ТЕХНОЛОГИЧЕСКОЕ ОБРАЗОВАНИЕ

Васильев Павел Константинович,
Соловьёв Игорь Павлович

ПРИМЕНЕНИЕ ИНЖЕНЕРИИ ЗНАНИЙ В СПЕЦИФИКАЦИИ ПРОГРАММНЫХ ПРОЕКТОВ

Одно из самых удивительных явлений наших дней заключается в том, что практически любая целесообразная деятельность приводит к составлению компьютерных программ, иногда маленьких, но чаще – очень и очень больших. Однако давно уже известно, что создание больших программных комплексов, так необходимых в мирной и военной индустрии, сопряжено с огромными трудностями, чревато коварными ошибками, не укладывается в кажущиеся разумными сроки проектирования и часто требует весьма основательной финансовой поддержки.

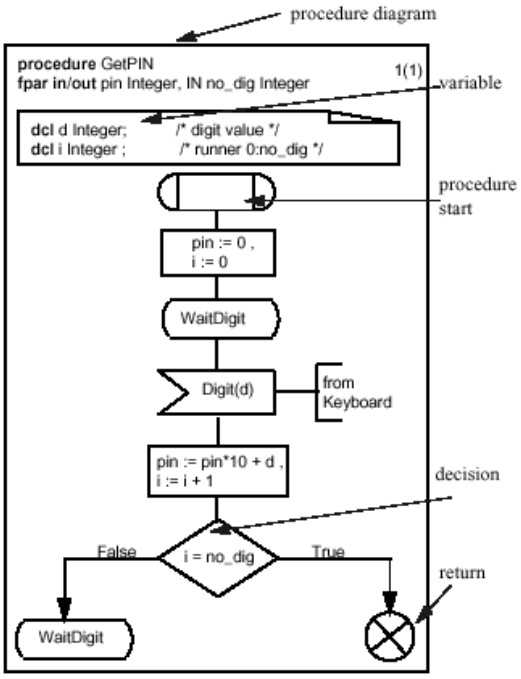
Существует много объективных и субъективных причин, порождающих все перечисленные проблемы. Но главная причина в том, что сам программный продукт изначально сложен. Во-первых, сложность программного продукта отражает сложность исходной реальной задачи. Во-вторых, требования заказчика часто оказываются неполными и противоречивыми, например, потому, что они сформулированы на живом разговорном языке, весьма образном, но, увы, неточном. Так, заказчик может потребовать особый графический интерфейс, предоставляющий необычные возможности ввода/вывода, плохо при этом представляя себе технические и технологические ограничения в реальных условиях. Другая трудность состоит в том, что программный проект, предложенный для решения целевой задачи, должен быть реализован на современных и вполне конкретных компьютерных (аппаратных) платформах и на основе имеющихся на сегодняшний день компьютерных технологий. Это значит, что стройная, а может быть, даже и изысканная архитектура программного проекта должна быть адаптирована (другими словами, искажена, иногда до неузнаваемости) с вполне определенной целью – заставить программный продукт работать, причем правильно и в требуемые сроки.

Таким образом, мы приходим к необходимости применять точные (по возможности математически точные) средства описания задачи, для решения которой заказчик попросил проектировщика найти компьютерную реализацию. К таким средствам обычно относят формальные языки, методы и даже целые технологии *спецификации*, или описания задачи и способа ее решения. Спецификация некоторого проекта обычно включает в себя описание поведения проектируемой системы, всех необходимых данных или программных объектов, моделирующих предметную область (тот реальный мир, в котором должно разворачиваться все действие), а также описание свойств и отношений объектов предметной области и все ограничения на них. Наличие такого точного описания свойств проекта также позволяет организовать их формализованную проверку или *верификацию* спецификации.

В некотором смысле история создания программных продуктов – это также и история поиска средств точной спецификации их программных проектов. Жесткие требования, предъявляемые к качеству и срокам выхода продукта на рынок, обычно

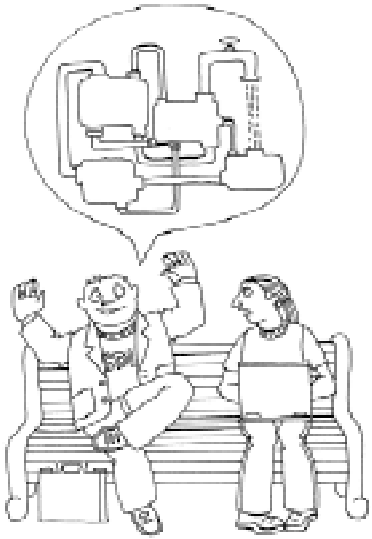
формулируются в виде особых промышленных стандартов, в соответствии с которыми ведется процесс проектирования (например, стандарты на технологии, языки, интерфейсы, протоколы и т. п. таких организаций и групп разработчиков как ISO, ANSI, IEEE, X/Open и др.). Эти требования заставляют специалистов постоянно пересматривать существующие подходы и предлагать усовершенствованные методы решения проблемы в виде новых языков и систем спецификации, успешно решающих задачу спецификации уже в новых условиях. В качестве примера можно привести такие известные и проверенные в индустрии программного проектирования методы спецификации как SDL, UML, MSC-диаграммы, развивающуюся технологию ASML и др., не говоря уже о многих экспериментальных университетских разработках.

Какими же средствами решали и решают задачу спецификации? Некоторые системы предоставляют возможности визуального моделирования программных проектов, строго определяя семантику (смысл) такого представления. К таким системам, например, относятся языки SDL и MSC (язык UML не всегда предоставляет точное описание). Некоторые системы позволяют специфицировать действующие прототипы программного продукта. К ним можно отнести язык AsmL и языки для описания конечных автоматов. В любом случае язык спецификации – это графический, логический или программный язык очень высокого уровня (по отношению к языку проектируемой системы), или *метаязык*.



Пример 1. Спецификация процедуры считывания PIN-кода на языке SDL (графическая форма записи) [1].

SDL

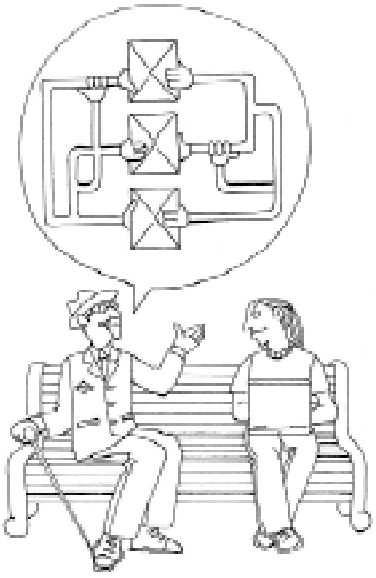


Язык SDL, один из наиболее распространенных и проверенных временем языков алгебраических спецификаций для формального описания программных продуктов (см. пример 1), был разработан и стандартизован организацией ITU-T (рекомендация Z.100) [2]. Первая версия языка SDL была выпущена в 1976 году, а обновления выходят каждые четыре года. В SDL были заложены главные свойства языка спецификации, такие как однозначность, ясность, точность, выразительность, отсутствие явного описания реализации. Язык SDL поддерживает концепцию модульности, дает возможность определять несколько уровней абстракции модели, так как SDL спецификация состоит из иерархии блоков. Взаимодействующие компоненты системы в языке

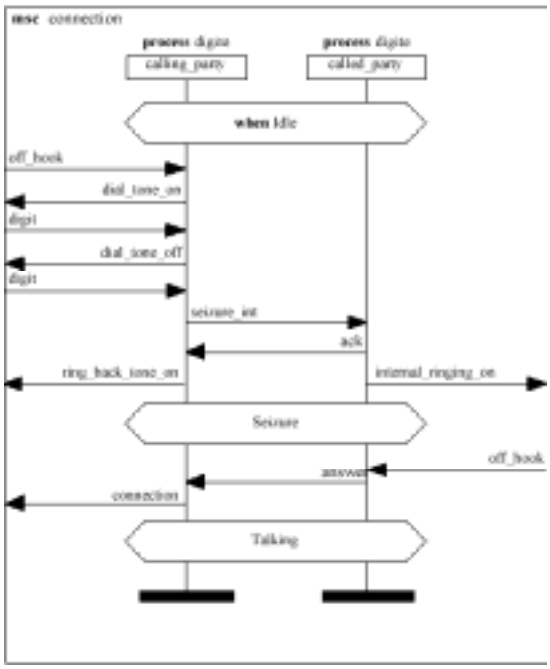
SDL моделируются с помощью *служб*, *процессов* и *блоков*. С помощью *каналов* блоки передают друг другу сообщения. Поведение блоков, являющихся конечными в этой иерархии, описывается с помощью одного или нескольких процессов, работа которых задается расширенными конечными автоматами. В настоящее время SDL поддерживает концепцию объектно-ориентированного подхода к проектированию на основе понятия *типа*, которое во многих системах и языках называется также *классом*. Кроме обычного текстового способа записи, язык SDL поддерживает также и графическую интерпретацию спецификаций, что позволяет программисту и заказчику визуально изучить описываемую модель системы, а также визуально сконструировать новую модель.

Язык SDL хорошо зарекомендовал себя в области телекоммуникаций, а сейчас широко применяется в проектировании систем реального времени, в распределенных и интерактивных системах. На данный момент существуют средства разработки, поддерживающие генерацию кода по SDL спецификациям, например, Telelogic Tau SDL Suite [5].

MSC



Язык описания Применение инженерии знаний в спецификации программных проектов MSC (Message Sequence Chart –



Пример 2. Спецификация взаимодействия на языке MSC (графическая форма записи).

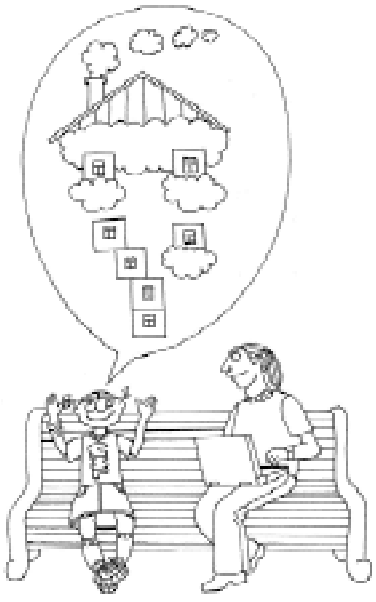
диаграмма последовательности сообщений) – это широко распространенный язык спецификации коммуникационного поведения компонентов некоторой системы и ее окружения посредством обмена сообщениями (см. пример 2). Первая рекомендация ITU по использованию MSC (Z.120) была опубликована в 1992 году [3]. Согласно этой рекомендации, язык MSC представляет собой мощное дополнение к языку SDL, описывая динамическое поведение проектируемой системы как результат трассировки ее SDL спецификации.

Основные средства описания диаграмм языка MSC – это *процессы*, *сообщения*, *условия* и *действия*. Каждая отдельная MSC диаграмма частично описывает поведение всей модели системы, а точнее, обмен сообщениями между процессами и выполнение некоторых действий в зависимости от условий. С помощью MSC можно осуществлять как графическое отображение работы симулятора SDL системы, так и верификацию модели на основании SDL спецификации. Хотя MSC удобно использовать вместе с SDL, область применения MSC не

ограничивается дополнением к SDL. Язык MSC может также служить для спецификации требований, накладываемых на программную систему, облегчения процесса проектирования путем выявления и документирования всевозможных случаев поведения, задания тестовых примеров. По диаграммам MSC можно строить тестовые модели поведения системы и с их помощью автоматизировать большую часть тестового процесса.

Документы MSC спецификации могут быть представлены в текстовой и в графической форме, что позволяет работать с ними как человеку, так и специализированной программе. Достоинствами языка MSC являются простота и наглядность, особенно в графической форме записи, поэтому его легко изучать и обрабатывать. В то же время MSC – достаточно мощный язык для использования в промышленных целях, особенно в совокупности с другими языками спецификаций, такими как SDL.

UML

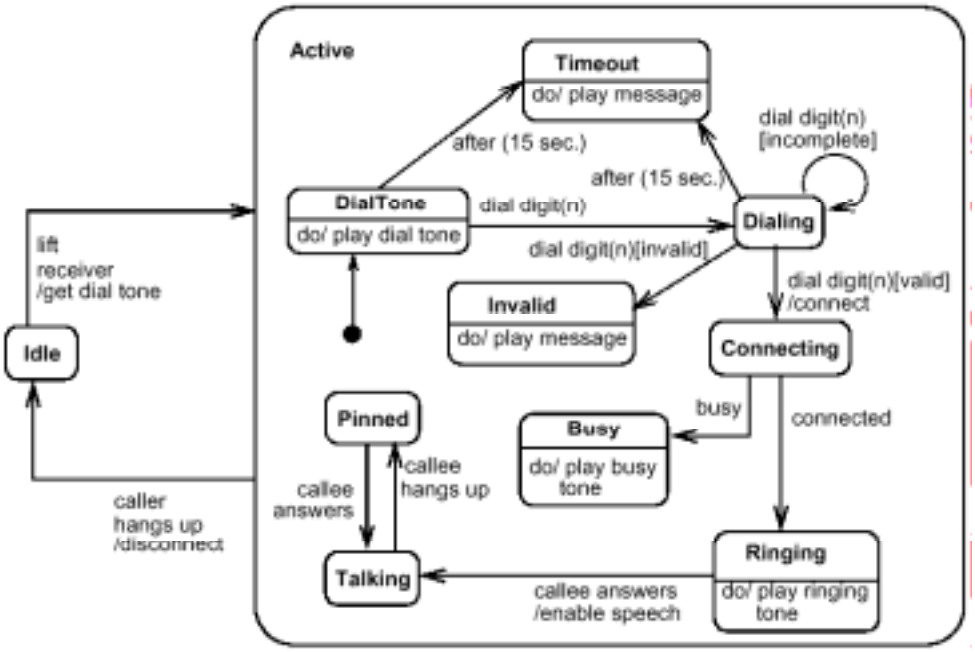


Язык UML [4] (Unified Modeling Language – унифицированный язык моделирования) является мощным средством полуформальной спецификации, визуализации, конструирования и документирования компонентов программных и электронных систем (см. пример 3). Он был разработан корпорацией Rational Software и ее партнерами в середине 90-х, объединив в себе основные концепции подходов Booch, OMT и OOSE, и был принят многими компаниями как стандарт, который охватывает такие области, как моделирование производственных и бизнес процессов, управление требованиями, анализ и проектирование, программирование и тестирование.

Язык UML сочетает в себе лучшие известные подходы в области моделирования больших и сложных систем. Главной особенностью UML является то, что он предлагает общие средства для моделирования практически любых систем. Моделирование в языке UML основывается на разных понятиях, таких как уровень абстракции, архитектура, область применения, жизненный цикл, технология реализации и так далее. Одной из основных целей разработчиков UML было создание общей метамодели языка, унифицирующей его семантику, а также общего способа записи, который позволяет человеку воспринимать эту семантику.

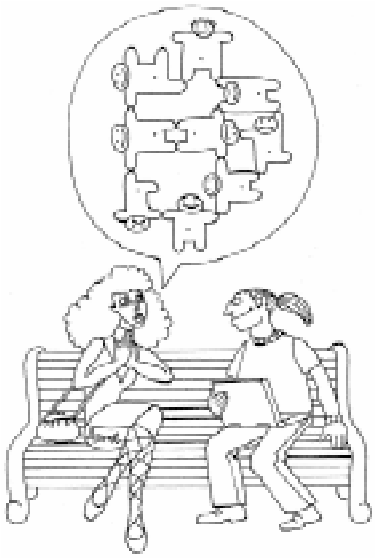
Создатели UML подчеркнули важность самого процесса разработки, который часто является ключевым фактором, определяющим, будет ли проект успешным или нет (подразумевается, что разработчики программных продуктов придерживаются некоторых общих правил работы и взаимодействия между собой). UML рекомендуется использовать для создания объектной модели системы на стадиях анализа и проектирования, если не требуется большой точности описания (которая присутствует, например, в SDL).

Для визуального описания моделей в языке UML существует несколько типов диаграмм, описывающих основные функции моделируемой системы, статическую структуру модели, взаимодействие объектов модели, состояния экземпляров классов моделируемой системы и переходы из одного состояния моделируемой системы в другое, особенности реализации и др.



Пример 3. Спецификация работы телефонного аппарата на языке UML. Диаграмма состояний.

ASML



Методология ASM (Abstract State Machine – машина абстрактных состояний) была разработана в начале 90-х в Мичиганском университете профессором Юрием Гуревичем [6]. За прошедшее десятилетие было проведено большое количество исследований, основанных на этой методологии. В конце 90-х в компании Microsoft под руководством Гуревича была разработана реальная система, использующая методологию ASM – язык AsmL [7] (Abstract State Machine Language – язык машин абстрактных состояний) (см. пример 4).

Основные достоинства AsmL – это простота, наглядность и возможность многоуровневой детализации программного проекта. Его синтаксис похож на синтаксис многих императивных языков, например, таких как Pascal или Basic, однако он обеспечивает намного больше возможностей, например, параллельное выполнение операций, гибкую работу с множествами, поддержку кванторов в логических выражениях и т. д.

AsmL является объектно-ориентированным языком и предназначен для моделирования, проектирования и прототипирования программных систем. Он позволяет точно описывать модель программной системы, явно задавая ее пошаговую работу.

AsmL – язык исполняемых спецификаций. Это означает, что разработчик спецификаций может исполнять построенные спецификации как обычный программный код с помощью подходящего интерпрета-

```
Main()
  initially F as File? = null
  initially FContents  = ""
  initially Mode       = "Initial" // Выделяем начальное состояние
  step until fixpoint
  // Повторяем, пока возможно применение одного из следующих правил
    if Mode = "Initial" then
    // Правило, которое выполнится один раз в первую очередь
      F    := Open("mfile.txt")
      Mode := "Reading"
    if Mode = "Reading" and Length(FContents) = 0 then
    // Читаем первый символ
      FContents := fread(F, 1)
    if Mode = "Reading" and Length(FContents) = 1 then
    // Читаем второй символ
      FContents := FContents + fread(F, 1)
    if Mode = "Reading" and Length(FContents) > 1 then
    // Переход в конечное состояние
      WriteLine(FContents)
      Mode := "Finished"
```

Пример 4. Спецификация программы, читающей содержимое файла mfile.txt на языке AsmL.

тора или генерировать исполняемый код с помощью компилятора, сразу же проверяя работоспособность и корректность поведения построенной модели.

Для иллюстрации возможностей AsmL на сайте компании Microsoft специалистам предлагаются несколько содержательных примеров готовых систем спецификаций, таких как Plug & Play протокол, спецификации языков SDL и UML. В качестве примера приведем фрагмент, описывающий работу с файлом.

Современная индустрия программного проектирования располагает целым рядом технологий создания программных продуктов, основанных на различных методах формальных спецификаций. Однако здесь мы сталкиваемся с неожиданной проблемой. Предлагаемые методы, включая приведенные выше примеры, часто используют синтаксически и семантически несовместимые языки спецификаций и способы представления знаний о программных проектах, что существенно затрудняет кооперацию групп разработчиков (из разных компаний или внутри одной организации).

Подобные трудности могут возникать при переносе накопленных результатов из старых проектов в новые, либо когда несколько компаний объединяют свои усилия для работы над одним проектом. Похожая проблема возникает при необходимости организовать обмен информацией (с минимальными ее потерями) между готовыми приложениями, спроектированными с применением различных технологий на основе различных систем понятий.

В качестве примера представим себе две базы с большим количеством разнообразной информации. Естественно, каждая из этих систем использует свой внутренний способ хранения информации. Теперь предположим, что создатели решили поделиться накопленной информацией или объединить два хранилища в одно. Оказывается, что автоматически это сделать очень сложно, так как изначально в этих базах использовались разные подходы к представлению информации. Например, в одном случае слово «мышь» использовалось для обозначения животного, а в другом подразумевалась компьютерная мышь.

Общепризнано, что задача преодоления барьера между различными методами спецификации под силу лишь человеческому интеллекту. Однако высокая трудоем-

кость этой задачи и настоятельная и каждодневная потребность в ее решении заставляют искать методы, по крайней мере, автоматизированного преобразования описаний проектных решений из одной технологии в другую. И именно здесь нам на помощь должен придти *искусственный интеллект* (ИИ) и методы, развитые в этой области компьютерной индустрии.

Для хотя бы частичного преодоления указанного синтаксического и семантического барьера между методами спецификации мы предлагаем для относительно близких областей информационных технологий определить базисный набор наиболее употребительных понятий и отношений и, насколько это возможно, стандартизировать методы их спецификации в виде *языка представления знаний* подходящей *базы знаний программных проектов* или *базы знаний спецификаций*, предназначенной для хранения фактических, декларативных и процедурных знаний о предметной области. Эти знания, сформулированные в достаточно общем виде и абстрагирующиеся от конкретного представления того или иного метода спецификации и дополненные «разумной» управляющей системой, должны исполнять роль интеллектуального ядра обобщенной технологии спецификации.

Экспорт и импорт. Представим, что несколько групп разработчиков используют N разных технологий спецификации. Чтобы свободно преобразовывать описания проектных решений из одной технологии в другую (и обратно) потребуется $N(N-1)$ преобразователей (конвертеров). С другой стороны, если использовать промежуточный язык спецификации, то потребуется $2N$ преобразователей. Очевидно, что если $N > 3$, то подход, использующий промежуточное представление, имеет существенное преимущество.

ИНТЕЛЛЕКТУАЛЬНЫЕ СИСТЕМЫ

Под *интеллектуальной системой* (ИС) понимают компьютерную систему (программу), способную имитировать те области деятельности человека, которые требуют мышления, определенного мастерства и накопленного опыта, другими словами, решать задачи ИИ. К задачам ИИ обычно относят задачи принятия решений, распознавания образов, понимания человеческого языка и многие близкие к ним задачи.

Методы ИИ уже давно и успешно применяются в таких областях, как органическая химия, поиск полезных ископаемых, медицинская диагностика, управление производством. При этом успех той или иной ИС в значительной мере определяется выбором конкретного способа представления знаний в этой системе. Чаще всего создатели ИС применяют такие методы представления знаний, как канонические системы с порождающими правилами (продукционные системы), языки функционального и логического программирования (например, Лисп и Пролог), графы, деревья, ассоциативные сети, фреймы, нейронные сети, онтологические библиотеки. В любом случае выбранный метод должен обеспечивать гибкие средства управления символьными данными, которые обычно предоставляют языки логического и функционального программирования.

С 90-х годов ведутся исследования и разработки систем спецификаций, направленных на работу со знаниями. В качестве примера можно привести такие системы и языки как KIF [9], PIF [12], PSL, SHOE [11]. Новые направления предлагают более общий подход к решению этой задачи, многие из них в качестве основы используют понятие *онтологии*, часто понимая под этим модель базы знаний некоторой предметной области, систематизирующую терминологию, понятия и отношения этой предметной области. В качестве примера таких систем следует упомянуть Ontolingua [8], Cyc [10], LOOM, KAKTUS [13].

БАЗА ЗНАНИЙ СПЕЦИФИКАЦИЙ

Прежде всего выберем язык спецификации программных проектов, поддерживающий описание понятий и отношений между понятиями, объектов и их свойств (атрибутов), а также процедурных (алгоритмических) знаний. Другими словами, мы выбираем *метаязык*, позволяющий описать синтаксис (формат хранения знаний) и семантику

языка представления знаний того или иного конкретного программного проекта. Синтаксис языка представления знаний можно позаимствовать из таких устоявшихся языков, как LISP или XML, поскольку они позволяют стандартным образом записывать информацию описательного характера, а также с легкостью расширять сам язык. В данном проекте мы останавливаем свой выбор на синтаксисе языка LISP. (Помимо прочего, простота внутреннего языка представления знаний дает возможность пользователю исправлять или добавлять информацию вручную).

LISP

Язык LISP – один из первых языков обработки символьных структур данных. Он был разработан в 1960 году и за четыре десятилетия модифицировался и расширился. Однако в своей основе LISP изменился мало. От прочих языков программирования LISP отличается тем, что основной структурой данных в нем является список; программы на этом языке также имеют списочную структуру, базовые операции языка – это операции над списками. Сейчас большинство языков программирования общего назначения тем или иным образом поддерживают операции со списочными структурами, хотя от программистов обычно требуется выделять память для формирования списка, а после работы возвращать выделенную память системе. В LISP еще на ранних стадиях развития в исполняющую систему была встроена поддержка автоматизации этого метода работы с памятью (сейчас этот механизм обычно называют «сборкой мусора»). В качестве примера программы на языке LISP приведем следующее выражение, которое возвращает максимальное из двух чисел *X* и *Y*: `(cond ((< X Y) Y) (T X))`.

Мы будем говорить, что выбранные нами метаязык и структура базы знаний определяют *онтологию* проектирования. Эта онтология, дополненная необходимыми логическими средствами формулирования и доказательства свойств программных проектов, и составляет нашу интеллектуальную систему спецификации.

Верификация (автоматическая или полуавтоматическая) правильности спецификаций. Нередко при спецификации или программировании возникают неточности, опечатки и, самое опасное, логические ошибки проектирования, которые незаметны пользователю и не могут быть обнаружены при синтаксическом анализе. С другой стороны такие ошибки могут быть выявлены при использовании некоторых эвристик, характерных для конкретной предметной области, и верификационного алгоритма, который может доказать или опровергнуть некоторые свойства модели. Это может быть полезно как при обычной проверке целостности спецификаций программного проекта, так и при доказательстве сложных утверждений о согласованности разнообразных знаний относительно какой-либо части проекта.

Рассмотрим некоторые примеры представления знаний о проектах. Каждый факт базы знаний представляется в виде списка внутреннего языка. А каждый список состоит из головы – имени сущности, свойства которой он описывает, и набора элементов, отображающих полную информацию о свойствах этой сущности, например:

```
(Project (id 1) (level 1) (name Simple_Stack_Machine)
(desc "Specification of a Simple Stack Machine."))
```

Листинг 1.

```
(Class (id 1001) (name Resource))
 (Attribute (id 5) (name flag) (classID 1001) (varType Integer)
            (value 1))
 (Function (id 2) (level 1) (name captureRes) (modifier public)
           (retType Bool) (classID 1001))
  (SeqBlock (id 7) (funID 2) (ownerType Function) (ownerID 2))
   (IfThen (id 8) (funID 2) (condition "flag<=0")
           (ownerType SeqBlock) (ownerID 7))
    (Return (id 9) (funID 2) (ownerType IfThen) (ownerID 8)
            (expr FALSE))
    (Else (id 10) (funID 2) (ownerType IfThen) (ownerID 8))
     (ParBlock (id 11) (funID 2) (ownerType Else) (ownerID 10))
      (Decrement (id 12) (funID 2) (ownerType ParBlock) (ownerID 11)
                 (varID 5))
      (Return (id 13) (funID 2) (ownerType ParBlock) (ownerID 11)
              (expr TRUE))
```

Голова списка (Project) говорит о том, что данный факт описывает свойства некоторого программного проекта. Остальные элементы задают конкретные значения этих свойств.

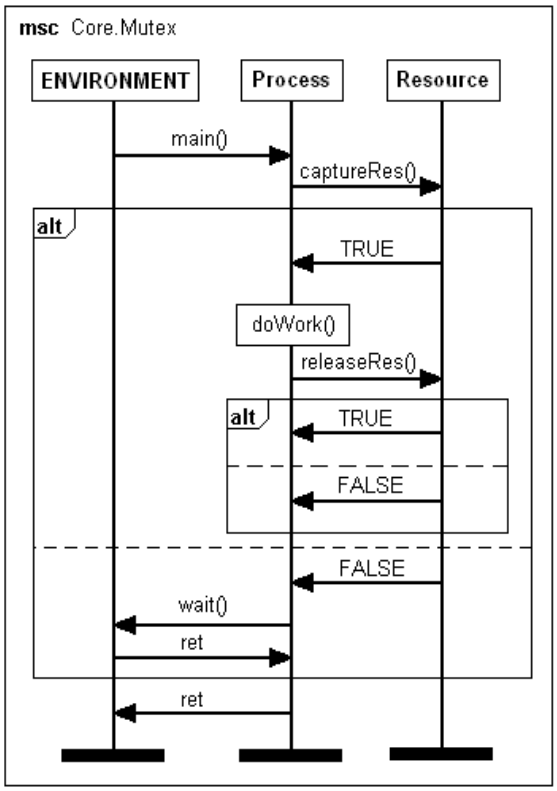
Изначально структура базы знаний содержит фиксированный набор понятий (например, Project), которые можно использовать при описании программного проекта. В то же время, эта структура предусматривает возможность дальнейшего расширения и уточнения системы понятий и отношений.

Разберем простой пример спецификации – фрагмент спецификации взаимоисключающего взаимодействия нескольких процессов с некоторым ресурсом. Данный фрагмент определяет функцию, с помощью которой ресурс захватывается некоторым процессом (см. листинг 1).

Приведенный фрагмент спецификации содержит только факты. Помимо фактов база спецификаций может содержать правила (процедурные знания). Правила состоят из двух частей. Левая часть сопоставляется с одним из фактов базы, а правая выполняется при удачном сопоставлении. Правила позволяют определенным образом интерпретировать и конвертировать накопленные знания в другие технологии и языки спецификации.

Покажем теперь, как работает механизм экспорта/импорта в интеллектуальной технологии спецификации. В качестве примера приведем фрагмент кода предыдущей спецификации, полученный при конвертировании в язык AsmL (см. листинг 2).

Пример 5 отображает результат конвертации приведенной спецификации в язык MSC.



Пример 5. MSC-диаграмма взаимодействия процесса с ресурсом.

Листинг 2.

```
class Resource
// определяем класс, инкапсулирующий данные и методы ресурса
    flag as Integer = 1
   // данный атрибут отображает занятость ресурса (1 = свободен)
    public captureRes() as Bool // метод для захвата ресурса
        step
            if ( flag <= 0 ) then
                return FALSE
            else
                flag—          // помечаем ресурс как захваченный
                return TRUE
    public releaseRes() as Bool // метод для освобождения ресурса
        step
            if ( flag >= 1 ) then
                return FALSE
            else
                flag++         // помечаем ресурс как свободный
                return TRUE
```

В заключение отметим, что полный набор конвертеров экспорта/импорта может быть получен только в условиях реального применения предлагаемой авторами интеллектуальной технологии спецификации.

Ссылки.

[1] TIMe Electronic Textbook v 4.0. Tutorial on SDL. 16.07.1999.
[2] Recommendation Z.100 (1993), CCITT Specification and Description Language (SDL).
[3] Recommendation Z.120 (1993), CCITT Message Sequence Charts (MSC).
[4] http://www.omg.org/technology/documents/formal/uml.htm
[5] http://www.telelogic.com
[6] http://research.microsoft.com/~gurevich
[7] http://research.microsoft.com/foundations/AsmL
[8] http://www-ksl-svc.stanford.edu
[9] http://logic.stanford.edu/kif
[10] http://www.cyc.com
[11] http://www.cs.umd.edu/projects/plus/SHOE/
[12] http://soa.cba.hawaii.edu/pif/
[13] http://www.swi.psy.uva.nl/projects/NewKACTUS/library/library.html

Васильев Павел Константинович,
аспирант математико-механического факультета СПбГУ,
сотрудник лаборатории исследования операций НИИММ СПбГУ,

Соловьев Игорь Павлович,
доцент кафедры информатики математико-механического факультета СПбГУ.