» и «упрощённого» анализа, позволит повысить скорость разбора текстов программного проекта.

1.3 Модель значимых элементов языка программирования Fortran

Дадим определение модели значимых элементов языка Fortran. ***Модель значимых элементов языка программирования Fortran*** описывает программные элементы данного языка, которые необходимо распознавать при анализе текста программы для предоставления контекстно-значимой помощи Fortran-программисту. Под контекстно-значимой помощью понимается не только предоставление всевозможных подсказок с краткими сведениями об элементе языка программирования, но и наполнение списков автодополнения с учётом места запроса помощи в файле текста программы.

Модель значимых элементов языка Fortran построена на основе проведённого автором анализа текста стандарта Fortran 2003 [6]. В качестве значимых элементов рассматривались именованные элементы Fortran, которые могут использоваться в программе по имени или псевдониму. Модель элементов языка Fortran, разработанная и реализованная во FRIS, учитывает вложенность областей видимости, описанную в стандарте. Данная модель является основополагающей, поскольку определяет содержание связанных с ней моделей — комментариев документирования и описания прикладных программных интерфейсов.

Вложенность областей видимости позволяет условно классифицировать все конструкции по уровню вложенности (рисунок 1.4).

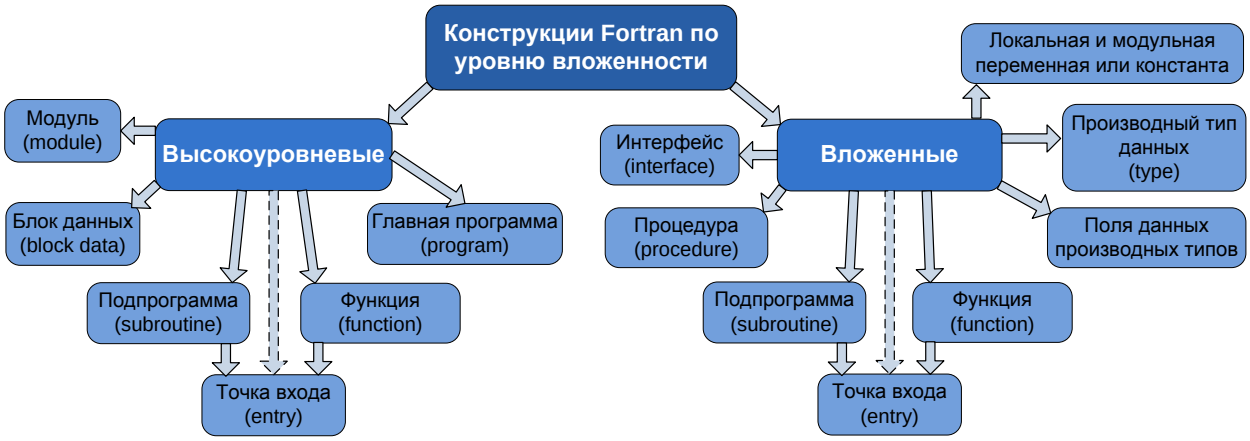

Рисунок 1.4 – Классификация конструкций Fortran по уровню вложенности

Поясним некоторые детали, отражённые на данном рисунке.

Во-первых, конструкция «блок данных» (block data) является необязательной и служит для инициализации начальными значениями содержимого common-блока. Отметим, что стандарт Fortran 2003 не рекомендуют использовать в новых программах common-блоки, так как для них существует более удобная альтернатива – модули (module).

Во-вторых, конструкция «точка входа» (entry) используется в подпрограммах и функциях для того, чтобы обеспечить им дополнительное имя вызова и зачастую иные аргументы. Эта конструкция также не рекомендуется к использованию в новых программах.

В-третьих, конструкция «процедура» (procedure) была введена в стандарте Fortran 2003 и обозначает, по сути, указатель на подпрограмму или функцию. Процедура обязана иметь лишь строго оговорённый интерфейс, но при этом она может быть настроена (ссылаться) на любую подпрограмму или функцию, имеющую данный интерфейс.

В Fortran понятие *интерфейс* используется исключительно по отношению к подпрограммам и функциям и служит для описания их сигнатуры. Под *сигнатурой* будем понимать описание аргументов и результирующего значения (для функций), включая их типы, имена и атрибуты.

В этом смысле необходимо отметить введение концепции «абстрактный интерфейс». Для её понимания нужно рассмотреть разницу между понятием «интерфейс», существовавшим в Fortran до выхода стандарта Fortran 2003. Ранее под интерфейсом понималось описание имени и аргументов *реально существующей внешней (глобальной)* подпрограммы или функции. Начиная со стандарта Fortran 2003, под *абстрактным ин-*

терфейсом понимается именно описание аргументов, их типов и атрибутов подпрограммы и функции, то есть то, что необходимо для её корректного вызова. *Имя абстрактного интерфейса используется лишь для его идентификации, то есть не существует внешней (глобальной) подпрограммы или функции с таким именем*. Введение абстрактных интерфейсов является необходимым условием для использования концепций объектно-ориентированного программирования, в частности для обеспечения работы механизма связанных с типом данных процедур и их дальнейшего переопределения в типах-наследниках (расширяющих типах) для базового типа.

В-четвёртых, под полями данных производного типа понимаются имена компонентов производного типа, которые не являются процедурами.

Особо стоит отметить роль, которую выполняют процедуры в производном типе данных. С введением концепций объектно-ориентированного программирования и процедур, обладающих абстрактным интерфейсом, тип данных получил возможность использовать два типа процедур:

1. *Процедуры-компоненты типа данных* являются указателями на подпрограммы и функции, могут быть многократно перенастроены во время выполнения программы. Типичный пример – использование подхода функций обратного вызова (или «событий»), когда при выполнении некоторого действия или изменения состояния необходимо уведомить об этом заинтересованного «подписчика».
2. *Связанные с типом данных процедуры* обеспечивают поведение типа данных и могут быть изменены только с использованием механизма наследования. Они уже не могут меняться во время выполнения программы, так как являются неотъемлемой частью объекта, его инфраструктурными элементами. В других языках программирования (например, C++, C#) – это методы класса.

Рассмотрим модель значимых элементов языка Fortran, использующуюся во FRIS (рисунок 1.5). Необходимо отметить, что данная модель содержит подавляющее большинство языковых конструкций, определённых стандартом Fortran 2003, поэтому позволяет предоставлять в дальнейшем наиболее объективную информацию о программном проекте. Сделаем важное замечание, касающееся удовлетворения требования учитывать специфику написания современных сложноструктурированных программ и предоставлять контекстно-зависимую помощь, в которой, помимо определения элемента языка программирования, должно присутствовать его смысловое описание. Для его реа-

лизации каждый значимый элемент содержит специальный атрибут для хранения текста смыслового описания своего предназначения. А за его получение отвечает модель комментариев документирования.

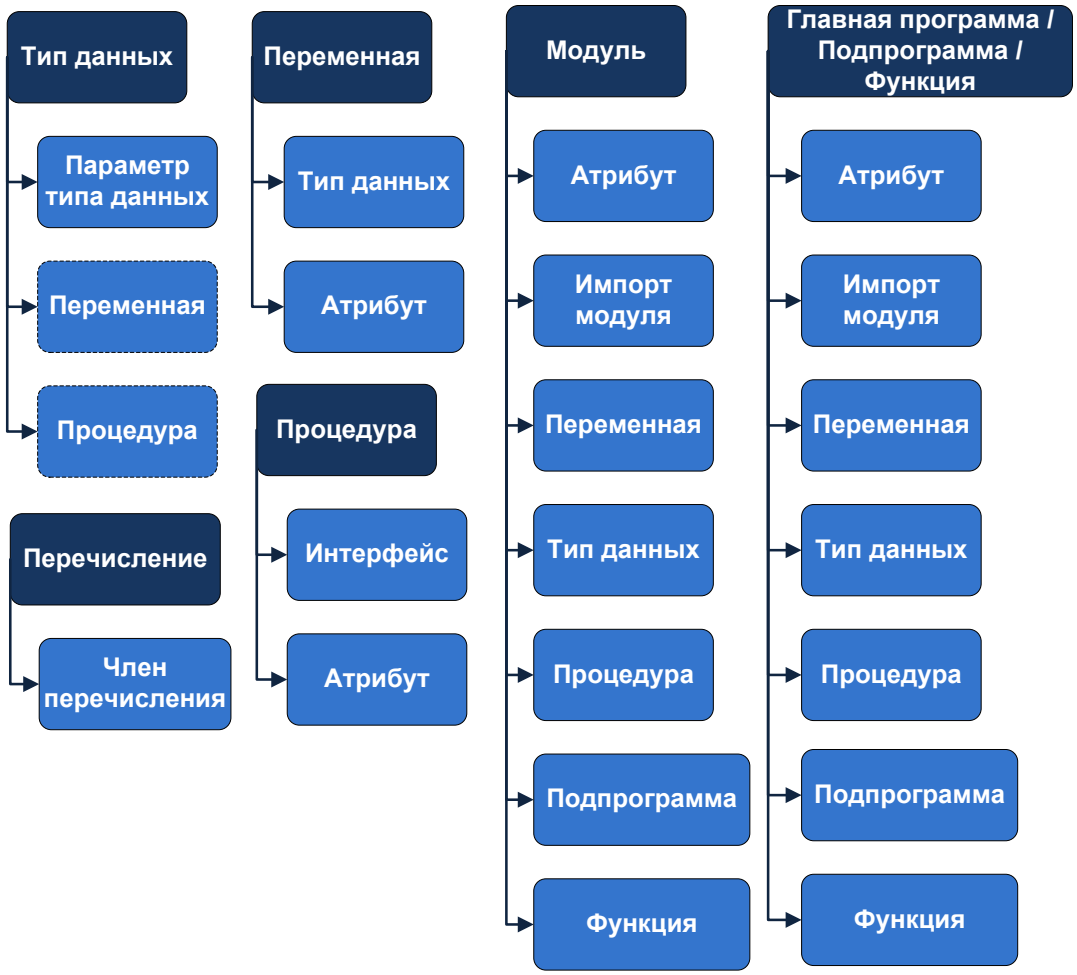

Рисунок 1.5 – Модель значимых элементов языка Fortran во FRIS

Сделаем важное замечание, касающееся ограничения именования различных объектов. Каждое имя в Fortran-программе должно быть уникальным (с несколькими оговорками для специальных случаев). Отсюда, в частности, следует, что в Fortran не допускается использования привычного для программистов на C++ способа перегрузки процедур (в C++ процедуры имеют одно имя, но отличаются списком и типами аргументов).

Модуль является наиболее интересным элементом с точки зрения построения Fortran-программ. Он может выполнять несколько функций:

1. Может служить для целей совместного использования данных – то есть быть контейнером данных, который постоянно хранит их в оперативной памяти во время ра-

боты программы и позволяет работать с ними всем потребителям модуля (как замена для common-блоков).

2. Может выступать в качестве поставщика интерфейсов для набора подпрограмм и функций. В этом качестве модули используют, например, реализации стандартов параллельного программирования MPI [23] и OpenMP [24].
3. Являться базовым инфраструктурным элементом для объектно-ориентированного программирования. Это обусловлено реализацией объектно-ориентированного программирования в Fortran. Согласно стандарту, расширяемые производные типы данных (классы в терминологии C++) могут содержаться только в модулях. Все присоединённые к типу данных процедуры должны содержаться внутри того же модуля, что и производный тип данных.

Fortran допускает всего два уровня доступа к любым объектам (5.1.2.1 в [6]): общий (public) и частный (private). При этом контроль доступа также осуществляется в рамках модуля. То есть внутри модуля все его объекты являются общедоступными, а вне его, доступ к ним регламентируется атрибутом доступа.

Ещё один дополнительный атрибут контроля использования объектов модуля, который может применяться к модульным переменным, включая процедуры, – это атрибут защищённого использования (protected, 5.1.2.12 в [6]). Он означает, что значение переменной может быть изменено только модульными подпрограммами и функциями, а вне модуля значение такой переменной доступно в режиме «только для чтения». Подключение модуля выполняется при помощи операции импортирования модуля – инструкция (statement) use.

Рассмотрим уровень подпрограмм и функций. Заметим, что главная программа (program), подпрограмма (subroutine) и функция (function) имеют практически полностью эквивалентную структуру, с небольшими оговорками. Главная программа не принимает аргументов и не возвращает результата, также у неё не может быть нескольких точек входа. Различие между подпрограммой и функцией заключается в том, что подпрограмма не возвращает значения, а функция – должна возвращать какое-либо значение как результат своей работы.

Стандарт Fortran 2003 выделяет следующие типы подпрограмм и функций (по месту их определения):

1) глобальные – определены в глобальной области видимости;

2) внешние – определены в глобальной области видимости, внешней по отношению к текущему программному проекту. Например, внешними являются подпрограммы и функции из используемых в проекте библиотек программ, написанных, возможно, на другом языке программирования;
3) модульные – определены внутри модуля;
4) внутренние – определены внутри глобальных или модульных подпрограмм и функций. Отметим, что стандарт допускает только один уровень вложенности для внутренних подпрограмм и функций. Это означает, что у внутренних подпрограмм и функций не может быть своих внутренних подпрограмм и функций;
5) интерфейсные – определены внутри блока описания интерфейса, в том числе абстрактного.

Как уже отмечалось ранее, Fortran, начиная со стандарта Fortran 2003, поддерживает объектно-ориентированное программирование. В этой связи производные типы данных Fortran претерпели существенные изменения по сравнению с предыдущим стандартом Fortran 95:

1. Производный тип данных стал параметризованным. В некотором смысле это позволяет использовать подобие шаблонизации. Параметры производного типа данных, по аналогии со встроенным типом, бывают двух разновидностей:
   1.1. Параметр размерности (kind) – используется для задания модели представления данных в памяти.
   1.2. Параметр длины (len) – используется для указания длины массивов.
2. Производный тип приобрёл поля процедуры, являющиеся указателями на подпрограммы и функции с заданным интерфейсом. В Fortran стало возможным использовать аналог указателей на функции из C/C++.
3. Производный тип приобрёл связанные с ним процедуры (ближайшая аналогия – это методы классов в C++). Стандарт допускает использование следующих 3-х типов процедур:
   3.1. Специализированные (specific) – имя связанной процедуры соответствует строго одной модульной подпрограмме или функции.
   3.2. Обобщённые или родовые (generic) – задают одно имя для множества подпрограмм и функций, отличающихся своими сигнатурами. Это основной способ перегрузки операций в Fortran. Помимо обычного имени, здесь можно определить

свой бинарный или унарный оператор или переопределить одну из встроенных операций (присваивание, сложение, умножение и т.д.).

3.3. Финализации (final) – аналог деструкторов в C++. Данная подпрограмма вызывается при уничтожении объекта в оперативной памяти для очистки занятых им ресурсов.

4. Производный тип стал расширяемым – стало возможным строить цепочки наследования типов (в терминологии Fortran – расширения типов). То есть объявлять базовые типы, в том числе и абстрактные (экземпляры которых нельзя создавать), и, потом, используя специальный синтаксис, расширять их (т.е. наследоваться от них).

Заключительный блок – интерфейс. По типу описываемых объектов интерфейсы бывают:

1) *специальными* – описывают имя и сигнатуру глобальных подпрограмм и функций;
2) *обобщёнными* или *родовыми* – объединяют подпрограммы и функции под одним именем. Служат для перегрузки подпрограмм и функций, а также для определения/переопределения унарных и бинарных операторов;
3) *абстрактными* – описывают лишь сигнатуру подпрограммы и функции. Имя интерфейса служит лишь для его уникальной идентификации, а не указывает на реально существующую подпрограмму или функцию.

Отметим связь разработанной во FRIS модели значимых элементов с самим языком программирования Fortran. Первое, на что необходимо обратить внимание при анализе любого языка программирования, это то, что его стандарт задаёт модель, используемую в каждом конкретном языке. К модели языка программирования относятся:

- множество базовых элементов и понятий – *абстракций или абстрактных объектов языка программирования*;
- множество правил оперирования с ними, *включая правила порождения новых (абстрактных) элементов из уже существующих*.

Таким образом, языки программирования обладают порождающим и системообразующим свойством, поскольку на основании чётко определённых базовых элементов и согласно строго определённым правилам оперирования позволяют строить сколь угодно сложные конструкции, обладающие свойствами, которые отсутствуют у каждой из составляющих новое целое частей по отдельности.

Чтобы относиться к данному языку программирования, программа должна *соответствовать (conform)* данному языку, то есть использовать его модель — множество базовых элементов и правил оперирования с ними. Если программа не удовлетворяет данным критериям, она не может называться программой на данном языке программирования.

Таким образом, чтобы успешно выполнять анализ программ какого-либо языка программирования, необходимо использовать его модель. Отметим, что *модель языка программирования* охватывает *все возможные, допустимые в нём операции и элементы*.

Модель значимых элементов FRIS является подмножеством модели языка программирования Fortran. К непосредственному предмету, рассматриваемому FRIS во время анализа текста программы, относятся элементы данных и области видимости. Обозначенные границы предметной области обусловлены тем, что помощь программисту необходимо оказывать по производным от базовых, в терминах модели языка программирования, элементам. К таким производным элементам относятся структуры данных, включая переменные, и исполняемые конструкции многоразового использования (функции и подпрограммы). Анализ же исполняемых инструкций производится в разрезе их синтаксической и частично семантической корректности, но поскольку FRIS не является ни компилятором, ни интерпретатором, то создавать модель для них и хранить их не представляется необходимым.

Из сказанного следует, что модель значимых элементов FRIS, учитывающая все аспекты областей видимости и элементов данных Fortran, является подмножеством модели языка программирования Fortran стандарта Fortran 2003 и в этой части предоставляет взаимно-однозначное, эквивалентное отображение абстракций своей модели и абстракций модели языка Fortran.

1.4 Модель комментариев документирования

Модель обеспечивает предоставление смыслового описания для элемента языка программирования. В качестве возможных рассматривались две наиболее распространённых модели и подхода к созданию комментариев документирования:

1. Комментарии в стиле Doxygen [84].
2. XML комментарии в стиле Visual Studio / Sandcastle [83].

Комментарии в стиле Doxygen представляют собой многострочный блок текста, где каждая строка начинается с ключевого тэга документирования и далее содержит его

текст. Фактически наличие ключевого тэга управляет интерпретацией следующего за ним текста. Заметим, что для Fortran ключевых тэгов не предусмотрено вовсе, что означает фактически использование только общего описания без дальнейшего его структурирования. В этом смысле модель комментариев Doxygen для Fortran уступает моделям этого же инструмента для других языков программирования, например C++.

XML комментарии в стиле Visual Studio / Sandcastle представляют собой многострочный блок текста, состоящий из XML-тэгов и их содержимого. Данный подход накладывает единственное ограничение: блок текста должен быть допустимым с точки зрения спецификации XML. Дополнительным преимуществом является возможность интеграции таких комментариев с инструментом создания автономной документации – Sandcastle. Поэтому данный формат был выбран во FRIS.

Поскольку документация создаётся для значимых конструкций Fortran и является по своей сути их дополнением, то при разработке модели документации уделялось особое внимание её соответствию модели значимых элементов.

Комментарий в Fortran начинается с символа восклицательного знака «!». Во FRIS для обозначения комментария документирования используется последовательность из 3-х восклицательных знаков «!!!», за которой может следовать один или несколько XML-тэгов документирования:

```
!!!<ИмяТэга Атрибут1="Значение атрибута">
!!!Текст комментария…
!!!</ ИмяТэга>
```

Допустимые в настоящее время тэги документирования показаны на рисунке 1.6, однако, благодаря использованию формата XML, в любое время можно ввести в модель новые тэги для поддержки новых возможностей.

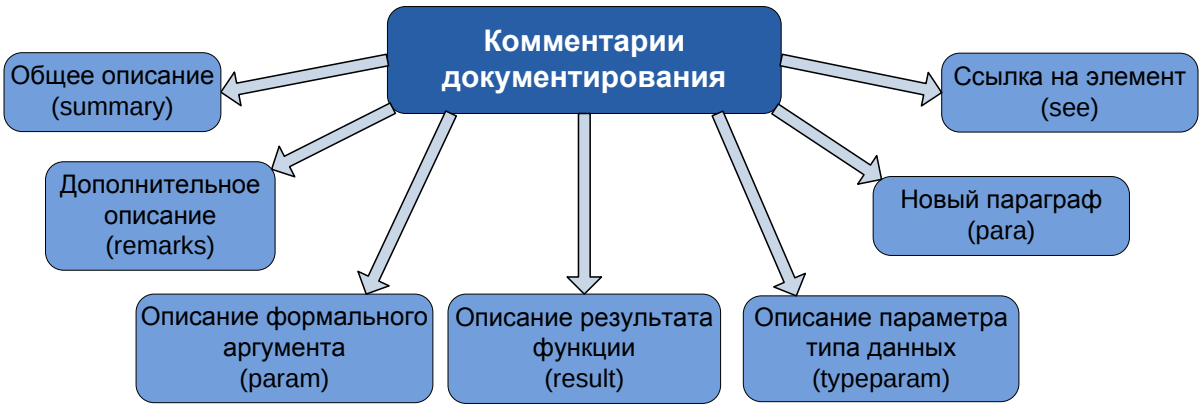

Рисунок 1.6 – Модель комментариев документирования

Все тэги можно разделить по группам, использующимся для описания различных значимых элементов Fortran (см. рисунок 1.7). Подробное описание тэгов приведено в таблице 1.5.

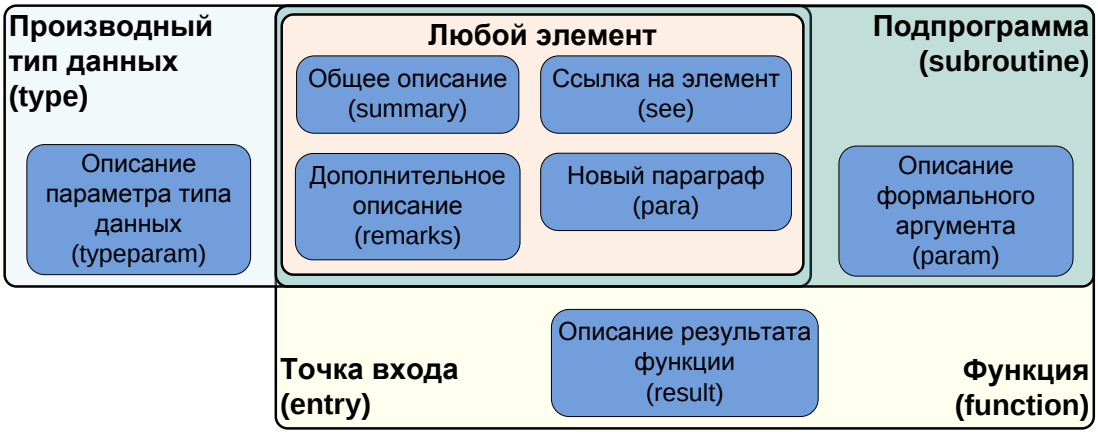

Рисунок 1.7 – Модель комментариев документирования с учётом модели значимых элементов Fortran

Таблица 1.5. Описание тэгов комментариев документирования

Тэг	Атрибуты	Описание	Пример
summary	-	служит для создания общего описания для любого элемента	!!!<summary>переменная для хранения типа методики</summary> integer(4) :: iMethod
remarks	-	служит для создания дополнительного описания (заметок) для элемента. Текст не включается в выводимые подсказки	!!!<remarks>Возможные значения: -1 – не указан, 0 – стандартный, …</remarks> integer(4) :: iMethod
param	name	служит для создания описания для формального аргумента подпрограммы и функции. Имя аргумента указывается как значение атрибута *name*	!!!<param name=”a”> Номер канала !!!для печати диагностики !!!</param> subroutine sub1(a)
result	-	служит для создания описания для возвращаемого значения функции	!!!<result> !!!Истина – если функция завершилась успешно, иначе – ложь. !!!</result> function fun() result(lVal)
typeparam	name	служит для создания описания для параметра параметризованного производного типа данных. Имя параметра указывается как значение атрибута *name*	!!!<typeparam name=”k”> !!!Размерность полей данных !!!</typeparam> type :: MyType(k)
para	-	служит для создания нового параграфа в каком-либо из тэгов. Т.е. текст, заключённый в данный тэг, будет выведен с новой строки	!!!<summary> !!!Внимание! !!!<para>Изменение данного значения запрещено!</para> !!!</summary> integer(4) :: iErrorCode

Таблица 1.5. Продолжение

see	cref	служит для создания ссылки на какой-либо именованный значимый элемент Fortran. Имя элемента указывается в качестве значения атрибута *cref*. Данный тэг предназначен в основном для целей генерации справочной документации	!!!<summary> !!!Переменная производного типа !!!<see cref=”MyType”/>. !!!</summary> type(MyType(4)) :: element

Таким образом, все тэги можно разделить на следующие группы (таблица 1.6).

Таблица 1.6. Классификация тэгов документирования

Группа	Подгруппа	Тэги	Описание
По уровню использования	**Высокоуровневые**	summary, remarks, param, result, typeparam	не могут быть вложены в любые другие тэги
	Вложенные	para, see	не могут использоваться вне других тэгов
По назначению	**Информационные**	summary, remarks, param, result, typaram, see	служат контейнерами для основной информации
	Форматирования	para	служат для целей форматирования текста других тэгов
По времени использования	**Времени разработки программного проекта**	summary, param, result, typeparam, para	предназначены для использования во время написания текста программы, для получения оперативной информации по значимым элементам Fortran
	Создания документации программного проекта	remarks, see	предназначены преимущественно для включения в систему документации, создаваемой при помощи специальных средств на основе текстов проекта

Необходимо отметить, что для тэгов *param* и *typeparam* необходимо проводить проверку на соответствие значений их атрибутов *name* фактическим именам формальных аргументов и параметров типов. Например, если подпрограмма не содержит формального аргумента с именем *arg1*, но он указан в качестве значения атрибута *name* тэга *param*, то следует трактовать это как ошибку в комментарии документирования. Подобного рода контроль на соответствие документации документируемым элементам позволит предоставлять Fortran-программисту актуальную информацию и своевременно предупреждать его о допущенных ошибках при документировании текста программы.

Далее приведена формальная спецификация[7] комментариев документирования, поддерживаемых во FRIS:

comment :

summary //должно встречаться только один раз

*param** //только для подпрограмм и функций, описание формальных аргументов

result? //только для функций, описание возвращаемого значения

*typeparam** //только для параметризованных производных типов данных

remarks? //для заметок

summary : <summary> (*text* | *innerTag*)* </summary>

remarks : <remarks> (*text* | *innerTag*)* </remarks>

param : <param name=”*name*”> (*text* | *innerTag*)* </param>

result : <result> (*text* | *innerTag*)* </result>

typeparam : <typeparam name=”*name*”> (*text* | *innerTag*)* </typeparam>

text : любой текст

innerTag : *para* | *see*

para : <para> (*text* | *see*)* </para>

see : <see cref=”*name*”/>

name : имя именованного элемента

1.5 Модель расширенной поддержки внешних библиотек программ

Модель обеспечивает реализацию следующих требований по учёту специфики написания современных сложно структурированных программ:

- иметь встроенную поддержку внешних, общеиспользуемых библиотек;
- иметь встроенную поддержку современных средств параллельного программирования: MPI, OpenMP и SIMD-операций.

Модель расширенной поддержки программных библиотек, реализованная во FRIS, состоит из следующих взаимодополняющих частей, модели:

- описания прикладных программных интерфейсов (API) библиотеки;
- описания документации для элементов библиотеки;

[7] При описании правил используются следующие обозначения: «*» — обозначает повторение правила стоящего слева ноль и более раз; «+» — повторение правила стоящего слева один и более раз; «?» — повторение правила стоящего слева ноль или один раз.

- визуального выделения элементов библиотеки в файле исходного кода.

1.5.1 Модель описания прикладных программных интерфейсов (API) библиотеки

Модель описания прикладных программных интерфейсов (API) библиотеки используется в ситуациях, когда её программные интерфейсы не описаны в соответствующих файлах исходного кода.

Подчеркнём глубинную взаимосвязь модели значимых элементов, рассмотренной в пункте 1.3, и модели API библиотеки или программного проекта. Она основывается на том, что обе модели работают с одной и той же предметной областью – элементами языка программирования Fortran, используемыми в программном проекте. Модель описания API выступает эквивалентом (эквивалентным представлением или проекцией) и внутренне полностью соответствует модели значимых элементов Fortran.

В основе разработанной модели описания прикладных программных интерфейсов лежит модель, используемая в инструменте генерации документации Sandcastle. Она выбрана ввиду того, что, во-первых, хорошо проработана и является относительно универсальной; во-вторых, её использование впоследствии упростит процесс использования Sandcastle для автоматического создания документации программных проектов.

Отметим, что данная модель в дальнейшем позволит решить и обратную задачу по описанию API к Fortran-программам для их использования из других языков программирования (при помощи специализированного средства преобразования Fortran API, скажем, в C API).

Рассмотрим общую структуру файла описания API (который является выражением модели значимых элементов из пункта 1.3 в виде тэгов языка XML). Подробная аннотированная XSD (XML Schema Definition) [86] схема всех его элементов приведена в Приложении А. Здесь будут рассмотрены примеры для описания некоторых наиболее важных элементов (рисунок 1.8, листинг 1.1-1.2, таблица 1.7-1.9).

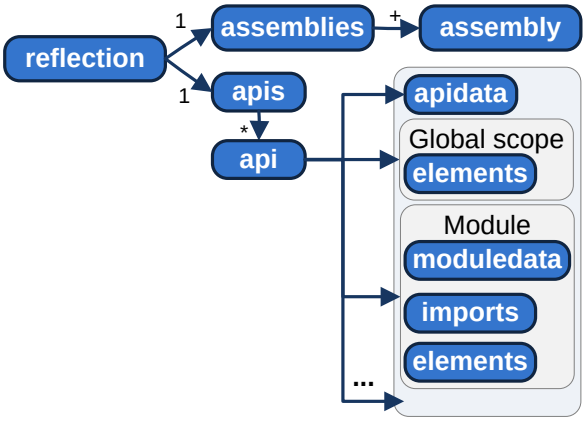

Рисунок 1.8 – Модель Fortran API

Таблица 1.7. Описание тэгов модели Fortran API

Тэг	Описание
reflection	корневой элемент. Состоит из assemblies (сборки) и apis (прикладные программные интерфейсы)
assemblies	служит для указания имён всех программных проектов (библиотек), информация об интерфейсах которых содержится в данном файле
assembly	позволяет указать имя библиотеки или программного проекта – атрибут name, а также язык программирования, на котором данная библиотека реализована, – атрибут language
apis	служит для непосредственного описания прикладных программных интерфейсов. Его элементы составляют основное содержание файла
api	используется для описания любого значимого элемента. Для идентификации элементов используется их полностью квалифицированное имя и префикс в формате: <Префикс>:<Область Видимости 1>.<Область Видимости 2>….<Область Видимости N>.<Имя Элемента> Полное описание префиксов имён приведено в таблице 1.9
apidata	позволяет более подробно классифицировать элемент при помощи своих атрибутов: • name – имя элемента; • group – основная группа элемента; • subgroup – подгруппа элемента; • subsubgroup – подподгруппа элемента (зарезервирована, сейчас не используется). Полный список элементов и соответствующих им групп приведён в таблице 1.10
elements	список имён элементов, которые принадлежат текущему элементу
element	служит для описания одного элемента. Содержит идентификатор элемента в области видимости
methoddata	содержит ряд общих атрибутов для всех методов: подпрограмм и функций
functiondata	данные о функции; конкретизирует, что метод является функцией
parameters	параметры, формальные аргументы – содержит перечень формальных аргументов
parameter	описывает формальный аргумент. Атрибуты: • name (имя) – служит для указания имени аргумента; • intent (цель) – позволяет задать использование аргумента «in» - входной, «out» - выходной, «inout» - входной и выходной; • optional (опциональный, необязательный) - указание, что аргумент является необязательным
variabledata	указывает, что описывается переменная
type	указывает тип данных. Атрибут api – содержит полностью квалифицированное имя типа данных. В скобках указываются параметры типа данных

Таблица 1.7. Продолжение

result	описывает возвращаемое функцией значение. Имеет тот же смысл, что и parameter. Стандарт Fortran позволяет задавать произвольное имя для результата функции, поэтому в элементе присутствует атрибут name. Если данный атрибут не задан, то считается, что имя переменной результата функции совпадает с именем функции
containers	служит для указания обратной связи, т.е. того элемента, которому принадлежит данный элемент
library	служит для указания конкретной библиотеки, которой принадлежит данный элемент

```
<?xml version="1.0" encoding="utf-8"?>
<!--Общее описание информации о прикладных программных интерфейсах (API) -->
<reflection>
  <!--Указываем имена библиотек, описываемых в данном файле-->
  <assemblies>
    <!--Описание библиотеки/проекта: имя и язык программирования-->
    <assembly name="intrinsics" language="Fortran"/>
  </assemblies>
  <!--Секция программных интерфейсов. Содержит описания для каждого значимого элемента.
Все описания с этой точки зрения являются равноправными и присутствуют в этой секции как
элементы верхнего уровня-->
  <apis>
    <!--Как правило, сначала указывается содержимое глобальной области видимости. Здесь
указывается только признак, что это область имён-->
    <api id="N:">
      <!-- Конкретизация, что это за элемент. Указывается условное имя, группа - простран-
ство имён, подгруппа - область видимости.-->
      <apidata name="Global" group="namespace" subgroup="scope"/>
      <!--Для области видимости указывается перечень её элементов-->
      <elements>
        <!--Указываем полностью квалифицированное имя элемента-->
        <element api="M:ACHAR(I, [KIND])"/>
      </elements>
    </api>
  </apis>
</reflection>
```

Листинг 1.1 – Пример Fortran API для встроенных процедур

Таблица 1.8. Перечень префиксов имён элементов.

Буква	Описание
N	глобальная область видимости или модуль.
M	подпрограмма или функция.
T	тип данных.
V	переменная
P	процедура
I	обобщённый интерфейс
R	ссылка
S	ассоциация уровня хранения данных (Storage Association): для namelist-ов и common-блоков

```
<!--Описание интерфейса функции-->
<api id="M:ACHAR(I, [KIND])">
  <!--Указываем общую информацию об элементе -->
  <apidata name="ACHAR" group="method" subgroup="function" />
  <!--указываем что это элементарный метод-->
  <methoddata elemental="true"/>
  <!--Указываем, что это функция-->
  <functiondata />
  <!--Описываем параметры функции-->
  <parameters>
    <!--Имя, использование - только входной-->
    <parameter name="I" intent="in">
      <!--Указываем, что это переменная-->
      <variabledata/>
      <!--Указываем тип данных. Целочисленный, с размерностью по умолчанию-->
      <type api="T:integer"/>
    </parameter>
    <!--Имя, использование - только входной, необязательный-->
    <parameter name="KIND" intent="in" optional="true">
      <!--Указываем, что это переменная-->
      <variabledata/>
      <!--Указываем тип данных. Целочисленный, с размерностью по умолчанию-->
      <type api="T:integer"/>
    </parameter>
  </parameters>
  <!--Указываем имя результирующего значения-->
  <result name="ACHAR">
    <!--Указываем, что это переменная-->
    <variabledata/>
    <!--Указываем тип данных. Символьный, единичной длины, с размерностью KIND -->
    <type api="T:character(len=1,kind=KIND)"/>
  </result>
  <!—Указываем, где содержится элемент, т.е. обратную связь-->
  <containers>
    <!--Имя библиотеки и модуля. Они совпадают-->
    <library assembly="intrinsics"/>
    <!--Имя базового элемента. Т.к. глобальное - то ничего не указываем-->
    <element api="N:"/>
  </containers>
</api>
```

Листинг 1.2 – Пример Fortran API для встроенной функции ACHAR

Таблица 1.9. Классификация элементов по группам.

Элемент	Разновидность	Группа	Подгруппа
глобальная область видимости	-	namespace	global
модуль	-	namespace	module
ссылка	-	reference	-
переменная	общий случай	variable	-
	поле типа данных	variable	member
тип данных	-	type	-
процедура	общий случай	procedure	-
	поле типа данных	procedure	member
	связанная с типом специфическая	procedure	specific
	связанная с типом обобщённая	procedure	generic

Таблица 1.9. Продолжение

процедура	связанная с типом финальная	procedure	final
интерфейс	оператор	interface	operator
	присваивание	interface	assign
	ввода/вывода	interface	io
подпрограмма	-	method	subroutine
функция	-	method	function
namelist	-	storage	namelist
common-блок	-	storage	commonblock

1.5.2 Модель описания документации для элементов библиотеки

Модель описания документации (Fortran Doc) для элементов внешней библиотеки выступает полным эквивалентом модели комментариев документирования. Она так же предоставляет дополнительное смысловое описание для значимых элементов библиотеки. Отметим, что модель описания документации включает в себя все теги документирования, рассмотренные в пункте 1.4. Необходимо подчеркнуть её связь и с моделью значимых элементов, поскольку документация относится к конкретно взятому значимому элементу языка программирования.

Рассмотрим структуру модели документации (рисунок 1.9) на примере встроенной функции ACHAR (листинг 1.3, таблица 1.10).

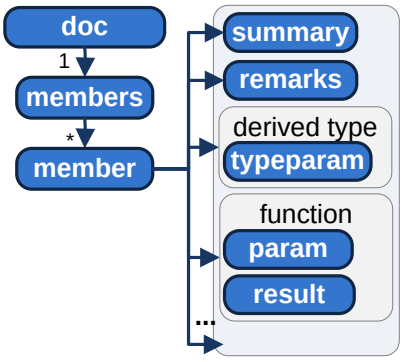

Рисунок 1.9 – Модель Fortran Doc

```
<?xml version="1.0" encoding="utf-8" ?>
<doc>
  <!-- Опционально указываем имя библиотеки -->
  <assembly>
    <name>intrinsics</name>
  </assembly>
  <!-- Указываем все элементы, для которых есть документация -->
  <members>
    <!-- Документация для одного элемента -->
    <!-- 13.7.2 ACHAR (I [, KIND]) -->
    <member name="M:ACHAR(I, [KIND])">
      <summary>
        Возвращает символ, находящийся в указанной позиции в схеме упорядочения ASCII.
        <para>Это инверсия значения функции IACHAR. </para>
      </summary>
      <param name="I"> Номер позиции, типа integer </param>
      <param name="KIND">
        Необязательный параметр, размерность типа возвращаемого значения
      </param>
      <result>Символ character(len=1,kind=[kind])</result>
    </member>
  </members>
</doc>
```

Листинг 1.3 – Описание документации для встроенной функции ACHAR

Таблица 1.10. Описание тэгов модели Fortran Doc

Тэг	Описание
doc	корневой элемент
members	контейнер для всех элементов документации
member	содержит документацию для одного элемента
summary	краткое описание элемента
remarks	дополнительная информация
see	внутренний тэг, создаёт ссылку на указанный элемент
para	внутренний тэг, создаёт параграф в родительском тэге
typeparam	описывает параметр производного типа данных
param	описывает аргумент подпрограммы или функции
result	описывает результирующие значение функции

Данный формат выбран, потому что он наиболее наглядно отражает структуру документации; прост и понятен для человеческого восприятия; совместим с форматом комментариев, создаваемых Visual Studio (для других языков программирования) и потребляемых потом инструментом создания документации – Sandcastle.

1.5.3 Модель визуального выделения элементов библиотеки

Модель визуального выделения элементов библиотеки в файле исходного кода относится к расширенной поддержке произвольных библиотек, подключаемых к проекту. Она является дополнением модели значимых элементов Fortran и модели описания API библиотеки. Её назначение заключается в визуальном выделении – обозначении цветом элементов библиотеки в тексте программы. Это позволяет акцентировать внима-

ние пользователя на том, что данные элементы являются в некотором смысле особенными. Модель состоит из следующих компонентов (таблица 1.11).

Таблица 1.11. Компоненты модели визуального выделения

Компонент	Описание
Имя модели	соответствует имени подключаемой библиотеки
Набор цветов для выделения значимых элементов	вообще говоря, каждый значимый элемент (см. пункт 1.3) может быть обозначен своим уникальным цветом. Набор цветов позволяет при необходимости определить для каждого класса значимых элементов свой уникальный цвет. К наиболее важным классам элементов можно отнести: • модули; • производные типы данных; • подпрограммы; • функции; • интерфейсы.
Список значимых элементов библиотеки, которые нужно выделять цветом	список всех общедоступных элементов библиотеки. Задаётся исходя из способа её подключения к программному проекту. Можно выделить два основных способа: • Задание списка файлов исходного кода, в которых содержится описание программного интерфейса библиотеки. Этот способ используется, если подключение библиотеки производится с помощью файлов исходного кода. При этом список элементов получается исходя из анализа указанных файлов. • Задание файла модели описания API библиотеки и документации. Этот способ используется при подключении библиотеки с помощью бинарных файлов, включая способ подключения библиотеки без каких-либо файлов поддержки вообще.

Выводы к главе 1

1. Перечислены задачи, которые должен решить языковой сервис FRIS для учёта специфики написания современных сложноструктурированных программ.
2. Разработана абстрактная модель языкового сервиса, обеспечивающая расширенную поддержку языка программирования и предназначенная для построения языковых сервисов для различных языков программирования и интегрированных сред разработки. Конкретизация данной модели для языка программирования Fortran 2003 и среды разработки Microsoft Visual Studio позволяет решить задачи по учёту особенностей написания современных сложноструктурированных программ. Модель состоит из 5-ти типовых блоков:
   - блока интеграции со средой разработки, реализующего интерфейсы, необходимые для встраивания языкового сервиса в целевую IDE. Он отвечает за подписку языкового сервиса на события редактирования текста пользователем в редакторе и за соответствующие отклики, абстрагирует остальные части языкового сервиса от конкретных деталей работы IDE;

- блока анализа, отвечающего за проведение анализа текстов на целевом языке программирования и сбор необходимой информации как для построения эквивалентной структуры программы, так и для подсветки синтаксиса;
- блока хранения распознанных элементов, являющегося централизованным хранилищем всех распознанных элементов, полученных как при анализе текстов программ, так и при работе с XML-описаниями API и документации для сторонних библиотек программ;
- блока сериализации (сохранения) и десериализации (восстановления) элементов, отвечающего за работу с XML-представлением API и документации как сторонних библиотек программ, так и текущего программного проекта;
- блока модели представления элементов, отвечающего за приведение элементов блока хранения к виду, необходимому блоку интеграции с IDE для предоставления необходимой информации для работы возможностей технологии IntelliSense, подсветки синтаксиса и т.д.

3. Разработана модель значимых элементов языка программирования Fortran 2003, позволяющая строить эквивалентное представление программы в виде дерева значимых элементов в оперативной памяти для использования в качестве источника информации для различных возможностей технологии IntelliSense. Модель отражает динамически меняющуюся в процессе редактирования пользователем структуру Fortran программы и обеспечивает удовлетворение требования по учёту специфики написания современных сложноструктурированных программ в части предоставления контекстно-зависимой помощи, в которой, помимо определения элемента языка программирования, должно присутствовать его смысловое описание.
4. Разработана модель описания прикладных программных интерфейсов Fortran 2003, служащая как для описания Fortran API внешней библиотеки программ, так и для описания Fortran API программного проекта. Данная модель принципиально позволяет решить и обратную задачу по описанию API к Fortran программам для их использования из других языков программирования. Модель обеспечивает удовлетворение требования по обеспечению встроенной поддержки средств распараллеливания MPI, OpenMP и SIMD-операций.
5. Разработана модель XML комментариев документирования для Fortran 2003, позволяющая программисту писать документацию к элементам программы непосредст-

венно в тексте программы и получать в дальнейшем данное описание в различных видах контекстной помощи. Также данная модель вместе с моделью описания Fortran API позволяет разрабатывать системы автоматической генерации документации разработчика и пользователя библиотеки или программы. Модель обеспечивает удовлетворение требования по учёту специфики написания современных сложноструктурированных программ в части встроенной поддержки MPI, OpenMP и SIMD-операций; предоставления контекстно-зависимой помощи, в которой, помимо определения элемента языка программирования, должно присутствовать его смысловое описание.

6. Разработана модель расширенной поддержки сторонних библиотек программ, основанная на использовании языка XML, позволяющая выполнять анализ программы с использованием любых средств обработки структурированных XML-данных, состоящая из модели описания Fortran API и модели XML комментариев документирования. Модель обеспечивает удовлетворение требования по учёту специфики написания современных сложноструктурированных программ в части встроенной поддержки внешних библиотек программ.

Глава 2. Программная реализация языкового сервиса FRIS

2.1 Общая схема языкового сервиса FRIS

Языковой сервис FRIS имеет сложную структуру, которая является специализированной реализацией абстрактной модели языкового сервиса (см. пункт 1.1, рисунок 1.2), и учитывает обозначенную в 1-й главе специфику написания современных сложно структурированных программ. Организация FRIS представлена на рисунке 2.1.

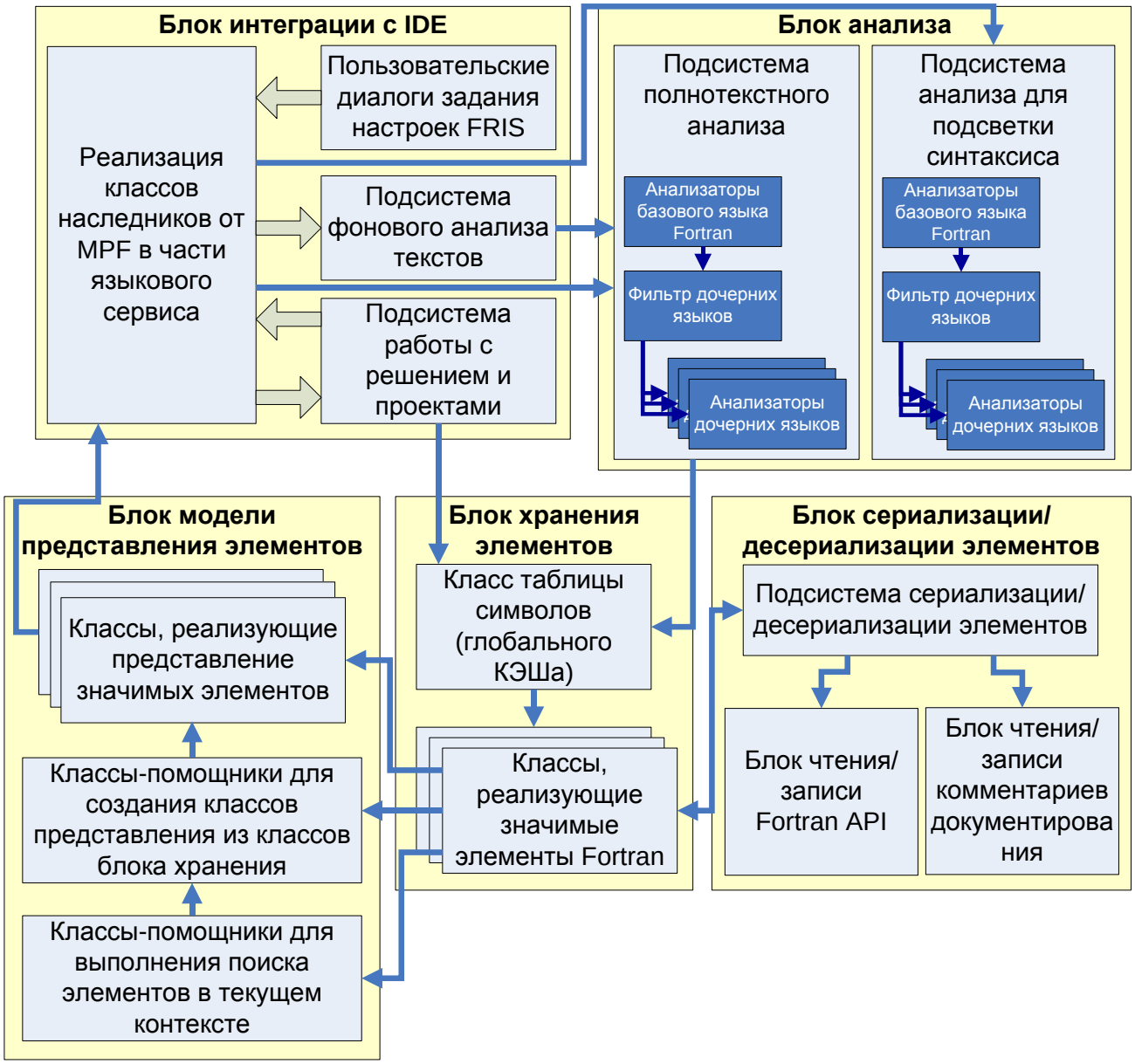

Рисунок 2.1 – Организация языкового сервиса FRIS

Основная часть блока интеграции с IDE заключается в реализации классов-наследников от базовых классов Managed Package Framework (MPF) [87] в части языко-

вого сервиса (см. пункт 2.2), которые обеспечивают взаимодействие с IDE и отклики на различные события: редактирования текста (подсветка синтаксиса и полнотекстный анализ), выбора пунктов меню и других.

Пользовательские диалоги предназначены для настройки работы языкового сервиса (рисунок 2.2, таблица 2.1).

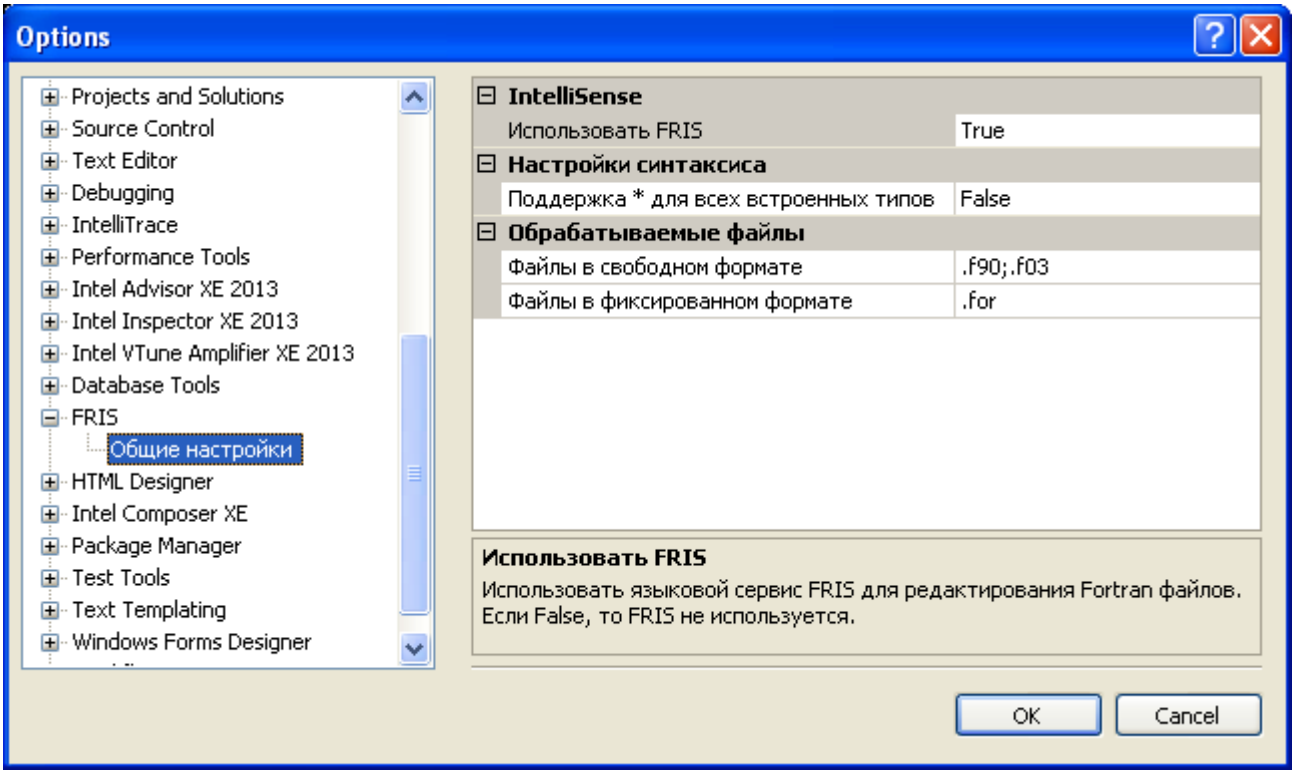

Рисунок 2.2 – Общие настройки FRIS

Таблица 2.1. Описание общих настроек FRIS

Управляющее значение	Описание
Секция IntelliSense	
Использовать FRIS	Основной управляющий параметр: True/False – использовать/не использовать FRIS в качестве языкового сервиса для языка программирования Fortran
Секция Настройки синтаксиса	
Поддержка * для всех встроенных типов	Указывает **FRIS**, что нужно поддерживать нестандартные объявления переменных встроенных типов: **type_name *** целое число **type_name: integer**, **real**, **logical**, **complex** Принимает одно из двух значений: True/False – использовать нестандартные/стандартные объявления встроенных типов данных

Таблица 2.1. Продолжение

Секция Обрабатываемые файлы	
Файлы в свободном формате	Список расширений файлов исходного кода в свободном формате. Принимает строку, в которой содержится перечисление расширений файлов интерпретируемых как файлы Fortran в свободном формате. Расширения перечисляются через символ «;». Например: .f90;.f03
Файлы в фиксированном формате	Список расширений файлов исходного кода в фиксированном формате. Принимает строку, в которой содержится перечисление расширений файлов интерпретируемых как файлы Fortran в фиксированном формате. Расширения перечисляются через символ «;». Например: .for

Подсистема фонового анализа текстов программ предназначена для предварительного сбора информации обо всех файлах, входящих в состав всех программных проектов, осуществляемого при загрузке проекта средой разработки (см. пункт 2.3.3). Подсистема фонового анализа запускает анализаторы подсистемы полнотекстного анализа (входящие в блок анализа) для каждого файла, который необходимо обработать. Множество файлов для обработки задаётся фильтрами на расширения обрабатываемых файлов в пользовательском диалоге настроек FRIS (пункты: файлы в свободном формате, файлы в фиксированном формате).

Подсистема работы с проектами и решениями отвечает за работу с программными проектами и составляющими их файлами. Она отслеживает изменения, производимые с проектами: добавление/удаление/переименование файлов, настройка зависимостей проектов друг от друга. Она также отслеживает связи файлов программных проектов и их распознанных представлений в блоке хранения (таблице символов). Данная подсистема также хранит виртуальные программные проекты, соответствующие внешним библиотекам программ, полученным из блока сериализации/десериализации. Таким образом, обеспечивается реализация требований по учёту специфики написания современных сложно структурированных программ: иметь встроенную поддержку внешних проблемно ориентированных библиотек программ, для РФЯЦ-ВНИИЭФ – это библиотеки УРС-ОФ и ЕФР; иметь встроенную поддержку средств параллельного программирования MPI, OpenMP и SIMD-операций.

Блок анализа состоит из двух частей (см. пункт 2.3): подсистемы полнотекстного анализа (см. пункт 2.3.3) и подсистемы анализа для подсветки синтаксиса (см. пункт

2.3.4). Ключевым отличием этих подсистем является то, что подсветка синтаксиса выполняется постоянно (после каждого напечатанного символа), а полнотекстный анализ выполняется с большей периодичностью (определяется Visual Studio). Таким образом, для подсветки синтаксиса нужно использовать только лексические анализаторы (для обеспечения приемлемой скорости работы), с набором управляющих правил, в то время как полнотекстный анализ использует полную схему: лексический анализатор, препроцессор, синтаксический и семантический анализаторы. Результат анализа для подсветки синтаксиса сразу же применяется к тексту программы средствами самой Visual Studio, а результаты полнотекстного анализа в виде внутреннего представления дерева значимых элементов – заносится в таблицу символов, и регистрируются в подсистеме работы с проектами и решениями. Таким образом, обеспечивается реализация требований по учёту специфики написания сложных пактов программ: предоставлять контекстно-зависимую помощь, в которой помимо определения элемента языка программирования должно присутствовать его смысловое описание; обеспечивать визуальное выделение элементов указанных библиотек; работать в процессе написания текстов программ.

Блок хранения элементов предназначен для хранения и предоставления информации о значимых элементах (см. пункт 2.4). Он состоит из таблицы символов (глобального кэша элементов) и конкретных элементов (определённых в пункте 1.3). Блок наполняется из двух источников из подсистемы полнотекстного анализа, блока анализа и из подсистемы сериализации/десериализации элементов, блока сериализации/десериализации.

Блок сериализации/десериализации элементов предназначен для поддержки работы с внешними программными библиотеками, чьи исходные тексты недоступны. В своей работе использует два блока чтения и записи Fortran API (см. пункт 1.5.1), а также комментариев документирования (см. пункт 1.5.2). Он позволяет из внешних файлов XML специального формата, создавать элементы модели значимых элементов и помещать их в блок хранения и в подсистему работы с решениями и проектами из блока интеграции с IDE.

Блок модели представления значимых элементов (см. пункт 2.5) отвечает за предоставление данных для отображения пользователю с учётом текущего контекста (места в файле программы, откуда была запрошена помощь). Для каждого конкретного элемента блока хранения, блок модели представления содержит модель отображения пользова-

телю. Учёт контекстов реализован в специализированных алгоритмах поиска нужных элементов с учётом требований изложенных в разделе 16 стандарта Fortran 2003 [6].

Языковой сервис FRIS реализован на языке программирования C# [88] для платформ .NET Framework 3.5 SP1 и .NET Framework 4.0. Объём программирования с учётом библиотек поддержки составил порядка 350000 строк программного кода на языке программирования C#. Необходимо отметить, что подобный объём программирования обусловлен реализацией языкового сервиса для трёх версий Microsoft Visual Studio 2005/08/10. Таким образом, для одной версии Visual Studio объём программирования составляет ~ 120000 строк, из которых ~ 30000 строк занимает описание грамматик.

Языковой сервис FRIS зарегистрирован в реестре программ для ЭВМ, о чём получены соответствующие свидетельства [74-75] (Приложение В-Г).

2.2 Блок интеграции с IDE

2.2.1 Расширение Visual Studio и языковые пакеты. Место языкового сервиса в языковом пакете

Добавление новых функциональных возможностей в VS обозначается термином расширяемость Visual Studio (Visual Studio Extensibility) [89-90]. Ключевым блоком, при помощи которого производится введение в VS новых возможностей, является так называемый Пакет Visual Studio (***VSPackage***) (далее просто *пакет*). Пакет взаимодействует с ядром IDE практически напрямую и помогает решить любую задачу по её расширению. Отметим, что интеграции всех поставляемых Microsoft языков программирования, таких как Visual C++, Visual C#, Visual Basic реализованы с помощью пакетов. К задачам, решаемым с помощью пакетов, относятся языковые сервисы, проектные системы, отладчики, задачи для сборки проектов, интеграции с системами контроля версий и т.д. FRIS является пакетом Visual Studio.

Рассмотрим ключевые аспекты интеграции в Visual Studio нового языка программирования (который ещё не поддерживается VS). Для того чтобы новый язык программирования можно было ввести в инфраструктуру среды разработки, необходимо чтобы пакет интеграции для него содержал так называемую проектную систему. Проектная система – это совокупность интерфейсов и реализующих их объектов, которые позволяют Visual Studio выполнять базовые операции по управлению проектами. Проект – совокупность файлов метаданных, файлов исходного кода, а так же правил для их сбор-

ки, отладки и т.п. Помимо проектной системы пакет интеграции языка программирования (далее языковой пакет) может включать языковые сервисы, сервисы отладки, модель сборки проекта, сервисы работы с системой контроля версий и т.д. Типовая схема такого языкового пакета приведена на рисунке 2.3.

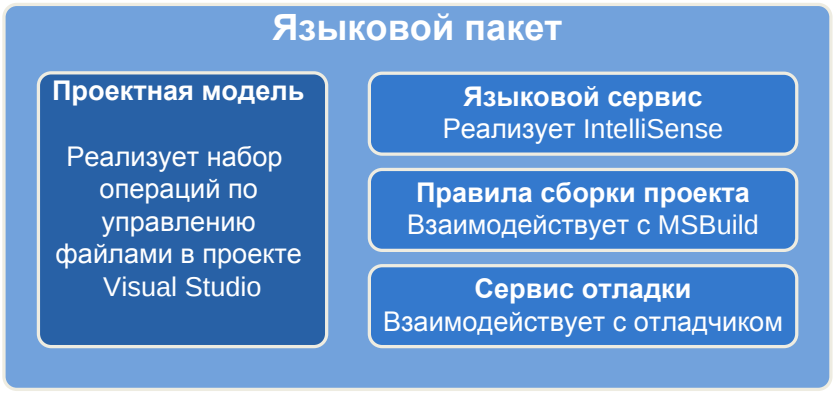

Рисунок 2.3 – Типовая схема языкового пакета

Отметим, что один пакет Visual Studio не обязан содержать сразу все компоненты языкового пакета. Как правило, языковой пакет содержит проектную модель, а так же правила сборки проектов проектной модели. Остальные компоненты, могут содержаться в других пакетах Visual Studio.

2.2.2 Языковой сервис как компонент блока интеграции с IDE

Языковой сервис (Language Service) [47] является составной частью языкового пакета и предназначен для поддержки пользователя при редактировании текста программы на целевом языке программирования. Его цель – «научить» стандартный текстовый редактор, встроенный в Visual Studio работать с целевым языком программирования.

Базовый языковой сервис обеспечивает лишь подсветку синтаксиса и состоит из двух объектов [48]:

- непосредственно языкового сервиса, реализующего интерфейс **IVsLanguageInfo**. Данный интерфейс отвечает за предоставление информации о целевом языке, включая его имя, расширение ассоциированных с ним файлов, менеджер окна кода и компонент подсветки синтаксиса (colorizer).
- колорайзера – компонента подсветки синтаксиса, который реализует интерфейс **IVsColorizer**. Этот интерфейс отвечает за предоставление посимвольной информа-

ции о цветах буферизованного представления текста программы в оперативной памяти.

Под расширенным языковым сервисом понимают такой языковой сервис, который помимо базовой реализации, реализует набор дополнительных интерфейсов и объектов [91] (таблица 2.3), соответствующих возможностям технологии IntelliSence [49].

Таблица 2.3. Ключевые интерфейсы технологии IntelliSense

Интерфейс	Описание
IVsCodeWindowManager	управляет боковыми элементами окна кода, такими как панели с выпадающими списками. Каждому окну кода соответствует строго один объект **IVsCodeWindowManager**
IVsMethodData	отвечает за реализацию возможности **IntelliSense** - ***отображение сведений о параметрах процедур***
IVsCompletionSet	отвечает за реализацию возможностей **IntelliSense** – ***автодополнение*** и ***построение списка элементов сложного объекта***
IVsTextViewFilter	отвечает за реализацию возможностей **IntelliSense** – ***отображение кратких сведений об элементе языка программирования***. Позволяет производить модификации в текстовом представлении, используя обработчики команд. Класс, реализующий данный интерфейс должен так же реализовывать интерфейс **IOleCommandTarget**. Для каждого текстового представления должен быть один объект **IVsTextViewFilter**
IOleCommandTarget	отвечает за перехват команд (событий) Visual Studio. Использование данного интерфейса является частью базовой инфраструктуры **IntelliSense** – для начала сессий автодополнения, отображения членов сущности и краткой информации

Необходимо отметить, что для реализации полнофункционального языкового сервиса недостаточно реализовать лишь указанные интерфейсы, так как, согласно документации Microsoft [91], реализация какого-либо из них требует реализации дополнительных интерфейсов, отвечающих за тот или иной набор функциональных возможностей. Также накладываются требования и по совместной реализации нескольких интерфейсов для одного объекта.

Таким образом, реализация всех программных контрактов, выраженных в интерфейсах, которые ожидает Visual Studio [91], для объектов полнофункционального языкового сервиса является сложной и весьма трудоёмкой задачей. Для упрощения подобных задач Microsoft предоставляет библиотеку Managed Package Framework (MPF) [87], содержащую базовые (абстрактные) классы, реализующие необходимые интерфейсы для различных частей расширения VS с использованием так называемого управляемого кода [85], включая и языковые сервисы.

Данная библиотека предоставляет широкие возможности для взаимодействия с Visual Studio. Набор её базовых классов отвечает за предоставление информации для подсветки синтаксиса и своевременной инициации семантического разбора файла исходного кода, за перехват событий технологии IntelliSense и предоставление данных для отображения пользователю средствами Visual Studio.

Ещё одно важное замечание по поводу MPF состоит в том, что при реализации FRIS возникла необходимость в гибкой настройке различных частей языкового сервиса, однако выяснилось, что реализация от Microsoft не позволяет выполнить такую настройку в виду особенностей программной реализации. Ряд методов базовых классов в реализации MPF от Microsoft нельзя модифицировать, поскольку они являются частными (private), в авторской реализации модификатор private был заменён модификаторами protected virtual (защищённый виртуальный), что позволило переопределять реализацию в классах наследниках.

Таким образом, блок интеграции с IDE во FRIS был реализован с использованием несколько модифицированной версии MPF в части языкового сервиса [92-103] (см. таблицы Б.1-Б.3).

2.3 Блок анализа текстов Fortran-программ

Блок анализа текста программы во FRIS состоит из двух частей, первая выполняет полнотекстовый анализ для наполнения таблицы символов – глобального кэша значимых элементов, а вторая предназначена для поддержки возможности подсветки синтаксиса.

Все анализаторы FRIS реализованы с использованием инструмента распознавания языков ANTLR (**An**other **T**ool for **L**anguage **R**ecognition) [55-56]. Такой выбор обусловлен несколькими факторами. Во-первых, ANTLR является бесплатным, свободно распространяемым с открытым исходным кодом и кроссплатформенным. Во-вторых, он обладает гибкими возможностями настройки для конкретных нужд, позволяя, например, автоматически создавать деревья синтаксического разбора, преобразовывать их любым необходимым способом и т.д. В-третьих, ANTRL поддерживает множество различных стратегий разбора (LL(1), LL(k), LL(*)) [104] и обработки ошибок (исключения, вставки, замены, режим паники и пользовательский режим обработки).

ANTLR состоит из 2-х частей: инструмента генерации анализаторов и библиотеки поддержки времени исполнения. Инструмент генерации анализаторов ANTLR реализо-

ван только на Java, а библиотека поддержки времени исполнения разработана для Java, C#, C/C++, Ruby и множества других языков программирования. Для построения анализаторов (лексических, синтаксических и анализаторов деревьев) используется специальный язык описания грамматик ANTLR. В грамматику допустимо встраивать код на целевом языке программирования, который будет выполняться в процессе разбора при наступлении определённых событий разбора.

Рассмотрим ключевые особенности языка Fortran 2003 и сложности, возникающие при анализе текстов программ, написанных на нём. Все обсуждаемые далее сложности заложены самим стандартом языка, закреплённым Международной организацией по стандартизации (ISO).

2.3.1 Международный стандарт Fortran 2003. Сложности лексического, синтаксического и семантического анализа

Язык программирования Fortran обладает рядом особенностей, делающих его сложным для проведения различного рода анализа (лексического, синтаксического и семантического). Рассмотрим эти особенности, изложенные в тексте стандарта Fortran 2003 [6]. Напомним, что Fortran допускает использование двух форм написания текста программ:

- свободной – является рекомендованной для использования;
- фиксированной – не является рекомендованной и поддерживается только для обратной совместимости со стандартом Fortran 77 [3].

Их отличия друг от друга приведены в таблице 2.4.

Таблица 2.4. Сравнение свободной и фиксированной форм Fortran.

Особенность	Свободная форма	Фиксированная форма
Конец инструкции	инструкция завершается либо в конце строки (символом конца строки), либо терминальным символом «;»	
Поддержка многострочных инструкций	поддерживаются. Для переноса инструкции на новую строку используется символ «&» **в конце переносимой строки**. Если в начале строки продолжения не указан символ «&», то строка продолжается с первой позиции, иначе строка продолжается с первой позиции после символа «&»	поддерживаются. Для переноса инструкции на новую строку используется любой символ кроме пробела или 0, **в 6-й позиции строки продолжения**. В продолжаемой (переносимой строке) никаких символов указывающих, что она продолжается не ставится

Таблица 2.4. Продолжение

Использование комментариев при переносе инструкций	при переносе инструкций можно использовать комментарии, т.е. между переносимой строкой и строкой переноса допустимы комментарии. Строки, состоящие только из пробельных символов, или пустые строки, также считаются строками комментариев	
Поддержка многострочных лексем (токенов)	есть. Любая лексема (токен), состоящая более чем из одного символа может быть перенесена на другую строку (строки). Для переноса лексемы на другую строку, используется символ «&», следующий за символом, начиная с которого будет осуществляться перенос. В строке продолжения должен указываться символ «&», символы, следующие за ним, считаются продолжением лексемы (токена)	нет
Использование комментариев при переносе лексем (токенов)	так же, как для инструкций	запрещены, поскольку запрещён перенос лексем (токенов)
Максимальное число строк переноса	допустимо использовать для переноса не более 255 строк	
Максимальная длина строки	132 символа в стандартной кодировке, или зависит от реализации, если используются символы не стандартной кодировки	72 символа в стандартной кодировке, или зависит от реализации, если используются символы не стандартной кодировки
Ограничения на начало текста	нет. Текст программы начинается с любой позиции в строке	есть, первые 6 позиций зарезервированы: • 1-5 позиции – числовая метка; • 6 позиция – для переноса инструкций. Текст программы начинается с 7-й позиции в строке
Формат комментариев	комментарий начинается с символа «!» и продолжается до конца строки	комментарием считается строка если в её первой позиции указан символ «*» или «C», или символ «!» в любой позиции кроме 6-й. Комментарий продолжается до конца строки
Поддержка ключевых слов	ключевые слова поддерживаются, но не являются зарезервированными, т.е. можно использовать идентификаторы литерально соответствующие ключевым словам	
Пробельные символы	значимы, за исключением комбинированных ключевых слов	не значимы
Поддержка включения в текст программ внешнего текста	поддерживается. Для этого используется строка INCLUDE ‘имя_файла’ (3.4 [6]). Её действие аналогично литеральной вставке содержимого файла «имя_файла» в текст программы. Поддерживается множественное вложенное использование строк INCLUDE с условием отсутствия циклического импорта. Т.е. текст, включаемый одной строкой INCLUDE, может содержать другие строки INCLUDE внутри себя. Все импорты обрабатываются последовательно в порядке их появления	
Не чувствительность к регистру	все лексемы за исключением строковых литералов и комментариев не чувствительны к регистру. Это означает, что следующие идентификаторы эквивалентны: abc, ABC, AbC	

Из таблицы видно, что Fortran сложен как с лексической, так и с синтаксической точки зрения. Переносы лексем в совокупности с допустимостью использования в них

комментариев существенно усложняют лексический анализ. А отсутствие зарезервированных ключевых слов усложняет синтаксический анализ, поскольку выбор того или иного правила уже обуславливается не типом токена (т.к. ключевые слова являются лишь разновидностью идентификаторов и поэтому имеют тип – идентификатор), а его текстом и контекстом употребления.

Приведём примеры переноса инструкций и токенов. Слева указан условный номер строки.

Пример 1. Перенос инструкции присваивания и идентификатора long_id на несколько строк.

```
1       long&
2       !Комментарий
3
4       !Комментарий
3       &_&
4
5       &id = &
6       !Комментарий
7       1
```

Пример 2. Перенос строкового литерала на несколько строк.

```
1       "это очень&
2       !Комментарий
3
4       &\""длинная\"" строка"
```

В приведённом примере показано, как можно использовать кавычки в строковом литерале, который ограничен этим типом кавычек. Т.е. знак кавычек соответствует паре кавычек, в то время как одинарное использование знака кавычек закрывает литерал.

Рассмотренные примеры и описанные свойства относятся к языку Fortran как таковому.

2.3.2 Общий метод построения грамматики для анализа текстов программ в режиме реального времени

Главная особенность, которую необходимо учитывать при проектировании всех без исключения анализаторов, заключается в том, что разбор (или анализ) текстов программ будет проводиться непосредственно во время редактирования программы пользователем. Это означает, что в подавляющем большинстве случаев программа будет находиться в некорректном с точки зрения спецификации языка программирования состоянии. Так, например, анализ может быть запущен, в момент, когда пользователь мог начать вводить какую-либо инструкцию языка программирования. Поскольку он ещё не

закончил ввод, такая незавершённая инструкция, является некорректной с точки зрения синтаксиса языка программирования. Другим примером может служить ситуация, когда пользователь ввёл стартовую инструкцию блочной конструкции, но ещё не успел ввести парную завершающую инструкцию. Тогда будет нарушена синтаксическая и семантическая целостность программы (т.к. не найдена ожидаемая завершающая инструкция).

Таким образом, при проведении анализа во время редактирования программы пользователем возможно возникновение любых ошибок, и их необходимо корректно обрабатывать. Для дальнейшего обсуждения механизма обеспечения указанного режима работы необходимо обратиться к базовому определению языка программирования и его грамматики [50-56].

Конечное множество объектов будем называть *словарём* [54]. Элементы словаря будем называть *символами* [54]. *Цепочкой* (символов) над словарём [54] будем называть произвольную конечную последовательность символов словаря. Пустую цепочку (не содержащую ни одного символа) будем обозначать символом ε.

Пусть V – словарь, тогда множество всех возможных цепочек, составленных из символов словаря V, включая пустую цепочку, обозначим V*.

Грамматикой G называется множество компонентов *G=(T,N,R,S)*, где: *T* – конечное непустое множество *терминальных символов* или *токенов*[8](терминальный словарь); *N* – конечное непустое множество *нетерминальных символов*[9] или синтаксических переменных (нетерминальный словарь); *R* – множество *продукций* или *правил вывода*, каждая из которых состоит из нетерминала, называемого заголовком, или левой частью, стрелки и последовательности терминалов и/или нетерминалов, называемых телом, или правой частью продукции; *S* – нетерминальный символ, указываемый как стартовый или начальный.

Языком *L* порождаемым грамматикой *G* называется множество **терминальных** цепочек, выводимых из начального символа грамматики: $L(G) = \{\alpha \in T^* \mid S \Rightarrow_{*_G} \alpha\}$ [54]. Здесь *T* – терминальный словарь; *T** - множество всех возможных цепочек, включая пустую, составленных из символов терминального слова-

[8] Терминальный символ, токен или лексема – элементарные символы языка программирования.

[9] Нетерминал, или нетерминальный символ – множество строк терминалов.

ря; S – стартовый нетерминал грамматики G; символ $\Rightarrow_{*_G}$ - означает, что цепочка α выводима из грамматики G за конечное (в том числе и нулевое) число шагов.

Теперь легко классифицировать возможные ошибки анализа и способы борьбы с ними (таблица 2.5).

Таблица 2.5. Возможные ошибки анализа и способ их устранения

Вид ошибки	Описание	Способ устранения
Лексическая	в процессе получения терминала найден недопустимый для данного терминала символ	вернуться к началу разбора терминала и попробовать найти другой подходящий терминал
	встречен недопустимый в данном алфавите символ	определить в грамматике терминал, принимающий один любой символ, в том числе не допустимый в алфавите языка
Синтаксическая	для данной последовательности терминалов не найдена ни одна продукция	определить в грамматике наиболее общую продукцию, совпадающую с любой, даже некорректной последовательностью терминалов
	при разборе блочной конструкции была встречена стартовая продукция, но до конца файла не была встречена конечная продукция	определить новую ослабленную грамматику эквивалентную исходной, с набором управляющих правил и особым механизмом разбора
	при разборе была встречена недопустимая в данном контексте продукция	

Рассмотрим способ построения ослабленной версии G^* [71] исходной грамматики G для улучшенной обработки ошибок, а также несоответствия продукций лексическим и синтаксическим контекстам языка.

В новую грамматику G^* включим всё множество терминальных символов исходной грамматики (T), их неполные версии (T') и специальный терминал для совпадения с любым символом ($TANY \rightarrow .$, здесь «.» - означает любой символ). Рассмотрим терминальный символ для строки символов. Строка определяется как любое множество символов, находящееся между символами двойных кавычек (”). Это исходный терминал. Неполным терминалом для строки будет являться следующий: открывающаяся двойная кавычка, за которой следует любое множество символов кроме дублирующей закрывающей двойной кавычки.

Выделим во множестве нетерминальных символов N и правил вывода R грамматики G нетерминальные символы N^S и правила вывода R^S, описывающие правила вывода отдельных инструкций (statements), и включим их без изменений в новую грамматику

G^*. Дополним правила вывода R^* новой грамматики правилом вывода инструкции языка в самом общем виде:

$any \to .^* \; end\_of\_stmt$, где

«.» - означает любой терминал,

«*» - означает повторение 0 и более раз,

а нетерминал *end_of_stmt* соответствует продукции, означающей конец инструкции. В Fortran признаком конца инструкции является: либо переход на новую строку, либо символ «;».

Дополним также правила вывода R^* новой грамматики новой стартовой продукцией, состоящей из линейного перечисления всех продукций инструкций из множества R^S и продукции *any*:

$$start \to statement^* \; end\_of\_file$$
$$statment \to s_1 \mid s_2 \mid \ldots \mid s_n \mid any$$
$$s_i \in R^S$$

Здесь нетерминал *end_of_file* соответствует продукции, означающей конец анализируемого файла.

Таким образом, новая грамматика $G^* = \left(T^*, R^*, N^*, S^*\right)$, где $T^* = T \cup T' \cup TANY$, $R^* = R^S \cup start \cup statment \cup any$, $N^* = N^S \cup start \cup statment \cup any$, $S^* = start$.

При этом по построению все корректные предложения для исходной грамматики *G* будут также корректными для новой грамматики G^*. Однако для грамматики G^* будут также корректны и предложения, недопустимые в первоначальной грамматике, за счёт введения неполных терминалов, терминала *TANY*, правила вывода и нетерминала *any*. Также в данной версии грамматики отсутствуют продукции для распознавания блочных конструкций, которые по определению [6] не являются отдельными инструкциями.

Для того чтобы корректно обработать такую ситуацию, вначале необходимо классифицировать блочные конструкции в исходной грамматике, а затем ввести дополнительно к грамматике набор управляющих правил по работе с блочными конструкциями. Отметим, что управляющие правила не входят в определение грамматики, поэтому заслуживают отдельного обсуждения.

Перед их формулированием необходимо классифицировать все блочные конструкции исходной грамматики. Классифицируем их по 3-м основным группам (таблица 2.6).

Таблица 2.6. Группы блочных операторов Fortran

№	Группа	Описание	Перечень конструкций
1	**Высокоуровневые**	являются глобальными блоками	главная программа (program); блок данных (block data); модуль (module); подпрограмма (subroutine); функция (function)
2	**Регулярные**	обычные блочные конструкции	определение производного типа данных (type); определение перечисления (enum); блочная конструкция where; блочная конструкция forall; блочный условный оператор if; блочный оператор select (две разновидности: select case, select type); блок associate; блок цикла do; блок интерфейса (interface)
3	**Внутренние**	группа внутренних операторов в специальных регулярных операторах. Как правило, не имеют явного признака своего завершения	блок else (для блоков if и where); блок case (для конструкции select case); блок type guard (для конструкции select type) (8.1.5.1 [6]); блок contains

В приведённой таблице столбец «№» характеризует приоритет блочного оператора, при этом чем больше номер, тем ниже приоритет. Сформулируем управляющие правила для обработки блочных операторов:

- при обнаружении блочного оператора с таким же приоритетом — текущий блочный оператор завершается;
- при обнаружении блочного оператора с более высоким приоритетом — завершаются все блочные операторы с низшим приоритетом, начиная с текущего, и оператор с таким же приоритетом, что и вновь обнаруженный.

Заметим, что для обработки специальных случаев, зависящих от спецификации конкретного языка программирования, применяются дополнительные специальные правила.

Так, для Fortran группа высокоуровневых операторов, подпрограмма и функция, в ряде случаев может обрабатываться по другим правилам, поскольку в Fortran существуют 4 группы подпрограмм и функций:

- глобальные – находятся в глобальной области видимости;

- модульные – находятся внутри модуля;
- интерфейсные – находятся внутри блока описания интерфейса;
- внутренние – находятся внутри других подпрограмм и функций.

Соответственно, для обработки таких случаев необходимо, помимо приоритета, анализировать текущий контекст. Так, если при обработке одной из модульных функций была обнаружена инструкция начала блока подпрограммы, то очевидно, что, хотя и модуль, и подпрограмма имеют одинаковый приоритет, необходимо лишь прекратить обработку текущей функции, а обработку модуля прекращать не следует.

Рассмотрим общую схему работы с дочерними языками на примере поддержки работы с комментариями документирования. Как отмечалось ранее, комментарии документирования служат для описания сущностной информации для элемента языка программирования, и они не определены в спецификации самого языка программирования. То есть являются языком внутри языка.

Для того чтобы однозначно определить дочерний язык внутри текста на базовом языке, необходимо, чтобы продукции дочернего языка начинались со специального терминального символа. Таким образом, всё, что следует за данным терминалом, относится к дочернему языку.

Приведём общую схему работы анализаторов для поддержки дочерних языков, реализованную во FRIS (рисунок 2.4).

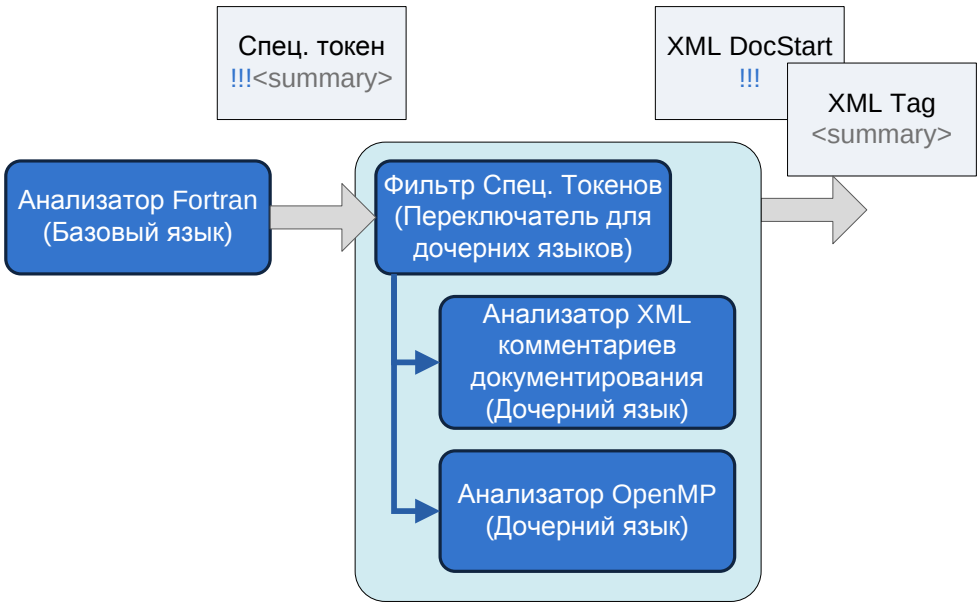

Рисунок 2.4 – Общая схема работы анализаторов

На рисунке представлена общая схема работы стека анализаторов на примере разбора части XML комментария документирования. Анализатор базового языка Fortran генерирует токены, которые затем пропускаются через фильтр токенов. Если токен совпадает с одним из зарегистрированных дочерних языков, вызывается соответствующий анализатор. На выходе получается множество полностью распознанных токенов для всех поддерживаемых языков.

Для построения синтаксического анализатора во FRIS использована «оптимистическая» стратегия разбора. Она заключается в том, что анализ файла ведётся с использованием ослабленной версии грамматики Fortran 2003 в режиме по одной инструкции. Для каждой инструкции строится абстрактное дерево разбора (AST). Если инструкция не может быть распознана, например вследствие того, что пользователь просто не успел её набрать, то для неё генерируется особое дерево разбора, включающее в себя все её лексемы (до признака конца инструкции включительно).

В паре с синтаксическим анализатором работает построитель полного дерева разбора (рисунок 2.5). Он на основе AST для индивидуальных инструкций и управляющих правил работы с блочными конструкциями строит полное дерево разбора. В задачу построителя входит отслеживание операций открытия и закрытия синтаксических контекстов, в частности их оптимистическое завершение согласно изложенной ранее схеме обработки блочных конструкций с учётом их приоритета.

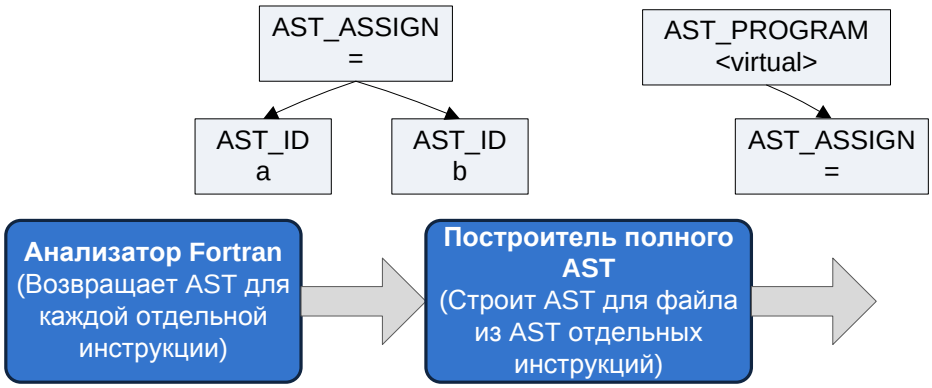

Рисунок 2.5 – Схема работы синтаксического анализатора FRIS

Таким образом, синтаксический анализатор на выходе всегда предоставляет корректное дерево разбора текущего файла, при этом нераспознанные инструкции помещаются в специализированные узлы дерева разбора. Это позволяет упростить алгоритм семантического анализа. Семантический анализатор обходит дерево разбора и собирает

информацию обо всех значимых элементах Fortran (см. пункт 1.3), которые затем помещает в блок хранения распознанных элементов (см. пункт 2.4).

2.3.3 Алгоритм анализа Fortran программ для сбора информации о значимых элементах

Данный алгоритм предназначен для анализа текста программы на языке Fortran с целью сбора информации о значимых элементах и помещении их в блок хранения распознанных элементов (см. пункт 2.4). Этот вид анализа выполняется для всех файлов, составляющих программный проект при открытии проекта в Visual Studio, и далее по мере их изменения.

Алгоритм предварительного анализа файлов следующий:

- получить список программных проектов;
- для каждого проекта получить список файлов, входящих в него;
- для каждого файла исходного кода на языке Fortran, запустить операцию предварительного фонового разбора.

При этом выполняется параллельная независимая обработка не только программных проектов, но и файлов внутри каждого проекта. Такое многоуровневое порождающее распараллеливание выполнено с использованием Библиотеки параллельных задач (TPL) [105] платформы .NET. Перед непосредственным запуском разбора файла выполняются все необходимые проверки: является ли этот файл физическим (находится на диске) и является ли он файлом, зарегистрированным как файл Fortran (проверяется его расширение).

2.3.3.1 Общая схема разбора

Механизм разбора, используемый во FRIS, выполняет сбор информации о значимых элементах. Он учитывает как особенности режима работы в реальном времени, так и особенности Fortran, такие как различные виды переноса инструкций, нечувствительность к регистру, возможность включения произвольного текста в файл и т.д., а так же особенности ANTLR (рисунок 2.6).

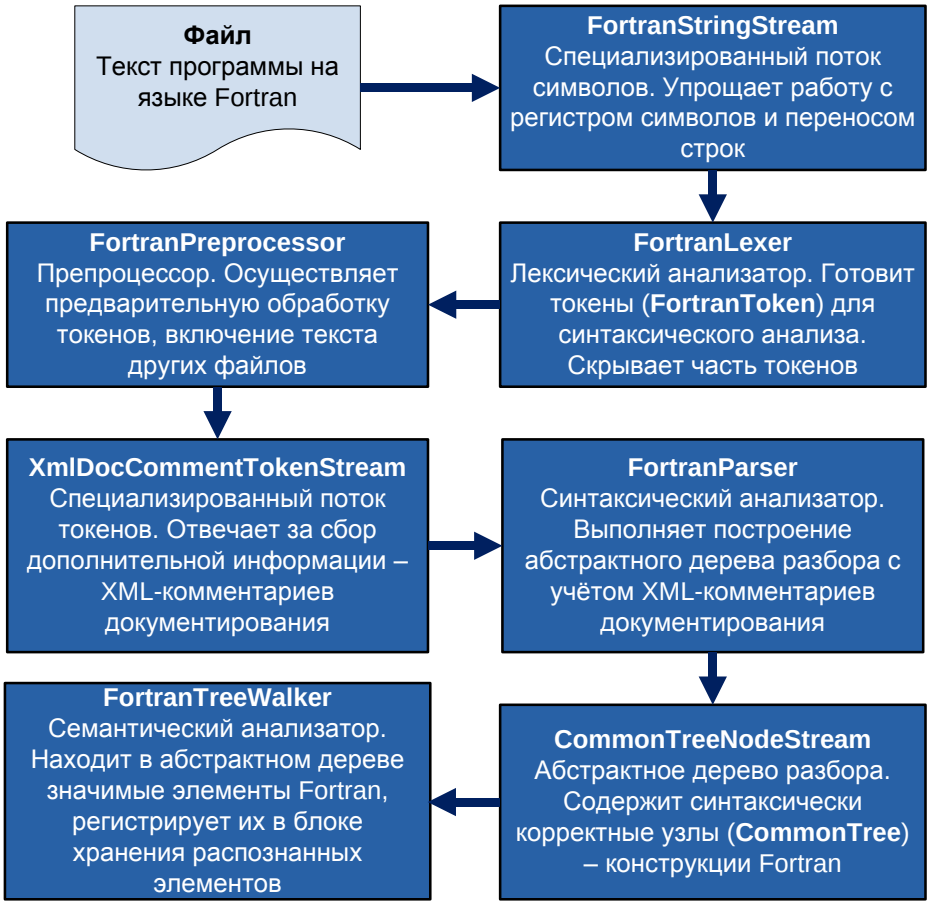

Рисунок 2.6 – Схема блока анализа с указанием классов для сбора информации о значимых элементах во FRIS

2.3.3.2 Специализированный поток символов

Специализированный поток символов отвечает за:

- обработку особенности Fortran – нечувствительности к регистру; в лексический анализатор выдаются символы в нижнем регистре;
- обработку различных вариантов переноса строк в фиксированной и свободной формах; вырабатывается единый признак переноса строки/литерала с учётом формы текста;
- определение формы потока символов – фиксированная или свободная.

Специализированный поток символов реализуют классы FortranStringStream и FortranFileStream (см. таблицу Б.5).

2.3.3.3 Алгоритм лексического анализа Fortran

Алгоритм лексического анализа на основе входного потока символов должен сконструировать поток распознанных токенов с учётом возможности работы с дочерними языками.

Реализованный лексический анализатор FortranLexer (см. таблицу Б.5) обладает следующими особенностями:

- использует ослабленные правила распознавания токенов:
  - обрабатывает незавершённые строковые и вещественные литералы (не содержат парной кавычки и т.п.);
  - обрабатывает незавершённые операторы в формате $Operator \rightarrow '.'('a'..'z')+'.'?$;
  - обрабатывает недопустимые символы, которые не соответствуют стандарту; для них не генерируются токены;
- определяет типы токенов для использования в лексическом, синтаксическом и других фазах анализа;
- обрабатывает многострочные литералы и создаёт специализированные токены, учитывающие особенности Fortran класс FortranToken (таблица Б.6);
- учитывает использование XML-комментариев документирования;
- учитывает использование директив препроцессора;
- скрывает (устанавливает номер скрытого канала, особенность ANTLR) [55] ряд токенов от дальнейшего использования в синтаксическом анализе:
  - все виды комментариев, включая XML комментарии документирования;
  - символы конца строки, если используется многострочная инструкция;
  - директивы препроцессора;
- поддерживает специфичную для компилятора от Intel возможность использования символа «$» в идентификаторах (стандарт запрещает использование данного символа).

Необходимо отметить ключевое значение возможности «скрытия» части токенов путём назначения им номера канала отличного от того, с которым будет работать синтаксический анализатор, без физического исключения их из потока. Данный механизм позволяет эффективно организовывать обработку других языков, вложенных в основной текст программы на Fortran.

Общий механизм подобной работы выглядит следующим образом:

- дочерний язык регистрирует специальную последовательность символов, все символы после которой до конца строки или другой специализированной последовательности символов считаются принадлежащими не Fortran, а дочернему языку;
- при обнаружении такой специальной последовательности символов, лексический анализатор формирует специальный токен и помещает его на скрытый либо на особо оговорённый канал, чтобы упростить алгоритмы дальнейшей обработки. При этом токен остаётся в общем потоке токенов, но эффективно скрывается от всех видов анализа, не работающих с тем каналом, к которому он приписан;
- для обработки специальных токенов на скрытых каналах используются специализированные лексические и синтаксические анализаторы, обрабатывающие их согласно правилам целевого интегрируемого языка.

Наиболее наглядно режим разбора с учётом возможных лексических ошибок, а также использования многострочных литералов можно продемонстрировать, рассмотрев правило для распознавания строкового литерала:

```
// Строка в самом общем виде с одинарной или двойной кавычками
GeneralString → '\''SingleQuoteStringContinued | '\"'DoubleQuoteStringContinued
// Продолжение строки с одинарной кавычкой
SingleQuoteStringContinued →
(// символы не являющиеся одинарной кавычкой и концом строки
 ch =~ ('\''|'\r'|'\n'){проверка переноса строки}
 // перенос строки возможен если данный литерал является многострочным
 |('\r'?'\n')=> {многострочный литерал}? => LineContinuation
 //вставка литерала ' в текст строки
 |('\''&'|'\''\'') =>'\''TokenContinuation? '\''TokenContinuation?
)*
// завершение строки : либо кавычкой, либо без неё (ослабленое правило)
|('\''|ε)
```

Здесь показано, как осуществляется автоматическое отслеживание допустимости использования многострочного литерала, использование перехода на другую строку, в случае необходимости — использование парной одинарной кавычки для вставки в строку литерала одинарной кавычки и окончание строкового (недописанного) литерала.

2.3.3.4 Препроцессор

Препроцессор предназначен для осуществления предварительной обработки токенов, генерируемых лексическим анализатором, а именно для обработки импортов файлов при помощи строки INCLUDE (3.4 [6]). Схема классов, входящих в блок препроцессора, приведена на рисунке 2.7.

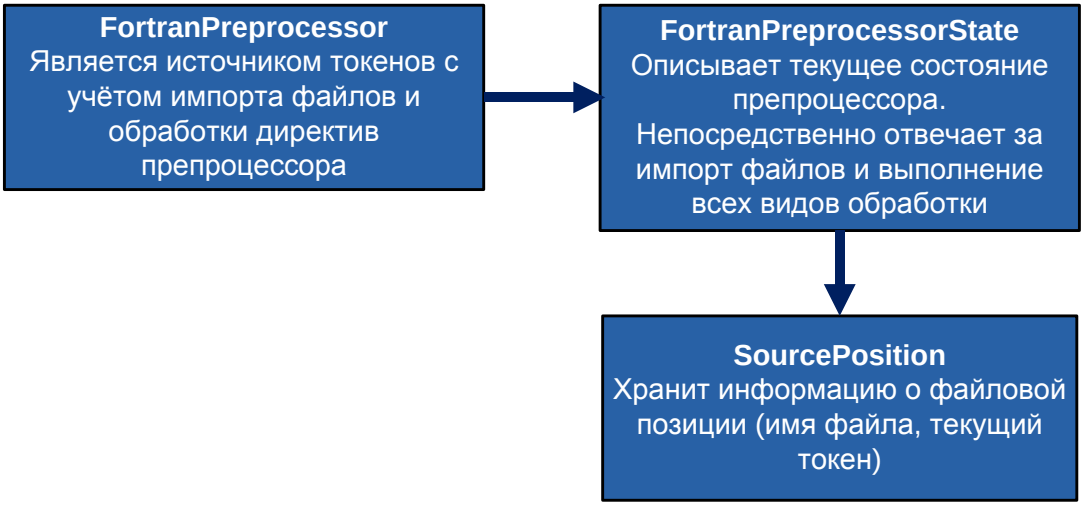

Рисунок 2.7 – Схема классов блока препроцессора

Расположение препроцессора в цепочке анализа файлов после лексического анализатора обусловлено соображениями удобства, поскольку уже распознанные токены, особенно с учётом их потенциальной многострочности, намного удобнее анализировать, чем исходный текст.

Согласно стандарту Fortran 2003, в текст любого файла может быть внедрён текст другого файла при помощи строки INCLUDE, при этом в самом внедряемом тексте, так же могут быть строки INCLUDE и так далее, с ограничением отсутствия циклических импортов. Рассмотрим схему работы препроцессора для обработки импортирования файлов (листинг 2.1).

```
function NextToken() returns(Token)
if (стек очередей импортированных токенов не пуст) then
  Token=получить токен из очереди на вершине стека
else
  Token=получить токен из исходного потока
endif
if (Token содержит текст «INCLUDE» и далее следует строковый литерал) then
  if (импортированный файл содержится в кэше импортированных файлов) then
    поместить содержимое импортированного файла на вершину стека очередей
импортированных файлов
  else
    провести лексический анализ импортируемого файла, поместить его в кэш им-
портированных файлов и на вершину стека очередей импортированных файлов
  endif
  Token = NextToken()
  модифицировать файловую позицию Token в соответствии с исходной файловой
позицией строки INCLUDE
endif
вернуть Token
end function
```

Листинг 2.1 – Схема работы препроцессора

Основные понятия, используемые в алгоритме (листинг 2.1):

- *исходный поток* – это токены, порождённые лексическим анализатором при анализе исходного файла программы;
- *стек очередей токенов* – каждый импортируемый файл является очередью токенов, последовательно идущих друг за другом, а поскольку таких импортов может быть сколь угодно много, то они эффективно образуют стек; при обнаружении в какой-либо очереди токенов строки «INCLUDE» файл, импортируемый данной директивой, эффективно вытесняет саму директиву – т.е. помещается на вершину стека; как только очередь на вершине стека заканчивается, она удаляется со стека, обработка возобновляется для очереди на вершине стека и так далее;

- *кэш импортируемых файлов* – это словарь, ключом которого является имя импортируемого файла, а значением – массив токенов, составляющих его содержимое; использование кэша позволяет уменьшать накладные расходы на многократный анализ часто используемых импортируемых файлов.

Таким образом, на выходе из препроцессора исходные токены могут быть модифицированы путём эффективной замены директив импорта файлов их содержимым с учётом требований стандарта Fortran 2003. Несмотря на это, их файловые позиции остаются неизменными, поскольку эффективно добавляемые токены перенимают свойства файловой позиции замещаемых токенов. Это особенно важно для работы в Visual Studio.

2.3.3.5 Специализированный поток токенов. Обработка дочерних языков

Специализированный поток токенов XmlDocCommentTokenStream (таблица Б.7) решает задачу по обработке дочерних языков, инициируя запуск соответствующих анализаторов. Как отмечалось ранее в пункте 2.3.3.3, токены дочерних языков помещаются лексическим анализатором на специальные скрытые или неактивные каналы.

Рассмотрим реализацию общей схемы (см. пункт 2.3.2) обработки специализированных токенов дочерних языков (рисунок 2.8). Синтаксический анализатор работает только с токенами, находящимися на активном канале (все каналы нумерованы, а активным считается канал, номер которого был установлен при создании синтаксического анализатора). Соответственно синтаксический анализатор запрашивает из потока только токены, находящиеся на интересующем его канале, а остальные токены, находящиеся на неактивных каналах, пропускаются. В момент пропуска синтаксическим анализатором не интересующих его токенов производится обработка дочерних языков. При необходимости проводится сбор многострочных токенов по специальным правилам.

Указанным образом обрабатываются XML-комментарии документирования, поскольку множество строк комментариев документирования, идущих подряд без пропусков (пустых строк), образуют единый логический блок. Как только находится специализированный токен – признак конца логического блока, операция сбора завершается, полученный токен помещается во внутреннюю очередь токенов и инициируется его полнотекстный анализ.

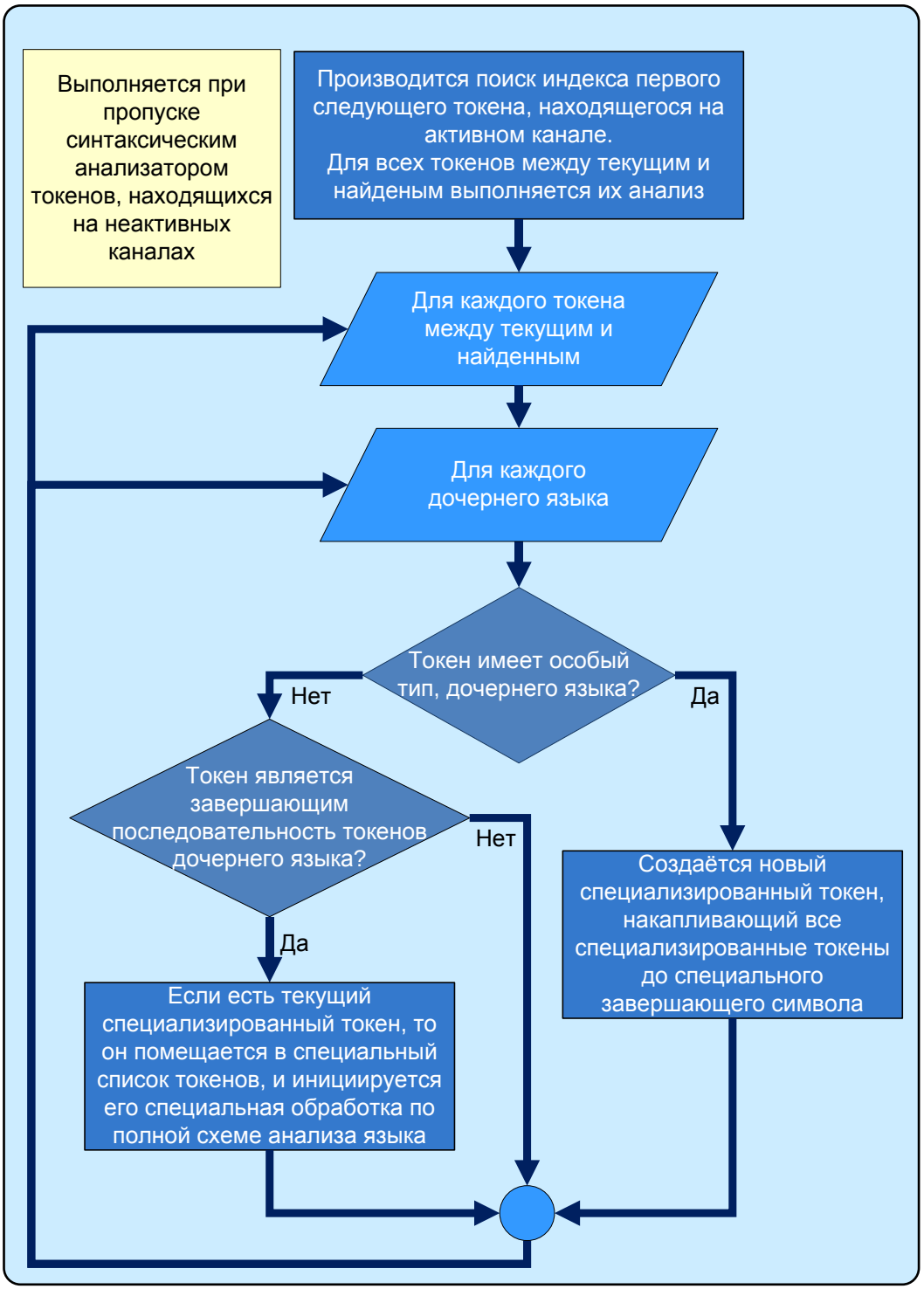

Рисунок 2.8 – Общая схема обработки специализированных токенов

Для хранения комментария документирования, который может являться многострочным, используется класс XmlDocComment (таблица Б.8). Для лексического и синтаксического анализа XML-комментариев документирования были разработаны облегчённые версии лексического (класс XmlLexer) и синтаксического (класс XmlParser) анализаторов, учитывающие возможность анализа во время редактирования пользователем текста комментария, когда он находится в заведомо лексически или синтаксически не-

верном виде. В XmlParser абстрактное синтаксическое дерево строится из специально разработанных классов описания структуры XML-тэгов (рисунок 2.9).

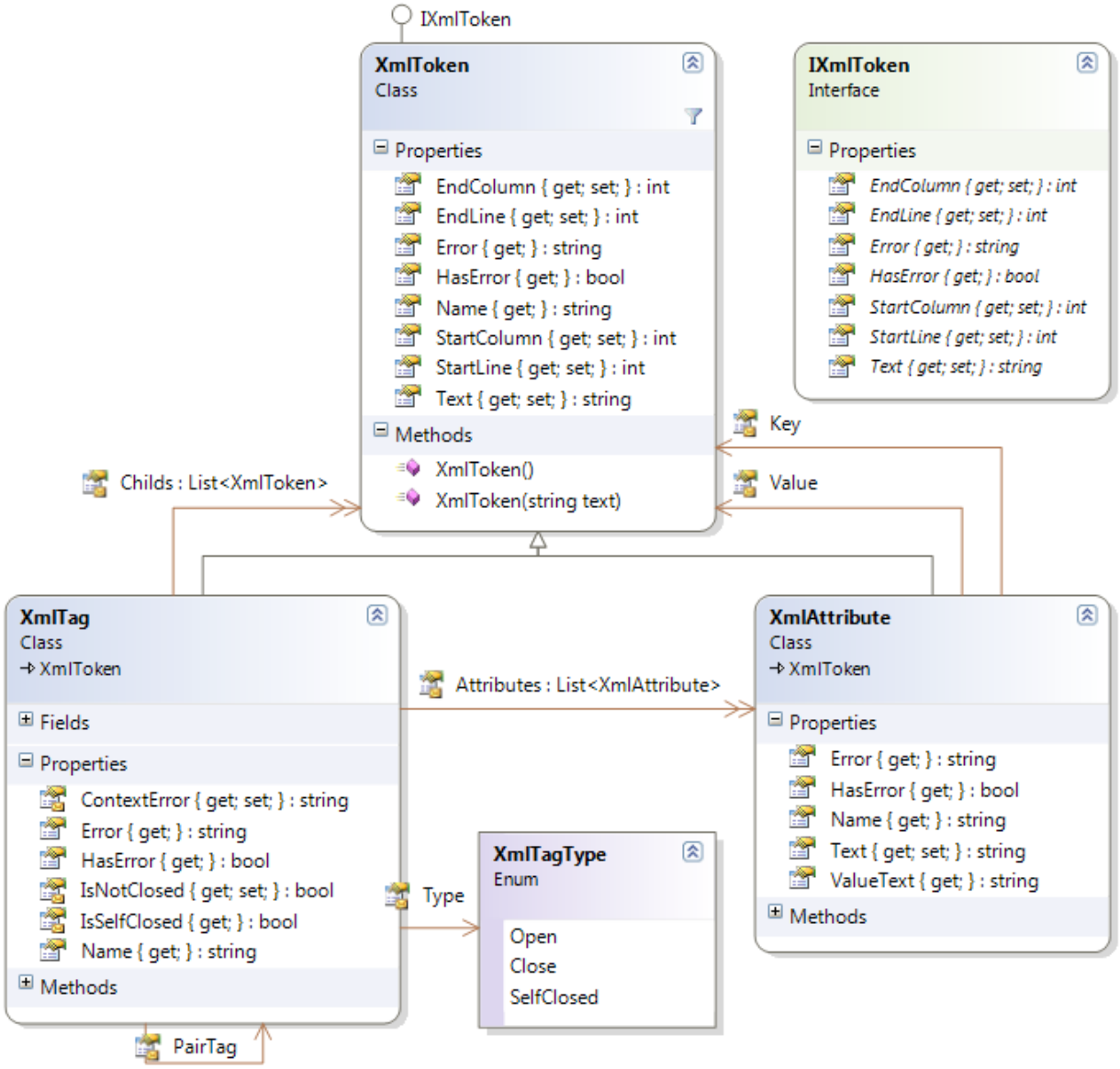

Рисунок 2.9 – Диаграмма классов узлов синтаксического дерева XML

Особенность разработанных классов заключается в учёте возможности использования многострочных XML-токенов и тэгов, а так же встроенной (в классы XmlTag и XmlAttribute) диагностики грубых лексических и синтаксических ошибок. Например, пропущена закрывающая угловая скобка «>», у атрибута нет значения (конструкция вида attribute или attribute=), у тэга нет парного тэга и т.п. Кроме того, поскольку тэг может быть снабжён атрибутами, то он учитывает наличие ошибок не только в нём, но и в каждом из его атрибутов.

Аналогичным образом осуществляется работа и для других дочерних языков.

2.3.3.6 Алгоритм синтаксического анализа и построение абстрактного синтаксического дерева

Самой сложной и трудоёмкой фазой анализа является синтаксический анализ, за его выполнение отвечает реализованный класс FortranParser, который формирует абстрактное синтаксическое дерево (AST) с учётом возможной синтаксической некорректности текста программы в связи с его разбором в момент редактирования пользователем (см. пункт 2.3.2). Здесь обнаруживается и обрабатывается большинство синтаксических и часть семантических ошибок, формируются узлы абстрактного синтаксического дерева, в которые подключаются узлы комментариев документирования и т.д.

Поскольку на этапе лексического анализа практически невозможно классифицировать часть идентификаторов как ключевые слова, то вся сложность обработки идентификаторов как ключевых слов ложиться на синтаксический анализатор.

Рассмотрим разработанную ослабленную версию грамматики синтаксического анализа Fortran, построенную согласно схеме, изложенной в пункте 2.3.2 и реализующую построчный разбор и передачу управления для дальнейшей специализированной обработки.

// стартовое правило грамматики
start → *labeled _ stmt* *
EOF {*обработать окончание разбора файла и вернуть AST для него*}
// основное правило разбора в режиме инструкция за инструкцией
// любая инструкция может быть снабжена числовой меткой – label
labeled _ stmt → *label* ? *all _ or _ any* {*обработать AST для одной инструкции*}
// универсальное правило, учитывает синтаксические ошибки
all _ or _ any → *all _ stmts* {*проверить была ли обнаружена ошибка разбора*}
|{*если была ошибка разбора, то обрабатываем инструкцию в режиме подавления ошибок*}? *any _ stmt*
// правило разбора всех корректных инструкций
all _ stmts → *end _ stmt* | *derived _ type _ stmt* | ...
// особое правило обработки инструкции в режиме подавления ошибок
// инструкией считается любая последовательность токенов
// завершающаяся либо токеном EOS либо EOF
any _ stmt →~ (*EOS* | *EOF*)* *end _ of _ stmt*
// конец инструкции
end _ of _ stmt → *EOS* |(*EOF*) => ε

Как видно из приведённой грамматики, после завершения разбора каждой инструкции (правило labeled_stmt) производится обработка распознанного узла синтаксического дерева для этой инструкции (в методе HandleNode), которая реализует алгоритм построения полного дерева разбора для текущего файла (см. пункт 2.3.2). В нём происходит формирование синтаксических контекстов и их вложенности друг в друга, здесь же производится часть семантического анализа, например, при обработке завершения модуля проверяется, как именно он завершился — инструкцией end, end module или как-то иначе, и в случае обнаружения ошибки генерируется соответствующее сообщение.

При достижении конца файла (правило start) производится закрытие всех открытых контекстов в режиме обнаружения ошибки, поскольку в нормальном состоянии при достижении конца файла не должно существовать ни одного открытого синтаксического контекста (метод HandleEOF).

Для отслеживания синтаксических контекстов был разработан специальный класс BlockConstruct (рисунок 2.9, таблица 2.7), представляющий собой отдельно взятый синтаксический контекст, а в синтаксическом анализаторе были организованы структуры данных, содержащие: полное абстрактное дерево разбора для всего файла, последнее обработанное дерево для отдельно взятой инструкции и стек синтаксических контекстов.

Таблица 2.7. Описание класса блочной конструкции BlockConstruct

Элемент	Описание
Свойства	
Kind	тип блочной конструкции: модуль, функция, тип данных, условный оператор и т.д.
Parent	ссылка на родительскую конструкцию, для конструкций в глобальной области видимости она нулевая
Childs	список блочных конструкций, являющихся дочерними элементами
Root	корневой, или стартовый узел синтаксического дерева для данной конструкции
End	конечный узел синтаксического дерева для данной конструкции
Name	имя блочной конструкции
Label	метка блочной конструкции
RootStopTokenIndexOriginal	предназначено для помощи в создании структуры кода для анализируемого файла
Методы	
AddChild(BlockConstruct)	служат для регистрации (добавления) дочерних элементов к текущему блоку
AddChild(ITree)	

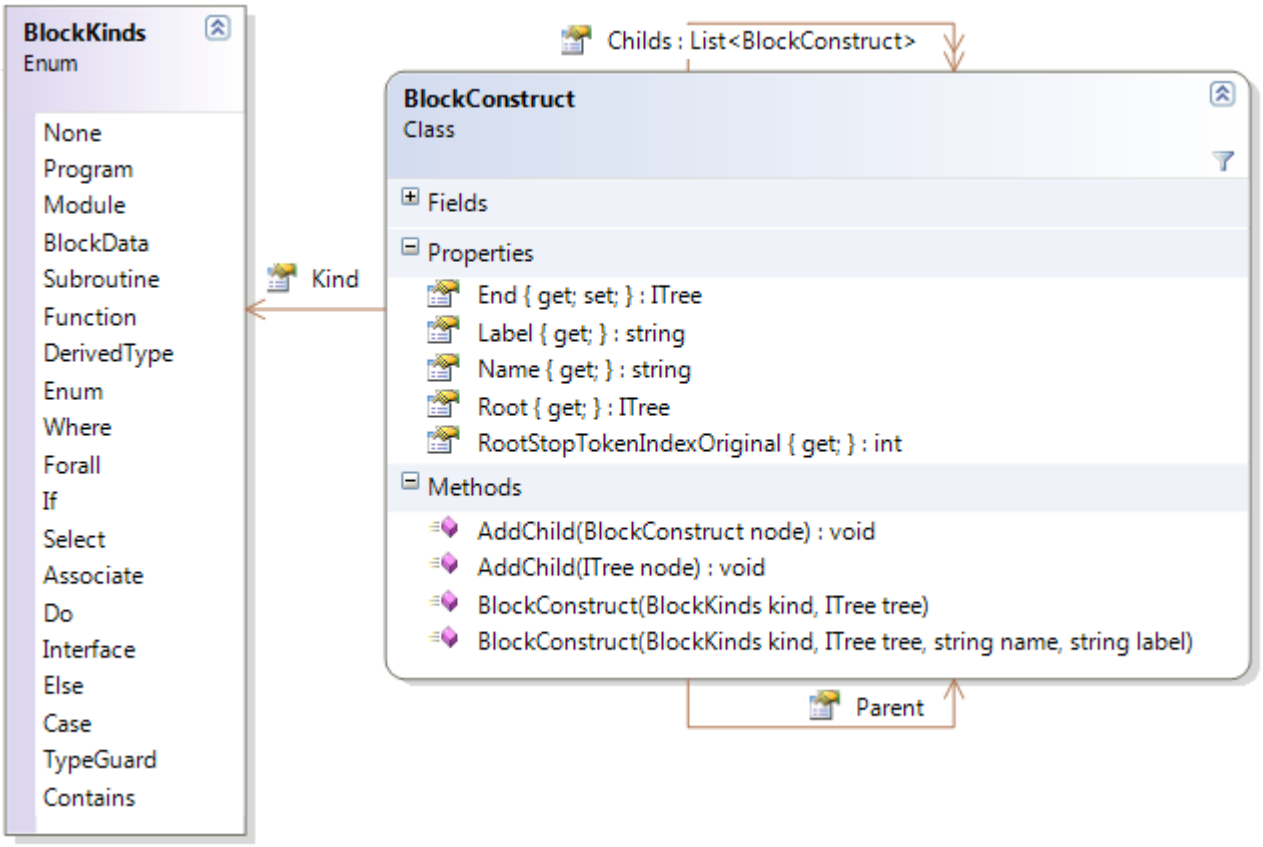

Рисунок 2.9 – Диаграмма классов блочных конструкций Fortran

Вложенность и отслеживание уровня иерархии блочных конструкций осуществляется при помощи свойств Parent и Childs.

Связь с исходными элементами, генерируемыми парсером, осуществляется при помощи свойств Root и End.

Особое внимание заслуживает факт специальной обработки поля End. Когда в результате анализа отдельной инструкции (проводимого в методе HandleNode) принимается решение о завершении блока, находящегося на вершине стека синтаксических блоков, для данного блока значение свойства End устанавливается равным тому узлу, для которого было принято решение, что он закрывает (завершает) данный блок. Класс BlockConstruct содержит логику, проверяющую, соответствует ли завершающий узел ожидаемому завершающему узлу. Например, когда на вершине стека содержится цикл do, а ближайшей обнаруженной инструкцией завершения блока явилась инструкция endif (завершения блока if). В случае соответствия завершающего узла ожидаемому производится нормальное закрытие блока. А в случае обнаружения ошибки генерируется узел AST_END_CONSTRUCT, говорящий, что блок завершился в нештатном режиме (в режиме ошибки). Подобное нештатное завершение может быть вызвано тем, что на

момент достижения конца файла стек синтаксических конструкций не пуст (эта проверка выполняется в методе HandleEOF). Или при разборе была обнаружена одна из высокоуровневых конструкций (например, module). В этом случае все конструкции стека также закрываются в нештатном режиме.

Рассмотрим алгоритм обработки абстрактного синтаксического дерева для отдельно взятой инструкции и конструирование на его основе полного абстрактного синтаксического дерева (метод HandleNode) (рисунок 2.10).

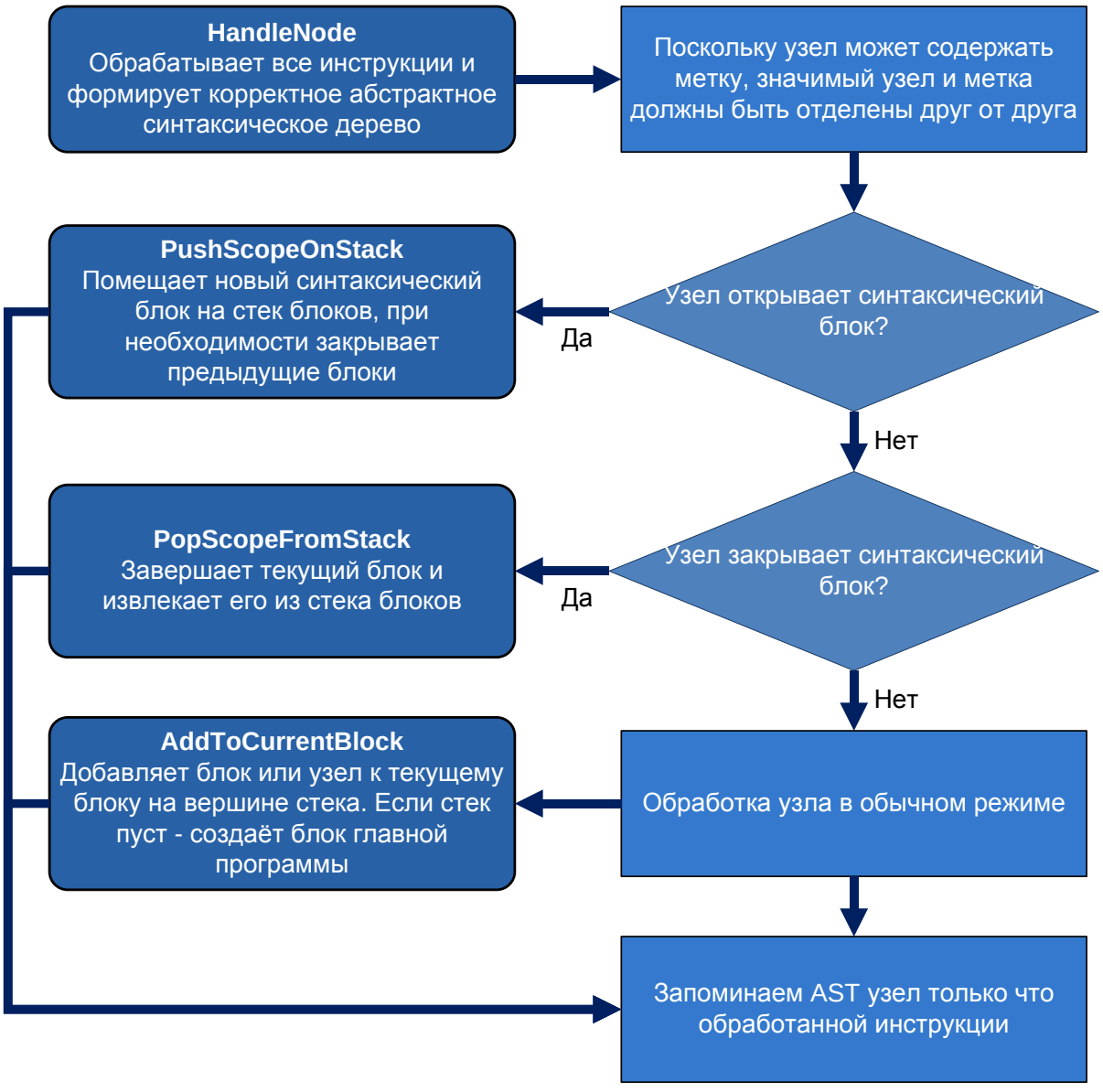

Рисунок 2.10 – Блок-схема работы метода HandleNode

Поскольку любая инструкция Fortran может начинаться с числовой метки, то сначала из анализируемого узла выделяется числовая метка, затем основной значимый узел. Далее производится анализ типа узла и, если он является началом нового синтаксиче-

ского блока, то производится создание новой блочной конструкции и помещение её на вершину стека. Если же узел является разновидностью инструкции END, то производится закрытие соответствующего блока, в противном случае производится добавление узла к текущему синтаксическому блоку (на вершине стека). По завершении обработки узла он запоминается как последний обработанный.

Набор управляющих правил по построению полного абстрактного синтаксического дерева (см. пункт 2.3.2) реализован в виде отдельных методов работы со стеком синтаксических контекстов (таблица 2.8). Они производят обработку ряда синтаксических ошибок, например, непредвиденного завершения синтаксических блоков.

Таблица 2.8. Методы класса FortranParser для построения полного AST файла

Метод	Описание
PushScopeOnStack	операция помещения блока на вершину стека. Особенность её работы заключается в том, что в случае регистрации нового глобального блока реализована возможность нештатного завершения уже существующих блоков (при помощи вызова метода **PopScopeToGlobal**)
PopScopeToGlobal	выполняет закрытие стека блоков до первого глобального, принимает во внимание тип добавляемого глобального блока. Если добавляется главная программа, модуль или блок данных, то стек очищается полностью (все его элементы завершаются внештатно). Если добавляется подпрограмма или функция, то возможно, что она добавляется в блок интерфейса или в другую высокоуровневую конструкцию (за исключением блока данных), которые допускают наличие внутренних процедур. В этом случае внештатном режиме закрываются все блоки, кроме любого допускающего наличие внутренних процедур. Закрытие блока производится при помощи метода **CloseUpperBlock**
PopScopeFromStack	обрабатывает инструкцию завершения синтаксического блока, учитывая тип закрываемого блока. Если встречена инструкция завершения высокоуровневого блока (например, end module), то проверяется, есть ли блок данного типа на стеке блоков, и если есть, то все открытые в данный момент блоки завершаются внештатном режиме, а сам высокоуровневый блок завершается в штатном режиме. Закрытие блока производится при помощи метода **CloseUpperBlock**
AddToCurrentBlock	является перегруженным и добавляет либо узел, либо другой блок в текущий синтаксический блок. В случае если в момент добавления на стеке нет блоков, создаётся блок главной программы. Выполняется проверка возможности закрытия цикла do с меткой «в особом режиме». В соответствии со стандартом, можно использовать специальные «помеченные» (labeled) циклы do, при этом несколько вложенных друг в друга циклов могут разделять одну и ту же числовую метку. В случае нахождения помеченной инструкции, числовая метка которой совпадает с меткой, использованной в объявлении цикла, производится закрытие этого цикла (или циклов, если метка является разделяемой). Для обработки данной ситуации реализован метод **CloseLabeledDoConstruct**
CloseLabeledDoConstruct	производит закрытие «помеченных» циклов do с учётом разделения циклами одной и той же числовой метки последней инструкции. Закрытие блока производится при помощи метода **CloseUpperBlock**

Таблица 2.8. Продолжение

CloseUpperBlock	закрывает блок (устанавливает свойство BlockConstruct.End), находящийся на вершине стека синтаксических блоков. Извлекает закрытый блок со стека и вызывает метод его окончательной обработки **ProcessClosingBlock**
ProcessClosingBlock	выполняет окончательную проверку блока извлекаемого со стека. Он вызывает соответствующий обрабатываемому блоку метод проверки синтаксической корректности: • **HandleProgram** – для главной программы; • **HandleModule** – для модуля; • **HandleMethod** – для подпрограммы и функции; • **HandleBlockData** – для конструкции блока данных; • **HandleDerivedType** – для производного типа данных; • **HandleInterface** – для интерфейса. Регистрирует блок как блок для выделения структуры кода.
Handle<XXX>	производят проверку блоков на наличие в них грубых синтаксических ошибок и в случае их обнаружения регистрируют соответствующее сообщение об ошибке. Типичный пример: в инструкции объявления блока использовалось имя Name1, а в инструкции завершения блока указано имя Name.

Рассмотрим алгоритм обработки инструкции достижения конца текущего файла исходного кода (метод HandleEOF) (рисунок 2.11).

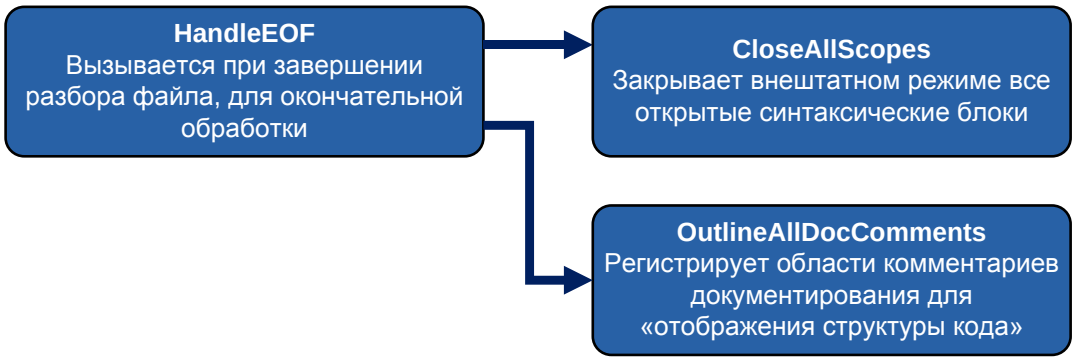

Рисунок 2.11 – Блок-схема работы метода HandleEOF

Основной задачей данного алгоритма является завершение операции разбора и формирование полного абстрактного синтаксического дерева с учётом возможных ошибок разбора. Так, если в стеке контекстов есть не закрытые контексты, то они завершаются в нештатном режиме (метод CloseAllScopes). Также здесь производится регистрация областей комментариев документирования для работы возможности «Отображение структуры кода» (метод OutlineAllDocComments).

Рассмотрим обработку комментариев документирования в алгоритме синтаксического анализа, а именно присоединение комментария документирования к тому узлу абстрактного синтаксического дерева, с которым он логически связан, на примере созда-

ния абстрактного синтаксического дерева для инструкции объявления функции. Необходимо отметить, что операции включения узлов с текстом комментариев документирования включены непосредственно в грамматику синтаксического анализатора.

```
// правило для объявления функции
function_stmt → prefix? 'FUNCTION' Identifier
{получить общее описание (doc_summary) и
описание результата (doc_result) функции, если они есть}
'(' arg_list? ')' suffix? end_of_stmt
{проверить правильность использования комментария документирования и
вернуть абстрактное синтаксическое дерево в виде
(AST_FUNCTION AST_TAG doc_summary
    (AST_NAME Identifier doc_result) (AST_ARGUMENTS arg_list?)
    (AST_ATTRIBUTES prefix? suffix?))
}
// правило для распознавания аргументов
arg_list → arg (',' arg)*
{вернуть абстрактное синтаксическое дерево в виде
(AST_NAME arg)+
}
// правило для распознавания одного аргумента
arg → Identifier {получить описание аргумента с текстом
идентификатора Identifier.text (doc_arg) и вернуть
абстрактное синтаксическое дерево в виде
(Identifier doc_arg)
}
// пустая продукция, соответствует пропущенному аргументу
| ε {вернуть абстрактное синтаксическое дерево в виде
(AST_MISSING)
}
```

Для получения различных данных из комментариев документирования (общего описания, описания параметров и результата), а также для проверки правильности использования комментария документирования для отдельно взятого значимого элемента были разработаны специализированные алгоритмы (выраженные в соответствующих методах) (таблица 2.9).

Таблица 2.9. Методы FortranParser для работы с комментариями документирования

Метод	Описание
GetDocCommentSummary	возвращает узел AST с типом DOC_COMMENT и текстом, соответствующим XML-тэгу summary комментария документирования. Комментарий документирования для текущей позиции разбора возвращается при помощи метода XmlDocCommentTokenStream.GetComment
GetDocCommentResult	аналогичен методу GetDocCommentSummary, но возвращает значение XML-тэга result
GetDocCommentForParam	возвращает узел AST с типом DOC_COMMENT и текстом, соответствующим тэгу param для функций и typeparam для производных типов данных, для параметра с заданным именем
CheckDocCommentUsage	проверяет правильность использования комментария документирования, включая синтаксические и иные ошибки в тексте комментария, множественное документирование для параметров с одинаковым именем, наличие в комментарии параметров, отсутствующих в фактическом значимом элементе и т.п.

Таким образом, алгоритм синтаксического анализа формирует абстрактное синтаксическое дерево, снабжённое специальными узлами для нераспознанных инструкций, XML комментариями документирования и т.д.

2.3.2.7 Алгоритм семантического анализа

Семантический анализатор (реализованный в виде класса FortranTreeWalker) осуществляет обход нормализованного абстрактного синтаксического дерева, подготовленного в синтаксическом анализаторе, находит узлы, соответствующие значимым элементам, создаёт для них соответствующие элементы и регистрирует в блоке хранения распознанных элементов.

Необходимо особо отметить, что абстрактное синтаксическое дерево строится в алгоритме синтаксического анализа таким образом, чтобы максимально упростить его обработку на этапе семантического анализа.

В качестве примера рассмотрим обработку модуля в алгоритме семантического анализа.

// правило для обработки модуля
module → *AST _ MODULE*
moduleStart {*обработать инструкцию начала модуля*}
body?
moduleEnd {*обработать инструкцию завершения модуля*}
// правило для обработки начала модуля
moduleStart → *AST _ TAG doc _ summary* (*AST _ NAME Identifier*)
{*запомнить содержимое комментария документирования и имя модуля*}

```
// правило для обработки инструкции завершения модуля
moduleEnd →
// ошибочное завершение модуля инструкцией END
AST_END AST_TAG
// правильное завершение модуля инструкцией END MODULE
| AST_END (AST_TAG AST_MODULE)
        (AST_NAME Identifier {запомнить имя модуля})?
// ошибочное завершение модуля (любой последовательностью токенов)
| AST_END_CONSTRUCT .*
```

Обработка и создание значимых элементов ведутся в методах вида HandleXXX. Поскольку большинство значимых элементов являются областями видимости (могут содержать дочерние элементы), то в семантическом анализаторе используется стек областей видимости (таблица 2.10) в терминах глобальной таблицы символов (об этом будет сказано в пункте 2.4).

Таблица 2.10. Поля и методы класса FortranTreeWalker, отвечающие за работу с областями видимости

Элемент	Описание
Поля	
Stack<IScope> _scopes	стек областей видимости. Область видимости соответствует интерфейсу IScope
Методы	
EnterScope(IScope scope)	помещает область видимости на вершину стека
IScope LeaveScope()	удаляет текущую область видимости из стека
IScope GetCurrentScope()	возвращают текущую область видимости с учётом желаемого типа области видимости
T GetCurrentScope<T>()	
Handle<XXX>Start(…)	создаёт значимый элемент (XXX) и помещает его в стек областей видимости (если элемент является областью видимости). Метод вызывается в процессе работы алгоритма семантического анализатора как часть правил разбора
Handle<XXX>End(…)	обработка завершающей инструкции значимого элемента. Производит извлечение области видимости со стека, если элемент является областью видимости

2.3.4 Алгоритм анализа Fortran программ для обеспечения подсветки синтаксиса

Данный алгоритм [68] предназначен для предоставления Visual Studio информации о подсветке синтаксиса, иначе говоря о том, какие цвета соответствуют символам, отображаемым в редакторе текста программы.

Ключевой особенностью Visual Studio является осуществление построчного анализа для подсветки синтаксиса и связанное с этим требование к анализаторам, заключающееся в умении сохранять и загружать своё состояние для продолжения разбора.

Основная лексическая особенность Fortran заключается в возможности использования многострочных токенов (см. пункт 2.3.1).

Указанные особенности позволяют сделать главный вывод: для подсветки синтаксиса не подходит лексический анализатор, используемый в алгоритме анализа для сбора информации о значимых элементах, поскольку он не удовлетворяет предъявляемому требованию. Кроме того, поддержка дочерних или встроенных в Fortran языков и расширенная поддержка внешних библиотек программ требуют совместной работы не только лексического анализатора, но и других механизмов.

Первая задача, которую необходимо решить, это обеспечение в специализированном лексическом анализаторе поддержки сохранения состояния разбора и возможности продолжить разбор с сохранённого таким образом среза. Для хранения состояния лексического разбора доступно одно 32-х битное целое число, поэтому все состояния необходимо кодировать битами в этом числе.

Для кодирования типа токена используется перечисление ContinuatedTokens (таблица 2.11), а для кодирования состояний разбора лексического анализатора - перечисление ContinuatedRules (таблица 2.12). Совместно значения этих двух перечислений позволяют однозначно идентифицировать текущее состояние разбора и продолжить дальнейший разбор с указанного состояния. Для кодирования состояния — продолжается ли текущая строка — используется битовый флаг.

Таблица 2.11. Перечисление ContinuatedTokens для кодирования типа токена

Член перечисления	Описание
None	нет токена
Dot	точка (далее либо вещественное число, либо пользовательский оператор)
Identifier	идентификатор, далее либо BOZ-константы, либо идентификатор
DefinedOperator	пользовательский оператор. Совпали точка и буква
BinaryConst	двоичный литерал, совпали b'
OctalConst	восьмеричный литерал, совпали o'
HexConst	шестнадцатеричный литерал, совпали z'
String	строковый литерал, совпали ' или "
DigitString	целое число, совпала любая десятичная цифра
RealConst	вещественное число
HollerithConst	константа Холлерита, совпало, например: 2h/=

Таблица 2.12. Перечисление ContinuatedRules для кодирования состояний разбора лексического анализатора

Член перечисления	Описание
None	начальное состояние
DefinedOperatorContinued	продолжается пользовательский оператор
IdentifierContinued	продолжается идентификатор
GeneralString	ожидается продолжение строки, но ещё не было совпадений с кавычками
SingleQuoteStringContinued	продолжается строка с одинарной кавычкой
DoubleQuoteStringContinued	продолжается строка с двойной кавычкой
ExponentL1	продолжается экспонента на первой стадии
ExponentL2	продолжается экспонента на второй стадии
DigitStringContinued	продолжается строка символов
RealConstContinued	продолжается вещественная константа

Для описания состояния специально разработанного лексического анализатора для подсветки синтаксиса FortranColoribleLexer создана структура ScannerState (таблица 2.13). Данная структура непосредственно используется в его работе и отвечает за представление состояния разбора как в виде 32-х битного целого числа, так и в виде набора из 3-х свойств (кода токена, кода лексического правила и признака переноса строки).

Таблица 2.13. Описание полей и свойств структуры ScannerState.

Элемент	Описание
bool IsContinued	признак, что текущая строка продолжается
ContinuatedRules Rule	код лексического правила для текущего токена
ContinuatedTokens Token	код токена, разбор которого ведётся
int State	состояние анализатора как 32-х битное целое

Алгоритм специализированного лексического анализа (FortranColoribleLexer) был разработан с учётом необходимости вести построчный разбор и постоянной синхронизации своего внутреннего состояния. Отметим, что для реализации возможности продолжать разбор с произвольного состояния лексические правила были записаны специальным образом – они были разбиты на минимально допустимое количество уникальных непересекающихся частей.

Рассмотрим специализированный алгоритм лексического анализа на примере разбора вещественного литерала, который является наиболее сложным для разбора.

// правило разбора целочисленного, вещественного или
// строкового литерала в форме Холлерита
DRH →
// целое или вещественное число
Digit {код токена = DigitString, код правила = DigitStringContinued}
DigitStringContinued {установить тип токена, согласно его коду}
// вещественное число
|'.'{код токена = RealConst, код правила = RealConstContinued}
RealConstContinued {установить тип токена, согласно его коду}
// правило для разбора целого или вещественного литерала,
// а так же строкового литерала Холлерита
DigitStringContinued → Digit *
// если за целым числом идёт '.', то это вещественный литерал
('.'{код токена = RealConst} RealConstContinued ?
// если за целым числом идёт экспоненциальная запись, то это вещественный литерал
|Exponent {код токена = RealConst}
// если за целым числом идёт 'h', то это специальный строковый литерал
|'h'{код токена = HollerithConst; считываем столько цифр, сколько указано перед 'h'}
// пустая продукция
|ε)
// правило для разбора вещественного литерала
RealConstContinued → {код правила = RealConstContinued}
(Digit + Exponent ?|Exponent)
// правило для разбора экспоненциальной части вещественного литерала
Exponent → ('e'|'d'){код правила = ExponentL1} ExponentL1
// правило для продолжения разбора экспоненциальной части
ExponentL1 → ('+'|'−')?{код правила = ExponentL2} ExponentL2
// правило для завершения разбора экспоненциальной части
ExponentL2 → Digit +
// правило для распознавания цифр
Digit → '0'..'9'

Из приведённого фрагмента грамматики видно, как происходит синхронизация состояния лексического анализатора со структурой ScannerState в направлении «лексический анализатор → вектор состояния» путём установки кодов текущего токена и правила разбора. В свою очередь, пара кодов текущего правила и токена позволяют одно-

значно определить, как необходимо продолжить разбор на основе сохранённого состояния. Вначале анализируется код токена Token (таблица 2.14).

Таблица 2.14. Соответствие кодов токенов методам их дальнейшей обработки

Код токена	**Вызываемое правило**	
	Имя метода	**Описание**
Dot	HandleDotToken()	точка может принадлежать как DefinedOperator так и RealConst
Identifier	HandleIdentifier()	обработка идентификатора
DefinedOperator	HandleDefinedOperator()	обработка оператора '.'буквы'.'
BinaryConst	HandleBOZConstToken()	обработка b'…' или o'…' или z'…'
OctalConst		
HexConst		
String	HandleStringToken()	обработка строкового литерала
DigitString	HandleDigitString()	обработка строки цифр
RealConst	HandleRealConstToken()	обработка вещественного литерала
HollerithConst	не обрабатывается	

В зависимости от кода токена вызывается один из методов специальной обработки вида HandleXXX. Каждый из этих методов, в свою очередь, анализирует правило продолжения и в зависимости от этого вызывает те или иные методы лексического анализатора. Рассмотрим пример с обработкой вещественной константы (листинг 2.2).

```
function HandleRealConstToken() returns(void)
if (если достигнут конец файла) then
  код токена = нет токена; return
endif //далее проверка кода правила
//никакое правило не продолжается
if (код правила == нет правила) RealConstContinued()
//продолжается вещественный литерал
if (код правила == RealConstContinued) RealConstContinued()
//перешли к разбору экспоненциальной части
if (код правила == ExponentL1) ExponentL1()
//перешли к разбору порядка экспоненциальной части
if (код правила == ExponentL2) ExponentL2()
end function
```

Листинг 2.2 – Пример восстановления разбора на основе сохранённого состояния

Необходимо сделать важное замечание, касающееся алгоритма восстановления лексического анализа на основе сохранённого состояния. Как показано в листинге 2.2, коды текущих правил в точности соответствуют именам лексических правил, которые необходимо вызвать для продолжения разбора. Код токена необходим также для определения того, какой тип будет иметь полученный в результате анализа токен.

Таким образом, алгоритм специального лексического анализа формирует набор элементарных токенов для использования в дальнейшем анализе.

Рассмотрев схему работы лексического анализатора для языка Fortran, который является основным языком, рассмотрим, как он встраивается в инфраструктуру языкового сервиса, вызываемого из Visual Studio. В реализации возможности подсветки синтаксиса участвуют: языковой сервис, колорайзер и сканер (таблица Б.1). При этом именно сканер предоставляет построчную информацию о назначаемых символам строки цветах. Во FRIS сканером является специально разработанный класс FortranColorizerScanner.

Рассмотрим, как происходит процесс колоризации, включающий в себя возможности расширенной поддержки дочерних языков и внешних проблемно-ориентированных библиотек программ, в частности встроенную поддержку библиотек MPI, OpenMP, а также УРС-ОФ и ЕФР РФЯЦ-ВНИИЭФ (рисунок 2.12).

Рисунок 2.12 – Общая схема процесса подсветки синтаксиса

Метод SetSource вызывается перед началом разбора и устанавливает текст строки для анализа. Метод SkanTokenAndProvideInfoAboutIt (см. таблицу Б.2) возвращает информацию о каждом токене, найденном в строке; в качестве второго аргумента метод принимает ссылку на состояние лексического анализатора в момент разбора. Он содержит основную логику работы реализации возможности подсветки синтаксиса. Если производится первое обращение, то запускается лексический анализ, при этом учитывается возможность работы более чем с одним языком (рисунок 2.13).

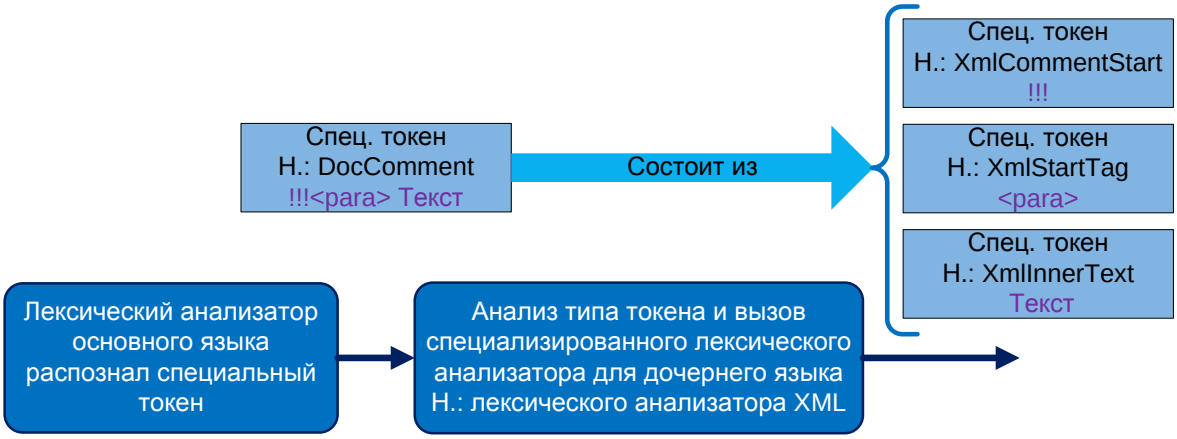

Рисунок 2.13 – Общая схема обработки дочерних языков в процессе подсветки синтаксиса во FRIS

За анализ токена и запуск лексических анализаторов дочерних языков отвечает метод ApplyFilters. Для работы с подсветкой синтаксиса XML-комментариев документирования был разработан специальный лексический анализатор XmlDocCommentColoribleLexer.

В некоторых случаях, например при обработке идентификаторов, производится более детальный анализ. В частности, не является ли идентификатор ключевым словом (метод IsKeyword), при необходимости привлекается информация из блока хранения распознанных элементов.

Если идентификатор является именем процедуры (проверка контекста выполняется в методе IsMethodNameContext), то проверяется, является ли он процедурой из поддерживаемых внешних библиотек программ. В частности: встроенной в язык (intrinsic) процедурой; процедурой прикладного программного интерфейса OpenMP; процедурой библиотеки MPI, УРС-ОФ или ЕФР.

Если же идентификатор является именем производного типа данных (проверка контекста осуществляется в методе IsTypeNameContext), то также проверяется, принадлежит ли он внешней библиотеке программ, например, УРС-ОФ.

Для всех специальных токенов устанавливаются отличительные цвета, визуально выделяющие их из остального текста.

Таким образом обеспечивается реализация требований по учёту специфики написания современных сложно структурированных программ: обеспечивать визуальное выделение элементов внешних библиотек программ: MPI, OpenMP, УРС-ОФ и ЕФР; работать в процессе написания текстов программ.

Важной особенностью сканера является также установка дополнительных атрибутов структуры TokenInfo (таблица Б.3). Наиболее важным из них является триггер события (поле Trigger). Например, при нахождении в тексте символа «%» можно однозначно утверждать, что пользователь хочет начать вводить имя компонента производного типа данных, и ему можно в этом помочь, автоматически активировав возможность IntelliSense «Построение списка элементов сложного объекта». Для этого взводится флаг TokenTriggers.MemberSelect. Аналогичные флаги взводятся при обнаружении открывающей и закрывающей круглых скобок и запятых внутри них – для поддержки возможности IntelliSense «Отображение сведений о параметрах процедур».

Реализованный во FRIS анализатор для подсветки синтаксиса полностью удовлетворяет требованиям, накладываемым Visual Studio на подобные анализаторы (скорость работы для анализа файла средних размеров порядка 10 мс). Проведённые испытания показали, что анализ 40000 строк Fortran-программы производится на ПЭВМ (Intel Pentium 4 3.1 ГГц, 1 Гбайт ОЗУ) за 76 мс, при этом анализ файла программы среднего размера 5000 строк производится менее чем за 10 мс.

2.4 Блок хранения распознанных элементов. Структура и основные классы

Блок хранения распознанных элементов является подобием базы данных всех известных в данном программном проекте элементов. Он наполняется из двух источников: в результате разбора файлов текстов программ и в результате десериализации информации о внешних библиотеках.

Данный блок по своей сути является разновидностью таблицы символов. При его проектировании необходимо принимать во внимание, что информация в нём будет непрерывно обновляться в результате редактирования пользователем различных файлов с

текстами программ. Рассмотрим типовую модель блока хранения, использующуюся во FRIS (рисунок 2.14).

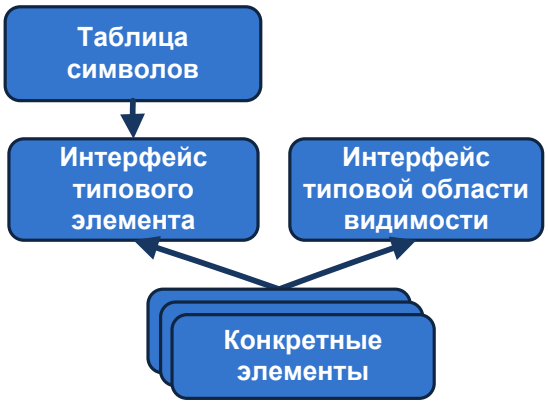

Рисунок 2.14 – Модель блока хранения распознанных элементов

В нём можно выделить следующие части:

- класс для описания таблицы символов (или кэша элементов);
- класс для описания интерфейса типового элемента языка программирования;
- класс для описания интерфейса типовой области видимости языка программирования;
- классы описания конкретных элементов языка программирования, реализующие интерфейсы типового элемента и типовой области видимости, для элементов, являющихся областями видимости.

Класс для описания таблицы символов должен быть построен по принципу индексированного хранилища данных для того, чтобы эффективно обрабатывать операции обновления и поиска элементов. Для максимальной гибкости он должен хранить ссылки на интерфейс типового элемента, а не на конкретные элементы. Они могут быть получены из абстрактного интерфейса с помощью операции приведения типов. Рассмотрим схему организации таблицы символов (таблица 2.15).

Таблица 2.15. Модель таблицы символов

Поле	Тип данных	Описание
Names	map<long, string>	словарь соответствий уникальных идентификаторов строкам
Elements	map<long, object>	словарь соответствий уникальных идентификаторов объектам
Projects	map<string, map<string, list<long>>>	словарь соответствий имён программных проектов словарю соответствий имён файлов, элементам этих файлов

Таблица 2.15. Продолжение

ProjectDependencies	map<string, list<string>>	словарь соответствий программных проектов списку программных проектов, от которых они зависят

При таком подходе, во-первых, есть доступ ко всем элементам (поле Elements). Во-вторых, для каждого проекта поддерживается список зависимостей данного проекта от других проектов, что позволяет упростить процедуру поиска необходимых элементов и не включать в результат поиска те элементы, которые не видны в целевом проекте. В-третьих, каждый проект содержит словарь входящих в него файлов и элементов, содержащихся в каждом из файлов, что позволяет эффективно выполнять операции обновления. Операция обновления является результатом операции разбора файла вследствие изменения его текста пользователем. Таким образом, поскольку известны все элементы, связанные с файлом, то их изъятие из остальных словарей и добавление в них вновь распознанных элементов является довольно простой задачей.

Рассмотрим интерфейс для типового элемента языка программирования (таблица 2.16).

Таблица 2.16. Модель интерфейса для типового элемента языка программирования

Поле	Тип данных	Описание
Name	string	имя элемента
Scope	Scope	внешняя область видимости элемента
Description	string	описание элемента, например из комментария документирования
Location	Location	местоположение элемента: местоположение объявления и определения. Местоположение состоит из имени файла и области. Область состоит из 4-х целых чисел: номер стартовой строки, номер стартового символа в строке, номер конечной строки, номер конечного символа в строке

Каждый элемент должен иметь по крайней мере имя, область видимости, в которой он определён, смысловое описание, например получаемое из комментариев документирования, и местоположение. Местоположение элемента состоит из местоположения объявления и местоположения определения. Каждое из них, в свою очередь, содержит имя файла и местоположение элемента в данном файле.

Рассмотрим интерфейс для типовой области видимости языка программирования (таблица 2.17). Область видимости в самом общем случае является контейнером элементов.

Таблица 2.17. Модель интерфейса для типовой области видимости

Поле	Тип данных	Описание
Scope	Scope	внешняя область видимости данной области видимости
Elements	list<Element>	список элементов области видимости

Каждая область видимости содержит ссылку на родительскую область видимости и список элементов, составляющих данную область видимости.

Любой конкретный элемент языка программирования должен наследоваться от интерфейса типового элемента и, если он является областью видимости, от интерфейса типовой области видимости. Заметим, что конкретные элементы полностью соответствуют модели значимых элементов, разработанной в первой главе. Блок хранения распознанных элементов во FRIS (рисунок 2.15) реализует модель значимых элементов для языка Fortran (изложенную в пункте 1.3) с учётом вышеизложенного подхода к организации блока распознанных элементов (рисунок 2.14).

Блок состоит из трёх составных частей:

- классы, описывающие значимые элементы Fortran (таблицы Б.10-Б.14), включая интерфейс типового элемента ICachedElement (таблица Б.10) и области видимости IScope (таблица Б.11);
- класс глобального кэша FortranCache (таблица Б.15), выполняющий хранение всех распознанных элементов во FRIS, аналог таблицы символов;
- вспомогательные классы и структуры, повышающие эффективность обработки и хранения данных.

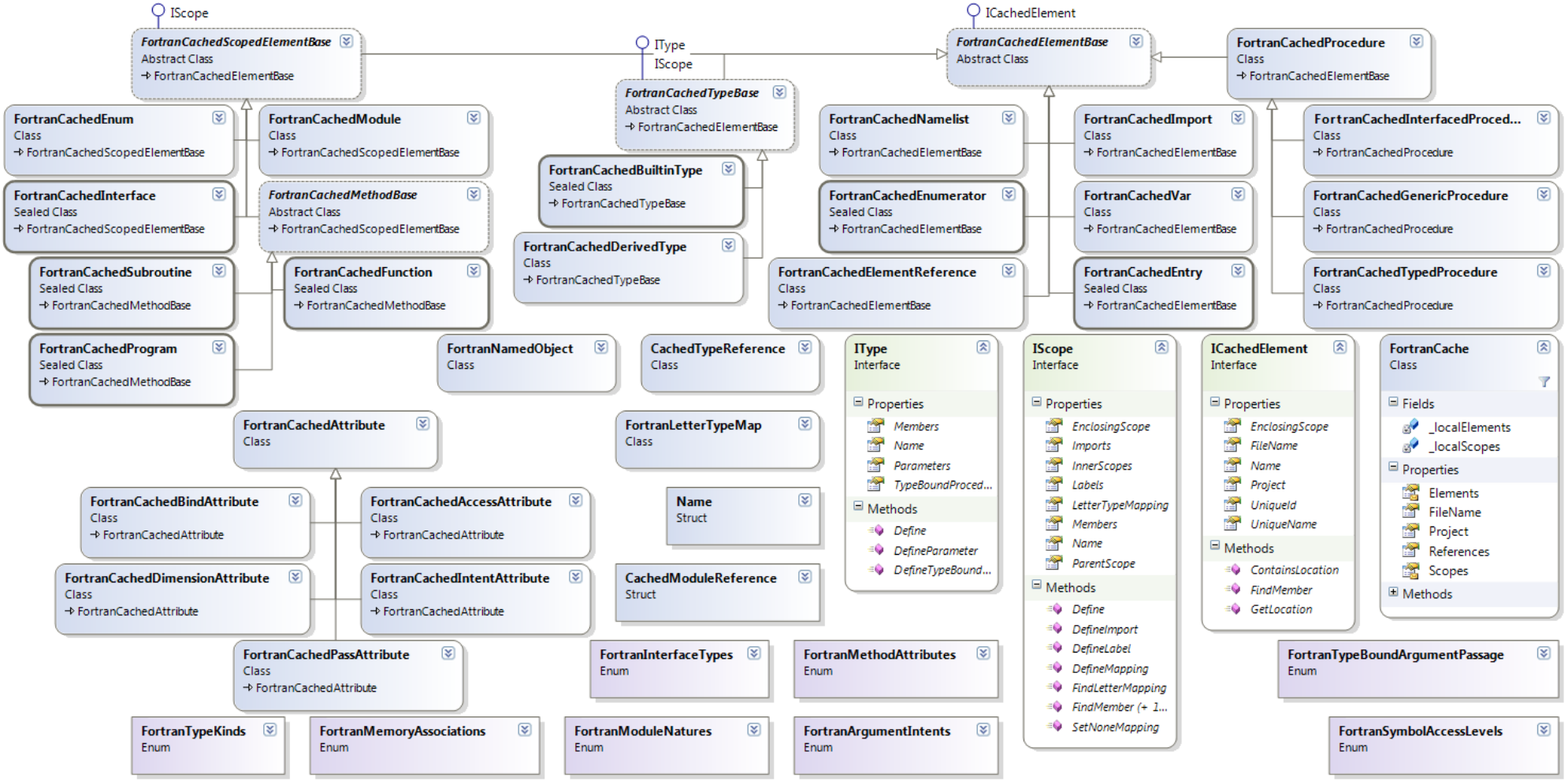

Рисунок 2.15 – Диаграмма классов значимых элементов и глобальной таблицы символов во FRIS

2.5 Блок модели представления значимых элементов

Блок модели представления является связующим звеном между блоком интеграции с IDE и блоком хранения данных. Он производит две основные функции: приводит данные блока хранения к виду, необходимому для IDE, и выполняет различные операции поиска информации в блоке хранения. Классы модели представления значимых элементов, реализованные во FRIS представлены на рисунке 2.17.

Операция приведения хранимых данных к виду, необходимому IDE, производит дополнение элементов свойствами визуального представления (рисунок 2.16). Например, задаются такие свойства, как цвет текста и иконка для обозначения элемента в различных списках автодополнения, и т.д. Иными словами, блок модели представления содержит аспекты представления данных пользователю. Поэтому структура блока модели представления аналогична блоку хранения. В нём так же определяются интерфейсы для типовых элементов представления (таблица Б.17), области видимости и набор конкретных их реализаций для каждого элемента блока хранения (таблица Б.18).

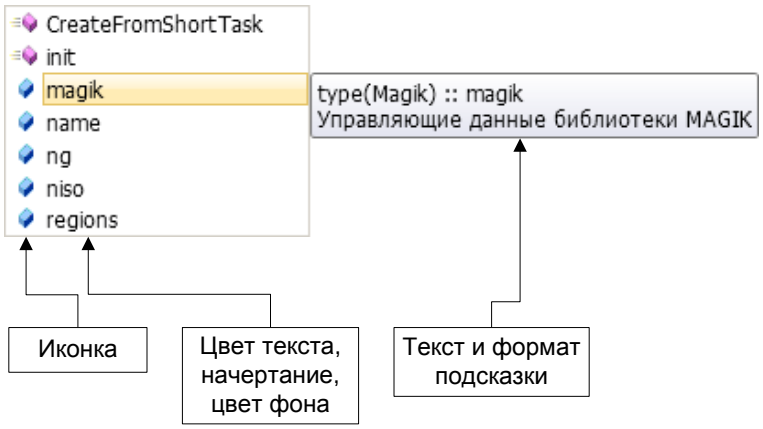

Рисунок 2.16 – Информация, необходимая для визуального представления значимых элементов пользователю

Вторая функция, которая реализуется в данном блоке, – это функция поиска с учётом текущей области видимости. То есть производится выборка данных из блока хранения с учётом различных аспектов языка программирования. Затем, выбранные данные приводятся к виду, необходимому для представления пользователю. Например, встретив выражение «myVal%Sub1(», необходимо сначала разобрать его на составные части: «myVal», «%», «Sub1» и «(». Далее найти в текущей области видимости переменную с именем «myVal».

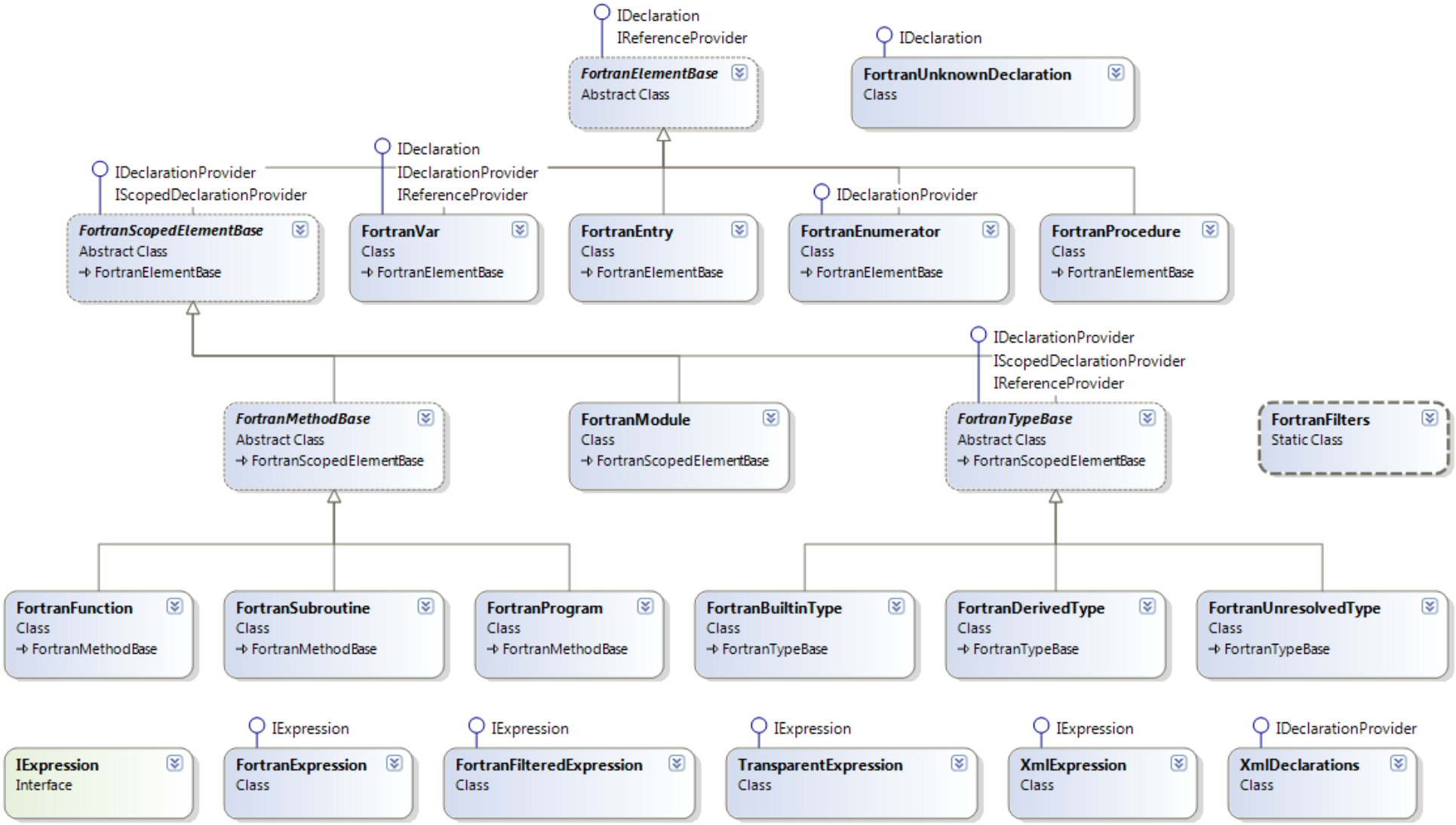

Рисунок 2.17 – Диаграмма классов модели представления значимых элементов

Найти определение её типа данных, в этом типе найти элемент с именем «Sub1» и далее, если это процедура, показать список её аргументов, а в случае, если это массив, ничего не предпринимать. Алгоритм для определения контекста и предоставления помощи приведён в листинге 2.3.

```
//функция возвращает список элементов для отображения пользователю
function GetContextHelp(line, col, reason) returns(ShowToUser)
startLine, startCol = определить номер начальной строки и символа в строке для ин-
струкции,  которой принадлежит файловая позиция (line, col)
instructionText = получить текст файла в диапазоне [(startLine, startCol); (line, col)]
AST = получить абстрактное дерево разбора для instructionText
CurrentElement = получить элемент, являющийся областью видимости, которому
принадлежит файловая позиция (line, col)
Context = FullContext = вычислить полный контекст для элемента CurrentElement
согласно правилам секции 16 [6]
if (reason не равен автодополнение) then
  for (node: узлы AST начиная с корневого)
    if (узел node является последним) then
      Context = найти все элементы с именем node.Name в контексте FullContext
    else
      Element = найти элемент с именем node.Name в контексте Context
      Element =найти тип элемента Element в контексте FullContext
      Context = получить все компоненты типа Element
    endif
  endfor
endif
ShowToUser = для всех элементов контекста Context создать элементы из блока
модели представления, с учётом фильтров, налагаемых обрабатываемой инструк-
цией
end function
```

Листинг 2.3 – Алгоритм определения текущего контекста и предоставления помощи

Для выполнения подобных операций была разработана инфраструктура классов (таблица Б.19-Б.21).

Особо отметим, что для удовлетворения требования по учёту специфики написания современных сложноструктурированных программ в части встроенной поддержки внешних проблемно-ориентированных библиотек программ и встроенной поддержки средств параллельного программирования MPI, OpenMP и SIMD-операций в контекст *FullContext* включаются также элементы этих внешних библиотек — MPI, OpenMP, а для РФЯЦ-ВНИИЭФ также УРС-ОФ и ЕФР; а во множество *ShowToUser* включаются сниппеты (шаблоны кода) OpenMP и SIMD-операций.

Наибольший интерес представляет алгоритм построения актуального контекста с учётом импорта элементов модулей. Каждый модуль знает лишь об именах тех модулей, которые он сам импортирует, но не знает об их содержимом Также он не знает о том, в каких элементах этот модуль используется (рисунок 2.18).

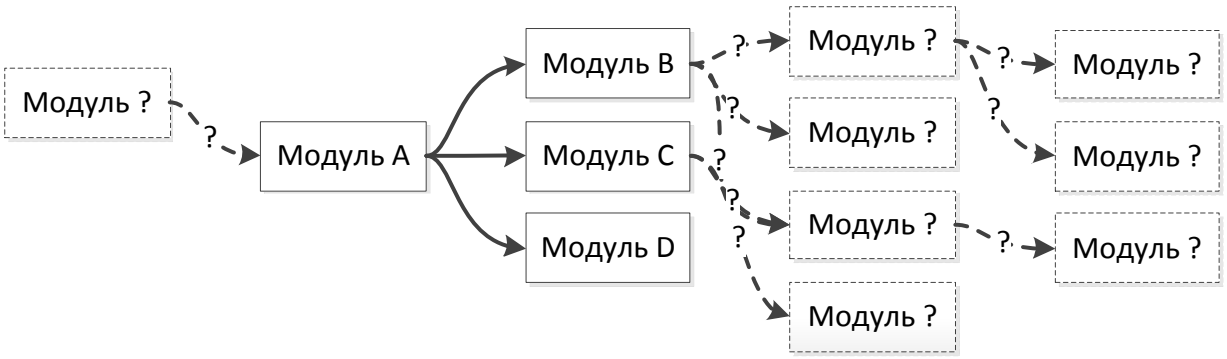

Рисунок 2.18 – Схема импорта модулей

При этом каждый импортируемый модуль может, в свою очередь, входит в цепочку импорта, т.е. импортировать содержимое произвольного количества других модулей. Более того, у модуля есть два контекста (управляемые атрибутами доступа private/public): внутренний – все элементы модуля, а также все публичные элементы, импортируемые из других модулей; экспортируемый, или внешний, – это те элементы, которые объявлены в модуле общедоступными. Например, модуль может импортировать к себе контексты, скажем, трех других моделей, но экспортировать только одну переменную, объявленную в нём самом.

Учитывая динамически изменяющийся характер информации в таблице символов для операции построения полного контекста, для модулей были разработаны специализированные структуры данных (рисунок 2.19).

Рисунок 2.19 – Схема структур данных для построения контекстов

Каждый модуль содержит список инструкций импорта, внутренние элементы, внутренний и внешний контексты, а также метку времени – когда модуль был помещён в таблицу символов. Каждая инструкция импорта содержит имя модуля, список импортируемых имён (с учётом переименований), а также импортируемый контекст – ссылку на экспортируемый контекст целевого модуля. Каждый контекст содержит словарь доступных значимых элементов: ключ – имя элемента, значение – список элементов с данным именем, а также метку времени – создание указанного контекста.

Для уменьшения времени операции построения контекста импортируемые в модуль контексты запоминаются в инструкциях импорта. Перестроение контекста производится методом поиска в глубину, когда запомненный контекст отличается от экспортируемого контекста целевого модуля. Таким образом, для большинства модулей удаётся пользоваться построенным и сохранённым контекстом.

Отметим, что блок сериализации/десериализации содержит для каждого класса из блока хранения распознанных элементов специализированные процедуры чтения и записи для Fortran API и для XML комментариев документирования. Поскольку работа с XML является стандартной, то её описание в рамках данной работы не представляется

необходимым. Отметим лишь, что формат записи изложен в пункте 1.5.1, 1.5.2 и приложении А.

Выводы к главе 2

1. В главе изложены подходы и программная реализация абстрактной модели языкового сервиса в языковом сервисе FRIS, обеспечивающем расширенную поддержку языка Fortran 2003 в Microsoft Visual Studio и учитывающем специфику написания современных сложноструктурированных программ.
2. Реализован блок анализа текста для сбора информации о значимых элементах Fortran:
   - построенный по многостадийной схеме транслятора и состоящий из потока символов, лексического анализатора, препроцессора, потока токенов (или терминалов), синтаксического анализатора, абстрактного синтаксического дерева, семантического анализатора;
   - учитывающий основные особенности языка Fortran, представляющие сложности для лексического и синтаксического анализа;
   - учитывающий необходимость выполнения анализа в реальном времени, когда текст анализируемой программы заведомо находится в некорректном с точки зрения спецификации языка программирования состоянии.
3. Реализован блок анализа текста в реальном времени для подсветки синтаксиса Fortran-программ с учётом построчного разбора, сохранения и восстановления состояния разбора, поддержкой выделения элементов внешних библиотек. В частности — технологий параллельного программирования MPI, OpenMP и общезначимых библиотек РФЯЦ-ВНИИЭФ УРС-ОФ, ЕФР.
4. Разработан блок хранения распознанных элементов, построенный по принципу таблицы символов и реализующий модель значимых элементов Fortran, изложенную в первой главе.
5. Реализован блок модели представления элементов, готовящий данные хранящиеся в блоке хранения, для использования в возможностях технологии IntelliSense, и являющийся, таким образом, промежуточным слоем между блоком хранения и блоком интеграции с Visual Studio.

Глава 3. Подтверждение работы FRIS

В РФЯЦ-ВНИИЭФ языковой сервис FRIS используется [76] при разработке следующих программ (таблица 3.1) для:

- моделирования работы специзделий в:
  - программах: КПД-1D, АРКТУР, ЛЭГАК-3D [106];
  - библиотеке программ расчёта макроконстант среды и кинетики ядер MAGIK [46] для программ: КПД-1D, АРКТУР и САТУРН [79-80];
- решения распределённых разреженных линейных алгебраических уравнений в библиотеке Newt [28,107];
- моделирования процессов в пакете прочностного анализа ЛЭГАК-ДК [108];
- моделирования воздушно-космического нападения и обороны в визуализационно-интегрирующей платформе Алькор [109].

Таблица 3.1. Основные сведения о программах

№	Программа	Объём, строк	Использование внешних библиотек	
			Количество	Языки
1.	**ЛЭГАК**	более 800 000	9	Fortran 90, C/C++
2.	**КПД-1D**	более 500 000	9	Fortran 90/2003, C++, Delphi
3.	**АРКТУР**	более 200 000	7	Fortran 90/2003, C++, Delphi
4.	**CONCORD**	более 150 000	6	Fortran 90, C++, Delphi
5.	**Алькор**	более 500 000	15	Fortran 90/2003, C/C++
6.	**MAGIK**	76 000	3	C++
7.	**Newt**	более 100 000	3	C++

Особо необходимо подчеркнуть, что во FRIS встроена поддержка:

- технологий параллельного программирования и высокопроизводительных вычислений MPI, OpenMP и SIMD-операций;
- внешних проблемно-ориентированных библиотек программ, которыми в РФЯЦ-ВНИИЭФ являются УРС-ОФ и ЕФР.

3.1 Встроенная поддержка технологий параллельного программирования и высокопроизводительных вычислений

При разработке FRIS уделялось внимание специфике написания сложноструктурированных параллельных программ и особенностям использования современных СуперЭВМ. В таблице 3.2 представлен ряд проблем по написанию программ для современных СуперЭВМ, выделенных сотрудниками РФЯЦ-ВНИИЭФ, и возможности FRIS,

позволяющие решить проблемы. Таким образом, FRIS способствует упрощению, повышению надёжности и скорости программирования и в целом модификации параллельных программ в сжатые сроки путём предоставления специального инструментального средства – проблемно-ориентированного языкового сервиса.

Таблица 3.2. Возможности FRIS, позволяющие упростить написание параллельных программ на языке Fortran для современных СуперЭВМ

Описание проблемы	Возможности FRIS позволяющие решить проблему
Необходимость эффективного распараллеливания программ при помощи средств OpenMP	Реализована поддержка спецификации OpenMP 4.0 (см. пункт 3.1.2)
Векторизация арифметических операций	Реализована поддержка SIMD-операций (см. пункт 3.1.3)
Раздельная память устройств	Реализована поддержка спецификации MPI 3.1 (см. пункт 3.1.1)
Учёт гетерогенности в виде различия отдельных блоков вычислений на процессорах и сопроцессорах для повышения производительности	Реализована поддержка в виде выделения цветом и жирным шрифтом вызова стандартной процедуры библиотеки MPI – **MPI_Get_processor_name**(name, resultlen, ierror) в аргументе name которой возвращается имя вычислительного устройства СуперЭВМ, выполняющего алгоритм. Анализ возвращаемого в этом аргументе значения, характеризующего, например, тип устройства, позволяет использовать оптимальную для данного устройства ветвь вычислительного алгоритма.
Необходимость переработки всех программ	FRIS обеспечивает поддержку программиста на этапе написания/модификации текста программ различными видами контекстной помощи, что позволяет переписывать программы в сжатые сроки. Так же во FRIS встроена поддержка общеиспользуемых в РФЯЦ-ВНИИЭФ библиотек УРС-ОФ и ЕФР (см. пункт 3.2)
Усложняется разработка программ, так как инструментальные средства разработчика не являются отработанными	FRIS позволяет упростить процесс программирования за счёт предоставления специализированного инструментального средства разработчика, учитывающего специфику разработки современных программ

3.1.1 Поддержка MPI

Во FRIS поддерживается стандарт MPI 3.1 [23] для распараллеливания вычислений на распределённой памяти. С точки зрения использования в программе MPI является набором функций следующих классов (групп):

1) программы для коммуникаций типа «точка-точка»;
2) программы для коллективных коммуникаций;
3) служебные программы для работы с коммуникаторами и группами процессов;
4) программы для создания топологий;
5) программы для работы с окружением;
6) программы для порождения и управления процессами;

7) программы для односторонних коммуникаций;
8) программы ввода/вывода.

На рисунках 3.1-3.3 приведены все основные возможности FRIS по поддержке MPI 3.1: выбор нужной процедуры из списка доступных процедур, выделение процедур MPI цветом, предоставление информации о параметрах процедур и их автодополнение.

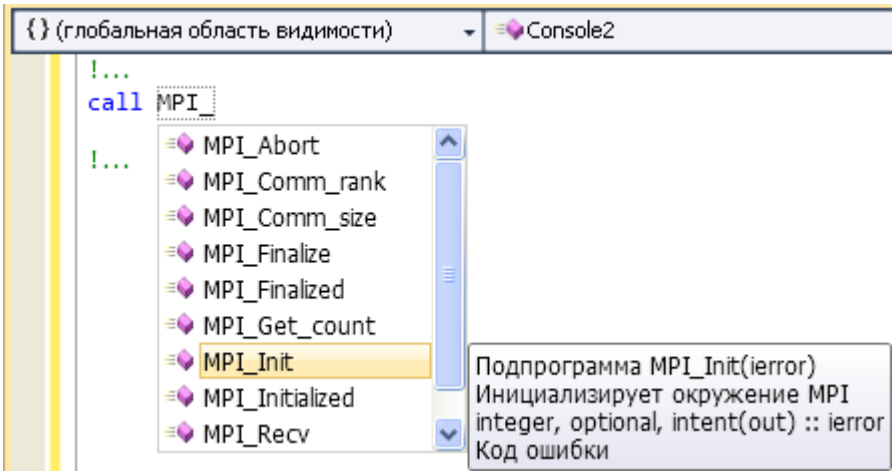

Рисунок 3.1 – Автодополнение имени процедуры для библиотеки MPI 3.1

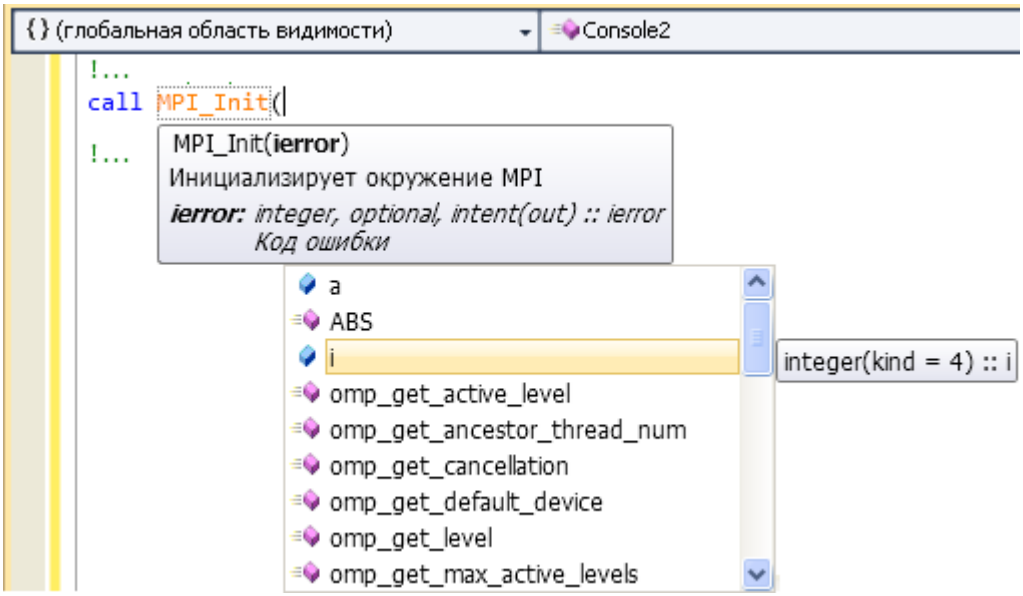

Рисунок 3.2 – Отображение сведений об аргументах процедуры и их автодополнение

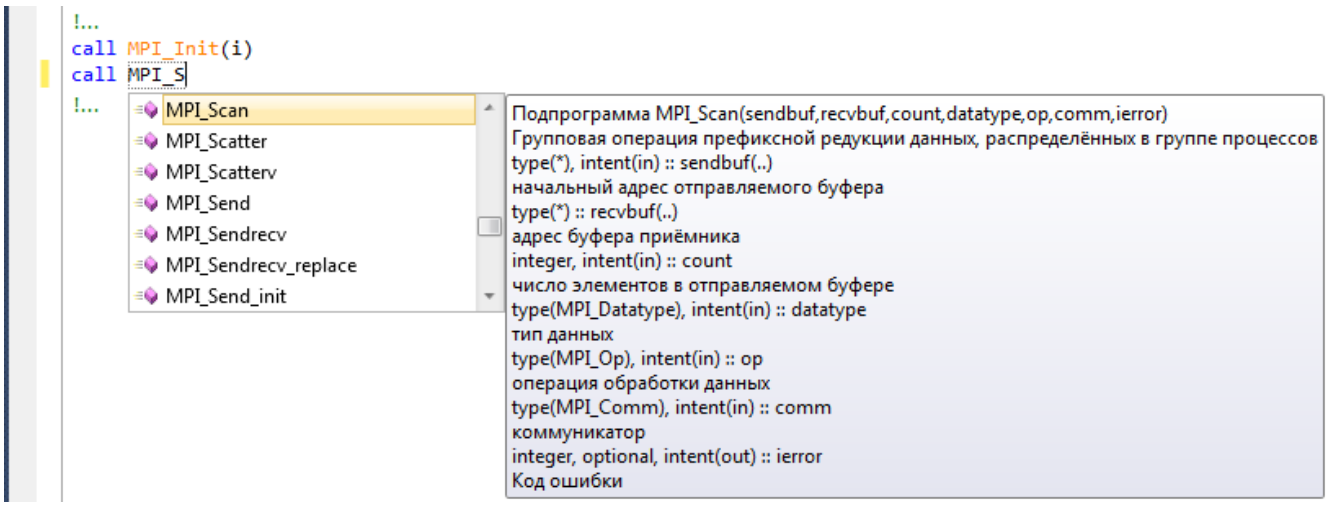

Рисунок 3.3 – Автодополнение имени процедуры и вывод краткой информации для MPI_Send

Особого внимания заслуживает пример функции отправки сообщения двухточечного блокирующего обмена MPI_Send (рисунок 3.3), а точнее описание её первого аргумента «buf» - буфера передаваемых данных. Для его описания используется расширение стандарта Fortran, утверждённое в технической поправке [110] к стандартам Fortran 2003 и Fortran 2008. В частности, запись «type(*)» означает использование любого типа данных, а запись «**(..)**» (в «buf**(..)**»), или, что эквивалентно «DIMENSION(..)» - использование массива любой размерности, или скаляра. Поддержка данных особенностей была добавлена во FRIS для полноценной поддержки MPI 3.1.

3.1.2 Поддержка OpenMP

Во FRIS встроена поддержка стандарта OpenMP 4.0 [24], который предназначен для обеспечения параллельной работы выделенных блоков программы в рамках одного программного процесса в многоядерной среде. С точки зрения использования в программе OpenMP состоит из двух частей:

- процедуры OpenMP, служащие для получения информации об окружении и его свойства, а также для управления окружением, в котором происходит выполнение параллельной программы;
- директивы компилятора, служащие для распараллеливания программы на общей памяти и других задач, связанных с этим процессом (контроль доступа к памяти и т.д.).

Рассмотрим поддержку стандарта параллельного программирования OpenMP 4.0 во FRIS. Прежде всего это касается наличия всех функций стандарта и выделения их со-

ответствующим цветом. На рисунке 3.4 показано выделение имени функции OpenMP зелёным цветом и предоставление помощи при заполнении её аргументов.

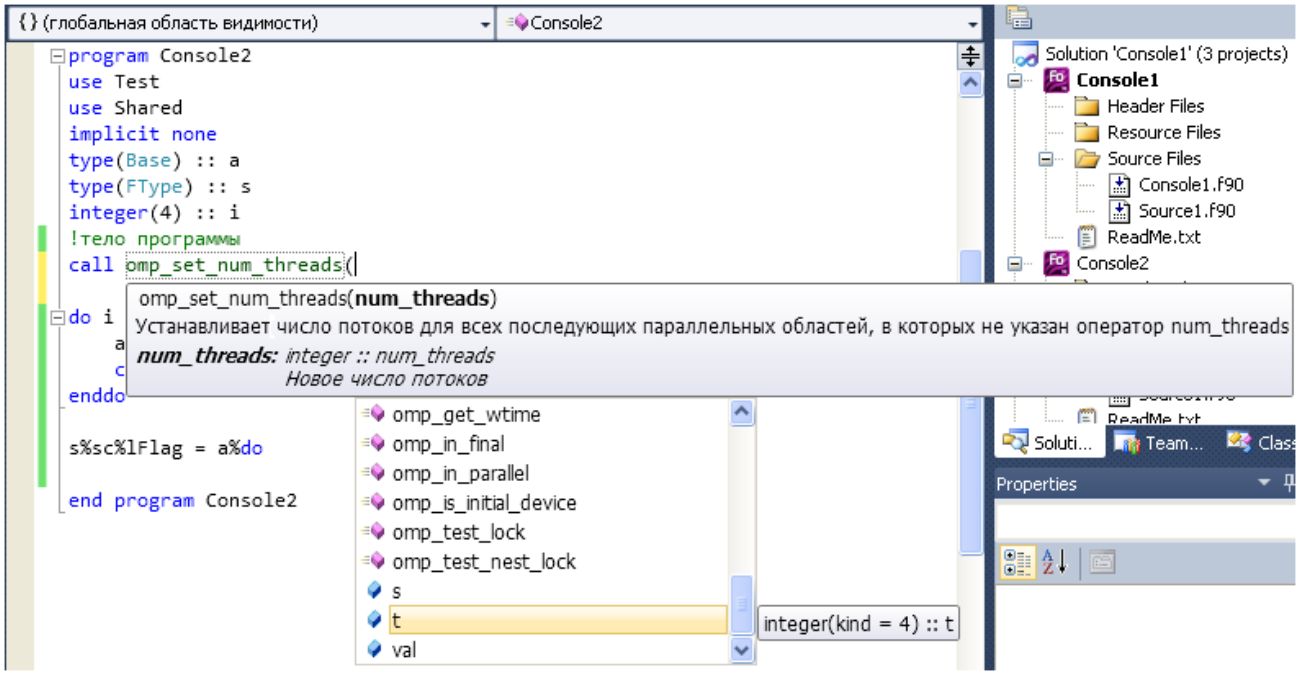

Рисунок 3.4 – Поддержка OpenMP 4.0 в части процедур (выделение цветом, описание аргументов процедуры и их автодополнение с учётом процедур OpenMP)

Вторая часть работы с OpenMP состоит в использовании специальных директив компилятора. Во FRIS были определены окружающие сниппеты для всех конструкций стандарта OpenMP 4.0.

Например, для распараллеливания цикла do сначала необходимо выделить всё тело цикла. Далее нажать правую кнопку мыши и в появившемся контекстном меню выбрать пункт «Surround With…» (рисунок 3.5). В появившемся списке найти нужный сниппет, например «parallel do» и нажать <Enter> (рисунок 3.6). После этого исходный цикл do будет помещён внутрь тела только что созданного сниппета (рисунок 3.7) и, соответственно, будет распараллелен на общей памяти.

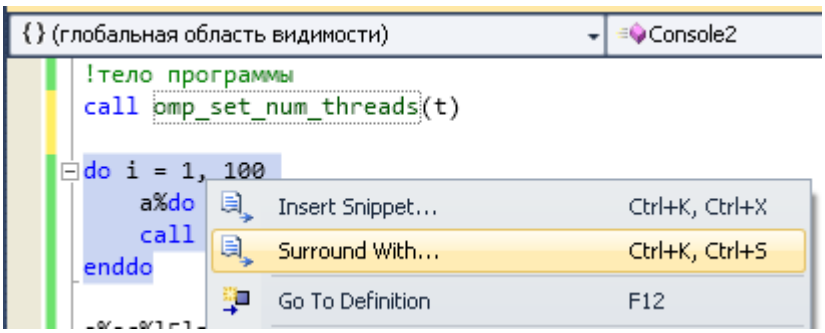

Рисунок 3.5 – Выделение тела цикла и выбор пункта меню «Окружающий сниппет»

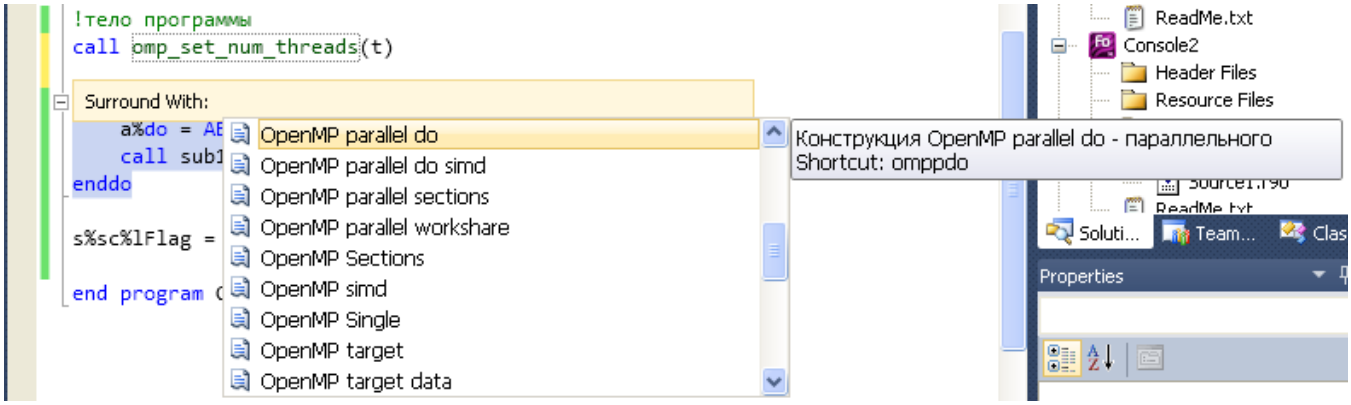

Рисунок 3.6 – Сниппеты для директив OpenMP во FRIS (выбор сниппета для конструкции OpenMP parallel do)

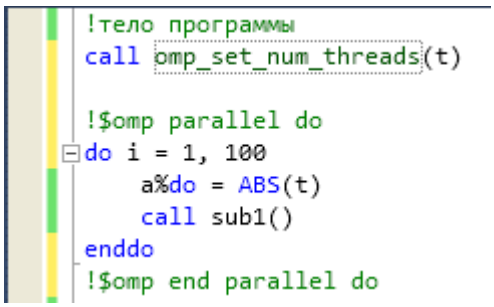

Рисунок 3.7 – Результат применения сниппета. Цикл do распараллелен на общей памяти

3.1.3 Поддержка SIMD-операций векторизации вычислений

Напомним, что под SIMD-операцией понимается выполнение одного вычислительного действия (сложение, вычитание, умножение, сдвоенная операция умножение-сложение) на векторных регистрах одновременно над массивом чисел фиксированной длины.

Хотя компилятор может сам расставлять инструкции по векторизации вычислений, в большинстве алгоритмов ему необходимо явным образом указывать, как вектори-

зовать счётные циклы. Для этого, в частности у компилятора Fortran фирмы Intel [34], применяются специализированные директивы компилятора, а для работы с предварительной загрузкой данных в кэшпамять рекомендуется применять специализированную подпрограмму mm_prefetch(). Наиболее часто используемые директивы компилятора для векторизации и запрета векторизации, а также описание подпрограммы mm_prefetch() и ускорители FRIS, используемые для них, приведены в таблице 3.3.

Таблица 3.3. Элементы, используемые для управления векторизацией вычислений, и ускорители (shortcuts) FRIS для их использования

Элемент	Ускоритель FRIS	Описание
!DIR$ IVDEP	**ivdep**	указывает компилятору, что все зависимости по данным в цикле происходят последовательно в их лексическом порядке. Позволяет компилятору применять расширенную эвристику оптимизации выполнения инструкций цикла
!DIR$ VECTOR ALWAYS	**vec**	указывает компилятору, что следующий за директивой цикл необходимо векторизовать
!DIR$ NOVECTOR	**novec**	указывает компилятору, что следующий за директивой цикл не нужно векторизовать
!DIR$ NOPREFETCH	**nopref**	указывает компилятору, что для следующего за директивой цикла не нужно генерировать автоматические вызовы подпрограмм загрузки данных, обрабатываемых в цикле, в кэш процессора
CALL MM_PREFETCH (address [,hint])	**-**	подпрограмма загружает данные из указанного адреса (address) оперативной памяти в кэш процессора. Аргумент hint (целочисленная константа) содержит указание, как планируется использовать загружаемые данные:
	pref0	0 (по умолчанию) – загружает данные в кэш 1-го (а так же 2-го и 3-го) уровня. Рекомендуется использовать это значение для целочисленных данных;
	pref1	1 – загружает данные в кэш 2-го (а так же 3-го) уровня. Вещественные данные используются из кэш 2-го уровня, а не из кэш 1-го уровня. Рекомендуется использовать это значение для вещественных данных;
	pref2	2 – загружает данные в кэш 2-го (а так же 3-го) уровня. Линия кэша в которую загружаются данные помечается как «для раннего вытеснения». Рекомендуется использовать это значение для данных, которые не планируется использовать часто;
	pref3	3 – загружает данные в кэш 2-го (но не 3-го) уровня. Линия кэша в которую загружаются данные помечается как «для раннего вытеснения». Рекомендуется использовать это значение для данных, которые не планируется использовать повторно

Поскольку директивы компилятора в общем виде являются совершенно одинаковыми и должны присутствовать перед каждым векторизуемым (или не векторизуемым) циклом, то они были оформлены в виде сниппетов. Приведём оценку ускорения программирования за счёт использования сниппетов из таблицы 3.3 (таблица 3.4).

Таблица 3.4. Оценка ускорения программирования за счёт сниппетов SIMD-операций

Элемент	Ускоритель FRIS	Ускорение от использования
!DIR$ IVDEP	**ivdep**	11/6 = 1.8
!DIR$ VECTOR ALWAYS	**vec**	19/4 = 4.75
!DIR$ NOVECTOR	**novec**	14/6 = 2.33
!DIR$ NOPREFETCH	**nopref**	16/7 = 2.28
CALL MM_PREFETCH (address [,hint])	**prefX**	19/6 = 3.16
В среднем на сниппет		2.86

Пример результата использования данных сниппетов при реализации блока расчёта констант среды библиотеки MAGIK приведён на рисунке 3.8.

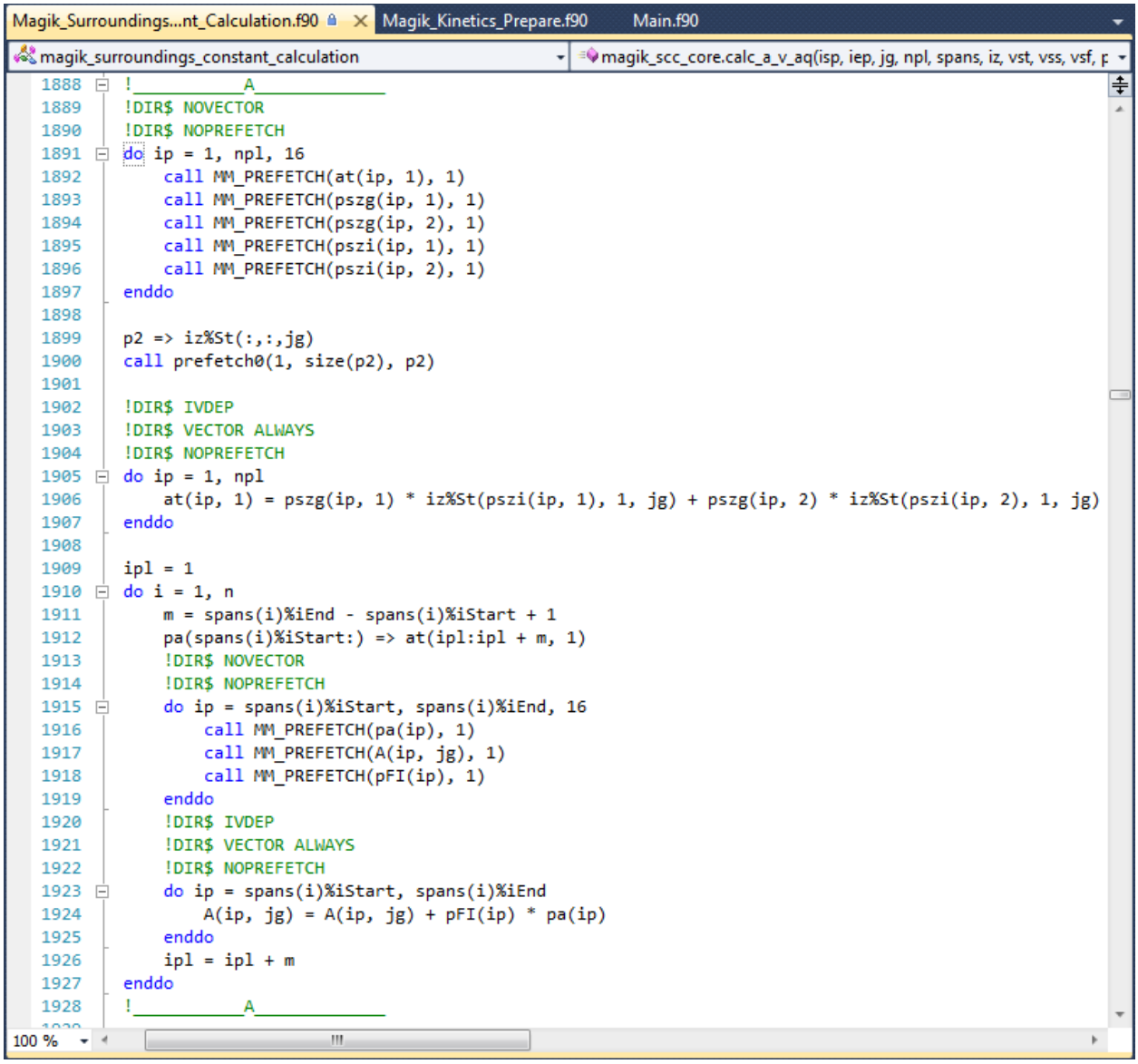

Рисунок 3.8 – Фрагмент программы расчёта макроконстант среды, полученный с использованием сниппетов FRIS для работы с SIMD-операциями

Использование сниппетов FRIS для SIMD-операций позволяет повысить производительность труда в этой части в среднем в 2.8 раза.

3.2 Встроенная поддержка работы с библиотеками РФЯЦ-ВНИИЭФ во FRIS

Одними из наиболее часто используемых в производственных программах и методиках РФЯЦ-ВНИИЭФ библиотек являются библиотеки УРС-ОФ и ЕФР. Во FRIS обеспечивается расширенная поддержка работы с ними.

3.2.1 Поддержка библиотеки УРС-ОФ

Основной целью библиотеки УРС-ОФ является расчёт характеристик уравнений состояния при моделировании различных изделий. Для подключения библиотеки служит 1 файл **urs_type.f90,** содержащий модуль **urs_of** с описанием типов данных библиотеки и интерфейсов библиотеки, которые были задокументированы с использованием комментариев документирования FRIS.

Во FRIS был реализован расширенный механизм подсветки синтаксиса для библиотеки УРС-ОФ. Для сравнения на рисунке 3.9 приведён файл исходного кода до и после реализации во FRIS подсветки синтаксиса для УРС-ОФ (визуально выделяются имена типов данных и имена процедур, при этом цвета могут быть настроены пользователем по своему усмотрению).

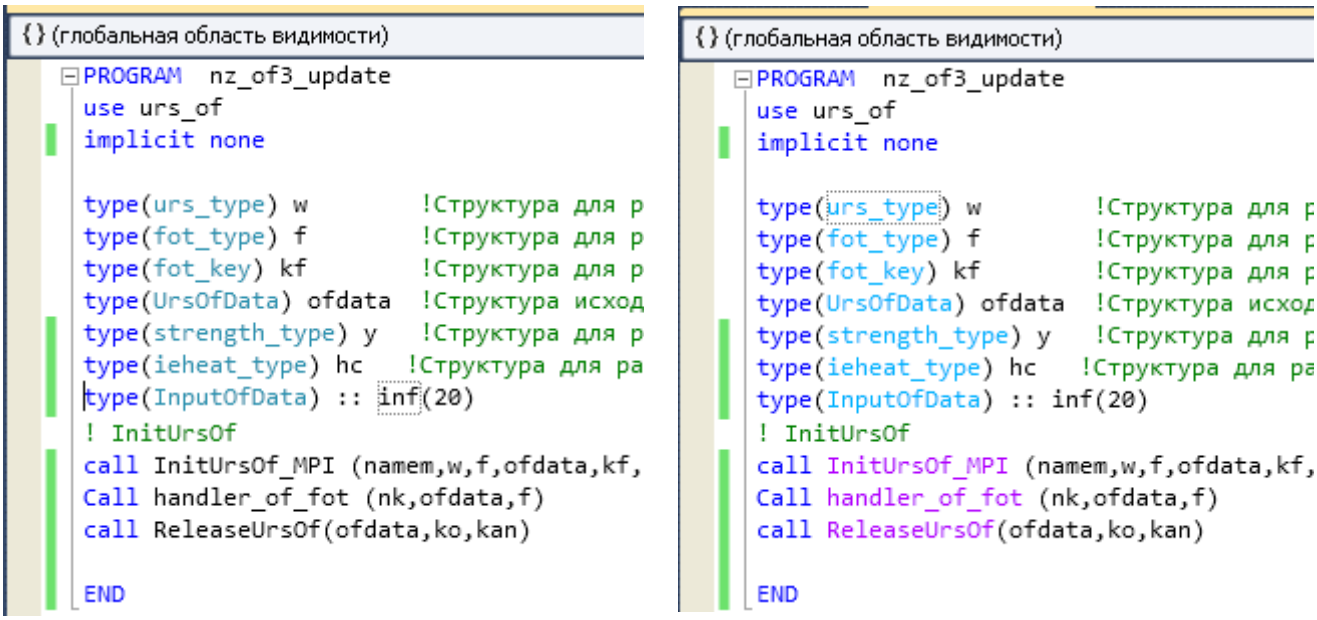

Рисунок 3.9 – Файл исходного кода до реализации подстветки синтаксиса для библиотеки УРС-ОФ (слева) и после реализации (справа)

Пример использования возможности построения списка членов типа данных **UrsOfData** приведён на рисунке 3.10.

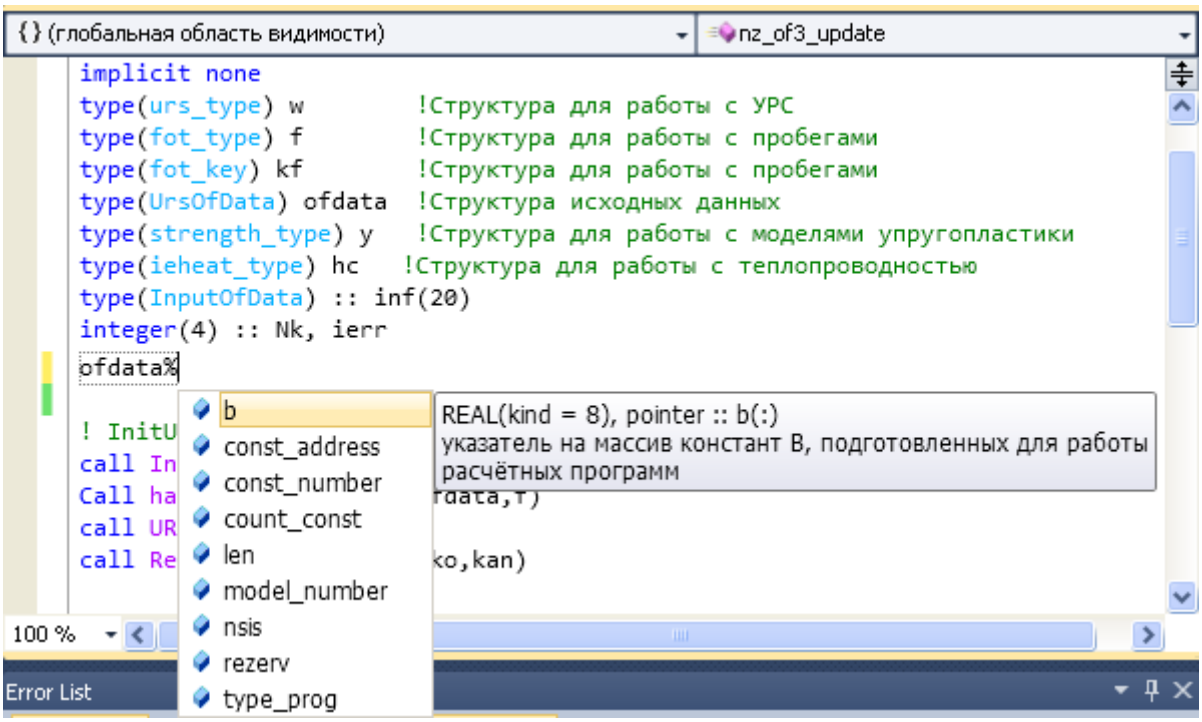

Рисунок 3.10 – Список членов типа данных **UrsOfData**

3.2.2 Поддержка библиотеки ЕФР

Основной целью библиотеки ЕФР является предоставление производственным программам и методикам РФЯЦ-ВНИИЭФ возможности чтения и записи своих данных в файле специального формата, для которого гарантируется работа в единой манере в различных операционных системах и на различных программных архитектурах.

Для подключения библиотеки служит 1 файл **efr.f90,** содержащий модуль **EFR**, в котором приведено описание констант и интерфейсов для подпрограмм и функций, предоставляемых библиотекой. Для обеспечения расширенной поддержки FRIS все значимые элементы библиотеки ЕФР были задокументированы.

Приведём примеры работы различных возможностей FRIS для библиотеки ЕФР: на рисунке 3.11 - автодополнение функций библиотеки ЕФР; на рисунке 3.12 - отображение аргументов функции по мере ввода; на рисунке 3.13 - работа с перегруженной (с использованием обобщённого интерфейса) подпрограммой.

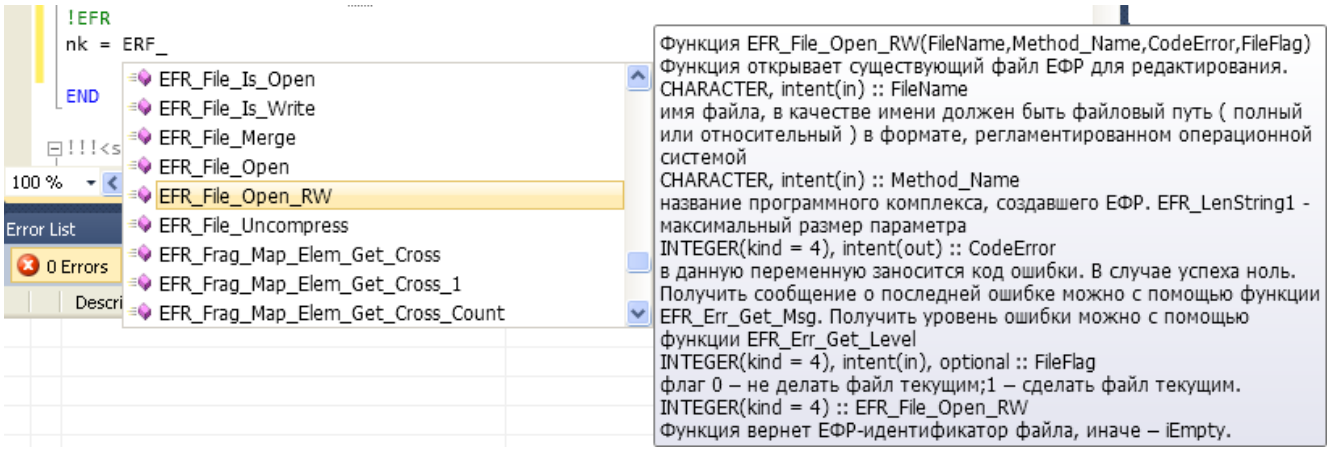

Рисунок 3.11 – Автодополнение имени функции ЕФР

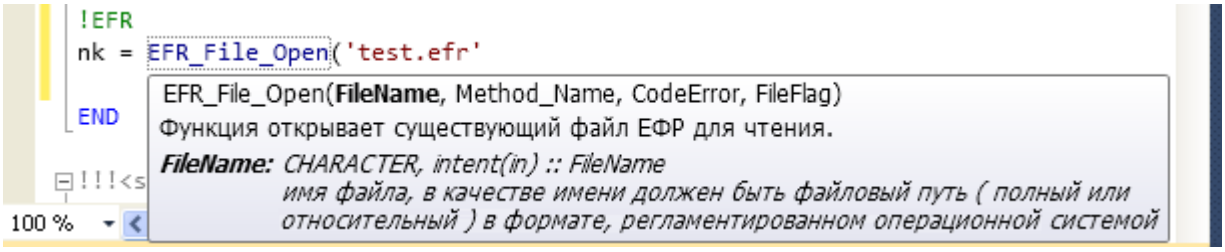

Рисунок 3.12 – Сведения об аргументах функции ЕФР по мере их ввода

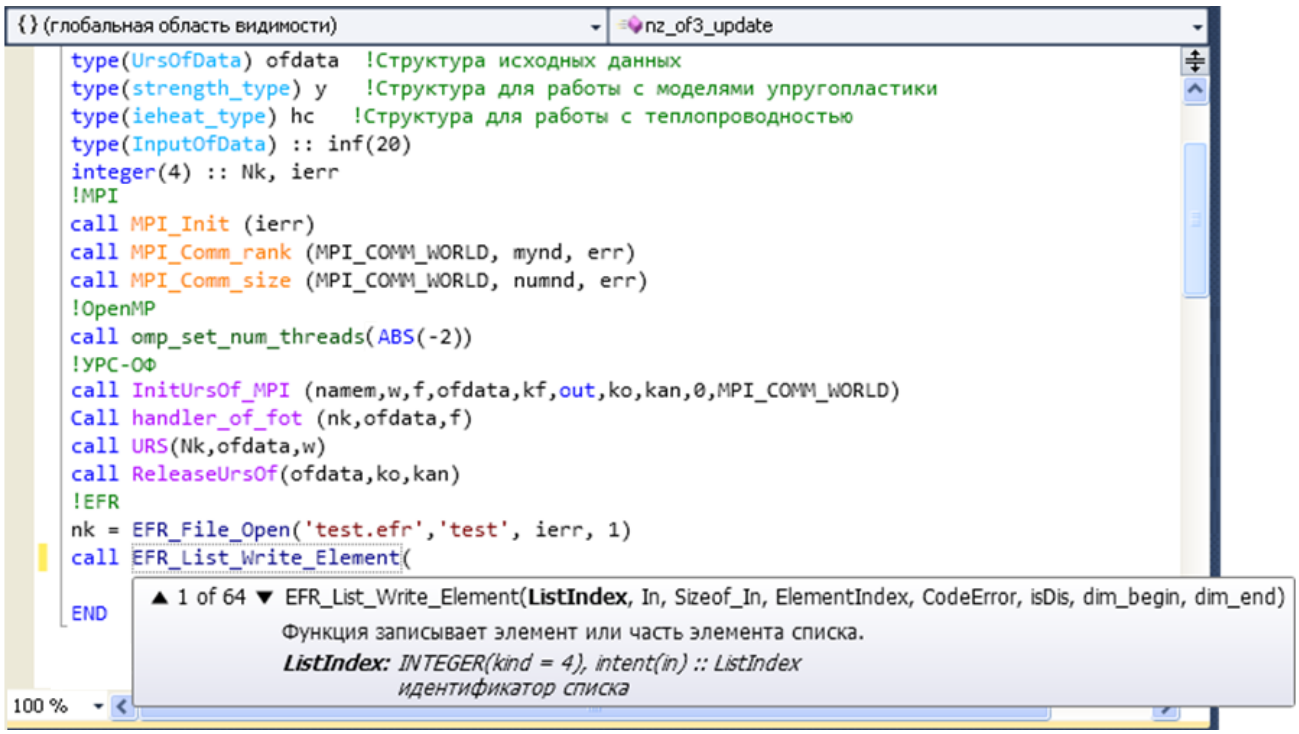

Рисунок 3.13 – Работа с перегруженными подпрограммами ЕФР

Таким образом, использование FRIS сокращает время написания текстов программ за счёт возможностей автодополнения, предоставления контекстной помощи и использования сниппетов, улучшает командную разработку проектов и работу с биб-

лиотеками РФЯЦ-ВНИИЭФ. Использование FRIS позволяет устранить необходимость в предварительном поиске нужных программных элементов, запоминании их устройства и т.п. Теперь всю необходимую информацию можно получить непосредственно в процессе написания инструкций и операторов языка программирования.

3.3 Применение FRIS в РФЯЦ-ВНИИЭФ

3.3.1 Применение FRIS при программировании библиотеки MAGIK

Отметим, что апробирование языкового сервиса FRIS началось в библиотеке MAGIK [46]. Библиотека MAGIK отвечает за расчёт макроскопических ядерно-физических констант среды, а также нейтронно-ядерной и термоядерной кинетики изотопов. Разработка библиотеки ведётся с использованием стандарта Fortran 2003 и технологии объектно-ориентированного программирования. Во FRIS обеспечена поддержка указанных возможностей.

Одной из задач при разработке библиотеки MAGIK являлось создание и использование динамически расширяющихся контейнеров, работающих по принципу двусвязных списков, для хранения данных об изотопах. При реализации данной задачи активно использовались возможности FRIS по работе со сниппетами.

Ключевой задачей при работе с динамически расширяющимися контейнерами является обеспечение потокобезопасного доступа при одновременной работе с контейнером из разных OpenMP потоков. Здесь так же использовались возможности FRIS по поддержке работы с OpenMP (описанные в пункте 3.1.2). В качестве наиболее простого механизма предотвращения одновременного доступа потоков к критически важным данным был использован механизм критических секций стандарта OpenMP.

Созданный таким образом контейнер для двусвязного списка был отлажен, преобразован в шаблон (сниппет) FRIS и использован для создания контейнеров изотопов подсистемы хранения и рабочих изотопов задачи (рисунок 3.14).

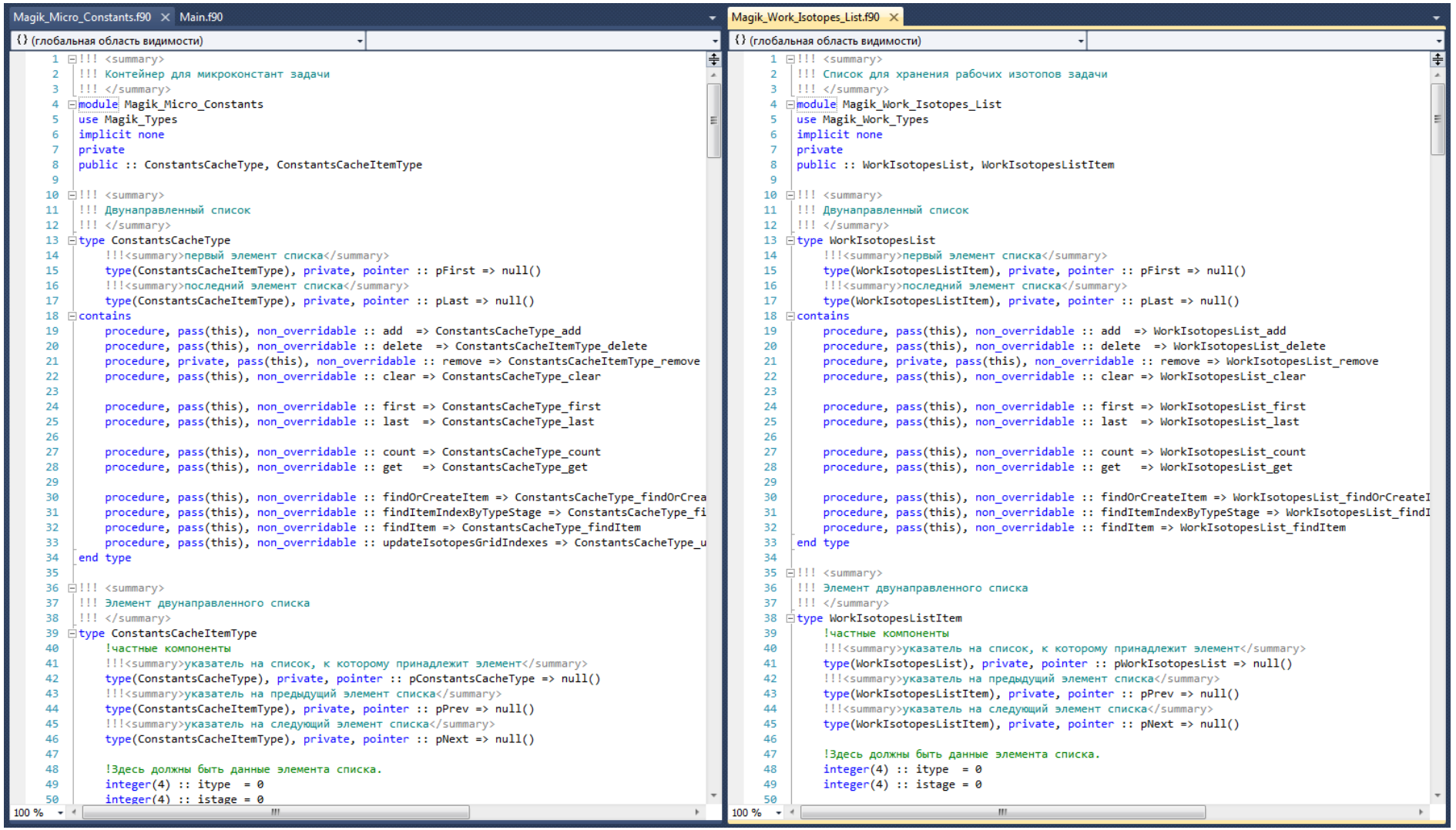

Рисунок 3.14 – Контейнеры для хранения изотопов, созданные с помощью шаблона (сниппета) двусвязного списка. Слева – для подсистемы хранения, справа — для рабочего экземпляра задачи

Отметим, что в данном случае в качестве шаблона выступает целый модуль языка программирования Fortran. Таким образом, при помощи сниппетов можно решить задачу создания шаблонизируемых элементов наподобие тех, которые содержатся, например, в стандартной шаблонной библиотеке (STL) языка C++, с тем отличием, что шаблоны в C++ «раскрывает» препроцессор, а в Fortran они настраиваются пользователем.

Второй задачей, которая была решена в MAGIK, стала адаптация основных счётных блоков к использованию трёх уровневой схемы распараллеливания вычислений, в частности к использованию SIMD-операций для векторизации вычислений (рисунок 3.8). При её решении также использовались все реализованные во FRIS возможности (см. пункт 3.1).

Дадим ориентировочную оценку ускорения (снижения трудозатрат) программирования библиотеки MAGIK за счёт использования возможностей FRIS (таблицы 3.5-3.6) [81].

Таблица 3.5. Оценка ускорения программирования библиотеки MAGIK за счёт использования сниппетов FRIS

Возможность	Конструкции и число их вхождений	Ускоритель FRIS	Ускорение
Использование директив OpenMP	parallel – 30 parallel do – 7 do – 8 critical – 6 Всего: 51 конструкция	omppar omppdo ompdo ompcrit	32 / 7 = 4.57 38 / 7 = 5.42 20 / 6 = 3.33 32 / 8 = 4 В среднем: 4.42
Использование SIMD-операций	ivdep – 261 noprefetch – 309 novector – 170 vector always – 242 mm_prefetch – 591 Всего 1501	ivdep nopref novec vec prefX	1.83 2.28 2.33 4.75 3.16 В среднем: 2.91
Использование двусвязного списка	использований – 3; объём кода сниппета – 270 строк – 8084 символа; настраиваемых частей 5: имя модуля, имя типа списка, имя типа элемента списка, имя модуля с типом данных хранимых в элементе списка, описание элемента данных, хранимых в списке	llist	8084 / 6 = 1347.33

Таблица 3.6. Оценка ускорения программирования библиотеки MAGIK за счёт использования возможностей IntelliSense во FRIS

Возможность	Описание	Характеристики	Ускорение
Построение списка элементов сложного объекта	автодополнение компонентов производных типов данных	средняя длинна имени компонента производного типа данных: 6 среднее число символов, позволяющее однозначно идентифицировать имя: 3	6 / 3 = 2
Автодополнение	завершение имени модуля	средняя длина имени: 20 среднее число символов для однозначной идентификации имени: 7	20 / 7 = 2.85
	завершение имени производного типа данных	средняя длина имени: 15 среднее число символов для однозначной идентификации имени: 7	15 / 7 = 2.14
	завершение имени переменной	средняя длина имени: 5 среднее число символов для однозначной идентификации имени: 3	5 / 3 = 1.66
	завершение имени процедуры	средняя длина имени: 15 среднее число символов для однозначной идентификации имени: 8	15 / 8 = 1.87

Для расчёта ускорения используется формула [81]:

$$S = \frac{N_{full}}{N_{entered}} \qquad (1)$$

где S - ускорение от использования возможности; N_{full} - полное число символов в результирующем тексте программы; $N_{entered}$ - количество введённых пользователем.

Для оценки усреднённого ускорения от использования FRIS выбран фрагмент библиотеки MAGIK, рассчитывающий макросечение с учётом температурной зависимости и зависимости микроконстант от резонансной структуры сечений (таблица 3.7, Приложение 3).

Как следует из таблицы 3.7, ускорение программирования за счёт использования FRIS в характерном счётном алгоритме библиотеки MAGIK составляет 1.458 раза (всего 1467 символов, из них набрано 1006 символов). При этом ускорение программирования отдельных частей может достигать для возможностей технологии IntelliSense до 2.85 раза, а для сниппетов вплоть до 1300 раз. Ускоряет процесс программирования также контекстно-зависимая справка FRIS об используемых элементах (типах данных, переменных, процедурах) в указанной курсором точке программы, что позволяет программисту (особенно не автору программы) получить сведения для корректной модификации кода, не отвлекаясь на поиск определений элементов.

Таблица 3.7. Оценка ускорения от использования FRIS на характерном коде MAGIK

№	Текст программы	Возможность FRIS	Символы			Ускорение
			Введённые	Дополнены FRIS	Длина строки	
1	**!DIR$ NOVECTOR**	*сн.:* novec	7	8	15	2.143
2	**!DIR$ NOPREFETCH**	*сн.:* nopref	8	9	17	2.125
3	do ip = 1, npl, 16	*сн.:* do	14	5	19	1.357
4	call MM_PREFETCH(at(ip, 1), 1)	*сн.:* pref1	17	16	33	1.941
5	call MM_PREFETCH(pszg(ip, 1), 1)	*сн.:* pref1, *доп.:* pszg	19	16	35	1.842
6	call MM_PREFETCH(pszg(ip, 2), 1)	*сн.:* pref1, *доп.:* pszg	19	16	35	1.842
7	call MM_PREFETCH(pszi(ip, 1), 1)	*сн.:* pref1, *доп.:* pszi	19	16	35	1.842
8	call MM_PREFETCH(pszi(ip, 2), 1)	*сн.:* pref1, *доп.:* pszi	19	16	35	1.842
9	call MM_PREFETCH(psigi(ip, 1), 1)	*сн.:* pref1, *доп.:* psigi	19	17	36	1.895
10	call MM_PREFETCH(psigi(ip, 2), 1)	*сн.:* pref1, *доп.:* psigi	19	17	36	1.895
11	enddo	*кц. do на стр.* 3	0	6	6	
12	p2 => iz%St(:,:,jg)	*доп.:* St, jg	20	0	20	1
13	call prefetch0_r4(1, size(p2), p2)	*доп.:* prefetch0_r4	26	9	35	1.346
14	**!DIR$ IVDEP**	*сн.:* ivdep	7	5	12	1.714
15	**!DIR$ VECTOR ALWAYS**	*сн.:* vec	5	15	20	4
16	**!DIR$ NOPREFETCH**	*сн.:* nopref	8	9	17	2.125
17	do ip = 1, npl	*сн.:* do	11	4	15	1.364
18	at(ip, 1) = pszg(ip, 1) * iz%St(pszi(ip, 1), psigi(ip, 1), jg) + pszg(ip, 2) * iz%St(pszi(ip, 2), psigi(ip, 1), jg)	*доп.:* pszg, St, pszi, psigi	116	2	118	1.017
19	enddo	*кц. do на стр.* 17	0	6	6	
20	**!DIR$ NOVECTOR**	*сн.:* novec	7	8	15	2.143
21	**!DIR$ NOPREFETCH**	*сн.:* nopref	8	9	17	2.125
22	do ip = 1, npl, 16	*сн.:* do, *доп.:* npl	14	5	19	1.357
23	call MM_PREFETCH(at(ip, 2), 1)	*сн.:* pref1	17	16	33	1.941
24	enddo	*кц. do на стр.* 22	0	6	6	
25	**!DIR$ IVDEP**	*сн.:* ivdep	7	5	12	1.714

Таблица 3.7. Продолжение

26	**!DIR$ VECTOR ALWAYS**	*сн.:* vec	5	15	20	4
27	**!DIR$ NOPREFETCH**	*сн.:* nopref	8	9	17	2.125
28	do ip = 1, npl	*сн.:* do	11	4	15	1.364
29	at(ip, 2) = pszg(ip, 1) * iz%St(pszi(ip, 1), psigi(ip, 2), jg) + pszg(ip, 2) * iz%St(pszi(ip, 2), psigi(ip, 2), jg)	*доп.:* pszg, St, pszi, psigi	116	2	118	1.017
30	enddo	*кц. do на стр.* 28	0	6	6	
31	**!DIR$ NOVECTOR**	*сн.:* novec	7	8	15	2.143
32	**!DIR$ NOPREFETCH**	*сн.:* nopref	8	9	17	2.125
33	do ip = 1, npl, 16	*сн.:* do, *доп.:* npl	14	5	19	1.357
34	call MM_PREFETCH(at(ip, 1), 1)	*сн.:* pref1	17	16	33	1.941
35	call MM_PREFETCH(at(ip, 2), 1)	*сн.:* pref1	17	16	33	1.941
36	call MM_PREFETCH(psigg(ip, 1), 1)	*сн.:* pref1, *доп.:* psigg	19	17	36	1.895
37	call MM_PREFETCH(psigg(ip, 2), 1)	*сн.:* pref1, *доп.:* psigg	19	17	36	1.895
38	enddo	*кц. do на стр.* 33	0	6	6	
39	**!DIR$ IVDEP**	*сн.:* ivdep	7	5	12	1.714
40	**!DIR$ VECTOR ALWAYS**	*сн.:* vec	5	15	20	4
41	**!DIR$ NOPREFETCH**	*сн.:* nopref	8	9	17	2.125
42	do ip = 1, npl	*сн.:* do	11	4	15	1.364
43	at(ip,1) = psigg(ip,1) * at(ip,1) + psigg(ip,2) * at(ip,2)	*доп.:* psigg	59	2	61	1.034
44	enddo	*кц. do на стр.* 42	0	6	6	
45	ipl = 1	*доп.:* ipl	8	0	8	1
46	do i = sp(1)%iSpan, sp(2)%iSpan	*сн.:* do, выбор iSpan	24	8	32	1.333
47	isp = max(spans(i)%iStart, sp(1)%igp)	*доп.:* spans, iStrat, igp	32	8	40	1.25
48	iep = min(spans(i)%iEnd, sp(2)%igp)	*доп.:* spans, iEnd, igp	31	7	38	1.226
49	m = iep - isp + 1		20	0	20	1
50	if(m < 1) cycle		18	0	18	1
51	pa(isp:) => at(ipl:ipl + m, 1)	*доп.:* isp,ipl	33	0	33	1
52	ptr(isp:iep) => A(isp:iep, jg)	*доп.:* isp,iep	33	0	33	1
53	**!DIR$ NOINLINE**		17	0	17	1

Таблица 3.7. Продолжение

54	call vector_kernel_sum_mult_2v(isp, iep, ptr, pfi, pa)	*доп.:* vector_...	37	20	57	1.541
55	ipl = ipl + m		16	0	16	1
56	enddo	*кц. do на стр.* 46	0	6	6	
		Итого:	**1006**	**461**	**1467**	**1.458**

Примечания к таблице 3.7.

В таблице использованы следующие обозначения: «*сн.:*» - использование сниппета с указанным именем; «*доп.:*» - использование возможности «Автодополнение» технологии IntelliSense; «*кц. do на стр.* XX» - конец цикла do, вставленного в результате использования сниппета для цикла do на строе с номером «XX».

Для написания циклов использовался сниппет FRIS с ускорителем do, который имеет следующий шаблон (Листинг 3.1).

```
do i = 1, npl
  |
enddo
```

Листинг 3.1 – Сниппет для цикла do

В листинге 3.1 курсивом отмечены настраиваемые части сниппета, а вертикальной чертой – место, куда будет переведён курсор после завершения редактирования сниппета. Таким образом, при использовании сниппета do вместо него будет вставлено 3 строки. Этим обусловлено то, что в строках 11, 19, 24, 30, 38, 44, 56 отсутствуют символы, вводимые пользователем.

Приведём оценку времени работы и объёма оперативной памяти, требуемых FRIS для анализа исходных кодов MAGIK. Получение оценок проводилось на персональной ЭВМ с центральным процессором Intel Core i5 2.8 ГГц и объёмом оперативной памяти 8 Гбайт. Исходные коды библиотеки MAGIK составляют 76000 строк в 79 файлах. Их анализ был выполнен FRIS за 3.21 секунды, при этом объём памяти для хранения значимых элементов составил 165 Мбайт.

3.3.2 Применение FRIS в других программах

Апробация FRIS с участием автора выполнялась в программе КПД-1D, в состав которой входит порядка 15 счётных методик различной направленности (рисунок 3.15). Программа обладает обширной и сложной структурой данных, в особенности для обеспечения совместной работы нескольких методик в рамках расчёта одной задачи.

Необходимо отметить, что разработкой каждой отдельно взятой методики занимается коллектив разработчиков, не участвующий в разработке других методик. Поэтому особенно актуальным вопросом является интеграция множества методик в одной программе и обеспечение их взаимодействия для согласованной работы в процессе счёта.

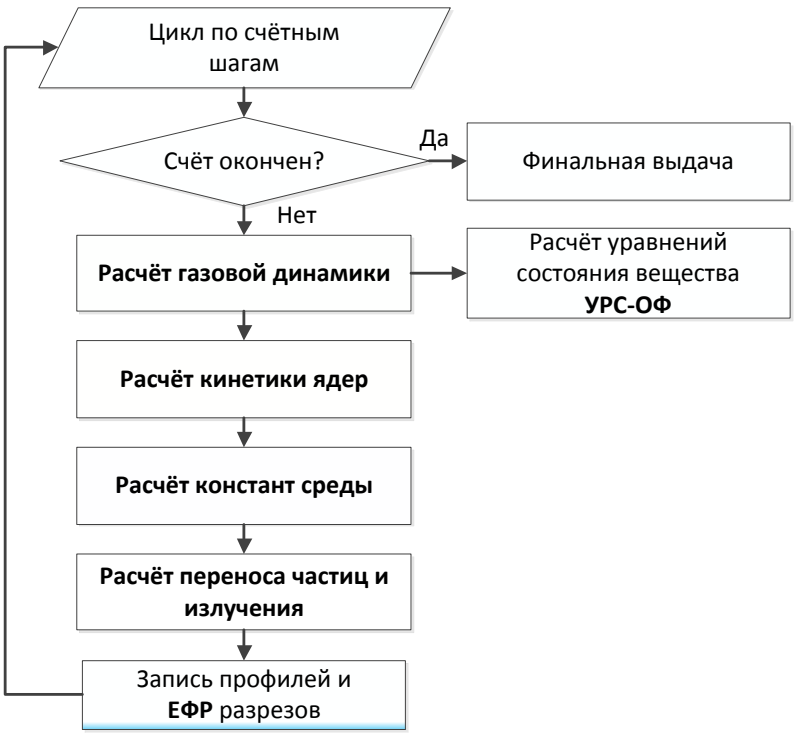

Рисунок 3.15 – Общая схема организации программы КПД-1D

Так, использование автодополнения имён переменных существенно ускоряет написание исходного кода, т.к. нужный элемент просто выбирается из списка доступных элементов (рисунок 3.16). Преимущество такого подхода заключается также и в том, что список конкретизируется по мере ввода имени нужного элемента.

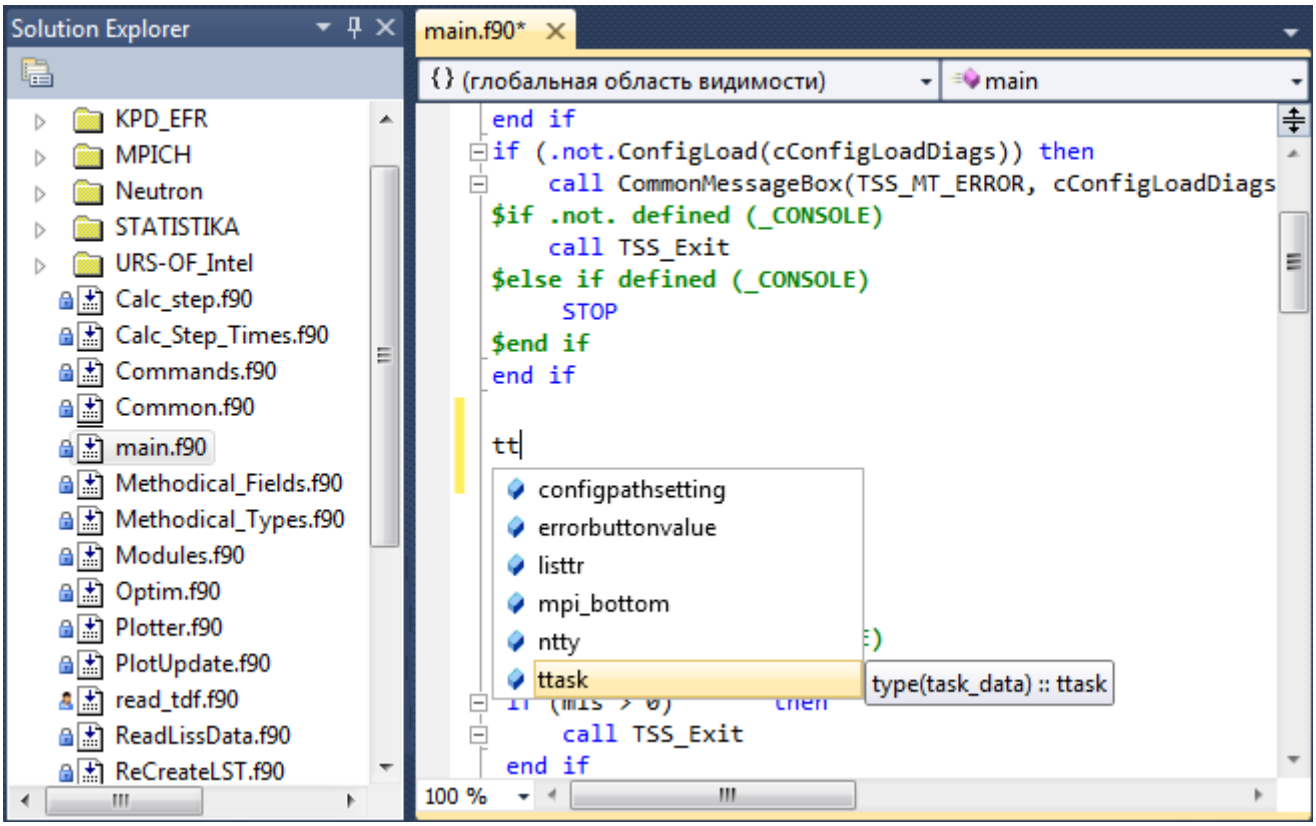

Рисунок 3.16 – Использование автодополнения при работе с главной программой

Функция построения списка членов производного типа данных позволяет работать с нужными компонентами, не обращаясь к определению типа (рисунок 3.17). При наведении курсора на нужный элемент выводится контекстная справка, содержащая основную информацию о нём. При необходимости можно воспользоваться функцией перехода к определению и рассмотреть интересующий элемент подробнее.

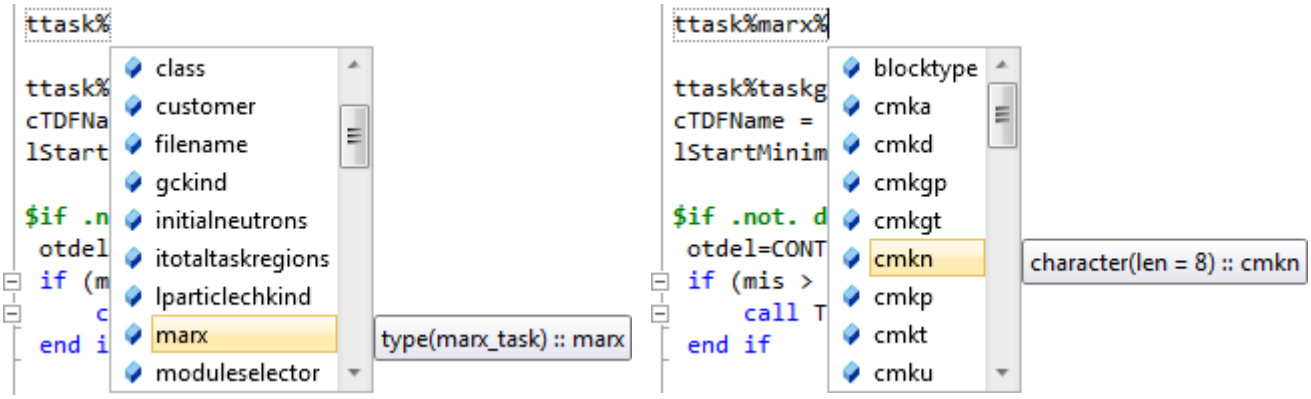

Рисунок 3.17 – Работа с компонентами производных типов данных

Работе с исходным кодом способствуют и функции свёртки различных структурных элементов, начиная от модулей и заканчивая условными операторами и циклами (рисунок 3.18).

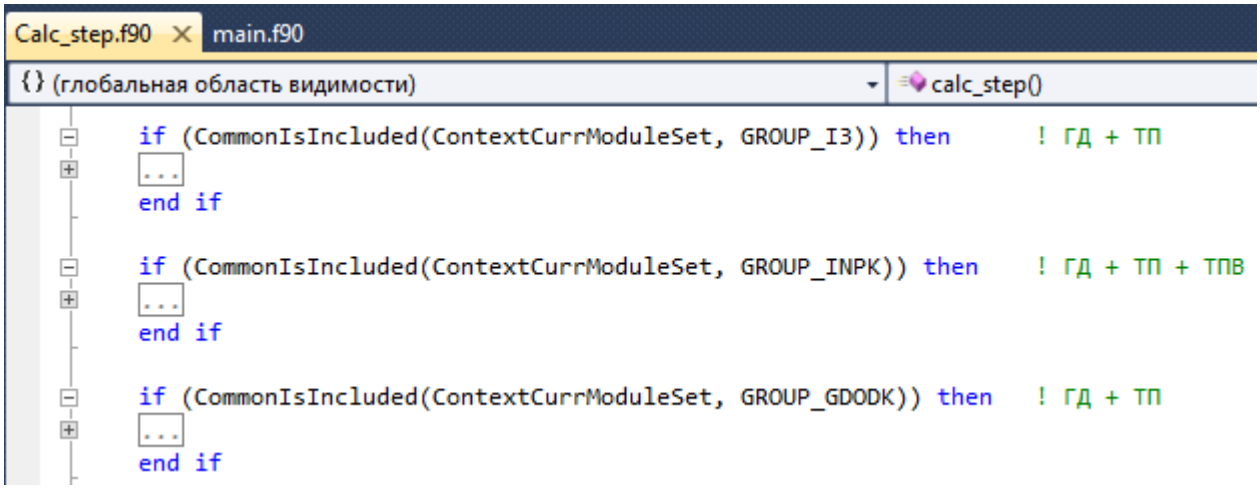

Рисунок 3.18 – Скрытие части операторов для лучшего анализа кода

Использование функций автодополнения и контекстных подсказок (рисунки 3.19-3.20) упростило программирование.

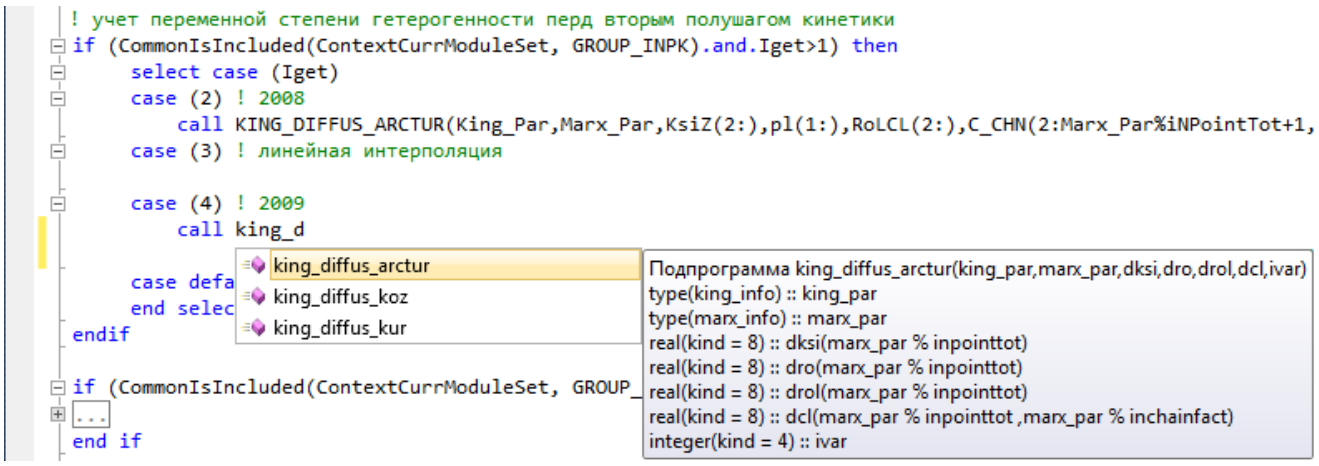

Рисунок 3.19 – Автодополнение при выборе нужной подпрограммы

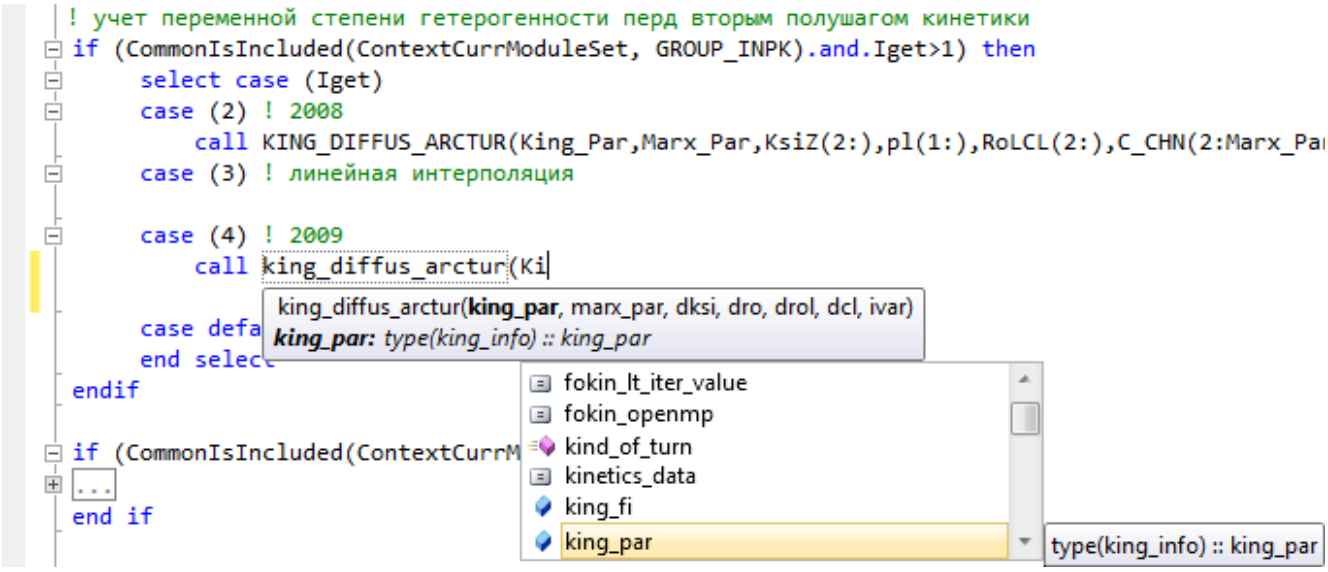

Рисунок 3.20 – Помощь по аргументам подпрограммы и автодополнение аргумента

Отметим, что подобное использование возможностей FRIS характерно также и для остальных программ [81] (Приложение 3). Перечислим наиболее востребованные возможности:

- автодополнение и построение списка элементов сложного объекта, что позволяет ускорить написание текста целевого программного продукта, а так же повышает его качество за счёт снижения вероятности допустить ошибку при программировании;
- получение расширенной контекстной помощи об элементах языка Fortran с учётом комментариев документирования;
- создание и использование сниппетов – готовых и отлаженных шаблонов программного кода (без необходимости повторного программирования) с модификацией для подстройки под конкретные нужды;

- документирование программы при помощи комментариев документирования, позволяющее получать расширенную контекстную помощь, особенно актуальную при командной работе над одним проектом связанных параллельных программ;
- переход к определению элемента языка программирования, что особенно полезно, например, при анализе состава элемента или модификации данного элемента;
- поддержка написания программ для высокопроизводительных вычислений с использованием MPI, OpenMP и SIMD-операций.

3.4 Сравнение языкового сервиса FRIS с аналогами

Проведём сравнение (таблица 3.9) языкового сервиса FRIS и его коммерческих [34-35,41] и свободно распространяемых аналогов [42-45]. Вначале представим их сводное описание (таблица 3.8).

Таблица 3.8. Сводные характеристики языковых сервисов для Fortran

Характеристика	**Visual Studio**				**Кроссплатформенные**		
	Intel	**Lahey**	**PGI**	**Fortran CodeNav**	**Eclipse Phortran**	**CodeBlocks Fortran**	**NetBeans Fortran**
Коммерческий	да	да	да	нет	нет	нет	нет
Доступен в исходных кодах	нет	нет	нет	нет	да	да	нет
На каком языке реализован (если известно)	неизвестно, предположительно:				Java	C++	Java
	C#	C#	C++	C#			

Как видно из таблицы 3.9, FRIS является единственным продуктом, который дает возможность работать с внешними программными библиотеками, предоставляет для этого соответствующий инструментарий, а также характеризуется бо̀льшим набором функциональных возможностей и их более полной реализацией.

Phortran [43] в Eclipse является единственным сервисом, поддерживающим возможности рефакторинга[10] программ, и в этой части имеет преимущество над FRIS. Однако он разрабатывался специально для этой цели [43].

[10] Рефакторинг – переписывание исходных текстов с целью улучшения внутренней структуры программного кода при сохранении его интерфейсов и функциональности.

Таблица 3.9. Сравнение языкового сервиса FRIS и зарубежных аналогов

Возможность	Visual Studio					Кроссплатформенные		
	FRIS	**Intel**	**Lahey**	**PGI**	**Fortran CodeNav**	**Eclipse Phortran**	**CodeBlocks Fortran**	**NetBeans Fortran**
Подсветка синтаксиса	есть + внешние библиотеки	есть	есть	есть	нет	есть	есть	есть
Построение списка элементов сложного объекта	есть	нет	нет	нет	нет	нет	есть	нет
Отображение сведений о параметрах процедур	есть	есть, кроме перегруженных процедур и процедур, связанных с типом данных	есть, кроме перегруженных процедур и процедур, связанных с типом данных	есть, только для встроенных процедур	есть, кроме перегруженных процедур и процедур, связанных с типом данных	нет	есть	нет
Отображение кратких сведений об элементе языка программирования	есть	есть, за исключением полей и процедур производных типов данных	есть, за исключением полей и процедур производных типов данных	есть, только для встроенных процедур	есть, за исключением полей и процедур производных типов данных	есть, за исключением полей и процедур производных типов данных	есть	нет
Автодополнение	есть	есть, только для имён модулей, подпрограмм и функций	есть	есть, только для ключевых слов	есть	есть, только для имён переменных, подпрограмм и функций	есть	нет
Поддержка комментариев документирования	есть, XML-комментарии	нет	нет	нет	нет	нет	есть, Doxygen	нет
Поддержка сниппетов кода	есть + автодополнение имён сниппетов	есть, только как команда меню	есть, только как команда меню	нет	нет	есть	нет	нет
Поддержка внешних библиотек	**есть**	нет	нет	нет	нет	нет	нет	нет
Рефакторинг кода	нет	нет	нет	нет	нет	есть	нет	нет

Fortran в CodeBlocks [44] является наиболее близким к FRIS по реализованным возможностям. Однако он не использует для анализа текстов программ каких-либо сложных механизмов, например, генераторов анализаторов и не учитывает важных особенностей, описанных в тексте стандарта Fortran 2003, например переноса лексемы на одну и более строк. Если в тексте программы есть такая лексема, данный языковой сервис полностью теряет свою функциональность. Необходимо отметить, что CodeBlocks работает с программой в представлении абстрактного синтаксического дерева, т.е. хранит деревья разбора, каждое из которых привязано к конкретному файлу с текстом программы. FRIS для работы с программой использует её представление в виде модели значимых элементов, которая не связана с абстрактным синтаксическим деревом (модель используется для представления всего программного проекта), каждый конкретный её элемент получается путём обработки абстрактного синтаксического дерева на стадии семантического анализа (о чём подробно сказано в пункте 2.3.3).

Отсутствие публикаций с описаниями моделей, реализованных в рассмотренных языковых сервисах, затрудняет сравнение в этой части. Свидетельством наличия подобных моделей могут являться зависящие от них функциональные возможности, широкого набора реализаций языковых сервисов (таблица 3.9). Наличие во FRIS расширенной поддержки внешних библиотек, не обязательно на языке Fortran, *может свидетельствовать о новизне и оригинальности* разработанных автором:

- абстрактной модели языкового сервиса;
- модели значимых элементов языка программирования Fortran 2003;
- концепции расширенной поддержки внешних библиотек;
- алгоритма подсветки синтаксических конструкций Fortran-программ, поддерживающего выделение элементов внешних библиотек.

Выводы к главе 3

1. Проведено сравнение языкового сервиса FRIS, учитывающего специфику написания современных сложноструктурированных программ, и других языковых сервисов Fortran. Показаны основные преимущества FRIS, который в полном объёме реализует функции технологии IntelliSense, а также работает с комментариями документирования и предоставляет расширенную поддержку внешних библиотек программ. Обоснованы элементы научной новизны диссертации.

2. Продемонстрирована поддержка во FRIS технологий параллельного программирования и высокопроизводительных вычислений, особенно актуальная при разработке параллельных программ для современных СуперЭВМ, для эффективного использования которых требуется работа со смешанной схемой распараллеливания: MPI, OpenMP и SIMD-операции.
3. Продемонстрирована реализованная во FRIS расширенная поддержка библиотек РФЯЦ-ВНИИЭФ УРС-ОФ и ЕФР, позволяющая упростить работу с данными библиотеками в программах за счёт визуального выделения элементов библиотек в тексте программ, а также предоставления контекстной помощи для их элементов.
4. Показан практический эффект от применения FRIS в многопоточных параллельных программах и библиотеках РФЯЦ-ВНИИЭФ:
   - FRIS используется при написании библиотеки MAGIK, программ КПД-1D, ЛЭГАК-3D и других;
   - получена усреднённая оценка ускорения от использования FRIS, составляющая 1.4 раза, при этом для отдельных возможностей IntelliSense до 2.8 раз, а для сниппетов вплоть до 1300 раз при использовании сниппета для программного модуля двусвязных списков;
   - применение FRIS повышает удобство программирования, позволяет существенно упростить внесение изменений и разработку новых возможностей, а также интеграцию программных алгоритмов различных методик друг с другом, улучшает наглядность программного кода и снижает количество ошибок, допускаемых при программировании.

Заключение

В диссертации исследованы проблемы автоматизации процесса написания текста программ на языке Fortran в интегрированной среде разработки Microsoft Visual Studio.

К основным полученным при выполнении данной работы результатам относятся:

1. Разработана абстрактная модель языкового сервиса, обеспечивающая расширенную поддержку языка программирования и предназначенная для построения языковых сервисов для языков программирования и интегрированных сред разработки; учитывающая работу с внешними программными библиотеками, реализованными на языках программирования, отличных от целевого.
2. Разработана модель значимых элементов языка программирования Fortran стандарта Fortran 2003, необходимых для реализации возможностей технологии IntelliSense, которая позволяет сопровождать значимые элементы смысловым описанием их предназначения.
3. Разработана модель описания прикладных программных интерфейсов (API) Fortran 2003, позволяющая языковому сервису отражать значимые элементы используемых в программе библиотек без анализа их исходного кода. Модель является эквивалентным представлением используемых в программе или библиотеке программ значимых элементов в специально разработанной структуре XML-файла, что позволяет анализировать Fortran-программы при помощи средств работы с XML. Данная модель в совокупности с моделью XML комментариев документирования позволяет обеспечить поддержку внешних библиотек программ, чьи исходные коды написаны не на Fortran или недоступны.
4. Разработана модель XML комментариев документирования для Fortran 2003, позволяющая снабжать значимые элементы описанием их предназначения, использования и любой иной информацией. Модель позволяет не только документировать значимые элементы непосредственно в тексте программ, но и использовать внешние файлы XML специального формата, содержащие документацию для значимых элементов.
5. Разработана концепция расширенной поддержки внешних библиотек программ, позволяющая предоставлять всю необходимую информацию для технологии IntelliSense и для алгоритма подсветки синтаксиса для визуального выделения значимых элементов библиотек в тексте программ, в которых они используются.

6. Разработана модель представления значимых элементов для использования в качестве источника информации для различных возможностей технологии IntelliSense.
7. Разработаны алгоритмы полнотекстного анализа Fortran-программ, написанных с использованием стандарта Fortran 2003, работающие при непосредственном редактировании текста программы, позволяющие обрабатывать лексически, синтаксически и семантически некорректные тексты программ и обеспечивающие подсветку синтаксиса в режиме построчного инкрементального разбора с возможностью сохранения и восстановления состояния разбора в произвольные моменты времени.
8. Реализован языковой сервис FRIS, учитывающий специфику написания на языке Fortran 2003 современных сложноструктурированных программ в интегрированной среде разработки Microsoft Visual Studio.
9. Языковой сервис FRIS используется для написания Fortran-программ в РФЯЦ-ВНИИЭФ, ОАО КБП и КБМ.
10. Использование FRIS позволяет сократить время написания текстов программ, по полученным оценкам, в среднем в 1.45 раза, скоординировать работу команды разработчиков над одним проектом, снизить количество допускаемых при программировании ошибок и повысить удобство программирования

В заключение автор хотел бы выразить благодарность своему научному руководителю, доктору физико-математических наук Бартеневу Юрию Германовичу за постоянное внимание к данной работе, ценные дискуссии и советы; Куделькину Вадиму Григорьевичу и Жильниковой Наталии Николаевне за помощь в адаптации FRIS к библиотеке УРС-ОФ; Олесницкой Ксении Константиновне и Петровой Марии Александровне за помощь в адаптации FRIS к библиотеке ЕФР; Касаткину Сергею Сергеевичу за помощь в тестировании FRIS; а также всем сотрудникам РФЯЦ-ВНИИЭФ, принимавшим участие в тестировании FRIS и использующим его в настоящее время в своей повседневной работе, за предложения по улучшению FRIS и адаптации к их работе; сотрудникам НИО 1, 2, 3 РФЯЦ-ВНИИТФ, сотрудникам ОАО «КБП» и ОАО «КБМ» за проявленный интерес к данной работе и обсуждение возможностей использования FRIS в их деятельности.

Список сокращений и условных обозначений

ЭВМ	Электронная вычислительная машина
ANSI	American National Standards Institute
ISO	International Organization for Standardization
LANL	Los Alamos National Laboratory (Лос-Аламосская национальная лаборатория)
BNL	Brookhaven National Laboratory (Брукхейвеновская национальная лаборатория)
CERN	European Organization for Nuclear Research (Европейский центр физики высоких энергий)
ФГУП	Федеральное государственное унитарное предприятие
РФЯЦ	Российский федеральный ядерный центр
ВНИИЭФ	Всероссийский научно-исследовательский институт экспериментальной физики
ВНИИТФ	Всероссийский научно-исследовательский институт технической физики
IDE	Integrated Development Environment (Интегрированная среда разработки)
VS	Microsoft Visual Studio
СуперЭВМ	Специализированная высокопроизводительная электронная вычислительная машина, используемая для решения сложных задач имитационного моделирования, недоступных для решения на Персональных ЭВМ
FRIS	Fortran Intelligent Solutions
ИТМФ	Институт теоретической и математической физики во ФГУП РФЯЦ-ВНИИЭФ
SIMD	(Single Instruction) Multiple Data stream processing - один поток команд - много потоков данных, архитектура SIMD
ИПМ	Ордена Ленина Институт прикладной математики имени М.В. Келдыша
OFP	Open Fortran Parser

SAX	Simple API for XML
API	Application Programming Interface, интерфейс прикладного программирования
XML	extensible markup language, расширяемый язык разметки
РФ	Российская Федерация
XSD	XML Schema definition
АО	Акционерное общество
ОАО	Открытое акционерное общество
НПК	Научно-производственная корпорация
КБМ	Конструкторское бюро машиностроения
КБП	Конструкторское бюро приборостроения
ПАО	Публичное акционерное общество
НПО	Научно-производственное объединение
AST	Abstract Syntax Tree, абстрактное дерево разбора
MPF	Managed Package Framework
ANTLR	Another tool for language recognition
TPL	Task Parallel Library
ПЭВМ	Персональная электронная вычислительная машина
ОЗУ	Оперативное запоминающее устройство

Список источников и литературы

1. The Fortran automatic coding system for the IBM 704 EDPM. Programmers reference manual. IBM, 1956.

2. Fortran 66 Standard; Fortran ANSI X3.9-1966.

3. Fortran 77 Standard; ANSI X3.9-1978, ISO 1539:1980, Information technology - Programming languages - Fortran - Part 1: Base Language.

4. Fortran 90 Standard; ISO/IEC 1539-1:1991, Information technology - Programming languages - Fortran - Part 1: Base Language.

5. Fortran 95 Standard; ISO/IEC 1539-1:1997, Information technology - Programming languages - Fortran - Part 1: Base Language.

6. Fortran 2003 Standard; ISO/IEC 1539-1:2004, Information technology - Programming languages - Fortran - Part 1: Base Language.

7. Fortran 2008 Standard; ISO/IEC 1539-1:2010, Information technology - Programming languages - Fortran - Part 1: Base Language.

8. Горелик А.М. Современный Фортран для компьютеров традиционной архитектуры и для параллельных вычислительных систем (аналитический обзор) [Электронный ресурс] // Препринт Института прикладной математики им. М.В.Келдыша РАН, № 29, Москва, 2003 г. – Режим доступа: http://www.keldysh.ru/papers/2003/prep29/preprint.asp?id=2003-29 (29.05.2014)

9. Горелик А.М. Средства поддержки параллельных вычислений в стандартах языка Фортран [Электронный ресурс] // Препринт Института прикладной математики им. М.В.Келдыша РАН, № 68, Москва, 2012 г. – Режим доступа: http://www.keldysh.ru/papers/2012/preprint.asp?id=2012-68 (29.05.2014)

10. The TIOBE Programming Community index for August 2016 [Электронный ресурс] – Режим доступа: http:// www.tiobe.com/ (16.08.2016)

11. The Top Programming Languages 2015 [Электронный ресурс] – Режим доступа: http://spectrum.ieee.org/static/interactive-the-top-programming-languages-2015/ (16.08.2016)

12. NAG [Электронный ресурс] – Режим доступа: https://www.nag.co.uk/ (16.08.2016)

13. LAPACK – Linear Algebra PACKage [Электронный ресурс] – Режим доступа: http://www.netlib.org/lapack/ (16.08.2016)

14. NJOY 2012 – Nuclear Data Processing System [Электронный ресурс] – Режим доступа: http://t2.lanl.gov/nis/codes/NJOY12/ (16.08.2016)

15. Los Alamos National Laboratory [Электронный ресурс] – Режим доступа: http://www.lanl.gov/ (05.02.2014)

16. Brookhaven National Laboratory [Электронный ресурс] – Режим доступа: http://www.bnl.gov/ (05.02.2014)

17. European Organization for Nuclear Research [Электронный ресурс] – Режим доступа: http://www.cenr.ch/ (05.02.2014)

18. Российский федеральный ядерный центр – Всероссийский научно-исследовательский институт экспериментальной физики [Электронный ресурс] – Режим доступа: http://www.vniief.ru/ (01.03.2016)

19. Российский федеральный ядерный центр – Всероссийский научно-исследовательский институт технической физики им. академика Забабахина [Электронный ресурс] – Режим доступа: http://www.vniitf.ru/ (01.03.2016)

20. Бартенев Ю.Г., Бондаренко Ю.А., Спиридонов В.Ф. Прогноз параметров подсистем вычислительной системы эксафлопсного класса // Вопросы атомной науки и техники. Сер.: Математическое моделирование физических процессов, 2012. Вып. 3. с. 15-23

21. TOP500 List [Электронный ресурс] // Список 500 самых мощных мировых суперкомпьютеров, Июнь 2016 – Режим доступа: https://www.top500.org/lists/2016/06/ (28.06.2016)

22. Суперкомпьютеры Top50 [Электронный ресурс] // Список 50 самых мощных суперкомпьютеров СНГ, 24 редакция от 29.03.2016 – Режим доступа: http://top50.supercomputers.ru/?page=rating (28.06.2016)

23. MPI: A Message-Passing Interface Standard Version 3.1; Message Passing Interface Forum; June 4, 2015

24. OpenMP Application Program Interface Version 4.0; OpenMP Architecture Review Board; July 2013

25. Копкин С.В., Крючков И.А. Алгоритм модернизированного многочастичного потенциала для молекулярно-динамического моделирования на графическом

арифметическом ускорителе // Вопросы атомной науки и техники. Сер.: Математическое моделирование физических процессов, 2010. Вып. 3. с. 73-82

26. Воронин Б.Л., Грушин С.А., Житник А.К., Залялов А.Н. и др. Программно-аппаратные комплексы на базе вычислительных систем с арифметическими ускорителями для моделирования методом Монте-Карло и методом молекулярной динамики // Вопросы атомной науки и техники. Сер.: Математическое моделирование физических процессов, 2011. Вып. 4. с. 66-71

27. Рыбкин А.С., Залялов А.Н., Малькин А.Г., Огнев С.П., Рослов В.И. Программный комплекс на базе гибридных вычислительных систем для расчёта критических параметров методом Монте-Карло // Молодёжь в науке. Сборник докладов девятой научно-технической конференции (г. Саров, 26-28 октября 2010 г.) / Под ред. В.П. Соловьёва, ФГУП «РФЯЦ-ВНИИЭФ», Саров, 2011 – с.120-125

28. Черных С.О., Королёв Р.А., Сухих А.С. Исследование эффективности блока модуля Newt, предназначенного для счёта систем линейных алгебраических уравнений методом переменных направлений на арифметических ускорителях // Молодёжь в науке. Сборник докладов десятой научно-технической конференции (г. Саров, 1-3 ноября 2011 г.) / Под ред. В.П. Соловьёва, ФГУП «РФЯЦ-ВНИИЭФ», Саров, 2012 – с. 616-624

29. Быков А.Н., Сизов Е.А. Адаптация расчета процессов газовой динамики в эйлерово-лагранжевых координатах по методике РАМЗЕС-КП на ЭВМ с арифметическими ускорителями // Супервычисления и математическое моделирование: Труды XIV Международной конференции / Под ред. Р.М. Шагалиева – Саров: ФГУП «РФЯЦ-ВНИИЭФ», 2013 – с.129-135

30. Быков А.Н., Гордеев Д.Г., Жильникова Н.Н., Куделькин В.Г. и др. Адаптация методики РАМЗЕС-КП для решения задач газовой динамики и теплопроводности на гибридных параллельных ЭВМ // Супервычисления и математическое моделирование: Труды XV Международной конференции / Под ред. Р.М. Шагалиева – Саров: ФГУП «РФЯЦ-ВНИИЭФ», 2015 – с.169-175

31. Быков А.Н., Гордеев Д.Г., Куделькин В.Г. и др. Организация взаимодействий прикладных программ и библиотеки УРС-ОФ расчёта теплофизических свойств веществ на ЭВМ с арифметическими ускорителями // Супервычисления и математическое моделирование: Труды XIV Международной конференции / Под ред. Р.М. Шагалиева – Саров: ФГУП «РФЯЦ-ВНИИЭФ», 2013 – с.120-128

32. Олесницкая К.К., Антипин И.А., Петрова М.А. Библиотека ЕФР как средство эффективного доступа к файловым данным на гибридных вычислительных системах и суперкомпьютерах // Супервычисления и математическое моделирование: Труды XV Международной конференции / Под ред. Р.М. Шагалиева – Саров: ФГУП «РФЯЦ-ВНИИЭФ», 2015 – с.346-355

33. Microsoft Visual Studio Integrated development environment [Электронный ресурс] – Режим доступа: http://www.visualstudio.com/ (05.01.2014)

34. Intel Fortran Composer (Visual Fortran) [Электронный ресурс] – Режим доступа: http://software.intel.com/en-us/articles/intel-fortran-composer-xe-2013-sp1-release-notes (05.01.2014)

35. PGI Visual Fortran [Электронный ресурс] – Режим доступа: https://www.pgroup.com/products/pvf.htm (05.01.2014)

36. Zhiyu S., Zhiyuan L., Pen-Chung Y. An emperical study of Fortran programs for parallelizing compilers. // IEEE Trans. on Parallel and Distributed Systems, July 1990

37. Воеводин В.В. Отображение проблем вычислительной математики на архитектуру вычислительных систем. // Вычислительные методы и программирование. Изд. МГУ, 2000, с. 105-112

38. Бахтин В.А., Бородич И.Г., Катаев Н.А. и др. Диалог с программистом в системе автоматизации распараллеливания САПФОР // Супервычисления и математическое моделирование: Труды XIII Международной конференции / Под ред. Р.М. Шагалиева – Саров: ФГУП «РФЯЦ-ВНИИЭФ», 2012 – с.80-84

39. Система САПФОР [Электронный ресурс] – Режим доступа: http://keldysh.ru/dvm/SAPFOR/ (20.03.2015)

40. Intel VTune [Электронный ресурс] – Режим доступа: https://software.intel.com/en-us/intel-vtune-amplifier-xe/ (19.08.2016)

41. Lahey Fortran [Электронный ресурс] – Режим доступа: http://www.lahey.com (20.03.2015)

42. Fortran Code Nav [Электронный ресурс] – Режим доступа: https://publicwiki.deltares.nl/display/FCN/Home (20.03.2015)

43. Eclipse Phortran [Электронный ресурс] – Режим доступа: http://www.eclipse.org/photran/ (20.03.2015)

44. CodeBlocks Fortran [Электронный ресурс] – Режим доступа: http://www.codeblocks.org; http://cbfortran.sourceforge.net (20.03.2015)

45. NetBeans Fortran [Электронный ресурс] – Режим доступа: https://netbeans.org (20.03.2015)

46. Касаткин С.С., Алексеев А.В., Мжачих С.В. Библиотека программ расчета многогрупповых макроскопических констант – MAGIK // Молодёжь в науке. Сборник докладов 12-й научно-технической конференции / Под ред. В.П. Соловьёва, ФГУП «РФЯЦ-ВНИИЭФ», Саров, 2014 – с.46-52

47. Language Services [Электронный ресурс] – Режим доступа: http://msdn.microsoft.com/en-us/library/bb165099.aspx (01.04.2012)

48. Model of a Language Service [Электронный ресурс] – Режим доступа: http://msdn.microsoft.com/en-us/library/bb166518(v=vs.100).aspx (01.04.2012)

49. Using IntelliSense [Электронный ресурс] – Режим доступа: http://msdn.microsoft.com/en-us/library/hcw1s69b(v=vs.110).aspx (01.04.2012)

50. Aho A., Ullman D.; Principles of Compiler Design. Addison-Wesley, 1977.

51. Ахо А., Ульман. Д.; Теория синтаксического анализа, перевода и компиляции. М.:Мир, 1978, т. 1, 612с. – т. 2, 487с.

52. Ахо А., Сети Р., Ульман Д.; Компиляторы: принципы, технологии и инструменты. : Пер. с англ. – М.: Вильямс, 2003 – 768 с.

53. Ахо А., Лам М., Сети Р., Ульман Д.; Компиляторы: принципы, технологии и инструментарий, 2-е изд. : Пер. с англ. – М.: Вильямс, 2008 – 1184 с.

54. Карпов Ю.Г. Теория и технология программирования. Основы построения трансляторов. – СПб.: БХВ-Петербург, 2005 – 272 с.

55. Terence Parr; The Definitive ANTLR Reference. Building Domain-Specific Languages; The Pragmatic Bookshelf, Texas, 2007 – 369 p.

56. Terence Parr; Language Implementation Patterns Create Your Own Domain-Specific and General Programming Languages; The Pragmatic Bookshelf, Texas, 2010 – 384 p.

57. ГОСТ 19781-90 // Обеспечение систем обработки информации программное. Термины и определения. /А.П. Гагарин, А.В. Багров, Н.А. Серова; Переиздание, М: Стандартинформ, 2010 – 11 с.

58. Баранова Т. П., Вершубский В.Ю. Использование библиотеки классов пакета 'SAGE' для анализа программ, написанных на языке ФОРТРАН [Электронный ресурс] // Препринт Института прикладной математики им. М.В.Келдыша РАН, № 69, Москва, 2004 г. – Режим доступа: http://library.keldysh.ru/preprint.asp?id=2004-69 (29.05.2014)

59. Баранова Т. П., Вершубский В.Ю. Анализатор программ, написанных на языке ФОРТРАН [Электронный ресурс] // Препринт Института прикладной математики им. М.В.Келдыша РАН, № 124, Москва, 2005 г. – Режим доступа: http://www.keldysh.ru/papers/2005/prep124/preprint.asp?id=2005-124 (29.05.2014)

60. Пакет для анализа C/C++ и Fortran программ SAGE [Электронный ресурс] – Режим доступа: http://www.extreme.indiana.edu/ (29.05.2014)

61. Craig E. Rasmussen, Christopher Rickett; Dan Quinlan; Matthew Sottile «Fortran Development Tools: Providing a Roadmap for Application Development on Advanced Computer Architectures» // Associate Directorate for Theory, Simulation, and Computation (ADTSC), ADTSC Science Highlights 2008, LA-UR-08-1690, pp. 2-5

62. David Brownell, SAX2, O'Reilly, ISBN 0-596-00237-8

63. Монахов В. Язык программирования Java и среда NetBeans. — 3-е издание. СПб.: «БХВ-Петербург», 2011. — С. 704.

64. Воронов Г.И., Горев И.В., Леонова Н.И. и др. Программное обеспечение функционирования пакета УРС-ОФ // Вопросы атомной науки и техники. Сер.: Математическое моделирование физических процессов, 1999. Вып. 3. с. 56-58

65. Волгин А.В., Красов А.В., Кузнецов М.Ю., Тарасов В.И. Библиотека ЕФР для универсального представления расчётных данных // Труды РФЯЦ-ВНИИЭФ, 2007. Вып. 11 с. 130-135

66. Олесницкая К.К., Антипин И.А., Шубина М.А. Библиотека ЕФР для масштабируемого доступа к файловым данным на многопроцессорных ЭВМ // XI Забабахинские науч. чтения: сб. тезисов. Снежинск, 2012. с. 335-336

67. Олесницкая К.К., Антипин И.А., Шубина М.А. Библиотека ЕФР для масштабируемого доступа к файловым данным на многопроцессорных ЭВМ // Супервычисления и математическое моделирование: Труды XIII Международной конференции / Под ред. Р.М. Шагалиева – Саров: ФГУП «РФЯЦ-ВНИИЭФ», 2012 – с.370-379

68. Раткевич И.С. Анализатор Fortran программы для реализации технологии IntelliSense в текстовом редакторе Microsoft Visual Studio // Системы управления и информационные технологии, Воронеж: Научная книга, 2015, № 1.1(59), с. 168-172

69. I.S. Ratkevich. FRIS language service for extended Fortran support in Microsoft Visual Studio. // Proceedings of the Institute for System Programming Volume 27 (Issue 3). 2015 y. pp. 9-28; ISSN 2220-6426 (Online), ISSN 2079-8156 (Print). DOI: 10.15514/ISPRAS-2015-27(3)-1

70. Раткевич И.С. Языковой сервис FRIS для эффективной разработки Fortran-приложений. Обзор возможностей // Вопросы атомной науки и техники, сер. Математическое моделирование физических процессов. 2015 г. Выпуск 4, с. 49-58

71. Раткевич И.С. Общая модель для расширенной поддержки языков программирования в интегрированных средах разработки // Успехи современной радиоэлектроники. 2016 г. Выпуск 2, с. 187-194

72. Раткевич И.С., Бартенев Ю.Г., Касаткин С.С.; Языковой сервис FRIS для эффективной разработки Fortran-приложений // Труды XV международной конференции «Супервычисления и математическое моделирование» /Под ред. Р.М. Шагалиева, г. Саров: ФГУП «РФЯЦ-ВНИИЭФ», 2015 г., ISBN 978-5-9515-0300-8, с.385-395

73. Раткевич И.С. Общая модель для расширенной поддержки языков программирования в интегрированных средах разработки // Итоги диссертационных исследований Том 3 – Материалы VII Всероссийского конкурса молодых учёных, посвящённого 70-летию Победы - М.: РАН, 2015; с. 27-38

74. Свидетельство о государственной регистрации программы для ЭВМ № 2015615397. Языковой сервис для эффективной разработки Fortran приложений в Microsoft Visual Studio («FRIS») / Раткевич И.С.; зарегистрировано в Реестре программ для ЭВМ 18.05.2015г.

75. Certificate of Registration computer program. Registration number TXu 1-936-258. FRIS (FoRtran Intelligent Solutions) / Irina Sergeevna Ratkevich; register of copyrights, United States of America, 31.07.2014

76. Выписка из акта внедрения программного обеспечения FRIS; Акт / ВНИИЭФ; Шагалиев Р.М.; Инв. № 195-2025-25/169122; Саров, 2016.

77. Акт реализации результатов диссертационного исследования младшего сотрудника ФГУП «Российский федеральный ядерный центр – Всероссийский научно-

исследовательский институт экспериментальной физики» Раткевич Ирины Сергеевны; Акт / Открытое акционерное общество «Научно производственная корпорация «Конструкторское бюро машиностроения»»; Грачиков Д.В., Шабалкин А.П., Асцатрян А.С., Родионов К.А; Инв. № 007-122/14431; Коломна, 2015.

78. Акт реализации результатов диссертационного исследования младшего научного сотрудника ФГУП «Российский федеральный ядерный центр – Всероссийский научно-исследовательский институт экспериментальной физики» Раткевич Ирины Сергеевны; Акт / Акционерное общество «Конструкторское бюро приборостроения им. Академика А.Г. Шипунова»; Семашкин В.Е., Симаков С.Ю., Болосов Д.А., Гусев А.В.; Тула, 2015.

79. Алексеев А.В., Евдокимов В.В., Шагалиев Р.М. Методика численного решения нестационарного трехмерного уравнения переноса частиц в комплексе САТУРН // Вопросы атомной науки и техники. Сер.: Математическое моделирование физических процессов, 1993. Вып. 3. с. 3-8

80. Алексеев А.В., Беляков И.М., Бочков А.И., Евдокимов В.В. и др. Методика САТУРН-2005. Математические модели, алгоритмы и программы решения многомерных задач переноса частиц и энергии // Вопросы атомной науки и техники. Сер.: Математическое моделирование физических процессов, 2013. Вып. 4. с. 17-30

81. Экспертное заключение. Оценка качественного и количественного эффекта от использования программного обеспечения FRIS; Акт / ВНИИЭФ; Алексеев А.В., Бнятов А.В., Горшихин А.А., Раткевич С.С., Шмелёв А.В; Инв. № 195-96/176478; Саров, 2016.

82. Каскадная модель жизненного цикла программного обеспечения [Электронный ресурс] – Режим доступа: http://www.intuit.ru/studies/courses/2195/55/lecture/1620/ (25.03.2015)

83. Michael Sorens; Taming Sandcastle: A .NET Programmer's Guide to Documenting Your Code [Электронный ресурс] – Режим доступа: https://www.simple-talk.com/dotnet/.net-tools/taming-sandcastle-a-.net-programmers-guide-to-documenting-your-code/ (25.03.2015)

84. Lambert M Surhone, Mariam T Tennoe, Susan F Henssonow Doxygen; Betascript Publishing, 2010 – 168 с.

85. What Is Managed Code? [Электронный ресурс] – Режим доступа: http://msdn.microsoft.com/en-us/library/windows/desktop/bb318664(v=vs.85).aspx (25.03.2015)

86. XML Schema Definition [Электронный ресурс] – Режим доступа: https://www.w3.org/TR/xmlschema11-1/ (25.03.2015)

87. Managed Package Framework Classes http://msdn.microsoft.com/en-us/library/bb164709.aspx (07.03.2015)

88. Троелсен Э.; Язык программирования C# 2010 и платформа .NET 4.0 5-е изд.: Пер. с англ. – М.: ООО «И.Д. Вильямс», 2011 – 1392 с.

89. Nayyeri K.; Professional Visual Studio® 2008 Extensibility, Wiley Publishing, Inc., Indianapolis, Indiana, 2008 – 520 с.

90. Snell M., Powers L.; Microsoft® Visual Studio®2010 Unleashed, Pearson Education, Inc., Indianapolis, Indiana, 2011 – 1196 с.

91. Language Service Interfaces [Электронный ресурс] – Режим доступа: http://msdn.microsoft.com/en-us/library/bb164598(v=vs.100).aspx (07.03.2015)

92. Code Snippets [Электронный ресурс] – Режим доступа: http://msdn.microsoft.com/en-us/library/ms165392.aspx (07.03.2015)

93. ExpansionProvider Class [Электронный ресурс] – Режим доступа: http://msdn.microsoft.com/en-us/library/microsoft.visualstudio.package.expansionprovider.aspx (07.03.2015)

94. ExpansionFunction Class [Электронный ресурс] – Режим доступа: http://msdn.microsoft.com/en-us/library/microsoft.visualstudio.package.expansionfunction.aspx (07.03.2015)

95. AuthoringSink Class [Электронный ресурс] – Режим доступа: http://msdn.microsoft.com/en-us/library/microsoft.visualstudio.package.authoringsink.aspx (07.03.2015)

96. ParseRequest Class [Электронный ресурс] – Режим доступа: http://msdn.microsoft.com/en-us/library/microsoft.visualstudio.package.parserequest.aspx (07.03.2015)

97. AuthoringScope Class [Электронный ресурс] – Режим доступа: http://msdn.microsoft.com/en-us/library/microsoft.visualstudio.package.authoringscope.aspx (07.03.2015)

98. ViewFilter Class [Электронный ресурс] – Режим доступа: http://msdn.microsoft.com/en-us/library/microsoft.visualstudio.package.viewfilter.aspx (07.03.2015)

99. TextTipData Class [Электронный ресурс] – Режим доступа: http://msdn.microsoft.com/en-us/library/microsoft.visualstudio.package.texttipdata.aspx (07.03.2015)

100. Declarations Class [Электронный ресурс] – Режим доступа: http://msdn.microsoft.com/en-us/library/microsoft.visualstudio.package.declarations.aspx (07.03.2015)

101. CompletionSet Class [Электронный ресурс] – Режим доступа: http://msdn.microsoft.com/en-us/library/microsoft.visualstudio.package.completionset.aspx (07.03.2015)

102. Methods Class [Электронный ресурс] – Режим доступа: http://msdn.microsoft.com/en-us/library/microsoft.visualstudio.package.methods.aspx (07.03.2015)

103. MethodData Class [Электронный ресурс] – Режим доступа: http://msdn.microsoft.com/en-us/library/microsoft.visualstudio.package.methoddata.aspx (07.03.2015)

104. Parr T., Fisher K. LL(*): the foundation of the ANTLR parser generator // Proceedings of the 32nd ACM SIGPLAN Conference on Programming Language Design and Implementation (San Jose, California, USA, June 04 - 08, 2011). PLDI '11. ACM, New York, NY, p.425-436

105. Task Parallel Library (TPL) [Электронный ресурс] – Режим доступа: http://msdn.microsoft.com/ru-ru/library/dd460717(v=vs.110).aspx (07.03.2015)

106. Бахрах С.М., Величко С.В., Спиридонов В.Ф. и др. Методика ЛЭГАК-3D расчёта трехмерных нестационарных течений многокомпонентной сплошной среды и принципы её реализации на многопроцессорных ЭВМ с распределённой памятью // Вопросы атомной науки и техники. Сер.: Математическое моделирование физических процессов, 2004. Вып. 4. с. 41-50

107. Чеботарь С.В., Сухих С.С. Параллельный решатель систем линейных алгебраических уравнений «Модуль Newt». Его применение в методике КОРОНА // Заба-

бахинские научные чтения: сборник тезисов докладов XII Международной конференции 2-6 июня 2014. РФЯЦ - ВНИИТФ, Снежинск, 2014 – с.333-334

108. Дьянов Д.Ю., Корсакова Е.И., Симонов Г.П., Циберев К.В., Шувалова Е.В. Результаты верификации моделей упругопластического деформирования и разрушения реализованных в пакете программ ЛЕГАК-ДК // Молодёжь в науке. Сборник докладов девятой научно-технической конференции (г. Саров, 26-28 октября 2010 г.) / Под ред. В.П. Соловьёва, ФГУП «РФЯЦ-ВНИИЭФ», Саров, 2011 – с.79-86

109. Анищенко А.А., Ермаков П.В., Касаткин С.С. и др. Визуализационно-интегрирующая платформа «АЛЬКОР» ФГУП «РФЯЦ-ВНИИЭФ» для создания систем имитационного распределенного моделирования // Седьмая всероссийская научно-практическая конференция «Имитационное моделирование. Теория и практика» (ИММОД-2015): Труды конф., Т.1.-М.: ИПУ РАН, 2015. -345 с. – ISBN 978-5-91450-172-0. с. 231-238

110. International Organization for Standardization, ISO/IEC/SC22/WG5 (Fortran), Geneva, TS 29113. TS on further interoperability with C, 2012. [Электронный ресурс] – Режим доступа: ftp://ftp.nag.co.uk/sc22wg5/N1901-N1950/N1917.pdf (07.03.2015)

Приложение А. Определение XML схемы (XSD) для описания прикладных программных интерфейсов (API) языка Fortran.

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:annotation>
    <xsd:documentation>
    Схема описания API для элементов языка Fortran (стандарт Fortran-2003).
    Схема разработана на базе схемы отражения используемой для "управляемых"
    (в терминологии Microsoft) языков в Sandcastle (https://shfb.codeplex.com/).
    Автор: Раткевич И.С.
    Версия: 1
    Дата создания: 01.08.2014
    Дата модификации: 14.08.2014
    </xsd:documentation>
  </xsd:annotation>
  <!-- Элементы API -->
  <!--Корневой элемент - контейнер для всех API библиотеки или программного проекта-->
  <xsd:element name="reflection">
    <xsd:complexType>
      <xsd:sequence>
        <!--Перечень всех библиотек, чьи API описаны в файле-->
        <xsd:element ref="assemblies" minOccurs="1" maxOccurs="1" />
        <!--Перечень всех API. Непосредственно блок их описания-->
        <xsd:element ref="apis" minOccurs="1" maxOccurs="1" />
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
  <!-- Данные о библиотеках -->
  <xsd:element name="assemblies">
    <xsd:complexType>
      <xsd:sequence>
        <!--Перечисление библиотек-->
        <xsd:element ref="assembly" minOccurs="0" maxOccurs="unbounded" />
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
  <!--Данные об одной библиотеке. Обязательные атрибуты: имя и язык программирования-->
  <xsd:element name="assembly">
    <xsd:complexType>
      <!--Имя библиотеки/программного проекта-->
      <xsd:attribute name="name" />
      <!--Версия (необязательный)-->
      <xsd:attribute name="version" use="optional"/>
      <!--Язык программирования, на котором создана библиотека-->
      <xsd:attribute name="language" />
      <!--Имя файла библиотеки (необязательно)-->
      <xsd:attribute name="filename" use="optional"/>
    </xsd:complexType>
  </xsd:element>
  <!-- Информация о прикладных программных интерфейсах - API -->
  <xsd:element name="apis">
    <xsd:complexType>
      <xsd:sequence>
        <!--Список всех программных интерфейсов.-->
        <xsd:element ref="api" minOccurs="0" maxOccurs="unbounded" />
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
  <!--Универсальный элемент для описания любого программного интерфейса - API-->
  <xsd:element name="api">
    <xsd:complexType>
      <xsd:sequence>
```

```
        <!--Информация об элементе: его имя, группа, подгруппа и т.п.-->
        <xsd:element ref="apidata" minOccurs="1" maxOccurs="1" />
        <!--Здесь описание для всех возможных программных интерфейсов-->
        <xsd:choice>
          <!--Глобальная область видимости-->
          <xsd:sequence>
            <!--Перечень элементов.
            При этом для каждого элемента указывается
            только имя его API-->
            <xsd:element ref="elements" minOccurs="0" />
          </xsd:sequence>
          <!--Модуль-->
          <xsd:sequence>
            <!--Элемент, явно указывающий, что это модуль-->
            <xsd:element ref="moduledata"/>
            <!--Импорты других модулей. Могут и не быть.-->
            <xsd:element ref="imports" minOccurs="0"/>
            <!--Далее элементы модуля: переменные, процедуры, типы данных, интерфейсы, подпро-
граммы и функции.-->
            <xsd:element ref="elements" minOccurs="0" />
          </xsd:sequence>
          <!--Ссылка-->
          <xsd:sequence>
            <!--Элемент, явно говорящий, что это ссылка-->
            <xsd:element ref="referencedata"/>
            <!--Ссылка может указывать в том числе и на функцию.
            Тогда вводим дополнительный элемент.-->
            <xsd:element ref="functionref" minOccurs="0"/>
            <!--Служит для обратной связи с элементом, внутри которого содержится
            данный элемент.-->
            <xsd:element ref="containers"  minOccurs="1" />
          </xsd:sequence>
          <!--Тип данных-->
          <xsd:sequence>
            <!--Далее элемент, явно говорящий, что это тип данных-->
            <xsd:element ref="typedata"/>
            <!--Опционально, имя типа данных во внешнем контексте. Для межъязыкового взаимо-
действия.-->
            <xsd:element name="bind" type="bindingType" minOccurs="0"/>
            <!--Для наследования. Указывается базовый тип.-->
            <xsd:element ref="family" minOccurs="0" />
            <!--Параметры типа, если есть-->
            <xsd:element ref="typeparams" minOccurs="0"/>
            <!--Элементы типа данных-->
            <xsd:element ref="elements" minOccurs="0" />
            <!--Обратная связь с родительским элементом-->
            <xsd:element ref="containers" minOccurs="1" />
          </xsd:sequence>
          <!--Переменная, в том числе и поле типа данных-->
          <xsd:sequence>
            <!--Элемент явно указывающий, что это переменная, а заодно и уточняющий,
            что это за переменная.-->
            <xsd:element ref="variabledata"/>
            <!--Опциональный тег, указывает, что переменная является членом
            типа данных-->
            <xsd:element name="memberdata" minOccurs="0"/>
            <!--Тип данных переменной-->
            <xsd:element ref="type" />
            <!--Опционально, размерность массива. Для скаляров не указывается.-->
            <xsd:element ref="dimension" minOccurs="0"/>
            <!--Опционально, имя переменной во внешнем контексте. Для межъязыкового взаимодей-
ствия.-->
            <xsd:element name="bind" type="bindingType" minOccurs="0"/>
            <!--Опционально, начальное значение переменной.
            Для полей производного типа данных - это значение по умолчанию.-->
```

```
            <xsd:element name="value" type="xsd:anyType" minOccurs="0"/>
            <!--Служит для обратной связи с элементом, внутри которого содержится
            данный элемент.-->
            <xsd:element ref="containers"  minOccurs="1" />
          </xsd:sequence>
          <!--Процедура, в том числе и поле типа данных-->
          <xsd:sequence>
            <!--Элемент, явно говорящий, что это процедура-->
            <xsd:element ref="proceduredata"/>
            <!--Если это поле типа данных, то указывается этот тэг-->
            <xsd:element ref="memberdata" minOccurs="0"/>
            <!--Опциональный атрибут - интерфейс-->
            <xsd:element name="interface" type="ProcedureInterface" minOccurs="0"/>
            <!--Дополнительные атрибуты, которые являются составными элементами-->
            <!--Опционально, имя процедуры во внешнем контексте. Для межъязыкового взаимодей-
ствия. Не может быть указан для процедуры - поля типа данных.-->
            <xsd:element name="bind" type="bindingType" minOccurs="0"/>
            <!--Только для связанных с типом данных процедур.
            Атрибут передачи экземпляра типа в процедуру-->
            <xsd:element name="pass" type="PassageType" minOccurs="0"/>
          </xsd:sequence>
          <!--Связанные с типом данных процедуры-->
          <xsd:sequence>
            <!--Данный тэг говорит, что процедура связана с типом данных-->
            <xsd:element ref="typebound" minOccurs="1"/>
            <!--Здесь выбор типа процедуры: Специфичная, Обобщённая, Финальная-->
            <xsd:choice>
              <!--Специфичная-->
              <xsd:sequence>
                <!--Данный тэг говорит, что это специфичная процедура-->
                <xsd:element ref="specific" minOccurs="1"/>
                <!--Элемент, явно говорящий, что это процедура-->
                <xsd:element ref="proceduredata"/>
                <!--Опциональный атрибут - интерфейс-->
                <xsd:element name="interface" type="ProcedureInterface" minOccurs="0"/>
                <!--Дополнительные атрибуты, которые являются составными элементами-->
                <!--Только для связанных с типом данных процедур.
                Атрибут передачи экземпляра типа в процедуру-->
                <xsd:element name="pass" type="PassageType" minOccurs="0"/>
              </xsd:sequence>
              <!--Обобщённая-->
              <xsd:sequence>
                <!--Данный тэг говорит, что это обобщённая процедура-->
                <xsd:element ref="generic" minOccurs="1"/>
                <!--Элемент, явно говорящий, что это процедура-->
                <xsd:element ref="proceduredata"/>
                <!--Это оператор ввода/вывода-->
                <xsd:element name="iooperator" type="IOFormatType" minOccurs="0"/>
                <!--Далее список имён специфичных процедур, составляющих данную процедуру-->
                <xsd:element name="items" type="GenericItemsType" minOccurs="1"/>
              </xsd:sequence>
              <!--Финальная-->
              <xsd:sequence>
                <!--Данный тэг говорит, что это финальная процедура-->
                <xsd:element ref="final" minOccurs="1"/>
                <!--Элемент, явно говорящий, что это процедура-->
                <xsd:element ref="proceduredata"/>
              </xsd:sequence>
            </xsd:choice>
            <!--Обратная связь с родительским элементом-->
            <xsd:element ref="containers" minOccurs="1" />
          </xsd:sequence>
          <!--Обобщённый интерфейс-->
          <xsd:sequence>
            <!--Говорим, что это интерфейс-->
```

```
            <xsd:element ref="interfacedata"/>
            <!--Это оператор ввода/вывода-->
            <xsd:element name="iooperator" type="IOFormatType" minOccurs="0"/>
            <!--Список элементов интерфейса-->
            <xsd:element ref="elements" minOccurs="0" />
            <!--Обратная связь с родительским элементом-->
            <xsd:element ref="containers" minOccurs="1" />
          </xsd:sequence>
          <!--Подпрограмма и функция-->
          <xsd:sequence>
            <!--Указываем, что это метод-->
            <xsd:element ref="methoddata"/>
            <!--Далее выбор: подпрограмма или функция-->
            <xsd:choice>
              <!--Подпрограмма-->
              <xsd:sequence>
                <!--Указываем, что это подпрограмма-->
                <xsd:element ref="subroutinedata"/>
                <!--Дополнительные сложные атрибуты-->
                <!--Данная подпрограмма является дополнительной точкой входа в другую подпро-
грамму-->
                <xsd:element name="entry" type="EntryPointType" minOccurs="0"/>
                <!--Опционально, имя процедуры во внешнем контексте. Для межъязыкового взаимо-
действия-->
                <xsd:element name="bind" type="bindingType" minOccurs="0"/>
                <!--Собсвтенно описание внутреннего устройства процедуры-->
                <!--Импорты других модулей. Могут и не быть.-->
                <xsd:element ref="imports" minOccurs="0"/>
                <!--Описание формальных аргументов-->
                <xsd:element ref="parameters" minOccurs="0" />
                <!--Далее элементы подпрограммы: Переменные, процедуры, типы данных, интерфей-
сы, подпрограммы и функции.-->
                <xsd:element ref="elements" minOccurs="0" />
              </xsd:sequence>
              <!--Функция-->
              <xsd:sequence>
                <!--Указываем, что это функция-->
                <xsd:element ref="functiondata"/>
                <!--Дополнительные сложные атрибуты-->
                <!--Данная функция является дополнительной точкой входа в другую подпрограмму-
->
                <xsd:element name="entry" type="EntryPointType" minOccurs="0"/>
                <!--Опционально, имя процедуры во внешнем контексте. Для межъязыкового взаимо-
действия-->
                <xsd:element name="bind" type="bindingType" minOccurs="0"/>
                <!--Собственно описание внутреннего устройства процедуры-->
                <!--Импорты других модулей. Могут и не быть.-->
                <xsd:element ref="imports" minOccurs="0"/>
                <!--Описание формальных аргументов-->
                <xsd:element ref="parameters" minOccurs="0" />
                <!--Описание возвращаемого значения-->
                <xsd:element ref="result" />
                <!--Далее элементы подпрограммы: переменные, процедуры, типы данных, интерфей-
сы, подпрограммы и функции.-->
                <xsd:element ref="elements" minOccurs="0" />
              </xsd:sequence>
            </xsd:choice>
            <!--Обратная связь с родительским элементом-->
            <xsd:element ref="containers" minOccurs="1" />
          </xsd:sequence>
          <!--Namelist-->
          <xsd:sequence>
            <!--Элемент, показывающий, что это список имён-->
            <xsd:element ref="namelistdata"/>
            <!--Список ассоциированных элементов-->
```

```
            <xsd:element ref="elements" minOccurs="0" />
            <!--Обратная связь с родительским элементом-->
            <xsd:element ref="containers" minOccurs="1" />
          </xsd:sequence>
          <!--Коммон-блок-->
          <xsd:sequence>
            <!--Элемент, показывающий, что это коммон-блок-->
            <xsd:element name="commonblockdata"/>
            <!--Список ассоциированных элементов-->
            <xsd:element ref="elements" minOccurs="0" />
            <!--Обратная связь с родительским элементом-->
            <xsd:element ref="containers" minOccurs="1" />
          </xsd:sequence>
        </xsd:choice>
      </xsd:sequence>
      <xsd:attribute name="id" type="apiId" use="required" />
    </xsd:complexType>
  </xsd:element>
  <!--Элемент, явно указывающий, что это модуль-->
  <xsd:element name="moduledata">
    <xsd:complexType>
      <!--Встроенный модуль, или пользовательский-->
      <xsd:attribute name="intrinsic" type="xsd:boolean" use="optional"/>
    </xsd:complexType>
  </xsd:element>
  <!--Импорты других модулей. Вообще говоря, они имеют силу
  только для модулей. Т.к. использование их внутри процедур,
  равносильно использованию полностью квалифицированных элементов из них.
  Использование модулей представляет собой импортирование всего их содержимого,
  включая все цепочки их импортов.-->
  <xsd:element name="imports">
    <xsd:complexType>
      <xsd:sequence>
        <!--Описание импорта одного модуля-->
        <!--Вариант с модификатором only не рассматривается,
        так как формально эквивалентен добавлению простых ссылок в список
        элементов модуля.
        Рассматривается вариант импорта всего содержимого модуля,
        с учётом возможного переименования импортируемых элементов.-->
        <xsd:element name="import" maxOccurs="unbounded">
          <xsd:complexType>
            <xsd:sequence>
              <!--Далее идёт список переименований.
              Буквально это перечисление элементов с новыми именами внутри данного модуля.-->
              <xsd:element ref="elements" minOccurs="0"/>
            </xsd:sequence>
            <!--Имя модуля-->
            <xsd:attribute name="api" type="xsd:string" use="required"/>
            <!--Встроенный? Варианты - встроенный, невстроенный, любой - если элемент не за-
дан-->
            <xsd:attribute name="intrinsic" type="xsd:boolean" use="optional"/>
          </xsd:complexType>
        </xsd:element>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
  <!--Элемент, явно говорящий, что это ссылка-->
  <xsd:element name="referencedata">
    <xsd:complexType>
      <!--Собственно цель ссылки.
          Полностью квалифицированное оригинальное имя.-->
      <xsd:attribute name="api"/>
    </xsd:complexType>
  </xsd:element>
  <!--Ссылка может указывать в том числе и на функцию.
```

```
Соответствует описанию переменной, с атрибутами intrinsic, или external-->
<xsd:element name="functionref">
  <xsd:complexType>
    <xsd:sequence>
      <!--Тип возвращаемого значения функции-->
      <xsd:element ref="type" />
      <!--Опционально, размерность массива - если функция возвращает массив чего-либо.
      Для скаляров не указывается.-->
      <xsd:element ref="dimension" minOccurs="0"/>
    </xsd:sequence>
    <xsd:attribute name="intrinsic" type="xsd:boolean" use="optional"/>
    <xsd:attribute name="external" type="xsd:boolean" use="optional"/>
  </xsd:complexType>
</xsd:element>
<!--Элемент, явно указывающий, что это переменная, а заодно и уточняющий,
    что это за переменная.-->
<xsd:element name="variabledata">
  <xsd:complexType>
    <!--Здесь указываются наиболее общие, простые атрибуты-->
    <!--Переменная – это именованная константа-->
    <xsd:attribute name="parameter" type="xsd:boolean" use="optional"/>
    <!--Переменная – это член перечисления, т.е. именованная целочисленная константа-->
    <xsd:attribute name="enumerator" type="xsd:boolean" use="optional"/>
    <!--Видимость: public/private. Для локальных переменных и аргументов не имеет смысла.-->
    <xsd:attribute name="visibility" type="visibilityType" use="optional"/>
    <!--Переменная размещаемая-->
    <xsd:attribute name="allocatable" type="xsd:boolean" use="optional"/>
    <!--Переменная используется в асинхронном вводе/выводе.-->
    <xsd:attribute name="asynchronous" type="xsd:boolean" use="optional"/>
    <!--Переменная используется как указатель на другую переменную.-->
    <xsd:attribute name="pointer" type="xsd:boolean" use="optional"/>
    <!--Переменная является защищённой. Атрибут применим только для модульных переменных.-->
    <xsd:attribute name="protected" type="xsd:boolean" use="optional"/>
    <!--Переменная сохраняет своё состояние между входом/выходом из контекста-->
    <xsd:attribute name="save" type="xsd:boolean" use="optional"/>
    <!--Атрибут указывает, что значение переменной может изменяться
        в терминах отличных от терминов Fortran-->
    <xsd:attribute name="volatile" type="xsd:boolean" use="optional"/>
  </xsd:complexType>
</xsd:element>
<!--Элемент явно указывает, что значимый элемент (переменная или процедура)
является полем производного типа данных-->
<xsd:element name="memberdata">
  <!--Место зарезервировано для дальнейших описаний -->
</xsd:element>
<!--Атрибут размерности переменной. Используется для задания массивов-->
<xsd:element name="dimension">
  <xsd:complexType>
    <xsd:sequence>
      <!--Каждый элемент dim - это одна из размерностей массива, определяемых
      слева направо. Значение элемента - это значение размерности.
      Количество элементов dim определяют размерность массива-->
      <xsd:element name="dim" minOccurs="0" maxOccurs="unbounded" type="xsd:string"/>
    </xsd:sequence>
    <!--размерность массива-->
    <xsd:attribute name="rank" type="xsd:positiveInteger" use="required"/>
  </xsd:complexType>
</xsd:element>
<!--Тип данных для атрибута BIND-->
<xsd:complexType name="bindingType">
  <!--Целевой язык. По умолчанию - C-->
  <xsd:attribute name="lang" type="xsd:string" default="c"/>
  <!--Внешнее имя в межъязыковом взаимодействии-->
  <xsd:attribute name="name" type="xsd:string" use="optional"/>
</xsd:complexType>
```

```
  <!--Элемент, явно говорящий, что это тип данных-->
  <xsd:element name="typedata">
    <xsd:complexType>
      <xsd:complexContent>
        <xsd:restriction base="xsd:anyType">
          <!--Здесь указываются наиболее общие, простые атрибуты-->
          <!--Это определение встроенного типа данных. Например, для описания базовой библио-
теки языка-->
          <xsd:attribute name="buildin" type="xsd:boolean" use="optional"/>
          <!--Тип данных - является простой структурой (не участвует в операции наследования)-
->
          <xsd:attribute name="sequence" type="xsd:boolean" use="optional"/>
          <!--Тип данных является абстрактным, участвует в наследовании,
          его экземпляры нельзя создавать напрямую-->
          <xsd:attribute name="abstract" type="xsd:boolean" use="optional"/>
          <!--Видимость: public/private. Для локальных типов данных не имеет смысла.-->
          <xsd:attribute name="visibility" type="visibilityType" use="optional"/>
        </xsd:restriction>
      </xsd:complexContent>
    </xsd:complexType>
  </xsd:element>
  <!--Описание параметров производного типа данных-->
  <xsd:element name="typeparams">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="typeparam" minOccurs="0" maxOccurs="unbounded"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
  <!--Описание одного параметра, типа данных-->
  <xsd:element name="typeparam">
    <xsd:complexType>
      <xsd:sequence>
        <!—Тип данных параметра: integer(...)-->
        <xsd:element ref="type"/>
        <!--Значение по умолчанию. Может и отсутствовать-->
        <xsd:element name="value" type="xsd:string" minOccurs="0"/>
      </xsd:sequence>
      <!--Имя параметра-->
      <xsd:attribute name="name" type="xsd:string" use="required" />
      <!--Тип параметра-->
      <xsd:attribute name="kind" type="typeParamKinds" use="required"/>
    </xsd:complexType>
  </xsd:element>
  <!--Процедуры, интерфейсы, подпрограммы и функции-->
  <!--Элемент, явно говорящий, что это процедура. -->
  <xsd:element name="proceduredata">
    <xsd:complexType>
      <!--Здесь указываются наиболее общие, простые атрибуты-->
      <!--Видимость: public/private. Для локальных процедур и аргументов не имеет смысла.-->
      <xsd:attribute name="visibility" type="visibilityType" use="optional"/>
      <!--Процедура используется как указатель на другую переменную.-->
      <xsd:attribute name="pointer" type="xsd:boolean" use="optional"/>
      <!--Процедура является защищённой. Атрибут применим только для модульных процедур.-->
      <xsd:attribute name="protected" type="xsd:boolean" use="optional"/>
      <!--Процедура сохраняет своё состояние между входом/выходом из контекста-->
      <xsd:attribute name="save" type="xsd:boolean" use="optional"/>
      <!--Далее опционально ссылка на процедуру или начальное значение.-->
      <xsd:attribute name="value" type="xsd:string" use="optional"/>
    </xsd:complexType>
  </xsd:element>
  <!--Тип данных для описания интерфейса процедуры-->
  <xsd:complexType name="ProcedureInterface">
    <xsd:choice>
      <!--Либо имя другой процедуры, на которую ссылается данная процедура-->
```

```
    <xsd:element name="name">
      <xsd:complexType>
        <xsd:attribute name="api" type="xsd:string"/>
      </xsd:complexType>
    </xsd:element>
    <!--Либо тип её возвращаемого значения для ссылки на функцию-->
    <xsd:element ref="type"/>
  </xsd:choice>
</xsd:complexType>
<!--Описание типа данных для передачи экземпляра типа данных,
для которого была вызвана процедура, через оператор %,
в связанную с этим типом данных процедуру-->
<xsd:complexType name="PassageType">
  <!--Режим передачи: Pass - передавать (по умолчанию), NoPass - не передавать-->
  <xsd:attribute name="mode" type="PassMode" use="optional" default="pass"/>
  <!--Если режим Pass - то дополнительно можно указать имя аргумента, в котором
  будет передан экземпляр типа.-->
  <xsd:attribute name="name" type="xsd:string" use="optional"/>
</xsd:complexType>
<!--Тип данных, для описания значений атрибута pass-->
<xsd:simpleType name="PassMode">
  <xsd:restriction base="xsd:string">
    <!--Передавать экземпляр типа в качестве одного из параметров в
    процедуру, присутствующую в типе данных (по умолчанию)-->
    <xsd:enumeration value="pass"/>
    <!--Не передавать экземпляр типа в качестве одного из параметров-->
    <xsd:enumeration value="nopass"/>
  </xsd:restriction>
</xsd:simpleType>
<!--Описание связанных с типом процедур-->
<!--Данный тэг говорит, что процедура связана с типом данных-->
<xsd:element name="typebound">
  <xsd:complexType>
    <!--Процедура не должна перечисляться в списке полей типа данных.
        Только для связанных с типом процедур.-->
    <xsd:attribute name="invisible" type="xsd:boolean" use="optional"/>
  </xsd:complexType>
</xsd:element>
<!--Данный тэг говорит, что это специфичная процедура-->
<xsd:element name="specific">
  <xsd:complexType>
    <!--Здесь описываются атрибуты, специфичные для этого типа процедуры-->
    <!--Процедура не переопределяется. Т.е. это её окончательная версия-->
    <xsd:attribute name="nonoverridable" type="xsd:boolean" use="optional"/>
    <!--Процедура отложена, применяется в абстрактных типах данных-->
    <xsd:attribute name="deferred" type="xsd:boolean" use="optional"/>
  </xsd:complexType>
</xsd:element>
<!--Данный тэг говорит, что это обобщённая процедура-->
<xsd:element name="generic">
  <xsd:complexType>
    <!--Это оператор?-->
    <xsd:attribute name="operator" type="xsd:boolean" use="optional"/>
    <!--Это оператор присваивания?-->
    <xsd:attribute name="assignment" type="xsd:boolean" use="optional"/>
  </xsd:complexType>
</xsd:element>
<!--список имён специфичных процедур, составляющих данную процедуру-->
<xsd:complexType name="GenericItemsType">
  <xsd:sequence>
    <!--Описание одного элемента (имени специфичной процедуры)-->
    <xsd:element name="item" minOccurs="1" maxOccurs="unbounded">
      <xsd:complexType>
        <!--Собственно имя процедуры-->
        <xsd:attribute name="name" type="xsd:string" use="required"/>
```

```
      </xsd:complexType>
    </xsd:element>
  </xsd:sequence>
</xsd:complexType>
<!--Данный тэг говорит, что это финальная процедура-->
<xsd:element name="final">
  <!--Зарезервировано для будущего использования-->
</xsd:element>
<!--Данные интерфейса-->
<xsd:element name="interfacedata">
  <xsd:complexType>
    <!--Здесь описание атрибутов интерфейса-->
    <!--Видимость: public/private. Для локальных интерфейсов не имеет смысла.-->
    <xsd:attribute name="visibility" type="visibilityType" use="optional"/>
    <!--Это оператор?-->
    <xsd:attribute name="operator" type="xsd:boolean" use="optional"/>
    <!--Это оператор присваивания?-->
    <xsd:attribute name="assignment" type="xsd:boolean" use="optional"/>
  </xsd:complexType>
</xsd:element>
<!--Типы операторов ввода/вывода-->
<xsd:simpleType name="IOMode">
  <xsd:restriction base="xsd:string">
    <!--Оператор вывода-->
    <xsd:enumeration value="write"/>
    <!--Оператор ввода-->
    <xsd:enumeration value="read"/>
  </xsd:restriction>
</xsd:simpleType>
<!--Режим ввода/вывода-->
<xsd:simpleType name="IOFormatMode">
  <xsd:restriction base="xsd:string">
    <!--Форматированный ввод/вывод-->
    <xsd:enumeration value="formatted"/>
    <!--Неформатированный ввод/вывод-->
    <xsd:enumeration value="unformatted"/>
  </xsd:restriction>
</xsd:simpleType>
<!--Тип данных для атрибута io-->
<xsd:complexType name="IOFormatType">
  <!--Режим ввод/вывод-->
  <xsd:attribute name="mode" type="IOMode" use ="required"/>
  <!--Модификатор - форматированный/неформатированный-->
  <xsd:attribute name="format" type="IOFormatMode" use="required"/>
</xsd:complexType>
<!--Тип данных, описывающий ссылку дополнительной точки входа.
Здесь устанавливается ссылка на оригинальную процедуру.-->
<xsd:complexType name="EntryPointType">
  <!--Имя родительской (оригинальной) процедуры-->
  <xsd:attribute name="ref" type="xsd:string" use="required"/>
</xsd:complexType>
<!--Так как и подпрограммы, и функции являются методами, то они разделяют общие свойства.-->
<xsd:element name="methoddata">
  <xsd:complexType>
    <xsd:attributeGroup ref="MethodAttributes"/>
  </xsd:complexType>
</xsd:element>
<!--Атрибуты подпрограмм и функций-->
<xsd:attributeGroup name="MethodAttributes">
  <!--Эта подпрограмма описана в блоке интерфейса-->
  <xsd:attribute name="interfaced" type="xsd:boolean" use="optional"/>
  <!--Эта подпрограмма описана в блоке абстрактного интерфейса-->
  <xsd:attribute name="abstract" type="xsd:boolean" use="optional"/>
  <!--Эта подпрограмма не имеет побочных эффектов (pure from side effects)-->
  <xsd:attribute name="pure" type="xsd:boolean" use="optional"/>
```

```
    <!--Эта подпрограмма предназначена для поэлементной обработки массивов.
    Все элементарные процедуры по определению являются чистыми.-->
    <xsd:attribute name="elemental" type="xsd:boolean" use="optional"/>
    <!--Подпрограмма является рекурсивной-->
    <xsd:attribute name="recursive" type="xsd:boolean" use="optional"/>
    <!--Видимость: public/private. Для локальных подпрограмм не имеет смысла.-->
    <xsd:attribute name="visibility" type="visibilityType" use="optional"/>
  </xsd:attributeGroup>
  <!--Элемент говорит, что это подпрограмма-->
  <xsd:element name="subroutinedata">
    <xsd:complexType>
      <!--Это главная программа-->
      <xsd:attribute name="main" type="xsd:boolean" use="optional"/>
    </xsd:complexType>
  </xsd:element>
  <!--Элемент говорит, что это функция-->
  <xsd:element name="functiondata">
    <xsd:complexType>
      <!—Зарезервировано для будущего использования-->
    </xsd:complexType>
  </xsd:element>

  <!--Описание namelist-а для ввода/вывода.-->
  <!--Элемент, показывающий, что это список имён-->
  <xsd:element name="namelistdata">
    <xsd:complexType>
      <!--Видимость: public/private. Для локальных списков имён не имеет смысла.-->
      <xsd:attribute name="visibility" type="visibilityType" use="optional"/>
    </xsd:complexType>
  </xsd:element>
  <!--Описание коммон-блока.-->
  <!--Элемент, показывающий что это список имён-->
  <xsd:element name="commonblockdata">
    <xsd:complexType>
     <!--Атрибут говорит, что содержимое common-блока должно сохраняться в памяти постоянно-->
      <xsd:attribute name="save" type="xsd:boolean" use="optional"/>
    </xsd:complexType>
  </xsd:element>
  <!--Группа описывает все внутренние элементы глобальной области видимости или модуля.-->
  <xsd:group name="memberContent">
    <xsd:sequence>
      <!--Информация об элементе: его имя, группа, подгруппа и т.п.-->
      <xsd:element name="apidata" minOccurs="1" />
      <!--Здесь может быть всё, кроме модуля и глобальной области видимости-->
      <xsd:choice>
        <!--Ссылка-->
        <xsd:sequence>
          <!--Элемент, явно говорящий, что это ссылка-->
          <xsd:element ref="referencedata"/>
          <!--Ссылка может указывать в том числе и на функцию.
            Тогда вводим дополнительный элемент.-->
          <xsd:element ref="functionref" minOccurs="0"/>
          <!--Служит для обратной связи с элементом, внутри которого содержится
            данный элемент.-->
          <xsd:element ref="containers"  minOccurs="1" />
        </xsd:sequence>
        <!--Тип данных-->
        <xsd:sequence>
          <!--Далее элемент, явно говорящий, что это тип данных-->
          <xsd:element ref="typedata"/>
          <!--Опционально, имя типа данных во внешнем контексте. Для межъязыкового взаимодей-
ствия.-->
          <xsd:element name="bind" type="bindingType" minOccurs="0"/>
          <!--Для наследования. Указывается базовый тип.-->
          <xsd:element ref="family" minOccurs="0" />
```

```
            <!--Параметры типа, если есть-->
            <xsd:element ref="typeparams" minOccurs="0"/>
            <!--Элементы типа данных-->
            <xsd:element ref="elements" minOccurs="0" />
            <!--Обратная связь с родительским элементом-->
            <xsd:element ref="containers" minOccurs="1" />
          </xsd:sequence>
          <!--Переменная, в том числе и поле типа данных-->
          <xsd:sequence>
            <!--Элемент, явно указывающий, что это переменная, а заодно и уточняющий,
              что это за переменная.-->
            <xsd:element ref="variabledata"/>
            <!--Опциональный тег, указывает, что переменная является
              членом типа данных-->
            <xsd:element name="memberdata" minOccurs="0"/>
            <!--Тип данных переменной-->
            <xsd:element ref="type" />
            <!--Опционально, размерность массива. Для скаляров не указывается.-->
            <xsd:element ref="dimension" minOccurs="0"/>
            <!--Опционально, имя переменной во внешнем контексте. Для межъязыкового взаимодейст-
вия.-->
            <xsd:element name="bind" type="bindingType" minOccurs="0"/>
            <!--Опционально, начальное значение переменной.
              Для полей производного типа данных - это значение по умолчанию.-->
            <xsd:element name="value" type="xsd:anyType" minOccurs="0"/>
            <!--Служит для обратной связи с элементом, внутри которого содержится
              данный элемент.-->
            <xsd:element ref="containers"  minOccurs="1" />
          </xsd:sequence>
          <!--Процедура, в том числе и поле типа данных-->
          <xsd:sequence>
            <!--Элемент, явно говорящий, что это процедура-->
            <xsd:element ref="proceduredata"/>
            <!--Если это поле типа данных то указывается этот тэг-->
            <xsd:element ref="memberdata" minOccurs="0"/>
            <!--Опциональный атрибут - интерфейс-->
            <xsd:element name="interface" type="ProcedureInterface" minOccurs="0"/>
            <!--Дополнительные атрибуты, которые являются составными элементами-->
            <!--Опционально, имя процедуры во внешнем контексте. Для межъязыкового взаимодейст-
вия.
              Не может быть указан для процедуры - поля типа данных.-->
            <xsd:element name="bind" type="bindingType" minOccurs="0"/>
            <!--Только для связанных с типом данных процедур.
              Атрибут передачи экземпляра типа в процедуру-->
            <xsd:element name="pass" type="PassageType" minOccurs="0"/>
          </xsd:sequence>
          <!--Связанные с типом данных процедуры-->
          <xsd:sequence>
            <!--Данный тэг говорит, что процедура связана с типом данных-->
            <xsd:element ref="typebound" minOccurs="1"/>
            <!--Здесь выбор типа процедуры: специфичная, обобщённая, финальная-->
            <xsd:choice>
              <!--Специфичная-->
              <xsd:sequence>
                <!--Данный тэг говорит, что это специфичная процедура-->
                <xsd:element ref="specific" minOccurs="1"/>
                <!--Элемент, явно говорящий, что это процедура-->
                <xsd:element ref="proceduredata"/>
                <!--Опциональный атрибут - интерфейс-->
                <xsd:element name="interface" type="ProcedureInterface" minOccurs="0"/>
                <!--Дополнительные атрибуты, которые являются составными элементами-->
                <!--Только для связанных с типом данных процедур.
                  Атрибут передачи экземпляра типа в процедуру-->
                <xsd:element name="pass" type="PassageType" minOccurs="0"/>
              </xsd:sequence>
```

```
            <!--Обобщённая-->
            <xsd:sequence>
              <!--Данный тэг говорит, что это обобщённая процедура-->
              <xsd:element ref="generic" minOccurs="1"/>
              <!--Элемент, явно говорящий, что это процедура-->
              <xsd:element ref="proceduredata"/>
              <!--Это оператор ввода/вывода-->
              <xsd:element name="iooperator" type="IOFormatType" minOccurs="0"/>
              <!--Далее список имён специфичных процедур, составляющих данную процедуру-->
              <xsd:element name="items" type="GenericItemsType" minOccurs="1"/>
            </xsd:sequence>
            <!--Финальная-->
            <xsd:sequence>
              <!--Данный тэг говорит, что это финальная процедура-->
              <xsd:element ref="final" minOccurs="1"/>
              <!--Элемент, явно говорящий, что это процедура-->
              <xsd:element ref="proceduredata"/>
            </xsd:sequence>
          </xsd:choice>
          <!--Обратная связь с родительским элементом-->
          <xsd:element ref="containers" minOccurs="1" />
        </xsd:sequence>
        <!--Обобщённый интерфейс-->
        <xsd:sequence>
          <!--Говорим, что это интерфейс-->
          <xsd:element ref="interfacedata"/>
          <!--Это оператор ввода/вывода-->
          <xsd:element name="iooperator" type="IOFormatType" minOccurs="0"/>
          <!--Список элементов интерфейса-->
          <xsd:element ref="elements" minOccurs="0" />
          <!--Обратная связь с родительским элементом-->
          <xsd:element ref="containers" minOccurs="1" />
        </xsd:sequence>
        <!--Подпрограмма и функция-->
        <xsd:sequence>
          <!--Указываем, что это метод-->
          <xsd:element ref="methoddata"/>
          <!--Далее выбор: подпрограмма или функция-->
          <xsd:choice>
            <!--Подпрограмма-->
            <xsd:sequence>
              <!--Указываем, что это подпрограмма-->
              <xsd:element ref="subroutinedata"/>
              <!--Дополнительные сложные атрибуты-->
              <!--Данная подпрограмма является дополнительной точкой входа в другую подпро-
грамму-->
              <xsd:element name="entry" type="EntryPointType" minOccurs="0"/>
              <!--Опционально, имя процедуры во внешнем контексте. Для межъязыкового взаимо-
действия-->
              <xsd:element name="bind" type="bindingType" minOccurs="0"/>
              <!--Собственно описание внутреннего устройства процедуры-->
              <!--Импорты других модулей. Могут и не быть.-->
              <xsd:element ref="imports" minOccurs="0"/>
              <!--Описание формальных аргументов-->
              <xsd:element ref="parameters" minOccurs="0" />
              <!--Далее элементы подпрограммы: переменные, процедуры, типы данных, интерфейсы,
подпрограммы и функции.-->
              <xsd:element ref="elements" minOccurs="0" />
            </xsd:sequence>
            <!--Функция-->
            <xsd:sequence>
              <!--Указываем, что это функция-->
              <xsd:element ref="functiondata"/>
              <!--Дополнительные сложные атрибуты-->
              <!--Данная функция является дополнительной точкой входа в другую подпрограмму-->
```

```
            <xsd:element name="entry" type="EntryPointType" minOccurs="0"/>
            <!--Опционально, имя процедуры во внешнем контексте. Для межъязыкового взаимо-
действия-->
            <xsd:element name="bind" type="bindingType" minOccurs="0"/>
            <!--Собственно описание внутреннего устройства процедуры-->
            <!--Импорты других модулей. Могут и не быть.-->
            <xsd:element ref="imports" minOccurs="0"/>
            <!--Описание формальных аргументов-->
            <xsd:element ref="parameters" minOccurs="0" />
            <!--Описание возвращаемого значения-->
            <xsd:element ref="result" />
            <!--Далее элементы подпрограммы: переменные, процедуры, типы данных, интерфейсы,
подпрограммы и функции.-->
            <xsd:element ref="elements" minOccurs="0" />
          </xsd:sequence>
        </xsd:choice>
        <!--Обратная связь с родительским элементом-->
        <xsd:element ref="containers" minOccurs="1" />
      </xsd:sequence>
      <!--Namelist-->
      <xsd:sequence>
        <!--Элемент, показывающий, что это список имён-->
        <xsd:element ref="namelistdata"/>
        <!--Список ассоциированных элементов-->
        <xsd:element ref="elements" minOccurs="0" />
        <!--Обратная связь с родительским элементом-->
        <xsd:element ref="containers" minOccurs="1" />
      </xsd:sequence>
      <!--Коммон-блок-->
      <xsd:sequence>
        <!--Элемент, показывающий, что это коммон-блок-->
        <xsd:element name="commonblockdata"/>
        <!--Список ассоциированных элементов-->
        <xsd:element ref="elements" minOccurs="0" />
        <!--Обратная связь с родительским элементом-->
        <xsd:element ref="containers" minOccurs="1" />
      </xsd:sequence>
    </xsd:choice>
  </xsd:sequence>
</xsd:group>
<!--Дополнительная информация об API используется для описания любых элементов.-->
<xsd:complexType name="optionalApiInformation">
  <xsd:choice minOccurs="0">
    <!--Информация о внутренних элементах-->
    <xsd:group ref="memberContent" />
  </xsd:choice>
</xsd:complexType>
<!--Описание информации об элементе-->
<xsd:element name="apidata">
  <xsd:complexType>
    <!--Имя элемента-->
    <xsd:attribute name="name" type="xsd:string" use="required" />
    <!--Основная группа элемента-->
    <xsd:attribute name="group" type="apiGroupType" use="required" />
    <!--Первая уточняющая подгруппа элемента-->
    <xsd:attribute name="subgroup" type="apiSubgroupType" use="optional" />
    <!--Вторая уточняющая подгруппа элемента-->
    <xsd:attribute name="subsubgroup" type="apiSubsubgroupType" use="optional" />
  </xsd:complexType>
</xsd:element>
<!--Описание формальных аргументов и возвращаемых значений-->
<xsd:element name="parameters">
  <xsd:complexType>
    <xsd:sequence>
      <!--Список параметров-->
```

```
        <xsd:element ref="parameter" minOccurs="0" maxOccurs="unbounded" />
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
  <!--Описание параметра подпрограммы/функции. Это может быть: переменная, процедура, подпро-
грамма или функция. -->
  <xsd:element name="parameter">
    <xsd:complexType>
      <!--Выбор: переменная, процедура, подпрограмма, функция-->
      <xsd:sequence>
        <xsd:group ref="ParameterChoiseGroup" minOccurs="1" />
      </xsd:sequence>
      <!--Имя параметра-->
      <xsd:attribute name="name" type="xsd:string" use="required" />
      <!--Указывает метод передачи аргументов-->
      <xsd:attribute name="intent" type="intentType" use="optional"/>
      <!--Указывает, что аргумент не обязательный-->
      <xsd:attribute name="optional" type="xsd:boolean" use="optional"/>
      <!--Указывает, что аргумент передаётся по значению-->
      <xsd:attribute name="value" type="xsd:boolean" use="optional"/>
    </xsd:complexType>
  </xsd:element>
  <!--Описание результирующего значения функции-->
  <xsd:element name="result">
    <xsd:complexType>
      <xsd:group ref="ResultChoiseGroup"/>
      <!--Имя результата. Fortran допускает указывать любую переменную
      в качестве результирующего значения функции. По умолчанию используется
      переменная, совпадающая по имени с именем функции.-->
      <xsd:attribute name="name" type="xsd:string" use="required" />
    </xsd:complexType>
  </xsd:element>
  <!--Группа элементов, которые могут являться возвращаемыми значениями функции-->
  <xsd:group name="ResultChoiseGroup">
    <!--Базовый выбор: переменная или процедура.-->
    <xsd:choice>
      <!--Переменная-->
      <xsd:sequence>
        <!--Элемент, явно указывающий, что это переменная, а заодно и уточняющий,
        что это за переменная. Здесь ссылка, т.к. используется в 3-х местах.-->
        <xsd:element ref="variabledata"/>
        <!--Тип данных переменной-->
        <xsd:element ref="type" />
        <!--Опционально, размерность массива. Для скаляров не указывается.-->
        <xsd:element ref="dimension" minOccurs="0"/>
      </xsd:sequence>
      <!--Процедура-->
      <xsd:sequence>
        <!--Элемент, явно говорящий, что это процедура-->
        <xsd:element ref="proceduredata"/>
        <!--Опциональный атрибут - интерфейс-->
        <xsd:element name="interface" type="ProcedureInterface" minOccurs="0"/>
        <!--Дополнительные атрибуты, которые являются составными элементами-->
        <!--Опционально, имя процедуры во внешнем контексте. Для межъязыкового
взаимодействия.-->
        <xsd:element name="bind" type="bindingType" minOccurs="0"/>
      </xsd:sequence>
    </xsd:choice>
  </xsd:group>
  <!--Группа элементов, которые могут являться аргументами подпрограмм и функций.-->
  <xsd:group name="ParameterChoiseGroup">
    <!--Это объединение:
    Либо один из элементов группы результирующего значения.
    Либо подпрограмма или функция.-->
    <xsd:choice>
```

```
      <!--Базовый выбор: переменная или процедура.-->
      <xsd:group ref="ResultChoiseGroup"/>
      <!--Дополнительные варианты выбора для параметров-->
      <xsd:sequence>
        <!--Указываем, что это метод. Поскольку и подпрограмма, и функция - это методы-->
        <xsd:element ref="methoddata"/>
        <!--Далее специализированный выбор: подпрограмма или функция.-->
        <xsd:choice>
          <!--Подпрограмма-->
          <xsd:sequence>
            <!--Указываем, что это подпрограмма-->
            <xsd:element ref="subroutinedata"/>
            <!--Дополнительные сложные атрибуты-->
            <!--Данная подпрограмма является дополнительной точкой входа в другую подпрограм-
му-->
            <xsd:element name="entry" type="EntryPointType" minOccurs="0"/>
            <!--Опционально, имя процедуры во внешнем контексте. Для межъязыкового взаимодей-
ствия-->
            <xsd:element name="bind" type="bindingType" minOccurs="0"/>
            <!--Собственно описание внутреннего устройства процедуры-->
            <!--Импорты других модулей. Могут и не быть.-->
            <xsd:element ref="imports" minOccurs="0"/>
            <!--Описание формальных аргументов-->
            <xsd:element ref="parameters" minOccurs="0" />
            <!--Далее элементы подпрограммы: переменные, процедуры, типы данных, интерфейсы,
подпрограммы и функции.-->
            <xsd:element ref="elements" minOccurs="0" />
          </xsd:sequence>
          <!--Функция-->
          <xsd:sequence>
            <!--Указываем, что это функция-->
            <xsd:element ref="functiondata"/>
            <!--Дополнительные сложные атрибуты-->
            <!--Данная функция является дополнительной точкой входа в другую подпрограмму-->
            <xsd:element name="entry" type="EntryPointType" minOccurs="0"/>
            <!--Опционально, имя процедуры во внешнем контексте. Для межъязыкового взаимодей-
ствия-->
            <xsd:element name="bind" type="bindingType" minOccurs="0"/>
            <!--Собственно описание внутреннего устройства процедуры-->
            <!--Импорты других модулей. Могут и не быть.-->
            <xsd:element ref="imports" minOccurs="0"/>
            <!--Описание формальных аргументов-->
            <xsd:element ref="parameters" minOccurs="0" />
            <!--Описание возвращаемого значения-->
            <xsd:element ref="result" />
            <!--Далее элементы подпрограммы: переменные, процедуры, типы данных, интерфейсы,
подпрограммы и функции.-->
            <xsd:element ref="elements" minOccurs="0" />
          </xsd:sequence>
        </xsd:choice>
      </xsd:sequence>
    </xsd:choice>
  </xsd:group>
  <!--Описание наследования-->
  <!--Базовый блок информации о наследовании-->
  <xsd:element name="family">
    <xsd:complexType>
      <xsd:sequence>
        <!--Базовые типы-->
        <xsd:element ref="ancestors" minOccurs="0" />
        <!--Потомки (наследники данного типа)-->
        <xsd:element ref="descendents" minOccurs="0" />
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
```

```
<!--Базовые типы-->
<xsd:element name="ancestors">
  <xsd:complexType>
    <xsd:sequence>
      <!--Описание типа-->
      <xsd:element ref="type" minOccurs="0" maxOccurs="unbounded" />
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
<!--Потомки (наследники данного типа)-->
<xsd:element name="descendents">
  <xsd:complexType>
    <xsd:sequence>
      <!--Описание типа-->
      <xsd:element ref="type" minOccurs="0" maxOccurs="unbounded" />
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
<!--Описание контейнеров (ссылок на базовые элементы)-->
<xsd:element name="containers">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="library" minOccurs="1" maxOccurs="1" />
      <xsd:element name="element" minOccurs="0" maxOccurs="1" >
        <xsd:complexType>
          <!--Ссылка на API, если API будет описано далее-->
          <xsd:attribute name="api" type="apiId" use="optional" />
        </xsd:complexType>
      </xsd:element>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
<!--Ссылка на библиотеку-->
<xsd:element name="library">
  <xsd:complexType>
    <!--Имя библиотеки-->
    <xsd:attribute name="assembly" type="xsd:string" />
  </xsd:complexType>
</xsd:element>
<!--Описание элементов-->
<xsd:element name="elements">
  <xsd:complexType>
    <xsd:sequence>
      <!--Последовательность элементов-->
      <xsd:element ref="element" minOccurs="0" maxOccurs="unbounded" />
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
<!--Описание элемента-->
<xsd:element name="element">
  <xsd:complexType>
    <xsd:complexContent>
      <!--Это расширение базового типа опциональной информации.
      Сделано для того, чтобы можно было не только делать ссылку на API,
      которое будет описано в дальнейшем, но и сразу же включать описание
      внутрь данного элемента-->
      <xsd:extension base="optionalApiInformation">
        <!--Ссылка на API, если API будет описано далее-->
        <xsd:attribute name="api" type="apiId" use="optional" />
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>
</xsd:element>
<!--Описание ссылок, например на типы данных-->
```

```
<!--Описание ссылки на тип данных-->
<xsd:element name="type">
  <xsd:complexType>
    <xsd:sequence>
      <!--Перечень параметров типа данных, если есть-->
      <xsd:element name="param" minOccurs="0" maxOccurs="unbounded">
        <xsd:complexType>
          <xsd:simpleContent>
            <!--Значение параметра - строка-->
            <xsd:extension base="xsd:string">
              <!--Имя параметра-->
              <xsd:attribute name="name" type="xsd:string" use="optional"/>
            </xsd:extension>
          </xsd:simpleContent>
        </xsd:complexType>
      </xsd:element>
    </xsd:sequence>
    <!--Полноквалифицированное имя типа данных, или *-->
    <xsd:attribute name="api" type="typeId" use="required" />
    <!--Классификатор типа: builin - встроенный, type - производный, class - полиморфный-->
    <xsd:attribute name="kind" type="typeKinds" use="optional" default="buildin"/>
  </xsd:complexType>
</xsd:element>
<!--Простые типы данных, используемые в схеме-->
<!--Тип параметра, производного типа данных-->
<xsd:simpleType name="typeParamKinds">
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="kind">
      <xsd:annotation>
        <xsd:documentation>
          Параметр типа размерности
        </xsd:documentation>
      </xsd:annotation>
    </xsd:enumeration>
    <xsd:enumeration value="len">
      <xsd:annotation>
        <xsd:documentation>
          Парамерт типа длинны
        </xsd:documentation>
      </xsd:annotation>
    </xsd:enumeration>
  </xsd:restriction>
</xsd:simpleType>
<!--Классификаторы типа данных в ссылках на тип данных-->
<xsd:simpleType name="typeKinds">
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="buildin">
      <xsd:annotation>
        <xsd:documentation>
         Встроенный (по умолчанию).
        </xsd:documentation>
      </xsd:annotation>
    </xsd:enumeration>
    <xsd:enumeration value="type">
      <xsd:annotation>
        <xsd:documentation>
          Производный
        </xsd:documentation>
      </xsd:annotation>
    </xsd:enumeration>
    <xsd:enumeration value="calss">
      <xsd:annotation>
        <xsd:documentation>
          Полиморфный
        </xsd:documentation>
```

```
        </xsd:annotation>
      </xsd:enumeration>
    </xsd:restriction>
  </xsd:simpleType>
  <!--Определение краткого имени для пространства имён-->
  <xsd:simpleType name="namespaceId">
    <xsd:restriction base="xsd:string">
      <xsd:pattern value="N:[_:\w]+(\.[_\w]*)*" />
    </xsd:restriction>
  </xsd:simpleType>
  <!--Определение краткого имени для типа данных-->
  <xsd:simpleType name="typeId">
    <xsd:restriction base="xsd:string">
      <xsd:pattern value="T:.+" />
    </xsd:restriction>
  </xsd:simpleType>
  <!--Определение краткого имени для метода (подпрограммы или функции)-->
  <xsd:simpleType name="methodId">
    <xsd:restriction base="xsd:string">
      <xsd:pattern value="M:.+" />
    </xsd:restriction>
  </xsd:simpleType>
  <!--Определение краткого имени для внутренних элементов-->
  <xsd:simpleType name="memberId">
    <xsd:restriction base="xsd:string">
      <xsd:pattern value="[VIPR]:.+" />
    </xsd:restriction>
  </xsd:simpleType>
<!--Определение краткого имени для common-блоков и namelist-ов-->
  <xsd:simpleType name="storageId">
    <xsd:restriction base="xsd:string">
      <xsd:pattern value="S:.+"/>
    </xsd:restriction>
  </xsd:simpleType>
  <!--Определение любого допустимого краткого имени-->
  <xsd:simpleType name="apiId">
    <xsd:union memberTypes="namespaceId typeId methodId memberId storageId" />
  </xsd:simpleType>
  <!--Описание типа данных основной группы-->
  <xsd:simpleType name="apiGroupType">
    <xsd:restriction base="xsd:string">
      <!--Область видимости или модуль-->
      <xsd:enumeration value="namespace" />
      <!--Тип данных-->
      <xsd:enumeration value="type" />
      <!--Ссылка-->
      <xsd:enumeration value="reference" />
      <!--Переменная-->
      <xsd:enumeration value="variable"/>
      <!--Процедура-->
      <xsd:enumeration value="procedure"/>
      <!--Интерфейс-->
      <xsd:enumeration value="interface"/>
      <!--Метод-->
      <xsd:enumeration value="method"/>
      <!--Ассоциативный блок хранения данных (для namelist и common block)-->
      <xsd:enumeration value="storage"/>
    </xsd:restriction>
  </xsd:simpleType>
  <!--Описание типа данных подгрупп.-->
  <xsd:simpleType name="apiSubgroupType">
    <xsd:restriction base="xsd:string">
      <!--Подгруппа областей имён-->
      <!--Глобальая область видимости-->
      <xsd:enumeration value="global"/>
```

```
      <!--Модуль-->
      <xsd:enumeration value="module"/>
      <!--Подгруппа для полей типа данных-->
      <!--Буквально член типа данных-->
      <xsd:enumeration value="member"/>
      <!--Подгруппа для процедур-->
      <!--Специфичная связанная с типом процедура-->
      <xsd:enumeration value="specific"/>
      <!--Обобщённая связанная с типом процедура-->
      <xsd:enumeration value="generic"/>
      <!--Финальная связанная с типом процедура-->
      <xsd:enumeration value="final"/>
      <!--Подгруппа для интерфейсов-->
      <!--Оператор-->
      <xsd:enumeration value="operator"/>
      <!--Оператор присваивания-->
      <xsd:enumeration value="assign"/>
      <!--Оператор ввода/вывода-->
      <xsd:enumeration value="io"/>
      <!--Подгруппа для методов-->
      <!--Подпрограмма-->
      <xsd:enumeration value="subroutine"/>
      <!--Функция-->
      <xsd:enumeration value="function"/>
      <!--Подгруппа для ассоциативных блоков хранения данных (для namelist и common block)-->
      <!--namelist-->
      <xsd:enumeration value="namelist"/>
      <!--common-block-->
      <xsd:enumeration value="commonblock"/>
    </xsd:restriction>
  </xsd:simpleType>
  <xsd:simpleType name="apiSubsubgroupType">
    <xsd:restriction base="xsd:string">
      <!--Зарезервировано для будущего использования-->
    </xsd:restriction>
  </xsd:simpleType>
  <!--Описание типа видимости элементов вне модуля-->
  <xsd:simpleType name="visibilityType">
    <xsd:restriction base="xsd:string">
      <!--Общедоступный (видим)-->
      <xsd:enumeration value="public" />
      <!--Частный (не видим)-->
      <xsd:enumeration value="private" />
    </xsd:restriction>
  </xsd:simpleType>
  <!--Описание типа передачи аргументов в процедуры-->
  <xsd:simpleType name="intentType">
    <xsd:restriction base="xsd:string">
      <!--Только входной-->
      <xsd:enumeration value="in"/>
      <!--Только выходной-->
      <xsd:enumeration value="out"/>
      <!--Входной и выходной-->
      <xsd:enumeration value="inout"/>
    </xsd:restriction>
  </xsd:simpleType>
</xsd:schema>
```

Приложение Б. Описание основных классов и интерфейсов.

Описание основных классов MPF в части языкового сервиса

Таблица Б.1. Классы MPF для поддержки реализации языкового сервиса

Класс	Описание
Классы для поддержки реализации подсветки синтаксиса	
Source	представляет абстракцию файла исходного кода с точки зрения языкового сервиса. Хранит информацию о редактируемом файле, в частности, содержит экземпляр класса Colorizer, отвечающего за подсветку синтаксиса
Colorizer	реализует интерфейс IVsColorizer. Используется стандартным редактором среды разработки для предоставления подсветки синтаксиса в текущем файле исходного кода. Для большей гибкости и абстракции колорайзера MPF от конкретного языка программирования используется понятие сканера
IScanner	интерфейс сканера. Каждый сканер по своей сути является специализированным лексическим анализатором, который должен уметь сохранять своё текущее состояние и восстанавливать его для продолжения разбора так, как если бы он осуществлял простой линейный разбор потока символов. Методы интерфейса описаны в таблице Б.2.
TokenInfo	описывает **токен** с точки зрения сканера (см. таблицу Б.3)
Классы поддержки реализации работы со сниппетами исходного кода (Code Snippets) [92]	
ExpansionProvider [93]	обеспечивает набор возможностей для выбора, вставки и редактирования сниппета. Сниппет [92]— это шаблон исходного кода с настраиваемыми частями, предназначенный для многократного повторного использования
ExpansionFunction [94]	представляет собой специальную функцию развёртывания, которая может использоваться при вставке сниппета в режиме редактирования. Обеспечивает реализацию базовых функций интерфейса IVsExpansionFunction и позволяет своим наследникам настраивать нужное поведение. Для поддержки функций развёртывания сниппета в языковом сервисе необходимо переопределить метод CreateExpansionFunction
Классы поддержки реализации возможностей технологии IntelliSense	
AuthoringSink [95]	предназначен для накопления информации в процессе разбора файла для поддержки следующих возможностей: • предоставление сообщений об ошибках; • обозначение парных элементов, например открывающей и закрывающей круглых скобок «(», «)»; • функции технологии IntelliSense в части сбора информации о расположении всевозможных идентификаторов в тексте программы; • поддержка скрываемых областей текста, например, для функции визуального выделения структуры кода; • управление содержимым окна отладчика «автоматические переменные»; • проверка допустимости точек останова.
ParseRequest [96]	используется для предоставления информации синтаксическому анализатору о конкретной операции анализа и для возвращения результатов выполнения этой операции
AuthoringScope [97]	абстрактный базовый класс, содержит информацию о файле исходного кода, полученную в результате проведения операции синтаксического анализа. Класс AuthoringScope является центральным местом для предоставления информации для базовых возможностей технологии IntelliSense (таблица Б.4)

Таблица Б.1. Продолжение

ViewFilter [98]	предназначен для обработки различных команд редактирования и команд технологии IntelliSense. Отвечает за инициализацию сессий отображения всевозможных списков и подсказок
TextTipData [99]	предоставляет реализацию по умолчанию интерфейса IVsTextTipData предназначенного для отображения всплывающих подсказок в Visual Studio. Отображает краткие сведения об элементе языка программирования, получаемые в результате вызова метода AuthoringScope.GetDataTipText
Declarations [100]	абстрактный класс, управляет списком определений, которые отображаются в выпадающем списке выбора, для поддержки функций IntelliSense завершения слова и списка элементов сложного объекта. Данный класс представляет собой абстракцию данных для отображения, но непосредственно их отображением он не занимается. Экземпляр класса Declarations возвращается из метода AuthoringScope.GetDeclarations
CompletionSet [101]	представляет список автодополнения (возможность технологии IntelliSense) который непосредственно отображается пользователю; отвечает за обработку событий выбора пользователем какого-либо из элементов списка. Работает с экземпляром класса Declarations
Methods [102]	абстрактный класс, представляет коллекцию сигнатур методов полученных в результате операции разбора в языковом сервисе. Используется для поддержки возможности IntelliSense «Отображение сведений о параметрах процедур» (Parameter Info). Так же, как и Declarations, является абстракцией данных, и не занимается непосредственным отображением сигнатур пользователю. Экземпляр данного класса возвращается методом AuthoringScope.GetMethods
MethodData [103]	предоставляет поддержку для операции отображения сведений о параметрах процедур в рамках технологии IntelliSense. В качестве источника данных для отображения этот класс использует экземпляр класса Methods

Таблица Б.2. Описание интерфейса IScanner

Метод	Описание
void SetSource(string source, int offset)	предназначен для инициализации строки для последующего разбора; здесь параметр offset указывает, с какого символа строки необходимо начинать разбор
bool ScanTokenAndProvideInfoAboutIt(TokenInfo tokenInfo, ref int state)	осуществляет разбор строки и возвращает текущий распознанный токен. Аргумент state является состоянием лексического анализатора, которое нужно использовать для возобновления разбора и сохранять при его завершении. Метод возвращает значение «истина», если токен найден, и «ложь», если в строке больше нет токенов

Таблица Б.3. Описание класса TokenInfo

Свойство	Описание
Color	индекс цвета, которым нужно выделять символы этого токена
StartIndex, EndIndex	задают номер первого и последнего символов, соответствующих лексеме токена в анализируемой строке
Token	идентификатор токена лексического анализатора (может отсутствовать)
Type	код типа токена: комментарий, строка, идентификатор, ключевое слово и т.д.
Trigger	целое число, в котором каждый бит означает наличие определённого действия. Например, в Fortran для доступа к полю или методу класса используется символ «%» (в C# для этой цели используется символ «.»). Поэтому если лексический анализатор встретит данный символ, то он может сделать вывод, что далее пользователь хочет увидеть список всех доступных полей или методов класса. Для этого он устанавливает специальный флаг-триггер, который далее приведёт к отображению списка доступных полей и методов

Таблица Б.4. Описание класса AuthoringScope и его связь с реализацией технологии IntelliSense

Метод	Описание
public abstract string GetDataTipText(int line, int col, out TextSpan span)	возвращает строку — описание элемента языка программирования, находящегося под курсором мыши. Данный метод соответствует функции IntelliSense – отображение кратких сведений об элементе языка программирования (Quick Info)
public abstract Declarations GetDeclarations(IVsTextView view, int line, int col, TokenInfo info, ParseReason reason)	возвращает список определений (объект класса Declarations) конкретизированный в соответствии с причиной разбора. Данный метод соответствует функциям IntelliSense «Построение списка элементов сложного объект» (List Members) и «Завершение слова» или «Автодополнение» (Complete Word)
public abstract Methods GetMethods(int line, int col, string name)	возвращает список сигнатур методов (объект класса Methods) с данным именем, включая их перегруженные версии. Указанный метод соответствует функции IntelliSense – отображение сведений о параметрах процедур (Parameter Info)
public abstract string Goto(VSConstants.VSStd97CmdID cmd, IVsTextView textView, int line, int col, out TextSpan span)	возвращает месторасположение файла, а также позицию в нем (строка, столбец), содержащего определение, объявление или ссылку для текста, находящегося под курсором мыши. Данная возможность относится к расширенному функционалу IntelliSense

Описание классов, реализующих блок интеграции с IDE в языковом сервисе FRIS

Таблица Б.5. Классы блока интеграции с IDE во FRIS

Класс	Описание
Классы для поддержки реализации подсветки синтаксиса	
FortranSource	наследник от класса Source, представляет абстракцию файла исходного кода
FortranColorizerScanner	класс сканера для обеспечения подсветки синтаксиса для языка Fortran. Реализует интерфейс IScanner
Классы поддержки реализации возможностей технологии IntelliSense	
FortranLanguageService	класс-наследник от LanguageService, реализует языковой сервис для языка Fortran
FortranConfiguration	класс для настроек языкового сервиса, специфичных для языка Fortran: • список цветов, используемых для подсветки синтаксиса; • признак, что язык не чувствителен к регистру; • настройки написания комментариев (с какого символа начинаются, поддерживаются ли многострочные и т.д.); • имя языка программирования; • признак, что языковой сервис поддерживает работу с отладчиком; • признак, что языковой сервис поддерживает фоновый предварительный разбор файлов; • признак, что языковой сервис поддерживает расширенные средства навигации.
FortranAuthoringScope	класс- наследник от AuthoringScope,содержит информацию о файле исходного кода, полученную в результате проведения операции синтаксического анализа
FortranViewFilter	класс- наследник от ViewFilter, предназначен для перехвата событий пользователя и IDE

Таблица Б.5. Продолжение

FortranDeclarations	класс-наследник от Declarations, представляет список определений элементов языка программирования. Список определений запрашивается из блока модели представления элементов (см. пункт 2.4)
FortranMethods	класс-наследник от Methods, представляет коллекцию сигнатур методов (процедур). Список определений запрашивается из блока модели представления элементов (см. пункт 2.4)
Поддержка разбора файлов при помощи ANTLR	
FortranStringStream	наследник от стандартного класса ANTLR – AntlrStringStream, представляющего абстракцию потока символов. FortranStringStream представляет специализированную абстракцию потока символов файла с текстом программы на языке Fortran.
FortranFileStream	наследник от класса FortranStringStream, представляет абстракцию физического файла Fortran
FortranLexer	класс предназначен для полнотекстного лексического анализа языка Fortran
FortranPreprocessor	главный класс, является поставщиком токенов, так же как и лексический анализатор (реализует интерфейс ITokenSource); перенаправляет основную работу помощнику FortranPreprocessorState, а затем, при необходимости, выполняет модификацию токенов для учёта импорта файлов
FortranPreprocessorState	внутренний класс, описывающий текущее состояние препроцессора, и непосредственно выполняющий функции предварительной обработки; например – импортирование файлов включаемых строкой INCLUDE
FortranToken	наследник класса CommonToken, содержит специфичные для Fortran свойства (н.: номер конечной строки и символа в строке)
XmlDocCommentTokenStream	наследник класса AntlrTokenStream, вводит поддержку работы с дочерними языками, например, с языком XML комментариев документирования
XmlDocComment	класс для хранения информации, полученной из одного многострочного XML комментария документирования. Так же данный класс проводит анализ текста комментария документирования, для проверки его корректности и получения всех необходимых данных
FortranParser	класс предназначен для полнотекстного синтаксического анализатора языка Fortran
FortranTreeAdaptor	класс наследник от CommonTreeAdaptor, предназначен для помощи в автоматическом построении абстрактного дерева разбора на основании токенов FortranToken
FortranTreeWalker	класс для проведения семантического анализа и регистрации значимых элементов Fortran в блоке хранения распознанных элементов
Поддержка других возможностей	
FortranEditorFactory	класс поддержки назначения языкового сервиса для работы в текстовом редакторе при открытии (создании) нового файла
Работа с проектами и решениями (см. рисунок 2.8)	
FrisSolution	класс для работы с текущим решением, загруженным в Visual Studio. Осуществляет слежение за добавлением и удалением проектов

Таблица Б.5. Продолжение

FrisProjectBase	базовый абстрактный класс, реализует интерфейс IProject, представляет собой программный проект. Содержит: • имя; • список содержащихся в нём значимых элементов; • список других проектов, от которых зависит данный проект
IProject	интерфейс для описания проекта, предназначен для связи блока анализа и блока хранения значимых элементов
FrisProject	класс наследник от FrisProjectBase описывает проект решения. Содержит: • список составляющих его файлов; • список значимых элементов принадлежащих каждому файлу
FrisProjectItem	класс описывает элемент программного проекта, файл исходного кода. Содержит: • полный путь к файлу; • признак, что файл является физическим (в VS некоторые элементы проекта являются виртуальными, например каталоги-фильтры в проекте Intel Fortran)
ExternalProject	класс-наследник от FrisProjectBase, предназначен для описания внешних библиотек. Является ключевым элементом инфраструктуры поддержки внешних библиотек
Сериализация/десериализация значимых элементов программных проектов и внешних библиотек в XML представление	
XmlCommonConsts	статический класс, содержит определения строковых констант, использующихся при чтении/записи специализированных файлов XML: Fortran API и комментариев документирования
XmlFortranApiSerialazier	класс сериализации/десериализации значимых элементов в формат XML Fortran API
XmlDocCommentsSerializer	класс сериализации/десериализации комментариев документирования значимых элементов в специализированный формат XML

Таблица Б.6. Описание класса FortranToken

Свойство	Описание
Channel	номер канала, на который помещён токен. Использование каналов позволяет эффективно скрывать токены без их физического удаления из потока ввиду того, что все анализаторы работают только с токенами из строго определённого канала.
InputStream	ссылка на исходный поток символов, из которого был получен данный токен
Line	номер начальной строки для токена
CharPositionInLine	номер символа в начальной строке, с которого начался токен
StartIndex	сквозной индекс символа начала токена в потоке символов
StopIndex	сквозной индекс символа завершения токена в потоке символов
Text	текст токена. В случае с FortranToken здесь находится «чистый» текст, из которого извлечены все комментарии, пустые строки и т.п.
TokenIndex	сквозной номер токена в потоке токенов, куда он помещается
Type	тип токена (идентификатор, строковый литерал, оператор и т.д.)
EndLine	номер завершающей строки для токена
EndCharPositionInLine	номер завершающего символа в завершающей строке для токена
RawText	первоначальный текст токена, включающий в себя все комментарии, пустые строки и т.д.

Таблица Б.7. Описание класса XmlDocCommentTokenStream

Элемент	Описание
Свойства	
AllComments	хранит все обработанные комментарии документирования
DocComment	хранит текущий комментарий документирования
Методы	
GetComment	получить комментарий документирования, находящийся в указанной позиции потока токенов
SkipOffTokenChannels	вызывается автоматически синтаксическим анализатором для пропуска всех токенов, находящихся на неактивных каналах (ненужных синтаксическому анализатору)

Таблица Б.8. Описание класса XmlDocComment

Элемент	Описание
Поля	
_rawCommentLines	накапливает содержимое подряд идущих строк с комментариями документирования, собирая, таким образом, один многострочных комментарий
_allErrors	массив ошибок, выявленных после анализа комментария документирования; каждая ошибка - это экземпляр класса FortranNameMessageSpan (таблица Б.9)
_parameterComments	массив комментариев документирования, описывающих назначение аргументов процедуры или параметров производного типа данных
_extraParameters	массив комментариев документирования, повторно описывающих назначение аргументов процедуры или параметров производного типа данных
Свойства	
CommentSpan	файловая позиция (номер начальной строки, номер символа в начальной строке, номер конечной строки, номер символа в конечной строке) комментария документирования
StartIndex	номер стартового и конечного токена в потоке токенов, которым соответствует данный комментарий документирования
EndIndex	
Summary	текст, содержащий общее описание следующего за комментарием документирования значимого элемента
Result	текст, содержащий описание результата функции (для функций)
Методы	
AddText(string, Token)	служит для «сбора» многострочного комментария документирования, добавляет к текущему комментарию новую строку
GetErrors()	возвращает массив всех ошибок, обнаруженных в результате анализа комментария документирования
GetExtraParameters()	возвращает массив всех комментариев документирования, повторно описывающих назначение аргументов процедуры или параметров производного типа данных
GetParameter(name)	возвращает описание параметра с указанным именем
GetParameters()	возвращает массив всех комментариев документирования, описывающих назначение аргументов процедуры или параметров производного типа данных
MakeReadOnly()	переводит комментарий в нередактируемое состояние и запускает полнотекстный разбор его содержимого

Таблица Б.9. Описание класса FortranNameMessageSpan

Элемент	Описание
Свойства	
Name	хранит имя
Message	хранит описание
Span	хранит файловую позицию (номер начальной строки, номер символа в начальной строке, номер конечной строки, номер символа в конечной строке)

Описание классов, реализующих блок хранения распознанных элементов

Таблица Б.10. Описание интерфейса ICachedElement

Элемент	Описание
Свойства	
string Name	имя элемента. Например, имя переменной или типа данных
string UniqueName	уникальное имя элемента. Полностью квалифицированное имя элемента с учётом всех областей видимости
int UniqueId	уникальный идентификатор элемента, выдаётся при помещении элемента в глобальный кэш
string FileName	имя файла, в котором расположен элемент. Если в какой-либо файл с текстом программы включается текст других файлов при помощи инструкции INCLUDE, то для всех импортируемых таким образом элементов в качестве имени файла устанавливается имя файла назначения
IScope EnclosingScope	область видимости (контекст), в которой расположен данный элемент
Методы	
bool ContainsLocation(line, col)	функция проверяет, содержится ли указанная позиция (номер строки, номер символа в строке) в данном элементе
ICachedElement FindMember(memberName, comparisonType)	функция возвращает элемент с указанным именем, если он содержится внутри данного элемента
TextSpan GetLocation(cmd)	функция возвращает позицию (включая протяжённость – номер начальной и конечной строки, а также номера начальной и конечной позиций символов в каждой из строк) данного элемента, принимая во внимание команду, которая запрашивает позицию

Таблица Б.11. Описание интерфейса IScope

Элемент	Описание
Свойства	
string Name	имя области видимости. Не является обязательным, поскольку некоторые области видимости являются неименованными
IScope ParentScope	родительская область видимости. Используется в производных типах данных для хранения ссылок на базовый тип данных
IScope EnclosingScope	внешняя область видимости, в которой определена данная область видимости. Для глобальных областей видимости – свойство возвращает null значение
IProject Project	ссылка на проект, которому принадлежит данная область видимости
List<ICachedElement> Imports	список элементов, импортируемых из различных модулей в данную область видимости
Dictionary<string, TextSpan> Labels	список всех числовых меток в текущей области видимости. Они могут использоваться в инструкциях go to или в операторах форматированного ввода/вывода
Dictionary<string, CachedTypeReference> LetterTypeMapping	список соответствий букв типам данных Fortran, определяемый при помощи инструкции IMPLICIT
List<IScope> InnerScopes	список внутренних областей видимости в данной области видимости. Например, у модуля могут быть внутренние процедуры
List< ICachedElement > Members	список всех членов области видимости
Методы	
Define(element)	регистрирует элемент в данной области видимости
DefineImport(element)	регистрирует инструкцию импорта элемента в текущей области видимости. Например, инструкцию USE или IMPORT

Таблица Б.11. Продолжение

DefineLabel(label, span)	регистрирует числовую метку и её местоположение в текущей области видимости
DefineMapping(letter, type)	регистрирует соответствие буквы и типа данных
SetNoneMapping()	обнуляет все зависимости букв и типов данных. Вызывается при обнаружении инструкции IMPLICIT NONE
ICachedElement FindMember(memberName, comparisonType)	функция возвращает элемент с указанным именем, если он содержится внутри данной области видимости
ICachedElement FindMember(memberName, comparisonType, includeInterfaces, includeImports)	функция возвращает элемент с указанным именем, если он содержится внутри данной области видимости, с возможностью исключения из поиска интерфейсов и импортируемых элементов
CachedTypeReference FindLetterMapping(letter)	функция возвращает тип данных, соответствующий указанной букве

Таблица Б.12. Описание интерфейса IType

Элемент	Описание
Свойства	
string Name	имя типа данных
List<ICachedElement> Parameters	список параметров типа данных. Любой тип может иметь параметры. Встроенные типы данных имеют параметр KIND – число байт, используемых для хранения значений данного типа, а тип CHARACTER имеет ещё и параметр LEN, задающий длину строки
List<ICachedElement> Members	список членов типа данных, к ним относится всё, что входит в тип данных, за исключением параметров типа данных и связанных с типом данных процедур
List<ICachedElement> TypeBoundProcedures	список связанных с типом данных процедур
Методы	
Define(element)	функция, регистрирует член типа данных
DefineParameter(parameter)	функция, регистрирует параметр типа данных
DefineTypeBoundProcedure(element)	функция, регистрирует связанную с типом данных процедуру

Таблица Б.13. Описание назначения классов модели значимых элементов

Класс	Описание
FortranCachedElementBase	абстрактный базовый класс для всех остальных классов модели, предоставляет базовую реализацию интерфейса ICachedElement
FortranCachedTypeBase	абстрактный класс, отвечает за работу с типами данных, реализует интерфейс IType. Определяет основное поведение и набор свойств, характерных для всех типов данных, как встроенных, так и производных
FortranCachedBuildinType	предназначен для хранения определений, встроенных в язык типов данных: INTEGER, LOGICAL, REAL, COMPLEX, CHARACTER. Всё поведение наследует от базового класса FortranCachedTypeBase
FortranCachedDerivedType	предназначен для хранения определений производных, т.е. созданных пользователем типов данных
FortranCachedVar	описывает все переменные, параметры производных типов данных, а также константы
FortranCachedProcedure	базовый класс для описания процедур Fortran

Таблица Б.13. Продолжение

FortranCachedGenericProcedure	предназначен для описания связанной с типом данных обобщённой процедуры и содержит, помимо имени процедуры, список имён специфичных процедур, составляющих эту обобщённую процедуру
FortranCachedInterfacedProcedure	предназначен для описания процедуры, у которой указывается интерфейс. Это может быть как связанная с типом данных специфичная процедура, так и обычная процедура, являющаяся указателем на другие процедуры
FortranCachedTypedProcedure	предназначен для описания процедуры, у которой указывается тип возвращаемого значения
FortranCachedImport	предназначен для хранения операторов импортирования элементов модулей в текущую область видимости. Он используется при обработке инструкций USE и IMPORT
FortranCachedElementReference	отвечает за хранение установленных ссылок на другие элементы, но с другими локальными именами
FortranCachedScopedElementBase	базовый абстрактный класс для описания всех областей видимости, реализует интерфейс IScope
FortranCachedEnum	предназначен для описания перечислений, для этого он содержит свойство Enumerators – список членов перечисления
FortranCachedEnumerator	описывает один член перечисления
FortranCachedInterface	описывает элемент интерфейса: специфичный, абстрактный, общий, присваивания или ввода/вывода
FortranCachedModule	описывает модуль
FortranCachedMethodBase	базовый абстрактный класс для описания всех исполняемых элементов: функций, подпрограмм и главной программы
FortranCachedEntry	описывает дополнительную точку входа в подпрограмму или функцию
FortranCachedFunction	служит для описания функции
FortranCachedSubroutine	описывает подпрограмму
FortranCachedProgram	описывает главную программу
FortranCachedNamelist	предназначен для описания именованного списка ввода/вывода. Поскольку именованный список и импорт коммон-блока практически идентичны по своим свойствам, этот класс используется и для хранения common-блоков.

Таблица Б.14. Классы для описания атрибутов значимых элементов

Класс	Описание
FotranCachedAttribute	базовый класс для описания атрибутов; практически все атрибуты Fortran содержат только имя и не требуют введения дополнительных свойств, за исключением 5-ти: DIMENSION, INTENT, PASS (NOPASS), BIND, PRIVATE (PUBLIC).
FortranCachedDimensionAttribute	предназначен для описания атрибута размерностей массива DIMENSION
FortranCachedIntentAttribute	предназначен для описания атрибута INTENT, определяющего «направление» использования формального аргумента процедуры (in – входной, out – выходной, inout – входной и выходной)
FortranCachedBindAttribute	предназначен для описания атрибута BIND, отвечающего за межъязыковое взаимодействие
FortranCachedAccessAttribute	является логическим объединением двух атрибутов - PRIVATE и PUBLIC, отвечающих за уровень доступа к элементам
FortranCachedPassAttribute	объединяет в логическую группу атрибуты передачи экземпляра класса в связанную с ним процедуру: NOPASS и PASS

Таблица Б.15. Описание класса FortranCache

Элемент	Описание
Поля	
Dictionary<int, ICachedElement> _elements	статический словарь высокоуровневых элементов кэша с поддержкой многопоточного доступа
Dictionary<int, string> _fileNames	статический словарь имён всех файлов, находящихся в кэше
Dictionary<int, int[]> _fileScopes	статический словарь уникальных идентификаторов элементов содержащихся в файле исходного кода и являющихся областями видимости
HashSet<int> _globalModules	статическое множество идентификаторов модулей
HashSet<int> _globalProcedures	статическое множество идентификаторов глобальных подпрограмм, функций и дополнительных точек входа в них
List<ICachedElement> _localElements	список высокоуровневых элементов обнаруженных в операции разбора
List<int> _localScopes	список уникальных идентификаторов элементов, являющихся областями видимости
Свойства	
string FileName	имя файла, которому соответствует локальный кэш
List<FortranCachedModule> GlobalModules	статическое свойство – список глобальных модулей
List<FortranCachedElementBase> GlobalProcedures	статическое свойство – список глобальных процедур (функции и подпрограммы, включая дополнительные точки входа в них)
Методы	
FortranCache(fileName)	частный (private) конструктор; создаёт новый локальный кэш для операции разбора указанного файла
FortranCache Begin(fileName)	статическая функция возвращает новый объект локального кэша для использования в операции синтаксического разбора файла с указанным именем
void AddElement(element)	функция добавляет элемент в локальный кэш
void AddScope(element)	функция регистрирует уже содержащийся в кэше элемент как являющийся областью видимости
void ApplyUpdate()	функция помещает содержимое локального кэша в глобальный кэш (в статические элементы)
void RemoveFile(fileName)	статическая функция удаляет из глобального кэша всё содержимое файла с указанным именем
void Reset()	статическая функция полностью очищает содержимое глобального кэша
ICachedElement GetElementScope(address)	статическая функция возвращает кэшированный элемент, являющийся областью видимости, по номеру строки и номеру символа в строке, которые входят в этот элемент
List<ICachedElement> GetScopesForFile(fileId)	статическая функция возвращает все области видимости, содержащиеся в файле с указанным идентификатором
List<ICachedElement> GetScopesForFile(filePath)	статическая функция возвращает все области видимости, содержащиеся в указанном файле
string GetFileName(fileId)	статическая функция возвращает имя файла по его уникальному идентификатору
FortranCachedModule GetModule(moduleId)	статическая функция возвращает кэшированный модуль по его уникальному идентификатору
FortranCachedModule GetModule(moduleName)	статическая функция возвращает модуль с заданным именем, если модуля ещё нет в глобальном кэше, создаёт «неразрешённый» модуль (пустышку)

Таблица Б.15. Продолжение

FortranCachedMethodBase **GetGlobalProcedure(procedureId)**	статическая функция возвращает глобальную кэшированную процедуру с указанным уникальным идентификатором
FortranCachedModule **GetUnresolvedModule(name)**	статическая функция возвращает или регистрирует «неразрешённый» модуль с заданным именем
FortranCachedModule **ResolveModule(moduleName)**	статическая функция возвращает кэшированный модуль по его имени

Описание классов, реализующих блок модели представления значимых элементов

Таблица Б.16. Описание интерфейса IDeclaration

Свойство	Описание
Color **BackgroundColor**	цвет фона
string **Description**	текст описания элемента
string **DisplayText**	краткое описание
Color **ForegroundColor**	цвет текста
IconImageIndex **Glyph**	иконка
string **InsertText**	текст, вставляемый вместо текущего
TextSpan **Location**	положение элемента в текущем файле

Таблица Б.17. Описание класса FortranElementBase

Элемент	Описание
Свойства	
FortranCachedElementBase **CachedElement**	абстрактное свойство – возвращает элемент кэша, которому соответствует данное представление
string **Name**	имя элемента
string **UniqueName**	уникальное (полностью квалифицированное) имя элемента
Color **BackgroundColor**	цвет фона для вывода текста данного элемента
string **Description**	текст описания элемента – получается из соответствующего поля элемента кэша
string **DisplayText**	краткое описание элемента
Color **ForegroundColor**	цвет текста для вывода
IconImageIndex **Glyph**	иконка - отображается слева от текста элемента в выпадающих списках
string **InsertText**	текст, вставляемый вместо текущего
CodeElementLocation **Location**	положение элемента в файле исходного кода
Методы	
FortranElementBase()	конструктор по умолчанию, имеет атрибут «защищённый» (protected)

Таблица Б.18. Классы модели представления элементов

Класс	Описание
FortranElementBase	базовый класс для описания элемента модели представления
FortranUnknownDeclaration	специальный класс для описания не найденных элементов блока хранения распознанных элементов
FortranProcedure	предназначен для обработки всех кэшированных процедур
FortranEnumerator	описывает элемент перечисления. Напомним, что в Fortran перечисления являются неименованными и служат лишь контейнерами для своих членов (энумераторов), которые являются именованными целочисленными константами
FortranEntry	работа с дополнительными точками входа

Таблица Б.18. Продолжение

FortranVar	работа с переменными
FortranScopedElementBase	базовый абстрактный класс для работы со всеми элементами, являющимися областями видимости
FortranModule	работа с модулями Fortran
FortranMethodBase	базовый, абстрактный, для работы с подпрограммами, функциями и главной программой
FortranFunction	работа с функциями
FortranSubroutine	предназначен для работы с подпрограммами
FortranProgram	работа с главной программой
FortranTypeBase	базовый абстрактный класс для работы с типами данных
FortranBuiltinType	работа со встроенными типами данных
FortranDerivedType	работа с производным типом данных
FortranUnresolvedType	специальный класс для представления неразрешённого (ненайденного) типа данных
Работа с XML комментариями документирования	
XmlDeclaration	предназначен для описания одного тэга XML комментария документирования
XmlDeclarations	предназначен для описания набора XML комментариев документирования

Таблица Б.19. Классы, выполняющие вычисление выражений

Класс	Описание
FortranExpression	простое вычисление выражения. Возвращает все элементы целевой области видимости
FortranFilteredExpression	вычисление выражения с предварительной фильтрацией результирующего списка. Например, вернуть только переменные или подпрограммы
TransparentExpression	формальная реализация интерфейса IExpression. Возвращает предварительно полученный список определений элементов
XmlExpression	работа с выражениями для XML комментариев документирования

Таблица Б.20. Интерфейс IExpression для описания выражений

Элемент	Описание
Свойства	
TextSpan Span	область определения выражения
Методы	
IEnumerable<IDeclaration> Evaluate(int line, int col, ParseReason reason)	производит вычисление результата выражения и возвращает список элементов модели представления, соответствующий данному выражению

Таблица Б.21. Описание класса FortranFilters

Элемент	Описание
Методы	
bool DerivedTypeFilter (IDeclaration decl)	функция; выбирает только определения, являющиеся производными типами данных
bool ModulesFilter (IDeclaration decl)	функция; выбирает только определения, являющиеся модулями
bool SubroutinesFilter (IDeclaration decl)	функция; выбирает только те определения, которые могут встретиться в инструкции CALL. Это могут быть подпрограммы, процедуры или переменные, являющиеся экземплярами производных типов данных, в которых есть компоненты процедуры

Таблица Б.21. Продолжение

bool ExecutableSectionDeclarationsFilter (IDeclaration decl)	функция; выбирает только те определения, которые могут встретиться в секции исполняемого кода. Это могут быть любые определения, не являющиеся типами данных
bool LeftSideDeclarationsFilter (IDeclaration decl)	функция; выбирает только те определения, которые могут встретиться с левой стороны от оператора присваивания. Это могут быть переменные, но не константы, и указатели на процедуры
bool RightSideDeclarationsFilter (IDeclaration decl)	функция; выбирает только те определения, которые могут встретиться с правой стороны от оператора присваивания. Это могут быть любые переменные и константы, члены перечислений, процедуры и функции

Приложение В. Копия свидетельства о регистрации программы в РФ.

РОССИЙСКАЯ ФЕДЕРАЦИЯ

RU 2015615397

ФЕДЕРАЛЬНАЯ СЛУЖБА
ПО ИНТЕЛЛЕКТУАЛЬНОЙ СОБСТВЕННОСТИ

ГОСУДАРСТВЕННАЯ РЕГИСТРАЦИЯ ПРОГРАММЫ ДЛЯ ЭВМ

Номер регистрации (свидетельства):
2015615397

Дата регистрации: **18.05.2015**

Номер и дата поступления заявки:
2015612492 30.03.2015

Дата публикации: **20.06.2015**

Контактные реквизиты:
ratkevichis@gmail.com, +7 960-167-5154

Автор:
Раткевич Ирина Сергеевна (RU)

Правообладатель:
Раткевич Ирина Сергеевна (RU)

Название программы для ЭВМ:
Языковой сервис для эффективной разработки Fortran приложений в Microsoft Visual Studio («FRIS»)

Реферат:
Программа представляет собой языковой сервис для Microsoft® Visual Studio®, который вводит полную поддержку базовых возможностей технологии IntelliSense для языка программирования Fortran, с использованием стандарта Fortran-2003. Программа предназначена для повышения эффективности работы Fortran программистов, использующих Visual Studio для разработки приложений. Программа работает в Microsoft Visual Studio 2005/2008/2010. Программа предназначена для широкой области применения при разработке любых приложений, в том числе для имитационного моделирования сложных физических, технических и иных процессов, на языке программирования Fortran с использованием стандарта Fortran 2003. К основным возможностям, реализованным в программе, относятся: подсветка синтаксиса; выделение структурных элементов кода; поддержка панели навигации в текущем документе; поддержка XML комментариев документирования; реализация функций технологии IntelliSense (поддержка списка элементов производных типов данных; поддержка списков автодополнения; отображение сведений о параметрах процедур; отображение кратких сведений об элементе языка программирования); поддержка перехода к определению элемента языка программирования; поддержка работы со сниппетами (шаблонами) исходного кода; поддержка работы с панелью Список ошибок; поддержка библиотек конечного пользователя.

Тип реализующей ЭВМ:	IBM PC - совмест. ПК
Язык программирования:	C#
Вид и версия операционной системы:	Windows XP/Vista/7
Объем программы для ЭВМ:	6,5 Мб

РОССИЙСКАЯ ФЕДЕРАЦИЯ

СВИДЕТЕЛЬСТВО

о государственной регистрации программы для ЭВМ

№ 2015615397

Языковой сервис для эффективной разработки Fortran приложений в Microsoft Visual Studio («FRIS»)

Правообладатель: ***Раткевич Ирина Сергеевна (RU)***

Автор: ***Раткевич Ирина Сергеевна (RU)***

Заявка № **2015612492**
Дата поступления **30 марта 2015 г.**
Дата государственной регистрации
в Реестре программ для ЭВМ ***18 мая 2015 г.***

Врио руководителя Федеральной службы
по интеллектуальной собственности

Л.Л. Кирий

Приложение Г. Копия свидетельства о регистрации программы в США.

Certificate of Registration

This Certificate issued under the seal of the Copyright Office in accordance with title 17, *United States Code*, attests that registration has been made for the work identified below. The information on this certificate has been made a part of the Copyright Office records.

Register of Copyrights, United States of America

Registration Number
TXu 1-936-258
Effective Date of Registration:
July 31, 2014

Title

Title of Work: FRIS (FoRtran Intelligent Solutions)

Completion/Publication

Year of Completion: 2014

Author

- **Author:** Irina Sergeevna Ratkevich
  Author Created: text, computer program
  Citizen of: Russia (Federation)
  Year Born: 1990

Copyright Claimant

Copyright Claimant: Irina Sergeevna Ratkevich
ul. Kurchatova d. 25 kv.54, Sarov, Russia (Federation)

Rights and Permissions

Name: Irina Sergeevna Ratkevich
Email: ratkevichis@gmail.com
Telephone: +79601675154
Address: ul. Kurchatova d. 25 kv.54
Sarov 607190 Russia (Federation)

Certification

Name: Irina S. Ratkevich
Date: July 31, 2014

Correspondence: Yes

Приложение Д. Копия выписки из акта внедрения программы в РФЯЦ-ВНИИЭФ.

РФЯЦ-ВНИИЭФ
Институт теоретической
и математической физики
(ИТМФ)
Факс 4-57-73 Тел. 4-57-78
E-mail solovyev@vniief.ru

Несекретно

УТВЕРЖДАЮ

Первый заместитель директора ФГУП «РФЯЦ-ВНИИЭФ» – директор ИТМФ, доктор физико-математических наук

_____ В.П. Соловьёв

11.10. 2016 г.

12.10.2016 № 195-2025-25/1169122

Выписка из акта о внедрении программного обеспечения FRIS

АКТ

(Выписка из акта внедрения программного обеспечения FRIS №195-96/124940-ДСП от 04.09.2015 г.)

ФГУП «РОССИЙСКИЙ ФЕДЕРАЛЬНЫЙ ЯДЕРНЫЙ ЦЕНТР – ВСЕРОССИЙСКИЙ НАУЧНО-ИССЛЕДОВАТЕЛЬСКИЙ ИНСТИТУТ ЭКСПЕРИМЕНТАЛЬНОЙ ФИЗИКИ» (ФГУП «РФЯЦ-ВНИИЭФ») ИНСТИТУТ ТЕОРЕТИЧЕСКОЙ И МАТЕМАТИЧЕСКОЙ ФИЗИКИ (ИТМФ)

Программное обеспечение FRIS разработанное Раткевич И.С. было установлено в Институте Теоретической и Математической Физики (ИТМФ) ФГУП РФЯЦ-ВНИИЭФ в декабре 2013 года и успешно используется при разработке приложений на языке программирования Fortran в Microsoft Visual Studio. В частности FRIS используется при разработке систем имитационного моделирования.

Программное обеспечение имеет необходимые функциональные возможности и может гибко настраиваться под нужды конкретного разработчика, что позволяет существенно сократить время разработки программ и повысить их качество за счёт использования встроенных механизмов:

- контекстно-зависимой помощи;
- авто дополнения имён программных конструкций;
- визуального выделения различных программных конструкций в тексте программы;
- возможности документирования программных элементов и включения данной документации в выводимую контекстную помощь;
- использования шаблонов кода;
- поддержки пользовательских библиотек программ.

Первый заместитель директора ИТМФ,
начальник отделения 08,
доктор физико-математических наук — Р.М. Шагалиев

_____ 2016 г.

Приложение Е. Копия акта внедрения программы в КБМ.

ОТКРЫТОЕ АКЦИОНЕРНОЕ ОБЩЕСТВО
«НАУЧНО–ПРОИЗВОДСТВЕННАЯ КОРПОРАЦИЯ
«КОНСТРУКТОРСКОЕ БЮРО МАШИНОСТРОЕНИЯ»
(ОАО «НПК «КБМ»)
Российская Федерация, 140402,
Московская область, г. Коломна,
Окский проспект, 42
Факс (496) 613-30-64, 615-50-04
Тел. (496) 616-36-69, 616-34-68
E-mail: kbm-kbm@mail.ru
http://www.kbm.ru
ОГРН 1125022001851

Директору
ФГУП «РФЯЦ-ВНИИЭФ»,
В.Е. Костюкову

607188, Нижегородская обл.,
г. Саров, пр. Мира, д. 37

от 17.09.2015 г. № 007-122/ 1443/
на № 195-35/133428 от 11.09.2015 г.

О направлении АКТа

Уважаемый Валентин Ефимович!

Направляю Вам акт реализации диссертационного исследования и внедрения программного обеспечение FRIS версии 0.9.5.4 младшего сотрудника Вашего предприятия Раткевич И.С. Наша организация заинтересована в применении предоставленного Вами программного обеспечения и в предоставляемых им функциональных возможностях.

Приложение: Акт реализации, на 1-ом листе, 2 экз. в адрес.

С уважением,

Первый заместитель управляющего директора
и генерального конструктора – директор по
НИОКР и инновационному развитию В.А. Коновалов

АКЦИОНЕРНОЕ ОБЩЕСТВО
«НАУЧНО–ПРОИЗВОДСТВЕННАЯ КОРПОРАЦИЯ
«КОНСТРУКТОРСКОЕ БЮРО МАШИНОСТРОЕНИЯ»
(АО «НПК «КБМ»)
Российская Федерация, 140402,
Московская область, г. Коломна,
Окский проспект, 42
Факс (496) 613-30-64, 615-50-04
Тел. (496) 616-36-69, 616-34-68
E-mail: kbm-kbm@mail.ru
http://www.kbm.ru
ОГРН 1125022001851
от _____ № _____

Первый заместитель
управляющего директора и
генерального конструктора –
директор по НИОКР и
инновационному развитию
_____В.А. Коновалов
«17» 09 2015 г.

А К Т

реализации результатов диссертационного исследования младшего сотрудника ФГУП «Российский федеральный ядерный центр - Всероссийский научно-исследовательский институт экспериментальной физики» **Раткевич Ирины Сергеевны**

Мы, нижеподписавшиеся, начальник научно-теоретического отделения Грачиков Д.В., начальник сектора д.т.н. Шабалкин А.П., заместитель начальника отдела к.т.н. Асцатрян А.С, ведущий инженер к.т.н. Родионов К.А. составили настоящий акт в том, что результаты диссертационного исследования Раткевич Ирины Сергеевны представленные в виде программного обеспечение FRIS версии 0.9.5.4 и отчете «FIRS. Руководство пользователя», внедрены в АО «НПК «Конструкторском бюро машиностроения» (г. Коломна), а именно:

Результаты диссертационного исследования	Где внедрены и реализованы
1. Программное обеспечение FRIS версии 0.9.5.4.	Установлено в КБ машиностроения. При выполнении НИОКР
2. Программное обеспечение FRIS версии 0.9.5.4 Руководство пользователя FRIS.	При разработке приложений на языке программирования Fortran в Microsoft Visual Studio. При выполнении НИОКР

Начальник отделения Д.В. Грачиков

Начальник сектора А.П. Шабалкин

Заместитель начальника отдела А.С. Асцатрян

Ведущий инженер К.А. Родионов

«16» сентября 2015 г.

Приложение Ж. Копия акта внедрения программы в КБП.

АКЦИОНЕРНОЕ ОБЩЕСТВО
«КОНСТРУКТОРСКОЕ БЮРО ПРИБОРОСТРОЕНИЯ
ИМ. АКАДЕМИКА А. Г. ШИПУНОВА»
INSTRUMENT DESIGN BUREAU

Россия, 300001, Тула, Щегловская засека, 59. Тел. (4872) 41-0068. Факс (4872) 42-6139, 46-9861. E-mail: kbkedr@tula.net

13.10.15 № 14803/ДИС

На № _____ от _____

Директору
ФГУП «РФЯЦ-ВНИИЭФ»
В.Е. Костюкову

607188, Нижегородская обл.,
г. Саров, пр. Мира, д. 37

О направлении Акта

Уважаемый Валентин Ефимович!

Направляю Вам акт реализации диссертационного исследования и внедрения программного обеспечения FRIS версии 0.9.5.0 младшего научного сотрудника Раткевич И.С. Вашего предприятия. Наша организация заинтересована в применении предоставленного программного обеспечения и в предоставляемых им функциональных возможностях.

Приложение: Акт реализации, на 1-ом листе в 2-х экземплярах в адрес.

С уважением,

Заместитель управляющего директора по
отраслевым направлениям А.В. Гусев

Исполнитель Понятский В.М.
Департамент информационных технологий
Контактный телефон: (4872) 46-94-16 (доб.52-16)
e-mail: kbkedr@tula.net (с пометкой для Понятского В.М.)

Головной отдел «Канцелярия РФЯЦ-ВНИИЭФ»
ЕОСДО вход. №30767/195-2025-24
21.10.2015 11:41

АКЦИОНЕРНОЕ ОБЩЕСТВО
«КОНСТРУКТОРСКОЕ БЮРО ПРИБОРОСТРОЕНИЯ
ИМ. АКАДЕМИКА А. Г. ШИПУНОВА»
INSTRUMENT DESIGN BUREAU

Россия, 300001, Тула, Щегловская засека, 59. Тел. (4872) 41-0068. Факс (4872) 42-6139, 46-9861. E-mail: kbkedr@tula.net

_____ № _____

На № _____ от _____

Заместитель управляющего
директора по отраслевым
направлениям, к.т.н.
_____ А.В. Гусев
_____ 2015 г.

АКТ

реализации результатов диссертационного исследования младшего научного сотрудника ФГУП «Российский Федеральный Ядерный Центр – Всероссийский Научно-Исследовательский Институт Экспериментальной Физики» **Раткевич Ирины Сергеевны**

Мы, нижеподписавшиеся, начальник отделения (научно-теоретического подразделения динамического проектирования комплексов вооружения) к.т.н. Семашкин В.Е., начальник отдела Симаков С.Ю., начальник сектора Болосов Д.А. составили настоящий акт в том, что результаты диссертационного исследования Раткевич Ирины Сергеевны представленные в виде программного обеспечения FRIS версии 0.9.5.0 и отчёте «FRIS. Руководство пользователя», внедрены в АО «Конструкторское бюро приборостроения им. академика А.Г. Шипунова» (г. Тула), а именно:

Результаты диссертационного исследования		**Где внедрены и реализованы**
1.	Программное обеспечение FRIS версии 0.9.5.0	Установлено в «КБП»
2.	Программное обеспечение FRIS версии 0.9.5.0 Руководство пользователя FRIS	При разработке приложений на языке программирования Fortran в Microsoft Visual Studio. При выполнении НИОКР

Начальник отделения _____ В.Е. Семашкин
Начальник отдела _____ С.Ю. Симаков
Начальник сектора _____ Д.А. Болосов

«___» _____ 2015 г.

Приложение 3. Копия экспертного заключения РФЯЦ-ВНИИЭФ об оценке практического эффекта от использования FRIS.

РФЯЦ-ВНИИЭФ
Институт теоретической
и математической физики
(ИТМФ)
Факс 4-57-73 Тел. 4-57-78
E-mail solovyev@vniief.ru

Несекретно

УТВЕРЖДАЮ

Первый заместитель директора
ИТМФ, начальник отделения 08,
доктор физико-математических наук
_____ Р.М. Шагалиев
17.10 2016 г.

21.10.2016 № 195-96/176478

Экспертное заключение

Экспертное заключение

ФГУП «РОССИЙСКИЙ ФЕДЕРАЛЬНЫЙ ЯДЕРНЫЙ ЦЕНТР – ВСЕРОССИЙСКИЙ НАУЧНО-ИССЛЕДОВАТЕЛЬСКИЙ ИНСТИТУТ ЭКСПЕРИМЕНТАЛЬНОЙ ФИЗИКИ» (ФГУП «РФЯЦ-ВНИИЭФ») ИНСТИТУТ ТЕОРЕТИЧЕСКОЙ И МАТЕМАТИЧЕСКОЙ ФИЗИКИ (ИТМФ)

Для оценки качественного и количественного эффекта от использования программного обеспечения FRIS, разработанного Раткевич И.С., при написании текстов Fortran-программ в ИТМФ ФГУП РФЯЦ-ВНИИЭФ, была сформирована экспертная комиссия:

председатель комиссии	начальник отдела, кандидат физико-математических наук	Горшихин А.А.
заместитель председателя комиссии	начальник лаборатории	Алексеев А.В.
члены комиссии:	научный сотрудник	Бнятов А.В.
	научный сотрудник	Раткевич С.С.
	научный сотрудник	Шмелёв А.В.

Исследования проводились с использованием текстов производственных программ отдела 0808 (таблица 1).

Таблица 1. Общие характеристики программ для тестирования работы FRIS

№	Программа	Объём	Связи
1.	**КПД-1D**	более 500 000 строк	MAGIK (Fortran 2003), MARX (Fortran 90), ЕФР (C++), УРС-ОФ (Fortran 90), TSS3 (Delphi), TSSD (C++), IRA (C++)

Таблица 1. Продолжение

2.	**АРКТУР**	более 200 000 строк	MAGIK (Fortran 2003), MARX (Fortran 90), ЕФР (C++), УРС-ОФ (Fortran 90), TSS3 (Delphi), IRA (C++)
3.	**CONCORD**	более 150 000 строк	KeffAll (Fortran 90), MARX (Fortran 90), ЕФР (C++), TSS3 (Delphi)
4.	**MAGIK**	76 000 строк	IRA (C++)

Комиссия выделила следующие характеристики работы FRIS (таблица 2).

Таблица 2. Характеристики работы FRIS

Характеристика	Описание
Количественные характеристики	
Ускорение написания текста программы	Оценка количества набранных на клавиатуре символов к общему числу символов. Единицы измерения: Безразмерная
Время отклика при получении контекстно-зависимой помощи	Оценка времени получения информации в виде выпадающего списка и/или всплывающей подсказки в текущем программном контексте. Единицы измерения: Секунды
Качественные характеристики	
Интерактивность	Соответствие времени отклика программы, ожиданиям пользователя (готовности пользователя ждать отклик на свои действия) Возможные значения: «очень хорошая», «хорошая», «удовлетворительная», «не удовлетворительная»
Полнота информации в контекстно-зависимой помощи	Соответствие выдаваемой в описании информации реальному описанию программной единицы (тип, интерфейс, комментарий). Возможные значения: «полная», «не полная»
Удобство работы с текущим программным окружением	Оценивается использование FRIS в качестве интерактивного помощника для исследования текущего программного окружения. Исследуются полнота выдаваемой контекстной помощи по всем программным элементам, доступным в текущем контексте, с учётом импортов модулей, использованию внешних программ (и библиотек), реализованных, возможно на других языках программирования. Возможные значения: «очень удобно», «удобно», «удовлетворительно», «не удобно»

Для оценки ускорения написания текста программы был выбран контрольный пример: типовой алгоритм расчёта макроскопических многогрупповых констант среды из программы MAGIK (таблица 3).

Для расчёта ускорения используются формула:

$$S = \frac{N_{full}}{N_{entered}} \qquad (1)$$

где S - ускорение от использования возможности; N_{full} - полное число символов в результирующем тексте программы; $N_{entered}$ - количество введённых пользователем.

Таблица 3. Оценка ускорения от использования FRIS на характерном коде MAGIK

№	Текст программы	Возможность FRIS	Символы			Ускорение
			Введённые	Дополнены FRIS	Длина строки	
1	**!DIR$ NOVECTOR**	*сн.:* novec	7	8	15	2.143
2	**!DIR$ NOPREFETCH**	*сн.:* nopref	8	9	17	2.125
3	do ip = 1, npl, 16	*сн.:* do	14	5	19	1.357
4	call MM_PREFETCH(at(ip, 1), 1)	*сн.:* pref1	17	16	33	1.941
5	call MM_PREFETCH(pszg(ip, 1), 1)	*сн.:* pref1, *доп.:* pszg	19	16	35	1.842
6	call MM_PREFETCH(pszg(ip, 2), 1)	*сн.:* pref1, *доп.:* pszg	19	16	35	1.842
7	call MM_PREFETCH(pszi(ip, 1), 1)	*сн.:* pref1, *доп.:* pszi	19	16	35	1.842
8	call MM_PREFETCH(pszi(ip, 2), 1)	*сн.:* pref1, *доп.:* pszi	19	16	35	1.842
9	call MM_PREFETCH(psigi(ip, 1), 1)	*сн.:* pref1, *доп.:* psigi	19	17	36	1.895
10	call MM_PREFETCH(psigi(ip, 2), 1)	*сн.:* pref1, *доп.:* psigi	19	17	36	1.895
11	enddo	*кц. do на стр.* 3	0	6	6	
12	p2 => iz%St(:,:,jg)	*доп.:* St, jg	20	0	20	1
13	call prefetch0_r4(1, size(p2), p2)	*доп.:* prefetch0_r4	26	9	35	1.346
14	**!DIR$ IVDEP**	*сн.:* ivdep	7	5	12	1.714
15	**!DIR$ VECTOR ALWAYS**	*сн.:* vec	5	15	20	4
16	**!DIR$ NOPREFETCH**	*сн.:* nopref	8	9	17	2.125
17	do ip = 1, npl	*сн.:* do	11	4	15	1.364
18	at(ip, 1) = pszg(ip, 1) * iz%St(pszi(ip, 1), psigi(ip, 1), jg) + pszg(ip, 2) * iz%St(pszi(ip, 2), psigi(ip, 1), jg)	*доп.:* pszg, St, pszi, psigi	116	2	118	1.017
19	enddo	*кц. do на стр.* 17	0	6	6	
20	**!DIR$ NOVECTOR**	*сн.:* novec	7	8	15	2.143
21	**!DIR$ NOPREFETCH**	*сн.:* nopref	8	9	17	2.125
22	do ip = 1, npl, 16	*сн.:* do, *доп.:* npl	14	5	19	1.357
23	call MM_PREFETCH(at(ip, 2), 1)	*сн.:* pref1	17	16	33	1.941
24	enddo	*кц. do на стр.* 22	0	6	6	
25	**!DIR$ IVDEP**	*сн.:* ivdep	7	5	12	1.714

Таблица 3. Продолжение

26	**!DIR$ VECTOR ALWAYS**	*сн.:* vec	5	15	20	4
27	**!DIR$ NOPREFETCH**	*сн.:* nopref	8	9	17	2.125
28	do ip = 1, npl	*сн.:* do	11	4	15	1.364
29	at(ip, 2) = pszg(ip, 1) * iz%St(pszi(ip, 1), psigi(ip, 2), jg) + pszg(ip, 2) * iz%St(pszi(ip, 2), psigi(ip, 2), jg)	*don.:* pszg, St, pszi, psigi	116	2	118	1.017
30	enddo	*кц. do на стр.* 28	0	6	6	
31	**!DIR$ NOVECTOR**	*сн.:* novec	7	8	15	2.143
32	**!DIR$ NOPREFETCH**	*сн.:* nopref	8	9	17	2.125
33	do ip = 1, npl, 16	*сн.:* do, *don.:* npl	14	5	19	1.357
34	call MM_PREFETCH(at(ip, 1), 1)	*сн.:* pref1	17	16	33	1.941
35	call MM_PREFETCH(at(ip, 2), 1)	*сн.:* pref1	17	16	33	1.941
36	call MM_PREFETCH(psigg(ip, 1), 1)	*сн.:* pref1, *don.:* psigg	19	17	36	1.895
37	call MM_PREFETCH(psigg(ip, 2), 1)	*сн.:* pref1, *don.:* psigg	19	17	36	1.895
38	enddo	*кц. do на стр.* 33	0	6	6	
39	**!DIR$ IVDEP**	*сн.:* ivdep	7	5	12	1.714
40	**!DIR$ VECTOR ALWAYS**	*сн.:* vec	5	15	20	4
41	**!DIR$ NOPREFETCH**	*сн.:* nopref	8	9	17	2.125
42	do ip = 1, npl	*сн.:* do	11	4	15	1.364
43	at(ip,1) = psigg(ip,1) * at(ip,1) + psigg(ip,2) * at(ip,2)	*don.:* psigg	59	2	61	1.034
44	enddo	*кц. do на стр.* 42	0	6	6	
45	ipl = 1	*don.:* ipl	8	0	8	1
46	do i = sp(1)%iSpan, sp(2)%iSpan	*сн.:* do, выбор iSpan	24	8	32	1.333
47	isp = max(spans(i)%iStart, sp(1)%igp)	*don.:* spans, iStrat, igp	32	8	40	1.25
48	iep = min(spans(i)%iEnd, sp(2)%igp)	*don.:* spans, iEnd, igp	31	7	38	1.226
49	m = iep - isp + 1		20	0	20	1
50	if(m < 1) cycle		18	0	18	1
51	pa(isp:) => at(ipl:ipl + m, 1)	*don.:* isp,ipl	33	0	33	1
52	ptr(isp:iep) => A(isp:iep, jg)	*don.:* isp,iep	33	0	33	1
53	**!DIR$ NOINLINE**		17	0	17	1

Таблица 3. Продолжение

54	call vector_kernel_sum_mult_2v(isp, iep, ptr, pfi, pa)	*доп.:* vector_...	37	20	57	1.541
55	ipl = ipl + m		16	0	16	1
56	enddo	*кц. do на стр.* 46	0	6	6	
		Итого:	**1006**	**461**	**1467**	**1.458**

Примечания к таблице 3.

В таблице использованы следующие обозначения: «*сн.:*» - использование сниппета с указанным именем; «*доп.:*» - использование возможности технологии IntelliSense автодополнение; «*кц. do на стр.* XX» - конец цикла do, вставленного в результате использования сниппета для цикла do на строе с номером «XX».

Для написания циклов использовался сниппет FRIS с ускорителем do, который имеет следующий шаблон (Листинг 1).

```
do i = 1, npl
  |
enddo
```

Листинг 1 – Сниппет для цикла do

В листинге 1 курсивом отмечены настраиваемые части сниппета, а вертикальной чертой – место, куда будет переведён курсор после завершения редактирования сниппета. Таким образом, при использовании сниппета do вместо него будет вставлено 3 строки. Этим обусловлено то, что в строках 11, 19, 24, 30, 38, 44, 56 отсутствуют символы, вводимые пользователем.

Таким образом (таблица 3), ускорение программирования за счёт использования FRIS составляет 1.45 раза (всего 1467 символов, из них набрано 1006 символов).

Аналогичные исследования для типовых алгоритмов были проведены в остальных программах и хорошо согласуются с указанным выше, поэтому не представлены в данном документе.

Приведём оценку ускорения от использования отдельных возможностей FRIS (таблицы 4-5).

Таблица 4. Оценка ускорения написания текстов программ за счёт использования сниппетов FRIS (в программе MAGIK)

Возможность	Конструкции и число их вхождений	Ускоритель FRIS	Ускорение
Использование директив OpenMP	parallel parallel do do critical	omppar omppdo ompdo ompcrit	32 / 7 = 4.5 38 / 7 = 5.4 20 / 6 = 3.3 32 / 8 = 4
Использование SIMD-операций	ivdep noprefetch novector vector always mm_prefetch	ivdep nopref novec vec prefX	11 / 6 = 1.8 6 / 7 = 2.2 14 / 6 = 2.3 19 / 4 = 4.7 19 / 6 = 3.1
Использование двусвязного списка (в библиотеке MAGIK)	Использований – 3 Объём кода сниппета – 270 строк – 8084 символа Настраиваемых частей 5: имя модуля, имя типа списка, имя типа элемента списка, имя модуля с типом данных хранимых в элементе списка, описание элемента данных, хранимых в списке	llist	8084 / 6 = 1347

Таблица 5. Оценка ускорения написания программ за счёт использования возможностей IntelliSense во FRIS (в программе MAGIK)

Возможность	Описание	Характеристики	Ускорение
Построение списка элементов сложного объекта	автодополнение компонентов производных типов данных	Средняя длинна имени компонента производного типа данных: 6 Среднее число символов, позволяющее однозначно идентифицировать имя: 3	6 / 3 = 2
Автодополнение	завершение имени модуля	Средняя длина имени: 20 Среднее число символов для однозначной идентификации имени: 7	20 / 7 = 2.85
	завершение имени производного типа данных	Средняя длина имени: 15 Среднее число символов для однозначной идентификации имени: 7	15 / 7 = 2.14
	завершение имени переменной	Средняя длина имени: 5 Среднее число символов для однозначной идентификации имени: 3	5 / 3 = 1.66
	завершение имени процедуры	Средняя длина имени: 15 Среднее число символов для однозначной идентификации имени: 8	15 / 8 = 1.87

Отметим, что подобные оценки справедливы и для остальных исследуемых программ, с небольшими отличиями, которые можно считать незначащими.

Приведём итоговую таблицу оценки характеристик FRIS (таблица 6).

Таблица 6. Оценка характеристик работы FRIS

Характеристика	КПД-1D	АРКТУР	CONCORD	MAGIK	Итог
Количественные характеристики					
Ускорение написания текста программы	1.41	1.36	1.42	1.45	1.41
Время отклика при получении контекстно-зависимой помощи	1 с.	1 с.	1 с.	менее 1 с.	1 с.
Качественные характеристики					
Интерактивность	«хорошая»	«хорошая»	«хорошая»	«очень хорошая»	«хорошая»
Полнота информации в контекстно-зависимой помощи	«полная»	«полная»	«полная»	«полная»	«полная»
Удобство работы с текущим программным окружением	«удобно»	«удобно»	«удобно»	«очень удобно»	«удобно»

Все проведённые исследования выполнялись на персональной ЭВМ с характеристиками: центральный процессор Intel Core i5 2.8 ГГц, объём оперативной памяти 8 Гбайт.

Вывод: Основываясь на результатах проведённых исследований, экспертная комиссия считает, что программа FRIS обладает хорошими качественными и количественными эксплуатационными характеристиками, и упрощает программирование современных сложно структурированных Fortran-программ.

Комиссия особо отмечает, что реализованные возможности существенно повышают удобство работы с текущим программным окружением, за счёт использования интерактивных подсказок в качестве справочной системы по доступным программным элементам.

Председатель комиссии _____ Горшихин А.А.
17.10 2016 г.

Заместитель председателя комиссии _____ Алексеев А.В.
2016 г.

Члены комиссии: _____ Бнятов А.В.
17.10 2016 г.

_____ Раткевич С.С.
17.10. 2016 г.

_____ Шмелёв А.В.
17.10 2016 г.

Раткевич С.С. 2-06-34 7 л. (1 экз.)
17.10.2016