«Оптика атмосферы», **2**, № 4 (1989)

УДК 681.306

В.Б. Новосельцев, В.Т. Калайда

КОНЦЕПЦИИ ПОСТРОЕНИЯ ПРОГРАММНОГО ОКРУЖЕНИЯ ДЛЯ РЕШЕНИЯ ЗАДАЧ АТМОСФЕРНОЙ ОПТИКИ

В статье на основании анализа современных методов конструирования программ и особенностей предметной области «оптика атмосферы» предлагается проект построения программного окружения, обеспечивающего поддержку решения задач атмосферной оптики. Анализируются требования к системе со стороны пользователя и разработчика и предлагается архитектура наиболее полно удовлетворяющая предъявленным требованиям.

Атмосферная оптика, как отмечается в [1], входит в ряд научных дисциплин, решение задач в которых сопряжено с обработкой больших объемов данных. Помимо этого существует еще ряд особенностей, среди которых следует отметить необходимость использования вычислительных комплексов сложной топологии, например, при обработке информации, которая поставляется сетью станций, обеспечивающих глобальную томографическую реконструкцию полей ряда атмосферно-оптических параметров. Эту задачу усложняет потребность анализа и обработки данных в режиме реального времени. Приведенные особенности предметной области определяют достаточно жесткие требования к информационно-программному обеспечению, используемому при решении задач атмосферной оптики с применением средств вычислительной техники.

Предлагаемые концепции построения комплекса программных средств отражают современный уровень «вычислительных наук» (*computer science*) и учитывают опыт создания специализированных систем, предназначенных для решения задач атмосферной оптики [2, 3]. В статье используется термин «система поддержки решения» — при этом подразумевается «решение задач» (*problem solving support system*).

Системы поддержки решения (СПР) являются интегрированными программно-аппаратными комплексами. При реализации подобных проектов наибольших затрат требует программная часть (*software*), что подтверждается целым рядом работ последних лет [4—6]. До определенной степени это связано с тем, что программирование сложных систем с большим количеством интерфейсов продолжает оставаться скорее искусством, чем производственным процессом.

Говоря о программном продукте, прежде всего следует учитывать два аспекта; взгляд на программный продукт со стороны пользователя и представление программиста-разработчика, реализующего проект.

Требования пользователей можно разбить на две группы. К первой отнесем требования общего характера, предъявляемые к современным программным системам, а во вторую включим те особенности, которыми должна обладать собственно СПР.

Требования общего характера включают: а) доброжелательность со стороны системы при общении; б) преимущественно интерактивный режим использования; в) малое время реакции системы. К особенностям, характерным для СПР — специальным требованиям — следует отнести: а) эффективные, мощные и удобные средства управления данными; б) возможность формализованного описания исходной предметной области (ПО), т.е. наличие средств представления концептуального и алгоритмического знания о ПО; в) аппарат логического вывода и экспертные системы, надстроенные над базой знаний ПО (БЗПО), а также г) проблемно-ориентированные интерфейсы между различными подсистемами СПР и пользователем, предусматривающие широкие сервисные возможности.

Таким образом, представление пользователя о функциональной архитектуре СПР в общем случае отражается схемой, приведенной на рис. 1. Пользователь СПР может ставить и решать свою задачу с помощью одной лишь экспертной системы (ЭС), которая наряду с функциями эксперта-специалиста обеспечивает корректное использование и сохранность данных и знаний. Кроме этого существует возможность прямого использования и модификации содержимого базы данных (БД) и базы знаний о ПО и смешанные стратегии работы с СПР, когда в ходе активного со стороны ЭС диалога пользователь «перехватывает» инициативу и вносит необходимые ему изменения в среду данных (знаний, правил ЭС). К интерфейсам пользователя предъявляются особые требования. Во-первых, необходимо минимизировать само их количество посредством разумной интеграции. Во-вторых, желательно, чтобы даже минимальный набор интерфейсов со стороны пользователя был унифицированным. В-третьих, интерфейсы должны быть настраиваемыми на конкретную ПО, т.е. проблемно-ориентированными. В-четвертых, помимо возможности настройки на содержательные понятия (термины—типы) выбранной ПО интерфейсы должны обеспечить пользователю определенную свободу и в оперировании более сложными объектами — связями между понятиями — на языке, близком к профессиональному арго для данной ПО.

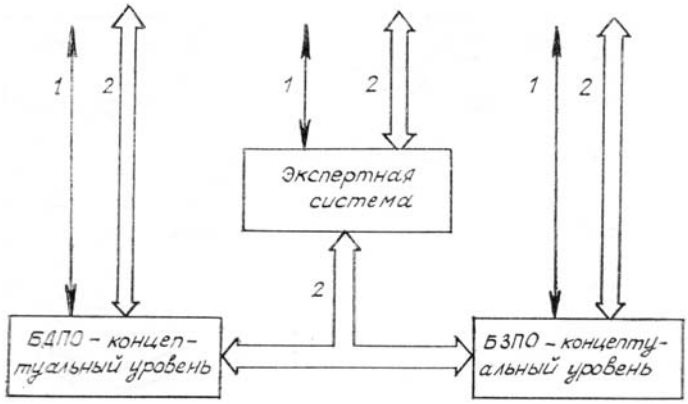

Рис. 1. Архитектура СПР (представление пользователя): *1* — интерфейсы пользователя; *2* — информационные потоки

Если представление пользователя об СПР сопоставить с видимой — надводной частью айсберга, то архитектуру (функциональную) системы с позиций программиста-разработчика можно сравнить с айсбергом целиком. В этом сравнении отражен чрезвычайно важный момент: при реализации системы нельзя пренебрегать представлением пользователя и заниматься лишь технической («подводной») стороной проблемы. Система должна быть сбалансирована так, чтобы ни одно из требований пользователя «не ушло под воду», но и ничего из относящегося к реализационной стороне не «всплыло на поверхность». Таким образом, принципиальная архитектура СПР с точки зрения разработчика представлена на рис. 2 и является детализацией рис. 1.

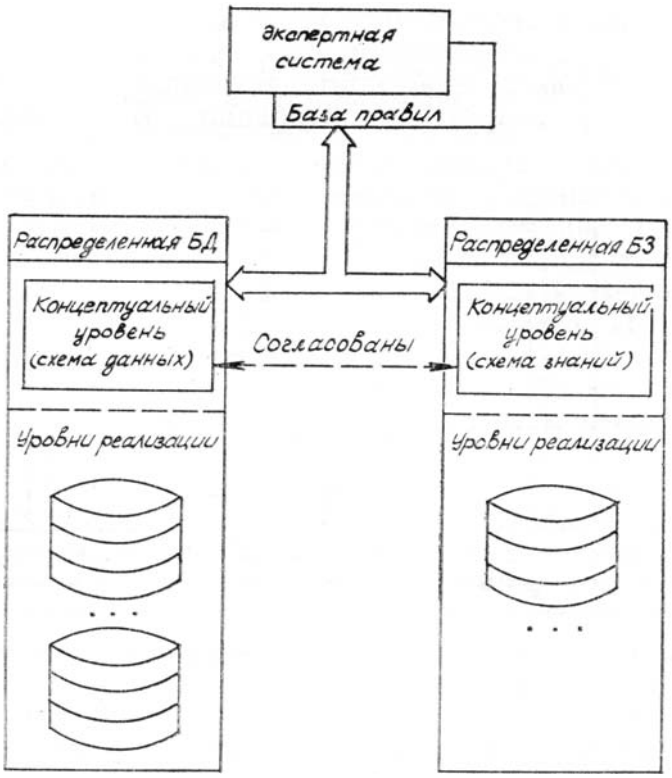

Рис. 2. Уточненная архитектура СПР

Следуя С.С. Лаврову [7], будем рассматривать три уровня знания о проблемной области:

1. Концептуальный уровень, на котором выделяются абстрактные объекты, соответствующие содержательным понятиям, и указываются логические связи между объектами.

2. Алгоритмический или процедурный. Часть связей предыдущего уровня носит характер отношений вычислимости, которые указывают на возможность получения одних значений по некоторым другим. Процедуры, реализующие соответствующие вычисления, составляют второй уровень.

3. Фактуальный или предметный. Значения, приписываемые определенным объектом ПО, могут быть получены в ходе эксперимента, из наблюдений и т.д. Эти значения хранятся в базе данных и составляют знание третьего уровня.

В первом уровне можно выделить два подуровня, различающиеся видами связей и степенью сложности понятий, между которыми эти связи установлены. Один из этих подуровней составляет вычислительная модель [7] данной ПО. В другом подуровне связи носят более сложный характер, определяются, возможно, на основании некоторых эвристик и строятся, главным образом, над агрегатами ПО (отношениями). При построении СПР концептуальное знание второго подуровня поддерживается экспертной системой, а знание первого подуровня погружается в базу знаний ПО, составляя описание вычислительной модели.

Организационно СПР представляет собой расширяемый комплекс мониторов, которые поддерживают внешние и внутрисистемные интерфейсы, а также реализуют прочие функции системы, кроме того в систему входит набор библиотек. Одним из главных принципов, положенных в основу организации СПР, является принцип иерархичности построения. Базисным элементом служит библиотека (рис. 3). Имеются стандартно организованные библиотеки, работа с которыми поддерживается а цтором библиотек системы (МБС), и нестандартные библиотеки, каждая из которых содержит в себе собственный обслуживающий монитор. В каталоге любой библиотеки стандартно расположен указатель на вид этой библиотеки и на «пускач» соответствующего монитора.

Элементами библиотеки служат модули. Модулями, в частности, являются каталог библиотеки и обслуживающий монитор. Для данной библиотеки модулем может быть и другая библиотека, а точнее ее адрес и адрес участка памяти, выделяемой для обмена информацией (порт). Модуль аналогичной структуры (адрес-порт) организуется в каждой библиотеке и для обмена информацией с библиотекой, которая актуализировала данную. Основным тактом работы на этом уровне является сеанс. Актуализация библиотеки открывает сеанс, при этом, в частности, создается копия библиотеки, инициализируется соответствующий порт и производится связывание мониторов. По окончании сеанса открывший его монитор, в случае необходимости, может произвести обновление библиотеки, не изменяя информации о ней в своем каталоге, либо организовать новую библиотеку в соответствии с предоставленными этому монитору полномочиями.

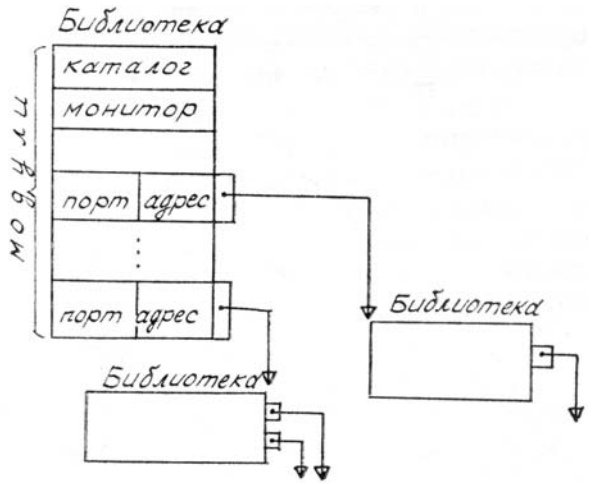

Рис. 3. Система библиотек

Описанная архитектура достаточно легко реализуема, хотя она и не является оптимальной во всех отношениях. Ее главным и, пожалуй, принципиальным недостатком является обилие ссылок и жесткие ограничения, накладываемые на межбиблиотечные интерфейсы. В то же время на обсуждаемую архитектуру уровня реализации естественным образом проецируется функциональная архитектура рис. 2, и при этом обеспечивается простота развития и поддержки системы.

Экспертная часть СПР представляет в действительности набор экспертных систем, который имеет иерархическую (древовидную) организацию, отражающую разбиение исходной проблемой области на изолированные подобласти.

Выбор подобной организации ЭС содержит ряд преимуществ:

1. Основой любой экспертной системы является набор правил. Проблемная область, сопоставимая с рассматриваемой, требует априорно достаточно большой базы правил. Поскольку эффективность получения экспертной оценки определяется второй или даже более высокой степенью общего количества правил в базе, то разбиение базы позволяет получать определенный выигрыш по ресурсам при использовании ЭС.

Это иллюстрируется неравенством:

$$\left(\sum_i n_i\right)^m > \sum_i n_i^m,\ m > 1.$$

2. Важно заранее предусмотреть наличие фазы продолжающейся разработки в жизненном цикле программного продукта. Существование этой фазы подразумевает, во-первых, отчуждение на некотором этапе продукта от группы разработчиков, и, во-вторых, открытость системы для модификации в ходе эксплуатации. Структуризация системы в целом и ее экспертной составляющей в частности способствует облегчению внесения изменений в систему штатом сопровождения. Узел дерева может быть модифицирован, удален или заменен другим — единственным требованием при этом является необходимость соблюдения соглашения о связях между узлами.

3. Экспертная система отражает некоторое знание об исходной предметной области. Реальная ПО, как правило, не является чем-то застывшим — появляется новая информация, изменяются соответствующие теории и т. Д. Иерархическая организация ЭС, точнее, свойственная такой организации модульность, обеспечивает простоту приведения концептуального знания о ПО в соответствие с меняющимися представлениями пользователя — специалиста в данной проблемной области.

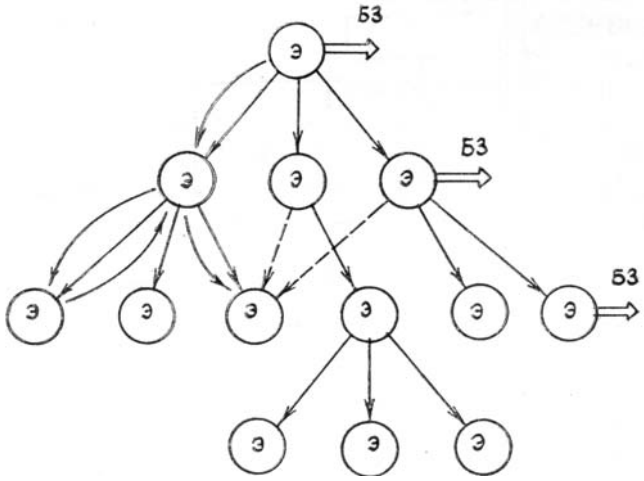

Рис. 4. Работа с ЭС

Работа пользователя с экспертной системой состоит из ряда консультаций с экспертами различных уровней (здесь и ниже под экспертами понимаются конкретные составляющие всей системы, расположенные в узлах дерева на рис. 4). При этом может производиться отказ от услуг — возврат на вышестоящий уровень, где эксперт обладает большей «широтой» знаний в ущерб «глубине», или детализация проблемы, т.е. обращение к эксперту следующего уровня, обладающему знаниями о некоторой частной задаче, которая сформулирована в ходе анализа, либо о специфическом подходе к решению исходной проблемы, возможность использования которого установлена предыдущими экспертами.

Описанный механизм использования ЭС весьма схож с процедурой «порождения и проверки» [8]. В то же время не следует забывать, что в предлагаемой трактовке каждый эксперт представляет собой независимую традиционно понимаемую экспертную систему со своей базой знаний и специфической реализацией метода поиска решений.

В дереве ЭС могут встречаться склеенные ветви, это вызывается наличием сходных подзадач (путей решения) для различных начальных конфигураций. Эксперты нижних уровней тесно взаимодействуют с вычислительной моделью и базой данных проблемной области. В ходе такого взаимодействия строится программа решения задачи пользователя, а также осуществляется управление а цесссом исполнения программы в среде локальной вычислительной сети.

База знаний (БЗ) системы поддержки решения содержит описание вычислительной модели предметной области. При разработке средств описания модели ПО мы стремились достичь «сбалансированности» использования процедурных и непроцедурных механизмов. «Чистый» Пролог [9] или Декарт[10] являются типичными представителями языков, поддерживающих непроцедурный стиль. В процессе развития как того, так и другого выяснилась необходимость добавления в них конструкций, характерных для традиционных языков программирования.

Языки, используемые для описания вычислительных моделей, следуя Э.Х. Тыугу [11], будем называть языками концептуального программирования,

Результат трансляции описания вычислительной модели в совокупности с реализацией элементов этой модели образует базу знаний (БЗ), ее содержимое используется при автоматизированном получении решений задач, которые в модели могут быть поставлены. Программа решения задачи строится на основании спецификации. Достаточно полная вычислительная модель позволяет задавать спецификацию в

непроцедурной форме, т.е. указывать лишь то, что известно и что в задаче необходимо найти. Язык аботы с моделью проблемной области наследует ряд концепций языков Декарт [10] и Утопист [12].

Неформально вычислительная модель, по Э.Х. Тыугу [13] («стандартная»), представляет собой набор имен понятий прикладной теории (элементов) и совокупность отношений над этими элементами, характеризующих вычислительные связи между понятиями. Как в языке Утопист, так и в Декарте предусмотрены средства структуризации при описании модели, однако эти средства, по существу, предназначены в основном для сокращения записи (текстуального представления модели). Дело в том, что исходное структурированное описание модели непосредственно перед применением процедур анализа (синтеза) в большинстве систем концептуального программирования приводится к виду «стандартной» вычислительной модели — это процесс обычно называют разверткой.

Несомненный интерес представляет такая модификация теории вычислительных моделей, при которой процедуры синтеза применяются непосредственно к структурированному описанию исходной ПО. В подтверждение этого приведем ряд соображений:

1. Использование рекурсивных конструкций при описании ПО может привести к тому, что процесс развертки не будет завершаться.

2. Даже если запретить рекурсивные конструкции, длина развернутой модели может оказаться экспоненциальной относительно длины исходного описания.

3. При использовании развертки затруднено предоставление пользователю возможности «содержательно» (в терминах описания ПО) анализировать синтезированную программу, что усложняет отладку описания ПО.

Аналогичные недостатки, присущие развертке, подчеркивались неоднократно. Как правило [14], в качестве альтернативного логического аппарата синтеза при этом предлагается использовать некоторый фрагмент исчисления предикатов.

Развиваемый нами подход опирается на формализм структурных *C*-вычислительных моделей, введенный в [15]. Модель при этом представляется конечной совокупностью схем отношений $M = \{T_1,\ldots,T_m\}$; схема T_i определяет структуру нетривиального объекта ПО. Говоря о структуре сложного объекта ПО, мы фиксируем список имен составляющих его понятий (элементов схемы) и набор предложений вычислимости вида F: $A \to X$ над этими именами. Предложение вычислимости (ПВ) отражает тот факт, что по значениям величин A применением программного терма с именем F можно вычислить значения величин X. В такой трактовке схема отношения мало отличается от стандартной вычислительной модели. Аналогия исчезает, если добавить, что типы элементов схемы могут определяться не только внешним по отношению к данной модели образом, но и посредством связывания элементов со схемами этой же модели. Таким образом, *C*-модель может адекватно отражать иерархию понятий исходной ПО и использовать рекурсивно-описанные понятия,.

В *C*-модели можно ставить задачи вида «по _ заданным _ A вычислить неизвестные _ X». На основании постановки задачи и с учетом информации о проблемной области соответствующая подсистема СПР предпринимает попытку автоматически синтезировать программу, реализующую задачу пользователя. Построенные таким образом программы могут содержать конструкции ветвления и рекурсии. Следует отметить, что процедуры синтеза достаточно эффективны для того, чтобы генерация программы пользователя производилась в интерактивном режиме.

Включение в архитектуру СПР базы данных на настоящем этапе не предполагает разработку оригинальной системы управления БД. Реализация программной части проекта, опирающаяся на принципы библиотечной организации, позволяет достаточно естественно использовать созданные ранее БД. Однако это не означает, что вопрос о собственных (штатных) базах данных является закрытым.

1. Зуев В.Е. //Оптика атмосферы. 1988. Т. 1. № 1. С. 5—12.
2. Войцеховская О.К., Зуев В.Е., Тютерев Вл.Г. //Оптика атмосферы. 1988. Т. 1. № 3. С. 3—15.
3. Комаров В.С, Мицель А.А., Михайлов С.А., Пономарев Ю.Н., Руденко В.П., Фирсов К.М. //Оптика атмосферы. 1988. Т. 1. № 5. С. 84—89.
4. Брукс Ф.П. Как проектируются и создаются программные комплексы М.: Наука, 1979. 152 с.
5. Фокс Дж. Программное обеспечение и его разработка. М.: Мир, 1985, 368 с.
6. Требования и спецификации в разработке программ /Под ред. В.Н. Агафонова. М.: Мир, 1984. 344 с.
7. Лавров С.С. //Микропроцессорные средства и системы. 1986. № 3. С. 14—19.
8. Элти Дж., Кумбс М. Экспертные системы: концепции и примеры. М.: Финансы и статистика, 1987. 191 с.
9. Colmerauer A., Kanout H., Pasero R. and Roussel P. Un system de communication Homme-machine en Francais. Research report. Groupe Intelligence Artificiel, Universite Aix Marseille, 1973. № 11. С. 136—144.
10. Бабаев И.О., Новиков Ф.А., Петрушина Т.И. Язык Декарт — входной язык системы СПОРА. Прикладная информатика. М., 1981. Вып. I
11. Тыугу Э.Х. Концептуальное программирование. М.: Наука, 1984. 256 с.
12. Кахро М.И., Калья А.П., Тыугу Э.Х. Инструментальная система программирования ЕС ЭВМ. (ПРИЗ). М.: Финансы и статистика, 1981. 158 с.
13. Тыугу Э. Х. //ЖВМ и МФ, 1970. Т. 10. № 3. С. 716—733.
14. Нейман В. С. //В кн.: Синтез программ: Тезисы докл. Всес. Школы, г. Устинов, 1985. С. 25—28.
15. Новосельцев В.Б. В кн.: Синтез программ: Тезисы докл. Всес. школы, г. Устинов, 1985. С. 75—77.

Институт оптики атмосферы
СО АН СССР, Томск

Поступила в редакцию
15 ноября 1988 г.

V. B. Novoseltsev, V. T. Kalaida. **Software Concepts for Solving Atmospheric Optics Problems**.

A programme environment project for a problem-solving support system in the atmospheric optics subject area is discussed. The proposed project is based on the analysis of the advanced approaches to program synthesis and specific features of the subject area referred to as atmospheric optics. The system architecture is shown to meet both the user requirements and the developer concepts.