УДК 004.91

И. С. Печенко, науч. сотр., e-mail: ivan.pechenko@intel.com
ЗАО "Интел А/О"

Способы представления спецификаций вычислительных систем: проблемы и возможности машинной обработки

Спецификации вычислительных систем могут состоять из множества документов с использованием различных форм описания, таких как естественные языки, диаграммы, схемы, таблицы и формальные языки. Процесс создания спецификаций может быть охарактеризован как процесс перехода от описанных на естественном языке требований к системе, не структурированных и содержащих обычно множество неточностей, к более формализованным спецификациям, не содержащим неточностей, двусмысленностей и противоречий. Описаны различные формы спецификаций, проблемы, связанные с каждой из них, а также возможности анализа и валидации спецификаций.

Ключевые слова: *требования, спецификация, валидация, естественный язык, формальный язык, машинная обработка, диаграмма UML, автоматическая трансляция, таблица, шаблон таблицы*

Введение

Сложность вычислительных систем многократно возросла за последние десятилетия. Они могут состоять из сотен взаимосвязанных аппаратных и программных компонентов. Все возрастающая сложность проектирования таких систем ведет к повышению важности создания их полных и точных спецификаций, в которых описано, каким образом компоненты системы взаимодействуют между собой и с окружением вычислительной системы. От качества таких спецификаций во многом зависит успех последующих этапов проектирования системы.

Процесс создания спецификаций включает в себя несколько этапов (рис. 1): выявление технических требований к системе, их анализ, документирование, верификация и дальнейшая поддержка изменений в документации [1, 2].

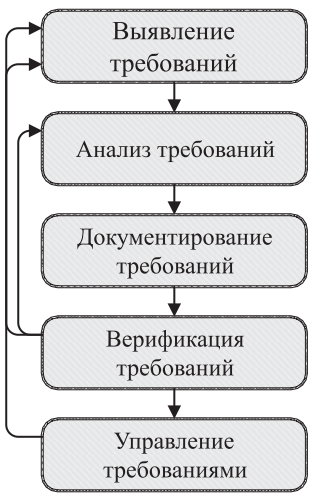

Рис. 1. Процесс создания спецификаций

Выявление технических требований к системе — это первый этап в процессе создания спецификаций, на котором необходимо определить потребности различных заинтересованных лиц — заказчиков системы и ее конечных пользователей. Этот этап особенно важен, так как часто именно в процессе выявления требований в спецификацию будущей системы могут закрадываться ошибки, противоречия и неточности. Обычно это связано с проблемами в коммуникации между системными архитекторами, с одной стороны, и заказчиками либо пользователями — с другой, с невозможностью точно описать некоторые идеи заинтересованных сторон. Такие ошибки в случае слишком позднего обнаружения или недостаточно тщательного устранения могут привести к катастрофическим для проекта последствиям [3—7]. Стоимость такой ошибки для проекта может возрасти, по некоторым данным, в десять и более раз за каждый этап разработки системы, на котором эта ошибка осталась невыявленной. В случае если такая ошибка остается невыявленной на момент выпуска продукта, последствия могут быть фатальными. Одним из известных примеров является крушение ракеты-носителя Ариан-5 4 июня 1996 г. вследствие ошибки в спецификации управляющего программного обеспечения.

Анализ требований к системе — второй этап в процессе создания спецификаций, включающий в себя проверку выявленных ранее требований к системе

на непротиворечивость, различные несоответствия, упущения и другие возможные ошибки, а также устранение таких ошибок.

На этапе *документирования требований* выявленные требования систематизируются, и на их базе создается спецификация будущей системы. Следует заметить, что на этом этапе спецификация еще не имеет окончательного вида вследствие итеративности процесса создания спецификаций. Ошибки, найденные на этапе верификации, а также изменения требований к системе требуют правок в спецификациях, после которых снова необходима верификация.

Верификация спецификаций состоит в проверке того, насколько описанная в спецификации функциональность системы соответствует ожидаемым требованиям. Кроме того, в процессе верификации спецификаций проверяется точность, полнота и непротиворечивость спецификации, а также соответствие описанных характеристик системы заявленным требованиям. Верификация спецификаций может выполняться как вручную, так и с использованием средств автоматизации, однако автоматизированная верификация может применяться лишь к спецификациям, представленным в формализованном виде. Ошибки, найденные в процессе верификации, приводят к необходимости изменения спецификации, а в некоторых случаях — и к коррекции требований к системе.

Важно отметить, что требования к вычислительным системам имеют тенденцию меняться и после создания спецификации, уже в процессе проектирования, что приводит к необходимости внесения изменений в спецификации, поддержки этих изменений и проверки их корректности.

Процесс создания спецификаций обычно состоит в переходе от полученных от заказчиков требований к формализованным спецификациям [8, 9]. Первичные требования обычно представляются в виде простого текстового описания, непрозрачны, могут включать двусмысленности и иметь множество различных трактовок. Финальная же версия спецификаций должна быть точной, корректной и недвусмысленной. Такой подход требует работы по трем направлениям:

- важно стремиться к созданию максимально полно описанной спецификации, включающей все детали будущей системы;
- необходимо добиться полного согласия заинтересованных сторон по поводу правильности и полноты спецификаций, для чего необходимо точно и недвусмысленно изложить в ней все требования к системе;
- спецификация должна быть представлена в формализованном виде для достижения однозначности изложенной в ней информации, а также для обеспечения возможности ее верификации.

В данной работе рассматривается главным образом третье направление работы над спецификациями, главная задача которого — формализация представления данных о системе. Инструменты такой формализации, методы создания формализованных спецификаций и языки описания требований могут быть подразделены на три категории [10]: объектно-ориентированные [11, 12], процесс-ориентированные [13] и поведенческие. Каждый из этих подходов имеет свои плюсы и минусы. Любого из этих методов создания и формализации спецификаций обычно недостаточно для того, чтобы полностью избежать проблем с целостностью, однозначностью, несвязностью и неточностью данных о системе.

Далее в работе будут рассмотрены различные способы представления спецификаций, классифицированные по степени формализации. Неформализованные и полностью формализованные способы представления спецификаций описаны относительно кратко, а основное внимание уделено частично формализованным способам представления спецификаций, в частности, диаграммному и табличному как наиболее интересным и перспективным с точки зрения оптимального соотношения затрат на создание и возможностей дальнейшей обработки и анализа.

Языки описания спецификаций

На сегодняшний день спецификация проекта обычно состоит из множества документов, специфицирующих отдельные аспекты работы проектируемой системы. Эти документы создаются с использованием различных языков и форматов описания поведения и характеристик системы, таких как текстовые описания на естественном языке, подразделяющиеся на неструктурированные и структурированные, различные диаграммы последовательности операций, таблицы, рисунки, а также формальные языки. При этом в процессе написания спецификации описания некоторых характеристик системы проходят определенную последовательность трансформаций, чаще всего от менее формализованной формы к более формализованной (рис. 2). К примеру, изначальные требования были оформлены в виде текста на естественном языке, затем на основе этого текста были составлены таблицы, с помощью которых на последних этапах создания спецификации был написан код на одном из языков программирования.

Тексты на естественном языке являются примером наименее формализованного описания системы и, соответственно, имеют все недостатки, связанные с неточностью, двусмысленностью и возможностью различных трактовок. Несмотря на все эти недостатки, тексты на естественных языках на ранних этапах процесса проектирования обычно составляют основную массу данных о системе. Есть несколько причин такого широкого распростране-

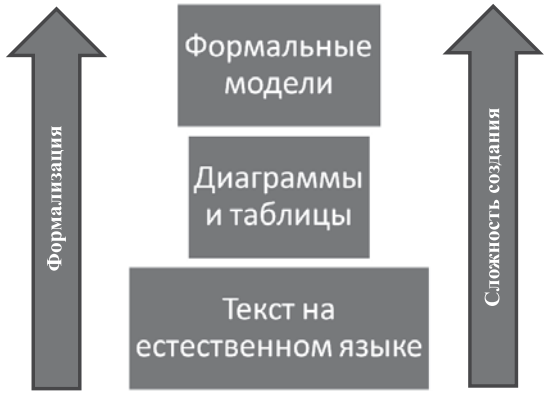

Рис. 2. Формы представления спецификаций

ния текстов на естественном языке в высокоуровневых спецификациях: простота создания, отсутствие семантических ограничений, позволяющих описывать с их помощью только отдельные аспекты системы, а также отсутствие необходимости серьезной подготовки специалистов, создающих документацию, связанной с освоением необходимых инструментов и языков.

Существенным недостатком текстов на естественном языке является сложность, а во многих случаях и невозможность автоматической трансформации их в другие, более формализованные формы спецификации. Однако существуют работы по переводу естественного языка в программный код [14—16]. Для корректной работы большинству таких переводчиков требуется правильное структурирование исходного текста и ограничения в используемой лексике, потому как задача обработки естественного языка не может быть решена с использованием существующих вычислительных систем и алгоритмов.

Формальные языки предоставляют возможность наиболее точного, лишенного проблем двусмысленности, описания системы. Обычно описания на формальном языке представляют собой строго описанные частные модели, отражающие определенные аспекты поведения будущей системы. Такие модели удобны тем, что их верификация может быть проведена автоматически, что является серьезным преимуществом в случае, когда сложность системы достаточно велика. Ряд работ описывает возможности проверки таких моделей [17—19].

Несмотря на преимущества спецификации поведения и характеристик системы с использованием формальных языков, обычно лишь небольшая часть спецификаций описана с их помощью. Это объясняется высокой сложностью освоения этих языков и создания с их помощью моделей системы, их плохой читабельностью, что приводит к невозможности быстрого обмена данными о будущей системе между заинтересованными сторонами, а также семантическими ограничениями, значительно сужающими область применения формальных языков.

Частично формализованные спецификации содержат как данные, описанные на формальных языках, так и дополнительную информацию на естественном языке. Примером такой спецификации является диаграмма последовательности операций: синтаксис и семантика диаграммы формально заданы, но подписи к объектам диаграммы на естественном языке составляют существенную часть представленных в ней данных. Такие формы документации имеют достоинства и недостатки обеих выше описанных категорий спецификаций. Более того, благодаря существующим методам автоматизации проектирования возможно нивелировать недостатки, присущие таким формам документации, вследствие чего частично формализованные формы спецификаций являются крайне интересным объектом для изучения.

Спецификации в форме диаграмм

Одной из самых распространенных и удобных форм создания частично формализованных спецификаций являются диаграммы. С их помощью можно описать и структуру будущей системы, и ее поведение. Наиболее распространенным визуальным языком создания диаграмм в сфере разработки спецификаций является UML (Unified Modelling Language), ставший практически стандартом для написания спецификаций програмных систем. В рамках UML существует множество типов диаграмм: структурные диаграммы для описания строения системы, а также различные диаграммы последовательности операций для описания ее поведения. Диаграммы UML чаще всего представляют собой спецификацию высокого уровня и не содержат детально и формализованно изложенной информации, и к ним нельзя применить формальные методы анализа. Вместо этого обычно модифицируют исходные диаграммы UML, добавляя в них дополнительные данные, а затем трансформируют их в более формализованные модели [20].

Ряд работ рассматривает различные области применения диаграм UML [21—24].

В частности, в работе [21] рассматривается процесс создания представленных в виде UML-диаграмм объектно-ориентированных моделей для спецификации систем управления данными о продукте и выявляются достоинства и недостатки таких моделей.

В статье [22] представлена методология создания спецификаций поведения киберфизических систем с использованием диаграмм UML. Для описания специфических для предметной области компонентов предлагается использовать такие механизмы расширения синтаксиса UML-диаграмм, как UML-профили и стереотипы.

В работах [23, 24] UML-диаграммы используются при создании спецификации логических контроллеров. Для обеспечения соответствия спецификации установленным пользователями требо-

ваниям к поведению контроллеров применяется формальная проверка модели, при которой требования приводятся к виду формул темпоральной логики, а диаграммы UML формализуются и переводятся в программный код.

В [25] также рассматривается интеграция инструментов формальной проверки моделей и методов спецификации, основанных на UML-диаграммах. Здесь целью является создание спецификаций встроенных систем, а в качестве примера рассматривается система управления медицинским оборудованием.

В работах [26, 27] рассматриваются возможности переиспользования UML-диаграмм в процессе разработки программного обеспечения. Будучи де-факто стандартом в этой области, UML-диаграммы используются практически повсеместно для описания программных систем, поэтому возможность их переиспользования чрезвычайно важна, так как позволяет экономить значительную часть времени работы аналитиков и архитекторов программных систем.

В статье [28] описывается исследование влияния различных факторов на читабельность и усваиваемость моделей предметных областей, созданных с использованием диаграмм UML. Исследование было проведено при участии более 100 студентов, от которых требовалось прочитать и понять специфицированные таким образом модели. Было выявлено, что явное руководство по использованию модели, включенное в спецификацию, серьезно улучшает ее читабельность и усваиваемость.

Ряд работ описывает возможности автоматической трансляции UML-диаграмм в более формализованные формы спецификации с помощью их дополнительной детализации. Обеспечение возможности такой трансформации — крайне перспективное направление исследований, так как приведение UML-диаграмм к более формализованному виду позволит использовать методы формального анализа для их проверки.

К примеру, работа [29] представляет систему, которая, пользуясь набором определенных правил, переводит диаграмму классов UML в программный код, специфицирующий поведенческие аспекты системы.

В работе [30] описывается метод генерации вариантов тестов для программной системы из диаграммы UML. Создание тестов обычно происходит на основе уже написанного кода, но их автоматическая генерация на основе спецификаций уменьшит затраты времени в фазе тестирования в разы. Для создания тестов из UML-диаграммы сначала генерируется набор сценариев, из которых далее берется вся необходимая информация для создания тестов.

Работа [31] описывает систему для оценки производительности будущей системы, представленной в виде UML-диаграммы. Для этого к имеющимся данным добавляются необходимые аннотации с помощью расширения для языка UML, OMG-профиля. Затем диаграмма трансформируется в промежуточную модель, которая затем и оценивается с точки зрения производительности.

Работа [32] также описывает возможность вычисления различных метрик производительности (времени отклика, времени прохождения сигнала, пропускной способности) модели в виде диаграммы UML. При этом диаграммы UML трансформируются в сети Петри — известную математическую абстракцию для описания динамических систем. Сети Петри являются формальным языком, что позволяет проводить их анализ, в том числе анализ производительности, с использованием формальных методов. Ряд работ описывает возможности по дальнейшей работе с сетями Петри [33—35].

В статье [36] предлагается метод перевода спецификаций в виде UML-диаграмм на естественный язык, что позволяет получить спецификацию на естественном языке, свободную от стандартных для естественного языка недостатков: неточности и двусмысленности. Несмотря на то что спецификации на естественном языке являются менее формализованными, а значит, такая трансформация будет "шагом назад" в процессе создания спецификации, итеративность такого процесса и изменчивость требований клиентов требуют вовлеченности будущих пользователей в процесс создания спецификаций на всех его стадиях, что не может быть достигнуто, если спецификации будут описаны на формальном или полуформальном языке. Именно поэтому важна возможность переводить спецификации из формализованных форм на естественный язык и обратно.

В работе [37] описана система, позволяющая создавать спецификации протоколов коммункационных систем из спецификаций сервисов в виде UML-диаграмм. Полученные в этой работе результаты позволяют автоматически генерировать крайне сложные коммуникационные протоколы, избегая больших временных затрат на последовательность операций по разработке протокола, его анализу, поиску и исправлению ошибок.

Интересна работа [38], в которой представлен основанный на диаграммах UML визуальный язык VLC. Он позволяет визуализировать дополнительные данные, не поддерживаемые в UML, а также управлять применением формальных техник к диаграммам посредством генерации формально описанной модели из диаграммы. Ряд работ [39, 40] описывает возможности применения языка VLC в рамках различных процессов создания спецификаций, а в работе [41] рассматривается возможность генерации Z-нотаций — строго формализованных форм спецификаций, используемых для моделирования программных продуктов, из диаграмм VLC.

Спецификации в форме таблиц

Еще одним удобным полуформализованным средством разработки спецификаций являются таблицы. Они обычно служат для детального описания определенных данных о структуре системы. Табличные данные формализуемы и могут быть подвергнуты анализу и валидации, если заданы структура каждой таблицы и ее связь с другими данными, содержащимися в спецификации.

Работы, касающиеся табличного формата данных, можно условно разделить на две категории: предлагающие возможности валидации табличных данных [42—45] и описывающие механизмы создания таблиц, корректных по построению [46—48]. Стоит отметить, что формальный анализ табличных данных при отсутствии информации о структуре таблицы невозможен в большинстве случаев, поэтому все подходы к валидации табличных данных, не полагающиеся на знания структуры таблиц и предметной области, требуют непосредственного участия пользователя.

В статье [42] рассматривается система полуавтоматической верификации табличных данных, выполненная в виде расширения для Microsoft Excel, самой распространенной системы работы с таблицами в мире. В данной системе обязанность по проверке корректности и указанию ошибок лежит на пользователе, а система подсказывает необходимые исправления. В [43] описывается похожая система полуавтоматической верификации таблиц, но с несколько другим подходом к разделению обязанностей между пользователем и системой. Здесь пользователь задает набор необходимых ограничений и ожидаемых значений для части ячеек таблицы, после чего система предлагает возможные исправления в табличных формулах. В работе [44] предлагается метод тестирования табличных данных, основанный на методологии потока данных. С помощью этого метода особенно удобно проводить тестирование таблиц с числовыми данными, так как формулы и их сложные зависимости являются основным объектом рассмотрения в методологии потока данных.

В статье [45] рассматривается возможность применения мутационного тестирования для валидации таблиц. Метод мутационного тестирования был разработан для оценки систем тестирования программного обеспечения и заключается в последовательном применении мутационных операторов, имитирующих программные ошибки, к разным областям исходного кода и последующем тестировании получившихся программ. Для того чтобы считаться достаточно полным, набор тестов должен выявить всех "мутантов". Ранее мутационное тестирование не получило широкого распространения вследствие своей огромной вычислительной сложности, но с развитием вычислительных технологий этот метод начал находить широкое применение в процессе создания пакетов тестов для программных систем. Работа [45] показывает, что этот метод можно с успехом применять и для оценки систем валидации табличных данных.

Все вышеописанные подходы к валидации таблиц являются полуавтоматическими, причем в большинстве из них пользователь сам должен указывать системе на ошибки в данных. Полная автоматизация тестирования таблиц может быть создана только при условии соответствия табличного синтаксиса и семантики определенной модели. Для решения этой задачи в последнее время предложено несколько подходов. Все они предлагают создавать таблицы в два этапа: сначала создаются шаблоны таблиц, а затем отдельные экземпляры, соответствующие этим шаблонам. Такие таблицы по построению свободны от многих типовых для табличных данных ошибок, например, некорректных ссылок, типов и пробелов в записях.

Самым известным проектом в этом направлении стала объектно-ориентированная система создания таблиц ClassSheets [49]. В ней созданию таблиц предшествует описание их модели.

Важным аспектом при создании таблиц на основе шаблонов является поддержка изменений этих шаблонов после создания таблиц. В этом случае необходимо применять к уже существующим экземплярам таблиц операторы, дублирующие изменения в их шаблонах. В частности, работа [47] описывает пары трансформаций, одна из которых применяется к модели (шаблону), а другая — к экземпляру таблицы. Статьи [48, 50, 51] описывают расширения для системы ClassSheets, также реализующие такие трансформации.

Заключение

В данной работе были рассмотрены различные формы представления спецификаций, среди которых основной акцент был сделан на диаграммной и табличной формах как на частично формализованных и представляющих наибольший интерес в качестве объекта анализа.

На основе приведенного обзора работ можно сделать вывод, что крайне перспективными направлениями исследований является поиск способов автоматической трансляции спецификаций в формальные модели и поиск возможностей создания корректных спецификаций на частично формализованных языках. Примером первого направления является генерация различных формальных моделей из диаграмм с добавлением некоторой дополнительной информации, а также генерация диаграмм из текстового описания [52]. Примером второго является генерация корректных по построению таблиц с помощью объектно-ориентированного подхода.

Список литературы

1. **Khan M. N. A., Khalid M., ul Haq S.** Review of requirements management issues in software development // International Journal of Modern Education and Computer Science (IJMECS). 2013. N. 5.1. 21 p.
2. **Pandey D., Suman U., Ramani A. K.** An Effective Requirement Engineering Process Model for Software Development and Requirements // International Conference on Advances in Recent Technologies in Communication and Computing Management. 2010. P. 287—291.
3. **Neetu K. S., Pillai A. S.** State-Of-The-Practice Survey And Comparison In Requirements Engineering In Healthcare IT Projects // International Conference on Advances in Computing, Communications and Informatics. Aug 2012.
4. **Lee Y. K., Kim N. H., Kim D., Lee D. H., In H. P.** Customer Requirements Elicitation based on Social Network Service // KSTT Transactions On Internet And Information Systems. Oct 2011. Vol. 5.10.
5. **Keller T.** Contextual Requirements Elicitation // Seminar in Requirements Engineering. Spring. Department of Informatics, University of Zurich. 2011.
6. **Dhungana D., Seyff N., Graf F.** Research Preview: Supporting End-user Requirements Elicitation Using Product Line Variability Models // Requirements Engineering Foundation for Software Quality. Mar 2011. Vol. 6606. P. 66—71.
7. **Neetu K. S, Pillai A. S.** A survey on global requirements elicitation issues and proposed research framework // Software Engineering and Service Science (ICSESS) // 4th IEEE International Conference on. IEEE. 2013. P. 554—557.
8. **Pohl K.** The three dimensions of requirements engineering // Seminal Contributions to Information Systems Engineering. Springer Berlin Heidelberg. 2013. P. 63—80.
9. **Pohl K., Ulfat-Bunyadi N.** The Three Dimensions of Requirements Engineering: 20 Years Later // Seminal Contributions to Information Systems Engineering. Springer Berlin Heidelberg. 2013. P. 81—87.
10. **Blekhman A., Wachs J. P., Dori D.** Model-Based System Specification With Tesperanto: Readable Text From Formal Graphics // Systems, Man, and Cybernetics: Systems // IEEE Transactions. 2015. Vol. 45 (11). P. 1448—1458.
11. **Kasser J.** Object-oriented requirements engineering and management // PhD diss., Systems Engineering Society of Australia. 2003.
12. **Liu Z., Jifeng H., Li X., Chen Y.** A relational model for formal object-oriented requirement analysis in UML // Formal Methods and Software Engineering. Berlin, Germany: Springer. 2003. P. 641—664.
13. **Bastani B.** Process-oriented abstraction of the complex evolvable systems // ACM SIGSOFT Softw. Eng. Notes. 2008. Vol. 33, N. 3. P. 1.
14. **Liu H., Lieberman H.** Metafor: Visualizing stories as code // Proc. 10th Int. Conf. Intell. User Interf., San Diego, CA, USA. 2005. P. 305—307.
15. **Harel D.** Can programming be liberated, period? // Computer. Jan. 2008. Vol. 41, N. 1. P. 28—37.
16. **Harel D.** From play-in scenarios to code: An achievable dream // Computer. 2001. Vol. 34, N. 1. P. 53—60.
17. **Gasparini L., Norman T. J., Kollingbaum M. J., Chen L., Meyer J. J. C.** Verifying Normative System Specification containing Collective Imperatives and Deadlines // Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems. International Foundation for Autonomous Agents and Multiagent Systems. 2015. P. 1821—1822.
18. **Plagge D., Leuschel M.** Seven at one stroke: LTL model checking for high-level specifications in B, Z, CSP, and more // International journal on software tools for technology transfer. 2010. Vol. 12.1. P. 9—21.
19. **Legay A., Delahaye B., Bensalem S.** Statistical model checking: An overview // Runtime Verification. Springer Berlin Heidelberg. 2010. P. 122—135.
20. **Amálio N.** Generative frameworks for rigorous model-driven development. PhD thesis, Dept. Computer Science, Univ. of York. 2007.
21. **Eynard B., Gallet T., Nowak P., Roucoules L.** UML based specifications of PDM product structure and workflow // Computers in Industry. 2004. Vol. 55.3. P. 301—316.
22. **Gavrilescu M., Magureanu G., Pescaru D., Jian I.** Towards UML software models for Cyber Physical System applications // Telecommunications Forum (TELFOR), 2012 20th. IEEE. 2012. P. 1701—1704.
23. **Grobelna I., Grobelny M.** UML Activity Diagrams in Requirements Specification of Logic Controllers // Conference: Design and Analysis of Control Systems (DACS 2015), At Athens, Greece. 2015. Vol. 1702, No. 1. 100005 p.
24. **Grobelna I., Grobelny M., Adamski M.** Model Checking of UML Activity Diagrams in Logic Controllers Design // Proceedings of the Ninth International Conference on Dependability and Complex Systems DepCoS-RELCOMEX. June 30—July 4, 2014, Brunyw, Poland. Springer International Publishing. 2014. P. 233—242.
25. **Daw Z., Cleaveland R., Vetter M.** Integrating model checking and UML based model-driven development for embedded systems // Electronic Communications of the EASST 66. 2014.
26. **Robles K., Fraga A., Morato J., Llorens J.** Towards an ontology-based retrieval of UML Class Diagrams // Information and Software Technology. 2012. Vol. 54.1. P. 72—86.
27. **Salami H. O., Ahmed M. A.** UML artifacts reuse: state of the art // International Journal of Soft Computing and Software Engineering. 2014. Vol. 3. P. 115—122.
28. **Reinhartz-Berger I., Sturm A.** Comprehensibility of UML-based software product line specifications // Empirical Software Engineering. 2014. Vol. 19.3. P. 678—713.
29. **Albert M., Cabot J., Gómez C., Pelechano V.** Generating operation specifications from UML class diagrams: A model transformation approach // Data & Knowledge Engineering. 2011. Vol. 70.4. P. 365—389.
30. **Linzhang W., Jiesong Y., Xiaofeng Y., Jun H., Xuandong L., Guo Z.** Generating test cases from UML activity diagram based on gray-box method // Software Engineering Conference, 2004. 11th Asia-Pacific. IEEE. 2004. P. 284—291.
31. **Distefano S., Scarpa M., Puliafito A.** From UML to Petri nets: The PCM-based methodology // Software Engineering, IEEE Transactions. 2011. Vol. 37.1. P. 65—79.
32. **Bernardi S., Merseguer J.** Performance evaluation of UML design with Stochastic Well-formed Nets // Journal of Systems and Software. 2007. Vol. 80.11. P. 1843—1865.
33. **Grobelna I., Wisniewska M., Wisniewski R., Grobelny M., Mroz P.** Decomposition, validation and documentation of control process specification in form of a Petri net // Human System Interactions (HSI), 2014 7th International Conference on. IEEE. 2014. P. 232—237.
34. **Karatkevich A.** Dynamic analysis of Petri net-based discrete systems // Lecture Notes in Control and Information Sciences. Springer Berlin Heidelberg, 2007. Vol. 356.
35. **Wiśniewski R., Karatkevich A., Adamski M., Kur D.** Application of comparability graphs in decomposition of Petri nets // 7th International Conference on Human System Interactions (HSI), Portugal. 2014. P. 216—220.
36. **Meziane F., Athanasakis N., Ananiadou S.** Generating Natural Language specifications from UML class diagrams // Requirements Engineering. 2008. Vol. 13.1. P. 1—18.
37. **Al Dallal J., Saleh K. A.** Synthesizing distributed protocol specifications from a UML state machine modeled service specification // Journal of Computer Science and Technology. 2012. Vol. 27.6. P. 1150—1168.
38. **Amálio N., Kelsen P., Ma Q.** The visual contract language: abstract modelling of software systems visually, formally and modularly // Univ. of Luxembourg, Tech. Rep. TRLASSY-10-03, 2010.
39. **Amálio N., Kelsen P.** Modular design by contract visually and formally using VCL. // Visual Languages and Human-Centric Computing (VL/HCC), 2010 IEEE Symposium. 2010. P. 227—234.
40. **Amálio N., Kelsen P., Ma Q., Glodt C.** Using VCL as an aspect-oriented approach to requirements modelling. // Transactions on aspect-oriented software development VII, Springer Berlin Heidelberg. 2010. P. 151—199.
41. **Amálio N., Glodt C., Kelsen P.** Building VCL models and automatically generating Z specifications from them // FM 2011: Formal Methods. Springer Berlin Heidelberg. 2011. P. 149—153.
42. **Abraham R., Erwig M.** Goal-directed debugging of spreadsheets // Visual Languages and Human-Centric Computing, 2005 IEEE Symposium on. IEEE. 2005. P. 37—44.
43. **Abraham R., Erwig M.** GoalDebug: A spreadsheet debugger for end users // Proceedings of the 29th international conference on Software Engineering. IEEE Computer Society. 2007. P. 251—260.

44. **Fisher M., Rothermel G., Creelan T., Burnett M.** Scaling a Dataflow Testing Methodology to the Multiparadigm World of Commercial Spreadsheets // Software Reliability Engineering, 2006. ISSRE'06. 17th International Symposium on. IEEE. 2006. P. 13—22.

45. **Abraham R., Erwig M.** Mutation operators for spreadsheets // Software Engineering, IEEE Transactions. 2009. Vol. 35.1. P. 94—108.

46. **Abraham R., Erwig M., Kollmansberger S., Seifert E.** Visual specifications of correct spreadsheets // Visual Languages and Human-Centric Computing, 2005 IEEE Symposium on. IEEE. 2005. P. 189—196.

47. **Mendes J.** Coupled evolution of model-driven spreadsheets // Software Engineering (ICSE), 2012 34th International Conference on. IEEE. 2012. P. 1616—1618.

48. **Cunha J., Fernandes J. P., Mendes J., Saraiva J.** MDSheet: A framework for model-driven spreadsheet engineering // Proceedings of the 34th International Conference on Software Engineering. IEEE Press. 2012. P. 1395—1398.

49. **Engels G., Erwig M.** ClassSheets: automatic generation of spreadsheet applications from object-oriented specifications // Proceedings of the 20th IEEE/ACM international Conference on Automated software engineering. ACM. 2005. P. 124—133.

50. **Cunha J., Fernandes J. P., Mendes J., Pacheco H., Saraiva J.** Bidirectional Transformation of Model-Driven Spreadsheets // ICMT. 2012. Vol. 12. P. 105—120.

51. **Stevens P.** Bidirectional model transformations in QVT: Semantic issues and open questions // Model Driven Engineering Languages and Systems. Springer Berlin Heidelberg. 2007. P. 1—15.

52. **Печенко И. С., Венгер О. В.** Способ автоматизированного создания диаграмм последовательности операций для сценариев поведения системы, описанных в виде текста // Информационные технологии. 2015. Том 21, № 1.

I. S. Pechenko, Research Associate, e-mail: ivan.pechenko@intel.com
ZAO "Intel AO"

Specification Representation Forms: Issues and Ways for Automatic Processing

The complexity of computer systems has grown dramatically over past thirty years. The growth of system design complexity leads to increasing importance of full and accurate system specifications creation. The quality of these specifications affect greatly on the product design success.

Requirement engineering process consists of several parts: requirements elicitation; requirements analysis; specification creation; specification validation and verification; requirements management.

Specification can consist of many documents written in different languages, for example: natural languages, structural and flow diagrams, spreadsheets and formal languages. The whole specification creation process can be characterized as a transition from natural language requirements got from stakeholders to more formal specification. Natural language requirements are commonly inaccurate, inconsistent, incomplete and cannot be automatically processed. Formal language specifications are the example of the most accurate and unambiguous system description. They also have another great advantage: they can be analyzed using formal methods. Despite this, usually only a small part of system specification is written in formal languages, because they are hard to learn and to read and have many semantic restrictions. Diagrams and spreadsheets stay somewhere between natural and formal languages: they have formal syntax, but also additional data in natural language.

Diagrams are one of the most common and convenient form of semi-formal specifications. UML is the most prevalent visual language in requirement engineering. There are both structural and behavioral types of UML diagrams. Specifications in UML contain usually high-level information and cannot be directly analyzed using formal methods. Instead of this, usually UML diagrams are modified, supplemented with additional data and then translated into some formal language. Many recent studies use this flow for diagram analysis.

Spreadsheets are another common form of semi-formal specifications. They usually serve as detailed description of system structure. They can be analyzed with formal methods only if the structure and syntax of the spreadsheet are set. There are two common directions in spreadsheets specification studies. One group of works consider spreadsheet validation capabilities. They use semi-automatic validation because, if spreadsheet structure is not set, formal validation methods cannot be applied to the spreadsheet. Another group of studies consider making spreadsheets correct by construction using spreadsheet templates.

Keywords: *requirements, specification, validation, natural language, formal language, automatic processing, UML diagram, automatic translation, spreadsheet, spreadsheet template*

References

1. **Khan M. N. A., Khalid M., ul Haq S.** Review of requirements management issues in software development, *International Journal of Modern Education and Computer Science (IJMECS)*. 2013, no. 5.1, 21 p.

2. **Pandey D., Suman U., Ramani A. K.** An Effective Requirement Engineering Process Model for Software Development and Requirements, *International Conference on Advances in Recent Technologies in Communication and Computing Management*, 2010, pp. 287—291.

3. **Neetu K. S., Pillai A. S.** State-Of-The-Practice Survey And Comparison In Requirements Engineering In Healthcare IT Projects, *International Conference on Advances in Computing, Communications and Informatics.* Aug 2012.

4. **Lee Y. K., Kim N. H., Kim D., Lee D. H., In H. P.** Customer Requirements Elicitation based on Social Network Service, *KSTT Transactions On Internet And Information Systems*, Oct 2011, vol. 5.10.

5. **Keller T.** Contextual Requirements Elicitation, *Seminar in Requirements Engineering.* Spring 2011. Department of Informatics, University of Zurich.

6. **Dhungana D., Seyff N., Graf F.** Research Preview: Supporting End-user Requirements Elicitation Using Product Line Variability Models, *Requirements Engineering Foundation for Software Quality*, Mar 2011, vol. 6606, pp. 66—71.

7. **Neetu K. S., Pillai A. S.** A survey on global requirements elicitation issues and proposed research framework, *Software Engineering*

and Service Science (ICSESS). 4th IEEE International Conference on. IEEE, 2013, pp. 554—557.

8. **Pohl K.** The three dimensions of requirements engineering, *Seminal Contributions to Information Systems Engineering. Springer Berlin Heidelberg*, 2013. pp. 63—80.

9. **Pohl K., Ulfat-Bunyadi N.** The Three Dimensions of Requirements Engineering: 20 Years Later, *Seminal Contributions to Information Systems Engineering. Springer Berlin Heidelberg*, 2013, pp. 81—87.

10. **Blekhman A., Wachs J. P., Dori D.** Model-Based System Specification With Tesperanto: Readable Text From Formal Graphics, *Systems, Man, and Cybernetics: Systems, IEEE Transactions,* 2015, vol. 45 (11), pp. 1448—1458.

11. **Kasser J.** *Object-oriented requirements engineering and management.* PhD diss., Systems Engineering Society of Australia. 2003.

12. **Liu Z., Jifeng H., Li X., Chen Y.** A relational model for formal object-oriented requirement analysis in UML, *Formal Methods and Software Engineering.* Berlin, Germany: Springer, 2003, pp. 641—664.

13. **Bastani B.** Process-oriented abstraction of the complex evolvable systems, *ACM SIGSOFT Softw. Eng. Notes*, 2008, vol. 33, no. 3. P. 1.

14. **Liu H., Lieberman H.** Metafor: Visualizing stories as code, *Proc. 10th Int. Conf. Intell. User Interf.*, San Diego, CA, USA, 2005. pp. 305—307.

15. **Harel D.** Can programming be liberated, period? *Computer*, Jan. 2008, vol. 41, no. 1, pp. 28—37.

16. **Harel D.** From play-in scenarios to code: An achievable dream. *Computer*, 2001, vol. 34, no. 1, pp. 53—60.

17. **Gasparini L., Norman T. J., Kollingbaum M. J., Chen L., Meyer J. J. C.** Verifying Normative System Specification containing Collective Imperatives and Deadlines, *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems. International Foundation for Autonomous Agents and Multiagent Systems*, 2015, pp. 1821—1822.

18. **Plagge D., Leuschel M.** Seven at one stroke: LTL model checking for high-level specifications in B, Z, CSP, and more, *International journal on software tools for technology transfer*, 2010, vol. 12.1, pp. 9—21.

19. **Legay A., Delahaye B., Bensalem S.** Statistical model checking: An overview, *Runtime Verification*. Springer Berlin Heidelberg, 2010, pp. 122—135.

20. **Amálio N.** *Generative frameworks for rigorous model-driven development.* PhD Liss, Dept. Computer Science, Univ. of York. 2007.

21. **Eynard B., Gallet T., Nowak P., Roucoules L.** UML based specifications of PDM product structure and workflow, *Computers in Industry*, 2004, vol. 55.3, pp. 301—316.

22. **Gavrilescu M., Magureanu G., Pescaru D., Jian I.** Towards UML software models for Cyber Physical System applications, *Telecommunications Forum (TELFOR), 2012 20th. IEEE*, 2012, pp. 1701—1704.

23. **Grobelna I., Grobelny M.** UML Activity Diagrams in Requirements Specification of Logic Controllers, *Conference: Design and Analysis of Control Systems (DACS 2015), At Athens, Greece*, 2015, vol. 1702, no. 1. P. 100005.

24. **Grobelna I., Grobelny M., Adamski M.** Model Checking of UML Activity Diagrams in Logic Controllers Design, *Proceedings of the Ninth International Conference on Dependability and Complex Systems DepCoS-RELCOMEX.* June 30—July 4, 2014, Brunyw, Poland. Springer International Publishing, 2014, pp. 233—242.

25. **Daw Z., Cleaveland R., Vetter M.** Integrating model checking and UML based model-driven development for embedded systems, *Electronic Communications of the EASST 66.* 2014.

26. **Robles K., Fraga A., Morato J., Llorens J.** Towards an ontology-based retrieval of UML Class Diagrams, *Information and Software Technology*, 2012, vol. 54.1, pp. 72—86.

27. **Salami H. O., Ahmed M. A.** UML artifacts reuse: state of the art, *International Journal of Soft Computing and Software Engineering*, 2014, vol. 3, pp. 115—122.

28. **Reinhartz-Berger I., Sturm A.** Comprehensibility of UML-based software product line specifications, *Empirical Software Engineering*, 2014, vol. 19.3, pp. 678—713.

29. **Albert M., Cabot J., Gómez C., Pelechano V.** Generating operation specifications from UML class diagrams: A model transformation approach, *Data & Knowledge Engineering*, 2011, vol. 70.4, pp. 365—389.

30. **Linzhang W., Jiesong Y., Xiaofeng Y., Jun H., Xuandong L., Guo Z.** Generating test cases from UML activity diagram based on gray-box method, *Software Engineering Conference, 2004. 11th Asia-Pacific. IEEE*, 2004, pp. 284—291.

31. **Distefano S., Scarpa M., Puliafito A.** From UML to Petri nets: The PCM-based methodology, *Software Engineering, IEEE Transactions*, 2011, vol. 37.1, pp. 65—79.

32. **Bernardi S., Merseguer J.** Performance evaluation of UML design with Stochastic Well-formed Nets, *Journal of Systems and Software*, 2007, vol. 80.11, pp. 1843—1865.

33. **Grobelna I., Wisniewska M., Wisniewski R., Grobelny M., Mroz P.** Decomposition, validation and documentation of control process specification in form of a Petri net, *Human System Interactions (HSI), 2014 7th International Conference on. IEEE*, 2014, pp. 232—237.

34. **Karatkevich A.** Dynamic analysis of Petri net-based discrete systems, *Lecture Notes in Control and Information Sciences*, Springer Berlin Heidelberg, 2007, vol. 356.

35. **Wiśniewski R., Karatkevich A., Adamski M., Kur D.** Application of comparability graphs in decomposition of Petri nets, *7th International Conference on Human System Interactions (HSI), Portugal*, 2014, pp. 216—220.

36. **Meziane F., Athanasakis N., Ananiadou S.** Generating Natural Language specifications from UML class diagrams, *Requirements Engineering*, 2008, vol. 13.1, pp. 1—18.

37. **Al Dallal J., Saleh K. A.** Synthesizing distributed protocol specifications from a UML state machine modeled service specification, *Journal of Computer Science and Technology*, 2012, vol. 27.6, pp. 1150—1168.

38. **Amálio N., Kelsen P., Ma Q.** The visual contract language: abstract modelling of software systems visually, formally and modularly, *Univ. of Luxembourg, Tech. Rep. TRLASSY-10-03.* 2010.

39. **Amálio N., Kelsen P.** Modular design by contract visually and formally using VCL, *Visual Languages and Human-Centric Computing (VL/HCC), 2010 IEEE Symposium*, 2010, pp. 227—234.

40. **Amálio N., Kelsen P., Ma Q., Glodt C.** Using VCL as an aspect-oriented approach to requirements modelling, *Transactions on aspect-oriented software development VII*, Springer Berlin Heidelberg, 2010, pp. 151—199.

41. **Amálio N., Glodt C., Kelsen P.** Building VCL models and automatically generating Z specifications from them, *FM 2011: Formal Methods.* Springer Berlin Heidelberg, 2011, pp. 149—153.

42. **Abraham R., Erwig M.** Goal-directed debugging of spreadsheets, *Visual Languages and Human-Centric Computing, 2005 IEEE Symposium on. IEEE*, 2005, pp. 37—44.

43. **Abraham R., Erwig M.** GoalDebug: A spreadsheet debugger for end users, *Proceedings of the 29th international conference on Software Engineering. IEEE Computer Society*, 2007, pp. 251—260.

44. **Fisher M., Rothermel G., Creelan T., Burnett M.** Scaling a Dataflow Testing Methodology to the MultiparadigmWorld of Commercial Spreadsheets, *Software Reliability Engineering, 2006. ISSRE'06. 17th International Symposium on. IEEE*, 2006, pp. 13—22.

45. **Abraham R., Erwig M.** Mutation operators for spreadsheets, *Software Engineering, IEEE Transactions*, 2009, vol. 35.1, pp. 94—108.

46. **Abraham R., Erwig M., Kollmansberger S., Seifert E.** Visual specifications of correct spreadsheets, *Visual Languages and Human-Centric Computing, 2005 IEEE Symposium on. IEEE*, 2005, pp. 189—196.

47. **Mendes J.** Coupled evolution of model-driven spreadsheets, *Software Engineering (ICSE), 2012 34th International Conference on. IEEE*, 2012, pp. 1616—1618.

48. **Cunha J., Fernandes J. P., Mendes J., Saraiva J.** MDSheet: A framework for model-driven spreadsheet engineering, *Proceedings of the 34th International Conference on Software Engineering. IEEE Press*, 2012, pp. 1395—1398.

49. **Engels G., Erwig M.** ClassSheets: automatic generation of spreadsheet applications from object-oriented specifications, *Proceedings of the 20th IEEE/ACM international Conference on Automated software engineering. ACM*, 2005, pp. 124—133.

50. **Cunha J., Fernandes J. P., Mendes J., Pacheco H., Saraiva J.** Bidirectional Transformation of Model-Driven Spreadsheets, *ICMT*, 2012, vol. 12, pp. 105—120.

51. **Stevens P.** Bidirectional model transformations in QVT: Semantic issues and open questions, *Model Driven Engineering Languages and Systems.* Springer Berlin Heidelberg, 2007, pp. 1—15.

52. **Pechenko I., Venger O.** Sposob avtomatizirovannogo sozdaniya diagram posledovatelnosty operaciy dlya scenariev povedeniya systemy, opisannih v vide texta, Informacionnye tehnologii, 2015, vol. 21, no. 1.