УДК 681.3.06

С. А. КОВАЛЕВ

ОСНОВНЫЕ ЧЕРТЫ ОБЪЕКТНО-ОРИЕНТИРОВАННОЙ ПАРАДИГМЫ ПРОГРАММИРОВАНИЯ

Изложим свою точку зрения на так называемую объектно-ориентированную парадигму программирования (ООПар). Предмет исследования рассмотрим более широко по сравнению с устоявшимся термином "объектно-ориентированное программирование" (ООП), сместив акцент с этапов реализации классов и сборки программной системы на этапы анализа и проектирования. Использование термина "парадигма" подчеркивает комплексность подхода к разработке программных систем. Например, Тимоти Бадд [1] говорит об "объектно-ориентированном мышлении". Большинство разработчиков утверждают об особом процессе разработки объектно-ориентированных программ, о специфическом объектном менеджменте. Некоммерческая организация Object Management Group (OMG) [2] была основана в апреле 1989 г. одиннадцатью ведущими компаниями рынка информационных технологий. Целью группы является создание и развитие рынка компонентного программного обеспечения посредством ускорения стандартизации объектного программного обеспечения. Наиболее известными разработками OMG являются Common Object Request Broker Architecture (CORBA) [3] – стандарт де-факто распределенных вычислений, обеспечивающий гибкую возможность взаимодействия и интеграции разнородных приложений, – также широко известен Unified Modeling Language (UML) [4, 5, 6], попытка стандартизации языка и графической нотации при объектно-ориентированном (ОО) анализе и последующем проектировании.

Широкое применение термина "парадигма" вызвано публикацией Томасом Куном "Структуры научных революций" [7]. Кун использовал это слово для описания набора стандартов, теорий и методов, которые совместно представляют собой способ организации научного знания – способ видения мира. Основное положение Куна состоит в том, что революции в науке происходят, когда старая парадигма пересматривается, отвергается и заменяется новой.

По отношению к программированию "парадигмы" применены в 1979 г. Робертом Флойдом. Т. Бадд [1] определяет программные парадигмы как "способ концептуализации, который определяет, как проводить вычисления и как работа, выполняемая компьютером, должна быть структурирована и организована". В настоящее время выделяют такие парадигмы, как директивная или процедурная (выражается в таких языках программирования, как Pascal, C), логическая (Prolog), функциональная (Lisp, FP, Haskel) и, естественно, последняя (в смысле новейшая, хотя апологеты данного подхода утверждают, что окончательная [8]) объектно-ориентированная. ООПар стремительно развивается, показателем может служить наличие более 170 объектно-ориентированных языков [9], разработанных за сравнительно короткое время.

Естественно, целью использования любой парадигмы программирования является разработка программ. Для ООПар концерн OMG определяет термин "объектный менеджмент" (object management) как разработку программного обеспечения, в ходе которой реальный мир моделируется посредством представления "объектов". Эти объекты включают в себя атрибуты, связи и методы идентифицируемых программных компонент. Результатом использования объектного менеджмента является более быстрая разработка программ, более легкая их поддержка и сопровождение, а также просто громадная масштабируемость и возможность повторного использования.

Итак, разрабатывая программное обеспечение (ПО), жизненный цикл программ чаще всего разбивается на три части: анализ, проектирование и программирование. В рамках ООПар используется объектно-ориентированный анализ (ООА – objectoriented analysis),

объектно-ориентированное проектирование (OOD – objectoriented design) и объектно-ориентированное программирование (OOP – objectoriented programming). Вот как Г. Буч, один из ведущих исследователей, занимающийся проблемами ООПар, определяет эти понятия. ООА – "методология, при которой требования к системе воспринимаются с точки зрения классов и объектов, выявленных в предметной области" [10, с. 54]. OOD – "методология проектирования, соединяющая в себе процесс объектной декомпозиции и приемы представления логической и физической, а также статической и динамической модели проектируемой системы" [10, с. 53]. OOP – "методология программирования, основанная на представлении программы в виде совокупности объектов, каждый из которых является экземпляром определенного класса, а классы образуют иерархию наследования" [10, с. 52]. Следовательно, на результатах ООА формируются модели, на которых основывается OOD; OOD в свою очередь, создает фундамент для окончательной реализации системы с использованием методологии OOP.

Для каждой парадигмы программирования характерны свое умонастроение, свой способ восприятия решаемой задачи. Так. писать программы в ОО стиле можно практически на любом языке программирования, но удобнее это делать на специально приспособленных, поддерживающих и поощряющих такой стиль разработки, ОО языках. Аналогично, используя только ОО язык, не принимая во внимание его концепции, мы не только потеряем их потенциал, скорее всего результат смеси ОО языка и, например, алгоритмической декомпозиции, будет даже хуже, чем при использовании просто алгоритмической парадигмы. Основой ООПар, ее концептуальной базой является объектная модель. Она имеет четыре неотъемлемых элемента (без них модель не будет объектной): абстрагирование, инкапсуляция, модульность, иерархия. Также полезны (но не обязательны) – типизация, параллелизм, сохраняемость.

Практический успех ОО технологии зависит от используемой методологии – интегрального, сформулированного в виде эмпирических методов, методик и приемов, подхода к основным фазам процесса разработки (ООА, OOD, OOP). В конце 80х – начале 90х наблюдался бум в создании новых методологий, которые разрабатывались, как правило, независимо друг от друга и практически без учета опыта. Создано около 50 ОО методов. Среди первых методологий можно выделить следующие (наряду с собственными названиями, в практике используются имена их создателей): Booch'91, OMT (Object Modelling Technique – автор James Rumbaugh), Object-Oriented System Analysis and Recursive Design (Shlaer/Mellor – Меллор разработал свой ОО язык – Eiffel), CRC (Class, Responsibility and Collaborations), Responsibility-Driven Design (разработчики одного их ОО языков Smalltalk – WirfsBrock, Beck/Cunningham), Object-Oriented Analysis and Design (Coad/Yourdon) и мн. др. Интенсивный обмен идеями во многочисленных публикациях, накопление опыта реальных разработок привели к появлению методологий второго поколения. Например, Booch'93 полностью вытеснил свой ранний вариант Booch'91, а методология Fusion использует CRC, Booch и OMT.

Только в первые несколько лет девяностых годов бурно расцвели около пятидесяти различных ОО методов. Настолько широкое распространение является показателем жизненности ОО технологии, но и также является плодом множества интерпретаций (взглядов) понятия объект. Отрицательной стороной данного изобилия методологий явилось поощрение беспорядка (путаницы), вынудившей пользователей занять позицию "подождем и посмотрим", что негативно отражается на развитии методов. Наилучший путь протестировать что-то – это использовать его, так как методы не застывают в камень – они развиваются в ответ на комментарии пользователей. К счастью, более пристальное рассмотрение доминирующих методов позволяет выявить единодушие относительно основных общих идей. Основные характеристики объектов, разделяемые многими методами, выражаются концепциями класса, ассоциации (описанной Джеймсом Рамбо), деления на подсистемы (Гради Буч)

и выражением требований, основанном на изучении взаимодействия между пользователями и системой (варианты использования Ивари Якобсона). В итоге, хорошо проработанные методы, такие, как Booch и OMT, были подкреплены опытом практических разработок и вобрали в себя наиболее высоко оцененные методологические элементы.

Второе поколение методов Booch и OMT, известные как Booch'93 и OMT-2, было более похоже друг на друга, чем их предшественники. Оставшиеся различия были незначительны и касались в основном терминологии и нотации. Booch'93 под влиянием OMT вобрал в себя ассоциации, диаграммы Харела (Harel) и отслеживание событий. В свою очередь OMT-2 под влиянием метода Booch ввел в свой технический арсенал потоки сообщений, иерархические модели и подсистемы, а также модельные компоненты. Однако более важным изменением было удаление диаграмм потоков данных из функциональной модели. Эти диаграммы были унаследованным из функциональной (алгоритмической) парадигмы программирования багажом и не сочетались с общим подходом OMT.

На этой стадии оба метода предлагали полное покрытие всего жизненного цикла программы, но расставляли разные акценты на этапы и методики. Booch'93 фокусировался на этапе реализации, а OMT-2 концентрировался на абстракциях и анализе. Тем не менее серьезных несовместимостей между этими двумя методами не было.

История ОО концепций часто очень сложна и запутана (что лишний раз подтверждает отсутствие четкой и всеобъемлющей теории). Эмпирические элементы разнообразных методологий имеют значительное влияние на эффективность и направление усилий по унификации методов (табл. 1).

Таблица 1

Элементы методологий

Происхождение	Элемент
Booch	Категории и подсистемы
Embley	Классы-синглетоны и композитные объекты
Fusion	Описание операций, нумерация сообщений
Gamma et al	Каркасы (frameworks), образцы (patterns) и примечания
Harel	Диаграмма состояний
Jacobson	Варианты использования (Use cases)
Meyer	Пред- и пост- условия
Odell	Динамическая классификация, акцент на события
OMT	Ассоциации
Shlaer-Mellor	Жизненный цикл объектов
Wirfs-Brock	Ответственности и сотрудничество (collaborations)

Непрерывное развитие всех методологий, проблемы в их поддержке со стороны CASEсредств привели к назревшей необходимости унификации методологий или хотя бы нотаций и терминов. Как показано ранее, непреодолимых препятствий к объединению и унификации у отцов-основателей не было. Не без усилий OMG под крышей компании Rational Software разрабатывается Unified Modeling Language (Унифицированный Язык Моделирования), также осуществляется унификация методов. В 1994 г. Джим Рамбо (OMT – 40 % американского рынка на 1994 г.) покинул General Electric и присоединился к Гради Бучу в Rational Software (Booch'93 – 11%). В конце 1995 г. к ним присоединился еще один известный методолог Ивар Якобсон (Ivar Jacobson), автор OOSE (ObjectOriented Software Engineering). Это методология (более известная под названием Objectory) особо подходит для проектирования больших систем, в частности телекоммуникационных (сказывается опыт рабо-

ты Якобсона с телефонными системами фирмы Erricsson). Объединными усилиями "троих друзей" постоянно выпускаются новые версии UML и проводится разработка унифицированного метода (выпуск книги с описанием которого анонсирован на конец 2000 г.).

Несмотря на разнообразие методологий, все же присутствуют общие идеи, понятия, хотя и с разнобоем в терминах. Вкратце опишем ключевые концепции ООПар. Термины, выбранные нами для использования, будем выделять курсивом.

Действие в ОО программах инициируется посредством передачи *сообщений* агенту (объекту), ответственному за действие. Сообщение содержит запрос на осуществление действия и сопровождается дополнительной информацией (аргументами), необходимой для его выполнения. Получатель сообщения, если он принимает его (несет *ответственность* за выполнение этого действия), запускает некоторый *метод* (процедуру, функцию) для удовлетворения принятого запроса. Маскировка информации (*инкапсуляция*) заключается в том, что посылающему запрос клиенту не требуется знать о фактических средствах, с помощью которых его запрос будет удовлетворен. Различие между вызовом процедуры и пересылкой сообщения состоит в том, что в последнем случае существует определенный получатель и интерпретация (т. е. выбор подходящего метода, который запускается в ответ на сообщение) может быть различной. В ОО языках это реализуется механизмом *позднего связывания* между сообщением (именем метода) и фрагментом кода, это происходит во время выполнения программы (а не компиляции). Такое связывание противопоставляется *раннему*, осуществляемому при традиционных вызовах процедур. *Ответственность* (или обязанность) выражается также в свободе выбора способа, которым получатель сообщения обеспечивает желаемый результат, заказанный в запросе. Полный набор обязанностей, связанных с определенным объектом, называется *протоколом*. Все объекты являются представителями (или *экземплярами*) *классов*. Все объекты одного класса используют одни и те же методы в ответ на одинаковые сообщения. Принцип, в соответствии с которым знание о более общей категории (классе) разрешается использовать для более узкой категории, называется *наследованием*. Классы организуются в иерархическую структуру с наследованием свойств. Дочерний класс (*подкласс*) наследует атрибуты родительского класса (*надкласса*), расположенного выше в дереве (или графе для множественного наследования). *Абстрактный класс* – класс, не имеющий экземпляров, – используется только для порождения подклассов. Информация, содержащаяся в подклассе, может переопределять (*перекрывать*) информацию, наследуемую из родительского класса.

Вот как определяет основные термины и концепции один из ведущих ученых [11, 12] – Эдвард Берард. *Объекты* – это материальные и концептуальные предметы, которые мы можем найти в окружающем универсуме. *Состояние* (state) объекта понимается как условие, состояние, режим, ситуация (condition) объекта и набор ограничений, описывающих объект. Для сложных объектов полное описание состояния может быть очень сложным. Когда мы используем объекты для моделирования реальных или представляемых ситуаций, мы, типично, ограничиваем возможные состояния объекта только теми, которые существенны (важны) для нашей модели. Состояние большинства объектов не изменяется до тех пор, пока чтолибо снаружи объекта не потребует сменить состояние. Такие объекты называются *пассивными*. *Активными* объектами (деятелями, актерами) называются объекты, способные спонтанно изменять свое состояние. Берард различает три точки зрения на понятие класса. В первой, "класс как резак для печенья", класс трактуется как образец, шаблон, проект для категории структурно идентичных элементов. Созданные таким образом элементы называются *экземплярами* (instance) и соответствуют "печенью" в предлагаемой метафоре. Другая точка зрения выражается фразой "класс как фабрика экземпляров". Класс состоит из образца и механизма создания элементов по этому образцу. Экземпляры есть индивидуальные элементы, "произведенные" "творительным" механизмом класса. С третьей точки зрения,

класс – совокупность элементов, созданная по определенному образцу. Иначе говоря, класс –это множество экземпляров этого образца. Берард описывает термином "объект" как классы, так и их экземляры.

С точки зрения "фабрики экземпляров" *метакласс* – класс, экземплярами которого являются другие классы. *Параметризированный класс* – шаблон для непараметризированного класса, для создания которого необходимо определить специфические компоненты шаблона.

Объекты рассматриваются как "черные ящики". Внутренняя реализация объекта прячется от тех, кто его использует. Пользователи класса не нуждаются в знании "*как*" работает объект, они должны использовать знание о том, "*что*" объект может для них сделать. Работа с объектом осуществляется через три *интерфейса*.

Общедоступный, публичный интерфейс открыт (виден) для всех. Интерфейс наследования доступен только при прямой специализации объекта (подклассы для класс-ориентированных ОО систем). *Параметрический интерфейс* определяет те параметры параметризированного класса, которые необходимо определить для создания экземпляра. Когда элемент описывается в публичном интерфейсе, мы говорим, что объект экспортирует этот элемент. Аналогично, если объекту требуется информация снаружи (например, параметры для параметризированного класса), то говорят об импорте этой информации.

Агрегация – это либо процесс создания нового объекта из двух или более объектов, либо сам объект, созданный таким образом. *Монолитный* объект имеет неразличимую снаружи структуру, он не кажется созданным из других объектов и рассматривается как единое целое. Ничто снаружи монолитного объекта не может прямо взаимодействовать с любыми (воображаемыми или реальными) объектами, находящимися внутри монолитного. *Композитные* объекты имеют структуру, различимую снаружи, к которой можно обращаться через публичный интерфейс. Объекты, составляющие композитный объект, называются *компонентными*. Композитные объекты соответствуют одному или двум критериям: на состояние композитного объекта прямо влияет наличие или отсутствие одного или более его компонентных объектов и/или на компонентные объекты можно прямо ссылаться через публичный интерфейс соответствующего композитного объекта. *Гетерогенные композитные* объекты состоят из концептуально разных компонентных объектов. Например, "дата" состоит из "дня", "месяца" и "года". *Гомогенные композитные* объекты состоят из концептуально единых компонент. Например, "список адресов" – гомогенный композитный объект.

Специализация – это либо процесс определения нового объекта, как правило, основанного на более узком определении уже существующего объекта, либо объект, определенный более узко, чем другой, относящийся к нему объект (называемый *обобщением*). В ОО контексте, мы говорим о специализации как о "наследовании" характеристик от соответствующих обобщений. *Наследование* может быть определено как процесс, посредством которого один объект приобретает характеристики одного и более других объектов, и, соответственно, разделяется на *одиночное* и *множественное* наследование. *Абстрактный класс* воплощает согласованную и связную, но неполную концепцию. Мы никогда не сможем создать экземпляр абстрактного класса, а должны специализировать его, дополнив посредством наследования.

Операция, находящаяся в публичном интерфейсе, рекламирует функциональную способность соответствующего объекта. На действительный алгоритм выполнения операции ссылаются как на *метод*. В отличие от операций, методы прячутся внутрь объекта. Иногда операции, находящиеся в общедоступной части интерфейса, называют *допустимыми операциями* или *экспортируемыми*. Существует три большие разновидности экспортируемых операций. *Селектор* – это операция, выдающая информацию о состоянии объекта и по определению не имеющая права его изменять. *Конструктор* – операция, способная изменять состояние объекта (некоторые ограничивают это понятие только операциями создания эк-

экземпляров класса). В контексте гомогенных композитных объектов *итератор* представляет собой операцию, позволяющую получить доступ к компонентным объектам. *Активные* (*открытые*) итераторы сами являются объектами. *Пассивные* (*закрытые*) реализованы в виде операций в интерфейсе объекта, к содержимому которого они будут предоставлять доступ. Опираясь на изложенное выше, пассивные *селекторные* операции не имеют права модифицировать состояние объектов, в отличии от *конструктивных*. *Примитивными* называют те операции, которые не могут быть реализованы еще проще (примитивнее), эффективнее и надежнее без прямого знания лежащей в основе (к тому же спрятанной) внутренней реализации объекта. Например, для объекта "список" операции "удалить элемент" и "добавить элемент" являются примитивными, а операция "обменять элементы" – нет. Следовательно, *композитными* (*составными*) являются операции, составленные из других (или имеющие возможность такого составления). Иногда объекты, чтобы поддержать свои характеристики, нуждаются в помощи. *Требуемые* (*импортируемые*) операции необходимы для поддержания внешне различимых характеристик, но которые объект предоставить сам не может.

Константы – это объекты, имеющие неизменное (константное) состояние. Часто это экземпляры простейших типов, предоставляемые в интерфейсе объекта для удобства его использования. Например, такой константой является число в публичном интерфейсе ограниченного списка, обозначающее максимально возможное количество элементов в таком списке. В некоторых случаях, таких, как алгоритмы шифрования, константы абсолютно необходимы.

Кроме операций и констант, в публичном интерфейсе объектов могут находиться также исключения (exceptions). *Исключение* имеет два разных определения: это событие, временно прекращающее нормальное выполнение приложения, а также набор информации, прямо относящейся к такой задержке работы программы. Когда встречается нестандартная ситуация (но не непредусмотренная), *активируется* соответствующее исключение. Также используются термины *возбуждать* и *бросать* исключения (raise, throw). Как только активируется исключение, прекращается нормальное выполнение программы и управление передается локальному *обработчику* соответствующего исключения. Если же такого нет или он не предназначен для оперирования данным исключением, говорят о *распространении исключения*, т. е. о передаче управления на более высокий уровень приложения. Основное отличие исключений от кодов ошибок, использовавшихся ранее, состоит в том, что исключения невозможно игнорировать. Не будучи обработанным локально, оно распространяется все выше и выше, пока не будет обработано или пока не прекратит функционировать все приложение. Обработчик проверяет тип исключения и при опознании предпринимает определенный набор действий, деактивируя, таким образом, исключение. Выполнение описанных действий и называется *обработкой исключения* (handling the exception).

Систему, компоненты которой весьма независимы, легче исправлять и развивать, чем систему с сильными взаимозависимостями между компонентами. Высокая независимость компонент возможна при минимальном сцеплении компонент и их большой связности. *Сцепление* (coupling) – это мера силы соединения между двумя частями системы. Чем больше один компонент знает о другом, тем теснее (хуже) сцепление между ними. *Связность*, сплоченность (cohesion) – мера логических соотношений частей компонента между собой и целым компонентом. Чем больше логически относятся друг к другу части, тем выше (лучше) связность этого компонента. *Объектное сцепление* описывает степень взаимосвязей объектов, составляющих систему. Для построения системы, мы должны связать (сцепить) до некоторой степени объекты, которые необходимы в системе. Это *необходимое объектное сцепление*. Тем не менее при проектировании одного объекта, снабжая его прямым знанием о других объектах, мы делаем лишние связи. *Ненужное объектное* сцепление уменьшает как возможность повторного использования отдельных объектов, так и надежность всей системы, составленной из таких объектов. С другой стороны, *объектная связность* – это мера

логического отношения компонент объекта друг с другом при рассмотрении снаружи этого объекта. Например, опишем объект "дата" как состоящий из объектов "день", "месяц", "год" и "зеленый цвет". Мы должны признать, что объект "зеленый цвет" не совсем подходит для определения "даты" и понижает связность описываемого объекта. Необходимо стремиться к более высокой связности объектов по двум причинам. Вопервых, обладающие низкой связностью объекты с высокой степенью вероятности будут изменяться в дальнейшем, более вероятны нежелательные побочные эффекты при таком изменении. Вовторых, плохая связность объектов не способствует их более легкому повторному использованию.

При разработке ОО моделей и программ быстро пришли к заключению о недостаточности одиночных классов и экземпляров. Необходимо создавать большие объекты и работать с ними. *Системы объектов* определяют как два или более взаимодействующих или взаимоотносящихся невложенных объекта. Таким образом, из определения исключаются простые агрегации композитных объектов. Системы объектов разбиваются на две большие категории: комплекты (наборы, конструкторы) и системы взаимодействующих объектов. *Комплект* (kit) – набор элементов (классов, метаклассов, параметризированных классов, неклассовых экземпляров, других комплектов и/или систем взаимодействующих объектов), предназначенных для поддержки единственной, большой, связной (coherent), объектно-ориентированной концепции, такой, как компьютерные графические окна или страховой полис. Несомненно, некоторые элементы такого набора могут быть физически сцеплены. Тем не менее комплекты обычно "зернистые". Хотя все компоненты набора логически связаны, существует мало физических связей, соединяющих их вместе. Комплекты больше похожи на библиотеки в алгоритмической парадигме. Например, коллекция разнообразных окон и оконных компонент может рассматриваться как комплект. *Система взаимодействующих объектов* (СВО) также представляет набор подобных элементов, поддерживающих, как и в комплектах, единую концепцию. Но некоторые элементы должны быть прямо или косвенно физически соединены. Более того, СВО имеют как минимум один внутренний, независимо выполняющийся поток управления. Наконец, СВО могут выставлять несколько полностью непересекающихся интерфейсов. Системы взаимодействующих объектов подобны приложениям. Например, необходимо создать систему управления лифтами в некотором здании. Мы собираем "лифты", "лампы", "панели" и другие объекты в одно работающее приложение. Оно является единым целым с высокой степенью связности и не может рассматриваться в качестве библиотеки. Такое приложение и является СВО.

В литературных источниках по ОО проблематике наиболее часто обсуждаются преимущества ООПар, со всех сторон на пытливого исследователя сыпятся хвалебные отзывы, рекламные прокламации. И очень мало информации о трудностях разработки самого ОО подхода, его недостатках и слабых местах. Приведем некоторую информацию [13] именно о недостатках ООПар (табл. 2) и ввиду ограниченности объема журнальной статьи перенесем обсуждение в следующие работы.

Основная цель данной статьи – описание основных черт объектно-ориентированной парадигмы программирования и выражение точки зрения автора – выполнена. Помимо этого необходимо отметить, что выполнены и другие сопутствующие задачи: даны определения основных терминов, их точное наименование и смысл, вкладываемый автором, эти достижения необходимы для более плодотворной дальнейшей работы и наиболее полного понимания читателем формулируемых в этой и последующих статьях мыслей. Показана необходимость и актуальность выполнения дальнейших исследований по данной тематике. Следующим шагом представляется привлечение идей системологии [14] к попытке решить выявленные проблемы, решению частных задач ООПар (повторное использование, идентификация ключевых абстракций и пр.) [15].

Таблица 2

Недостатки ООПар

Проблема	Краткое описание
Определение и обнаружение задач	ООПар слаба в определении проблем (задач) в слабоструктурированных проблемных областях
ООПар & натуральность	ООПар не так натуральна, как многие полагают
ООПар & натуральность 2	Прорехи в концептуальной модели могут обернуться неправильными применениями и ошибками
Множественные и противоречивые представления (views)	ООПар не предоставляет адекватную поддержку множественных представлений
SQL представления	Так же
Ad-hoc запросы	В чистой ОО модели нет поддержки ad-hoc запросов (запросы, составленные "по случаю", при неотложной необходимости получить результаты сейчас, в противопоставление жестко закодированным в ОО методах запросах к БД). Поддержка этого противоречит принципу инкапсуляции
OIDs и взаимодействие с реальным миром	OIDs (идентификаторы объектов) не взаимодействуют содержательно с реальным миром
OIDs и семантика PART-OF отношений	Никакой семантики к OIDs не присоединяется, как следствие, слабое представление сущностей (entities)
Проблема размещения методов (кода операций)	Размещение методов важно в базах данных – не рассматривается в рамках ООПар
ООПар и знания человека	В ООПар предположения о знаниях человека не настолько аккуратны, как утверждают многие
Рассмотрение всего в качестве объектов	Не подходит в области баз данных
Поддержка представлений	В ООПар нет поддержки представлений (то же для реляционных представлений)
Отсутствие формального основания	Следовательно, качество S/L (symbol level – символьный уровень)
Отсутствие декларативного языка	В ООПар нет эквивалента для SQL
Конфликт иерархии классов	Существующие механизмы в S/L требуют наличия K/L (knowledge level – уровень знаний) механизмов
Неопределенная информация	Выводы, касающиеся неопределенной и двузначной информации, кажется, игнорируются в ООПар
Результаты исследования связей	Слабая поддержка связей в ООПар. Необходим K/L для поддержки непрямой целостности
Проблема субъективных представлений	Акцентирование на объектах может привести к проблемам в интегрированных наборах программ – требуется также рассмотрение субъекта
Проблемы расширяемости	Некоторые типы расширений, необходимые на практике, не поддерживаются
Проблема коммуникации	Притязания ООПар на усовершенствованную коммуникацию не справедливы в слабоопределенных (ill-defined) областях
Рассмотрение конкретных случаев	Выявляет некоторые проблемы разных уровней – эффективность передачи сообщений, стоимость динамического связывания, параллелизм и т. д.

Список литературы: 1. *Бадд Т.* Объектно ориентированное программирование в действии: Пер. с англ. С.-Пб.: Питер, 1997. 464 с. 2. *The Object Management Group"Strategic Approach to Value Chain Integration"* / URL: http://www.omg.org, 2000. 3. *Орфали Р., Харки Д.* Java и CORBA в приложениях клиент-сервер. М: "ЛОРИ", 2000. 712 с. 4. *Фаулер М., Скотт К.* UML в кратком изложении. Применение стандартного языка объектного моделирования: Пер. с англ. М: Мир, 1999. 191 с. 5. *Буч Г., Рамбо Д., Джекобсон А.* Язык UML. Руководство пользователя: Пер. с англ. М: ДМК, 2000. 432 с. 6. *Inside Unified Modeling Language* / URL: http://www.rational.com, 2000. 7. *Кун Т.* Структура научных революций: Пер. с англ. Под общ. ред. С.Р. Микулинского. М: Прогресс, 1977. 300 с. 8. *Booch G.* The End of Objects and the Last Programmer /URL: htt://pwww.rational.com. 9. *A History of ObjectOriented Programming Language and their Impact on Program Design and Software Development* / URL: http://www. cyberdyneobjectsys.com. 10. *Буч Г.* Объектно-ориентированный анализ и проектирование с примерами приложений на C++: Пер. с англ. М: Бином, 1999. 560 с. 11. *Edward V. Berard* Basic ObjectOriented Concepts / The Object Agency. URL: http://www.toa.com/pub/OOBasic.pdf. 12. *Edward V. B.* What is a Methodology? / The Object Agency. URL: http://www.toa.com/pub/method.txt. 13. *Shajan M.* Critique of the Object Oriented Paradigm: Beyond ObjectOrientation / URL: http://members.aol.com/shaz7862/critique.htm, 1997. 14. *Мельников Г.П.* Системология и языковые аспекты кибернетики. М.: Сов. Радио, 1978. 368 с. 15. *Ковалев С.А.* Возможности использования системологии в объектно-ориентированной парадигме // 6-я международная конференция "Теория и техника передачи, приема и обработки информации" ("Новые информационные технологии"): Сб. научных трудов. Х.: ХТУРЭ, 2000. С. 78.

Поступила в редколлегию 6.10.2000

УДК 519.71

Е. И. КУЧЕРЕНКО

О ПОЛНОТЕ ПРИНИМАЕМЫХ РЕШЕНИЙ В НЕЧЕТКИХ УСЛОВИЯХ ФУНКЦИОНИРОВАНИЯ ТЕХНОЛОГИЧЕСКИХ ОБЪЕКТОВ

Введение

Технологические объекты в различных предметных областях характеризуются сложным параллельно-последовательным взаимодействием существенно нечетких процессов управления, которые реализуются с использованием как традиционных, так и интеллектуальных подходов [1], в частности, с использованием экспертных систем, баз знаний и машин логического вывода.

Такая тенденция к построению систем существенно расширяет возможности пользователя по управлению объектами, снижает затраты на требуемые ресурсы. Традиционные и интеллектуальные компоненты систем функционируют в тесной взаимосвязи, взаимно дополняя друг друга. Однако в таких ситуациях существенное значение приобретают нечеткие составляющие при описании процессов управления и принимаемых решений, что приводит к необходимости преодоления ряда трудностей в их реализации.

Постановка задачи

Рассмотрим проблемы, связанные с выявлением и устранением неполноты в реализации процессов управления и принимаемых решений. Качество решения этой проблемы часто является определяющим фактором, её анализу уделяется значительное внимание [2]. Однако ни вопросы её формализации, ни её решение в полной мере не реализованы.

Различают понятия внешней $\{Pl_n'\}, n \in N'$ и внутренней $\{Pl_n''\}, n \in N''$ полноты [2]. Эти понятия определим следующим образом:

Определение 1. *Внешней полнотой* $\{Pl_n'\}, n \in N'$ *будем называть ситуацию, когда на множестве решений* $\{\tilde{A}_u''\}, u \in U$ *присутствуют все ожидаемые решения* $\{\tilde{A}_r'\} r \in R, R \subseteq U$.

Определение 2. *Внутренней полнотой* $\{Pl_n''\}, n \in N''$ *будем называть ситуацию, когда на множестве решений* $\{\tilde{A}_u''\}, u \in U$ *нет таких решений и оценок, о которых нет достоверной информации.*

Анализ понятий внешней и внутренней полноты дает возможность сформулировать следующее утверждение.

Утверждение 1. *Внешняя полнота* $\{Pl_n'\}, n \in N'$ *и внутренняя полнота* $\{Pl_n''\}, n \in N''$ *в условиях существенно нечетких процессов управления взаимно дополняют друг друга и являются компонентами полноты* $\{Pl_n\}, n \in N$, *причем*

$$\{Pl_n\} = \{Pl_n'\} \cup \{Pl_n''\} \; N = N' \cup N''. \qquad (1)$$

Доказательство утверждения 1 основано на анализе сущности компонент $\{Pl_n'\}$, $\{Pl_n''\}$ и предметных областей.

Следствие 1. *В дальнейшем, если это не будет вызывать неопределенности, мы будем рассматривать полноту в смысле (1).*

Таким образом, на множестве нечетких процессов $\{\tilde{\Pi}_i\}, i \in I$ математическая постановка решения задачи может быть представлена в виде

$$\forall \tilde{\Pi}_i \in \{\tilde{\Pi}_i\} \left| (\forall Pl_n \in \{Pi_n\} \right| Pl_n = true\ ), i \in I, n \in N. \tag{2}$$

Исследование подходов к решению проблемы выявления и анализа полноты

Нечеткие процессы управления и их взаимодействия опишем нечеткой сетевой моделью (НСМ), построенной с использованием положений развитого аппарата теории расширенных интерпретированных сетей Петри (РИСП) [3] и теории нечетких множеств[4].

Определение 3. *Пространство состояний НСМ $\tilde{S}(f)$ определим множеством возможных достижимых маркирований*

$$\{\tilde{M}(f)_v\} \cup \left\{\tilde{M}(f)_{0_w}\right\} \cup \left\{\tilde{M}(f)_{k_q}\right\}, v \in V, w \in W, q \in Q, \tag{3}$$

где $\tilde{M}(f) = \left\{\tilde{M}(\tilde{p}_j) : Z_{\tilde{p}_j}(k)\right\}$ – вектор нечеткого текущего маркирования позиций $\tilde{p}_j \in \tilde{P}$, $\tilde{M}(\tilde{p}_j)$ – маркирование позиций $\tilde{p}_j \in \tilde{P}$, $j \in J$ НСМ $\tilde{S}(f)$; $\tilde{M}(f)_0 = \left\{\tilde{M}(\tilde{p}_j) : Z_{\tilde{p}_j}(k)\right\}$ – вектор нечеткого начального маркирования позиций $\tilde{p}_j \in \tilde{P}$; $\tilde{M}(f)_k = \left\{\tilde{M}(\tilde{p}_j) : Z_{\tilde{p}_j}(k)\right\}$ – вектор нечеткого конечного (терминального) маркирования позиций $\tilde{p}_j \in \tilde{P}$; $Z_{\tilde{p}_j}(k)$ – функция принадлежности маркирования $\tilde{p}_j$ позиции НСМ множеству (3); k – некоторая переменная, определяющая значение функции $Z_{\tilde{p}_j}(k)$.

Утверждение 2. *Решение задачи (1) основывается на моделировании процессов и анализе пространства состояний НСМ $\tilde{S}(f)$.*

Доказательство утверждения 2 непосредственно следует из определения полноты и пространства состояний НСМ $\tilde{S}(f)$.

Формализация и анализ полноты в пространстве состояний НСМ

В качестве исходных данных, как следует из изложенного выше, следует принять следующее:

– множество начальных состояний системы $\{\tilde{A}_{0l}''\}, l \in L$;

– множество ожидаемых решений системы $\{\tilde{A}_r{}'\}, r \in R$;

– множество фактических решений системы $\{\tilde{A}_u{}''\}, u \in U$;

– $\{\mu_{\tilde{\Pi}_i}(k)\}$ –множество функций принадлежности i-му процессу.

Утверждение 3. Если при заданном множестве $\{\tilde{A}_{0l}{}''\}$, для множеств $\{\tilde{A}_r{}'\}$ и $\{\tilde{A}_u{}''\}$ справедливо

$$\left|\{\tilde{A}_r{}'\}\right| \le \left|\{\tilde{A}_u{}''\}\right| \quad \text{и} \quad \{\tilde{A}_r{}'\} \cap \{\tilde{A}_u{}''\} = \{\tilde{A}_u{}''\} \tag{4, 5}$$

а также

$$\forall \tilde{A}_u{}'' \in \{\tilde{A}_u{}''\}, \; \forall \tilde{A}_r{}' \in \{\tilde{A}_r{}'\} \left| \mu_{A''_u}(k_0) \ge \right| \mu_{A''_r}(k_0), \tag{6}$$

где $\mu_{A''_u} \vee \mu_{A'_r}(k_0)$ – функция принадлежностей множеству соответственно фактических или ожидаемых решений, тогда система характеризуется полнотой принимаемых решений.

Доказательство утверждения 3 непосредственно следует из постановки задачи исследований.

Сформулируем утверждения, определяющие основные положения, связанные с интерпретацией компонент исходных данных.

Утверждение 4. *Множество начальных состояний* $\{\tilde{A}_{0l}{}''\}, l \in L$ *интерпретируется в пространстве состояний НСМ множеством векторов начальной маркировки* $\{\tilde{M}(f)_{0w}\}, w \in W$, *причем* $|L| \ne |W|$.

Утверждение 5. *Множество ожидаемых решений* $\{\tilde{A}_r{}'\}, r \in R$ *интерпретируется в пространстве состояний НСМ подмножеством ненулевых компонент ожидаемых векторов маркирования позиций* $\{\tilde{M}_1(f)''_q\} \cup \{\tilde{M}_1(f)'_{k_q}\}$ *НСМ:*

$$\begin{aligned}
&\{\tilde{M}_1(f)'_q\} \cup \{\tilde{M}_1(f)_{k_q}{}'\} = \{\{\tilde{M}(\tilde{p}_j) : Z_{\tilde{p}_j}(k)\}\}, q \in Q', \\
&\{\tilde{M}_1(f)_{k_q}{}'\} \subset \{\tilde{M}(f)_{k_q}{}'\}, q \in Q', \\
&\{\tilde{M}_1(f)_q\} \subset \{\tilde{M}(f)_q{}'\} q \in Q',
\end{aligned} \tag{7}$$

где $\{\tilde{M}(f)_q{}'\}$ – множество ожидаемых векторов текущего маркирования $\tilde{p}_j \in \{\tilde{p}_i(out)\}$; позиций; $\{\tilde{M}(f)_{k_q}{}'\}$ – множество ожидаемых векторов конечного маркирования позиций НСМ $\tilde{p}_j \in \{\tilde{p}_i(out)_s\}, \{\tilde{p}_i(out)_s\}$ – множество конечных (терминальных) позиций НСМ.

Утверждение 6. *Множество фактических решений* $\{\tilde{A}_u''\} u \in U$ *интерпретируется в пространстве состояний НСМ подмножеством ненулевых компонент фактических векторов конечного маркирования позиций* $\{\tilde{M}_2(f)''_q\} \cup \{\tilde{M}_2(f)''_{k_q}\}$ *НСМ:*

$$\begin{aligned}
&\left\{\tilde{M}_2(f)_{k_q}''\right\} = \left\{\left\{\tilde{M}\left(\tilde{p}_j\right) : Z_{\tilde{p}_j}(k)\right\}\right\}, q \in Q'', \\
&\left\{\tilde{M}_2(f)_{k_q}''\right\} \subset \left\{\tilde{M}(f)_{k_q}''\right\}, q \in Q'', \\
&\left\{\tilde{M}_2(f)_q''\right\} \subset \left\{\tilde{M}(f)_q''\right\} q \in Q'',
\end{aligned} \tag{8}$$

где $\{\tilde{M}(f)_q''\}$ – множество фактических векторов текущего маркирования позиций $\tilde{p}_j \in \{\tilde{p}_i(out)\}$; $\left\{\tilde{M}(f)_{k_q}''\right\}$ – множество фактических векторов конечного маркирования позиций НСМ $\tilde{p}_j \in \{\tilde{p}_i(out)_s\}, \{\tilde{p}_i(out)_s\}$ – множество конечных (терминальных) позиций НСМ.

Доказательство утверждений 4-6 основывается на постановке рассматриваемой задачи и интерпретации НСМ.

Тогда очевидно, что решение задачи (1) с учетом соотношений (4, 5, 6) и принятой интерпретации (7, 8) может быть определено в виде следующего утверждения:

Утверждение 7. *Система обладает полнотой принимаемых целевых (терминальных) решений* $Pl'_{kn} \subset \{Pl_n'\}$, *если в пространстве состояний НСМ, моделирующей процессы, при заданном множестве векторов* $\left\{\tilde{M}(f)_{0_w}\right\}$ *справедливы следующие условия:*

$$-\forall \tilde{M}_1(f)_{k_q}' \in \left\{\tilde{M}_1(f)_{k_q}'\right\}, \forall \tilde{M}_2(f)_{k_q}'' \in \left\{\tilde{M}_2(f)_{k_q}''\right\} \left\|\left\{\tilde{M}_1(f)_{k_q}'\right\}\right\| \le \left\|\left\{\tilde{M}_2(f)_{k_q}''\right\}\right\|, \tag{9}$$

$$-\left\{\tilde{M}_1(f)_{k_q}'\right\} \cap \left\{\tilde{M}_1(f)_{k_q}''\right\} = \left\{\tilde{M}_1(f)_{k_q}'\right\}, \tag{10}$$

$$-\forall \tilde{M}_1(f)_{k_q}' \in \left\{\tilde{M}_1(f)_{k_q}'\right\}, \forall \tilde{M}_2(f)_{k_q}'' \in \left\{\tilde{M}_2(f)_{k_q}''\right\} \left\|\left\{Z_{\tilde{p}_j}'(k_0)\right\}\right\| \ge \left\{Z_{\tilde{p}_j}''(k_0)\right\}, \tag{11}$$

где $\tilde{p}_j \in \{\tilde{p}_i(out)_s\}$.

Утверждение 8. Система обладает полнотой принимаемых промежуточных решений $Pl'_{in} \subset \{Pl_n'\}$, если в пространстве состояний НСМ, моделирующей процессы, при заданном множестве векторов $\left\{\tilde{M}(f)_{0_w}\right\}$ справедливы следующие условия:

$$-\forall \widetilde{M}_1(f)_q{}' \in \{\widetilde{M}_1(f)_q{}'\}, \forall \widetilde{M}_2(f)_q{}'' \in \{\widetilde{M}_2(f)_q{}''\} \| \{\widetilde{M}_1(f)_q{}'\} \le |\{\widetilde{M}_2(f)_q{}''\}|, \quad (12)$$

$$-\{\widetilde{M}_1(f)_q{}'\} \cap \{\widetilde{M}_1(f)_q{}''\} = \{\widetilde{M}_1(f)_q{}'\} \quad (13)$$

$$-\forall \widetilde{M}_1(f)_q{}' \in \{\widetilde{M}_1(f)_q{}'\}, \forall \widetilde{M}_2(f)_q{}'' \in \{\widetilde{M}_2(f)_q{}''\} \{Z_{\widetilde{p}_j}{}'(k_0)\} \ge \{Z_{\widetilde{p}_j}{}''(k_0)\}, \quad (14)$$

где $\widetilde{p}_j \in \{\widetilde{p}_i(out)\}$.

На множестве нечетких процессов

$$\forall \widetilde{\Pi}_i \in \{\widetilde{\Pi}_i\} (\forall Pl_n \in \{Pl_n\} | Pl_n = True), i \in I, n \in N. \quad (15)$$

В качестве примера рассмотрим НСМ, моделирующую фрагмент нечетких процессов, для которого необходимо определить степень полноты.

Пусть задано:

$\widetilde{M}(f)_0 = (1,0,1,1,0,...,0)$; $\widetilde{M}(f)'k = (0,0,0,0,...,0,1,1,1)$; $\widetilde{M}(f)''k = (0,0,0,0,...,1,1,1,1)$;

$\widetilde{M}_1(f)'k = (1,1,1)$; $\widetilde{M}_2(f)''k = (1,1,1,1)$;

$\forall \widetilde{M}(\widetilde{p}_j) \in \widetilde{M}(f)'k \left| Z_{\widetilde{p}_j}(k_0) = 0{,}80; \right.$ $\forall \widetilde{M}(\widetilde{p}_j) \in \widetilde{M}(f)''k \left| Z_{\widetilde{p}_j}(k_0) = 0{,}85. \right.$

В соответствии с утверждением 7 НСМ моделирует фрагмент нечетких процессов системы, который обладает полнотой принимаемых решений, так как выполняются условия

$$\left\{\widetilde{\widetilde{Z}}_{\widetilde{p}_j}{}'(k_0)\right\} = (1;1;1); \quad \left\{\widetilde{\widetilde{Z}}_{\widetilde{p}_j}(k_0)\right\} = (1;1;1;1),$$

где $\left\{\widetilde{\widetilde{Z}}_{\widetilde{p}_j}(k_0)\right\}$ – обычное подмножество функций принадлежности, ближайшее к нечеткому [5].

Тогда

$$\nu\left(\left\{\widetilde{\widetilde{Z}}_{\widetilde{p}_j}{}'(k_0)\right\}\right) = \frac{2}{3}(|0{,}85-1| + |0{,}85-1| + |0{,}85-1|) = 0{,}30;$$

$$\nu\left(\left\{\widetilde{\widetilde{Z}}_{\widetilde{p}_j}{}'(k_0)\right\}\right) = \frac{2}{4}(|0{,}85-1| + |0{,}85-1| + |0{,}85-1| + |0{,}85-1|) = 0{,}30$$

Условие (11) справедливо.

Анализ полноты принимаемых решений с использованием НСМ дает возможность пользователю на основе формальных подходов определять свойства полноты. Такой подход может быть положен в основу решения практических задач построения эффективных технологических комплексов в условиях существенно нечеткой среды взаимодействия процессов.

Выводы

1. На основе анализа условий и особенностей функционирования существенно нечетких процессов в сложных технологических комплексах обосновано и сформулировано представление о полноте принимаемых решений.

2. Предложен комплекс утверждений, позволяющих интерпретировать состояние нечетких процессов в терминах пространства НСМ.

3. Предложен подход к формализации процесса выявления полноты принятия решений в пространстве состояний НСМ.

4. Определены формальные условия полноты при взаимодействии процессов в нечеткой среде.

5. Подход может быть положен в основу построения эффективных инструментальных средств анализа процессов управления в сложных технологических комплексах.

Список литературы: 1. *Ямпольский Л.С., Лавров О.А.* Штучний інтелект у плануванні та управлінні виробництвом: Підручник. К.: Вища шк., 1995. 255с. 2. *Обработка нечеткой информации в системах принятия решений* /А.Н. Борисов, А.В. Алексеев, Г.В. Меркурьева и др. М.:Радио и связь, 1989. 304с. 3. *Кучеренко Е.И., Фадеев В.А.* Инструментальные средства моделирования процессов управления в сложных технологических комплексах //Авиационно-космическая техника и технология. Тр. Гос. аэрокосм. ун-та им. Н.Е. Жуковского. Харьков. Гос. аэрокосмич. ун-т, 2000. Вып.14.С. 166-168. 4. *Мурата Т.* Сети Петри: Свойства, анализ, приложения //ТИИЭР. 1989. Т.77, № 4. С. 41-85. 5. *Кофман А.* Введение в теорию нечетких множеств: Пер. с франц. М.: Радио и связь, 1982. 432с.

Поступила в редколлегию 18.10.2000