На правах рукописи

Карпов Андрей Николаевич

Технология настраиваемой генерации тестов по формальным спецификациям для встроенных приложений и программных интерфейсов, реализованных на Java-подобных языках

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

Автореферат
диссертации на соискание ученой степени кандидата технических наук

Санкт – Петербург - 2007

Работа выполнена в Государственном образовательном учреждении высшего профессионального образования «Санкт-Петербургский государственный политехнический университет».

Научный руководитель – кандидат технических наук, профессор
Котляров Всеволод Павлович

Официальные оппоненты – доктор технических наук, профессор
Лисс Александр Рудольфович

– кандидат физико-математических наук, доцент
Кознов Дмитрий Владимирович

Ведущая организация – Федеральное государственное унитарное предприятие "Научно-производственное объединение "Импульс"

Защита состоится « 15 » марта 2007 г. в 16.00 часов на заседании диссертационного совета Д 212.229.18 при ГОУ ВПО “Санкт-Петербургский государственный политехнический университет” по адресу: 195251, Санкт-Петербург, Политехническая ул., д.29, 9 уч. корп., ауд. 325.

С диссертацией можно ознакомиться в Фундаментальной библиотеке ГОУ ВПО “Санкт-Петербургский государственный политехнический университет”.

Автореферат разослан « 08 » февраля 2007 г.

Ученый секретарь диссертационного совета,
доктор технических наук, профессор Шашихин В.Н.

1. ОБЩАЯ ХАРАКТЕРИСТИКА РАБОТЫ

1.1 Актуальность работы. В рамках технологии производства программного обеспечения (ПО) важное значение имеет организация процессов тестирования, которые во многом определяют длительность, трудоемкость цикла разработки и качество ПО.

Возрастающая сложность ПО встроенных систем требует от соответствующих инструментальных средств:

- поддержки процессов тестирования, начиная с ранних этапов разработки;
- использования формальных спецификаций для ясного представления поведения тестируемой системы, а также проверки требований на непротиворечивость;
- поддержки автоматической генерации тестов;
- организации тестирования на разных этапах разработки с обеспечением преемственности (переиспользования) уже полученных результатов и наработок;
- поддержки тестирования в реальном окружении.

В связи с этим, при постоянном сокращении сроков разработки ПО и увеличении его сложности необходимо создание методик и инструментальных средств, которые позволили бы автоматизировать процесс тестирования.

Анализ предметной области показал, что в настоящее время широкое распространение имеют различные нотации для формального представления требований к тестируемой системе: UML, MSC, VDM-SL, Murφ, Alloy, T-VEC LF, LOTOS, Esterel, SCR, которые основаны на использовании таких известных моделей программ как сети Петри (Петри), конечные автоматы (Мили, Мура), темпоральные логики (Приора), алгебры параллельных процессов (Милнера, Хоара, Бергстры и Клоппа), транзиционные системы (Д. Парка). Существующий инструментарий, основанный на их использовании, TVG (от T-VEC Technologies), Rational Rose Test Realtime (от Rational), Rhapsody TestConductor (от I-Logix), UniTesK (ИСП РАН), GOTCHA-TCBEAN (от AGEDIS), MulSaw (от MIT), JUnit (от RoleModel Software) не удовлетворяет требованиям современного процесса производства ПО в области разработки встроенных Java программных интерфейсов и приложений, например, для мобильных телефонов, что является следствием следующих причин:

- трудоемко параметрическое описание взаимодействия процессов тестируемой системы и тестового окружения, а также функционалов контроля значений параметров этих процессов;
- отсутствуют подходы платформо-независимой интеграции тестовых наборов и тестируемого Java приложения;
- затруднено получение эффективного настраиваемого вида целевого кода тестов.

В диссертации автором разработана технология создания специализированных систем автоматизированного тестирования с помощью адаптивной генерации тестовых наборов и их интеграции с тестируемым приложением. Применение предлагаемой технологии позволяет существенно сократить длительность и трудоемкость цикла разработки, что и определяет актуальность диссертационной работы. Результаты анализа области исследования позволили сформулировать цели и задачи исследования и определить его структуру.

1.2 Цели и задачи диссертационной работы. Целью диссертационной работы является разработка комплекса методик и инструментальных средств технологии создания систем автоматизированного тестирования реактивных программных приложений мобильных Java телефонов, обеспечивающих сокращение трудозатрат и времени фазы тестирования при отсутствии регрессии качества. Для достижения цели в работе решены следующие задачи:

- разработка требований к технологии автоматизации тестирования ПО мобильных телефонов на основе проведенного анализа существующих методов и средств автоматизации тестирования;
- разработка модели взаимодействия тестируемого объекта и окружения на базе аппарата реактивных систем;
- создание методик разработки параметризованных тестовых сценариев, адаптивной генерации целевого кода тестов и их интеграции с тестируемой системой;
- создание программного инструментария, обеспечивающего: 1) адаптивную генерацию целевого кода тестовых сценариев по формальному описанию; 2) встраивание тестовых агентов в тестируемую систему для сопряжения с тестовым окружением;
- проверка работоспособности предложенных методик и инструментальных средств на 8 реальных проектах для 4 моделей мобильных телефонов.

Разработка и результаты решения этих задач выносятся на защиту.

1.3 Предметом исследования являются методы и инструментарий автоматизации тестирования встроенных Java программных интерфейсов и приложений.

1.4 Методы исследования. В диссертации используется теория реактивных систем и конечных автоматов, аппарат формальных спецификаций, концепция абстрактных тестовых сценариев. В коде применялись стандарты Message Sequence Charts (MSC), ANSI C и Java. Основными критериями являлись универсальность, адаптивность и простота автономного использования разрабатываемой технологии автоматизации тестирования, а также возможность интеграции с существующими инструментальными средствами. В основу исследований положен системный подход.

1.5 Обоснованность и достоверность полученных результатов обусловлена корректным использованием теории реактивных систем и конечных автоматов, аппарата формальных спецификаций; совпадением результатов по достижению качества программного обеспечения, полученных различными методами; положительными результатами использования предложенных методов и средств в реальных проектах и положительным опытом эксплуатации программного обеспечения, тестирование которого осуществлялось с использованием предложенных методик.

1.6 Научные результаты и их новизна: в диссертации разработаны методологические основы технологии создания систем автоматизированного тестирования. Суть этих результатов сводится к следующему.

1. Разработана модель взаимодействия тестируемой системы и тестового окружения на основе модели реактивных систем в виде системы временных переходов, расширенная средствами статической и динамической параметризации. Модель положена в основу функционирования систем автоматизированного тестирования и учитывает функциональность средств проверки параметров событий и сопряжения интерфейсов.
2. На базе стандарта MSC создана методика формализации требований к тестируемой системе посредством применения параметризованных, инвариантных тестовых сценариев, для последующего их автоматизированного преобразования в платформо-независимое представление.
3. Разработана методика генерации платформо-независимого абстрактного тестового набора в виде трасс и обобщенных сценариев для обеспечения возможности генерации целевого кода тестов на заданную платформу с помощью адаптируемого шаблона;
4. Предложена методика управления балансом выбора обобщенных сценариев и линейных трасс в совокупном тестовом наборе для удовлетворения ограничениям доступной памяти и времени выполнения тестового цикла;
5. Разработана методика создания шаблонов адаптивной генерации кода, позволяющая преобразовать платформо-независимое представление тестовых сценариев в платформо-зависимый целевой код тестов путем интерпретации команд абстрактного теста.
6. Предложена методика встраивания тестовых агентов в Java приложения для сопряжения интерфейсов тестируемой системы и тестового окружения, что позволило обеспечить эмуляцию тестовых воздействий и контроль откликов.

1.7 Практическая значимость работы. На базе полученных научных результатов разработан комплекс программных средств, предназначенных для автоматизации тестирования Java программных интерфейсов и приложений для мобильных телефонов. Программный комплекс использован в компании Motorola в 8 программных проектах в

таких областях как: разработка Java программных интерфейсов (JSRs) и базового приемочного (Sanity) тестирования, а также при разработке пользовательских (MIDlet) и системных (CORElet) Java приложений. Созданная технология внедрена в проекте по разработке аппарата оповещения о статусе систем контроля в ОАО “Интелтех” и проекте “Исследование применения мобильных технологий в задаче отслеживания движения поездов” в ЗАО “Северо-Западная лаборатория Лтд”. Созданные комплекс методик и программные средства являются универсальными и могут быть использованы для автоматизации тестирования Java программных интерфейсов и приложений различной направленности. Применение разработанной технологии автоматизации тестирования позволяет в среднем сократить время фазы тестирования в 2,5 раза.

1.8 Апробация работы. Основные результаты и выводы диссертации докладывались на следующих международных научных конференциях: “IEEE Russia Northwest Section, 110 Anniversary of Radio Invention conference” (СПб., 2005 г.); “2006 IEEE Tenth International Symposium on Consumer Electronics” (СПб., 2006 г.); Motorola Technology Day (Spb 2005, 2006); конференциях “Технологии Microsoft в теории и практике программирования” 2002, 2004, 2005 и 2006 гг.; конференциях XXIX–2001 г, XXX–2002 г, XXXII–2004 г, XXXIII–2005 г, недели науки СПбГПУ. По материалам диссертации опубликовано 8 печатных работ, в том числе статья в издании из списка ВАК, где достаточно полно изложены основные результаты работы.

1.9 Внедрение. Разработанная технология автоматизации тестирования внедрена в ЗАО “Северо-Западная лаборатория”, НПО “Интелтех”, ЗАО “Моторола” и использована при разработке учебно-методического комплекса СПбГПУ по курсу “Автоматизация тестирования Java на мобильных телефонах” на кафедре “Информационных и управляющих систем”. Практическое использование представляемых на защиту результатов подтверждено соответствующими актами о внедрении.

1.10 Структура и объем работы. Работа содержит введение, 5 глав, заключение и 5 приложений. Объем работы 145 страниц текста, количество иллюстраций 80, список использованной литературы содержит 95 наименований.

2. СОДЕРЖАНИЕ РАБОТЫ

В **первой главе** диссертации на основе анализа состояния предметной области исследования сделано заключение о современном состоянии теории, практики и средств автоматизации тестирования и необходимости их совершенствования:

1. Основным направлением развития теории и практики процесса тестирования является совершенствование существующих методов тестирования для решения задач его автоматизации.

2. Современная трактовка автоматизации тестирования включает не только автоматизированное выполнение тестовых наборов, но и автоматизацию их создания.
3. Автоматизация создания тестовых наборов основывается на применении формальных нотаций и методов для задания требований к тестируемой системе и обеспечения формального преобразования, т.е. генерации тестов по спецификациям.
4. Проведенный анализ позволил выявить следующие недостатки методов и средств автоматизации тестирования:

- использование не стандартизованной формальной нотации описания требований усложняет внедрение методов и средств автоматизации тестирования в процесс промышленного производства ПО и затрудняет интеграцию с другими подходами обеспечения качества ПО;
- необходимость задания полной модели объекта тестирования препятствует описанию поведенческих сценариев проверки взаимодействия тестируемой системы и окружения на выделенных интерфейсах;
- обеспечение генерации только в фиксированное множество целевых языков тестовых наборов и ограничения по адаптации вида генерируемого целевого кода препятствуют применению методов и средств тестирования для различных целевых платформ;
- применение подхода сопряжения тестируемой системы и тестового окружения по внешним программным интерфейсам затрудняет их использование для тестирования встроенных Java приложений;
- осуществление контроля работы тестируемой системы на основе сравнения простых типов данных затрудняет использование таких подходов при необходимости производить контроль отображения графических образов.

5. Дальнейшее направление развития автоматизации тестирования состоит в адаптации существующих общих подходов к различным целевым областям применения с задачей повышения качественного уровня процесса автоматизации тестирования.

Полученные результаты анализа позволили установить отсутствие и необходимость создания эффективных технологий для построения адаптивных систем автоматизированного тестирования Java приложений и программных интерфейсов встраиваемых вычислительных платформ.

Во **второй главе** разработана модель функционирования систем автоматизированного тестирования, основанная на взаимодействии тестируемой системы и тестового окружения на базе модели реактивных систем в виде системы временных переходов. Взаимодействие тестируемой системы и тестового окружения осуществляется на множестве интерфейсов I. Пусть интерфейс I_i специфицирован некоторой функцией

F_i с заданным множеством параметров $V_i = \{v_j\}$: $I_i = F_i(V_i)$. Использование интерфейса в заданный момент времени определяется сигналом – вызовом функции F_i с заданными значениями параметров $\forall v_j \in V_i$.

Обосновано расширение формального представления системы временных переходов для описания поведения реактивных систем с учетом специфики предметной области тестирования Java приложений и программных интерфейсов. Система временных переходов описывается кортежем: $S^T = \langle X, \Sigma, \mathrm{T}, \Theta, L, U \rangle$, где: X – множество переменных над областью определения D; $\Sigma : \{X \to D\}$ – множество состояний системы; $T \subseteq \{\Sigma \to \Sigma\}$ – конечное множество переходов; $\Theta \in \Sigma$ – множество начальных состояний, L, U – множества нижних и верхних границ интервалов временных задержек срабатывания активизированных переходов $\tau \in \mathrm{T}$: $L = \{l_\tau, \tau \in \mathrm{T}\}$; $U = \{u_\tau, \tau \in \mathrm{T}\}$; $0 \le l_\tau \le u_\tau$; $l_\tau, u_\tau \in R$.

В общем случае имеет место взаимодействие двух совокупных параллельных процессов тестируемой системы и окружения посредством обмена сигналами, определяющими значения параметров вызываемых интерфейсов (рис. 1).

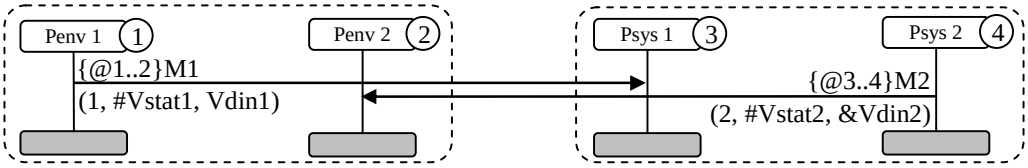

Рис. 1 Поведенческое взаимодействие тестируемой системы и тестового окружения

В связи с наличием множества взаимодействующих процессов, как со стороны окружения, так и со стороны тестируемой системы введена индексация взаимодействующих процессов: $P : (P_{sys}, P_{env})$, где P_{sys} (3,4) – множество процессов со стороны тестируемой системы, а P_{env} (1,2) – множество процессов со стороны тестового окружения. Для кодирования сигналов воздействия тестового окружения и контроля ответов от тестируемой системы предложено использовать статическую и динамическую параметризации $V : (V_{stat}, V_{din})$. Множество статических параметров V_{stat} предложено описывать макросредствами, что позволяет производить декомпозицию тестовых сценариев на этапе предварительной обработки посредством подстановки значений параметров из их области определения $D_{V_{stat}}$ для $\forall V_{stat_i}$ $\exists I_{V_{stat}} : V_{stat} \to D_{V_{stat}}$. Множество динамических параметров V_{din} предложено описывать переменными, значение которых вычисляется в процессе выполнения теста.

Для проверки в процессе тестирования значений параметров интерфейсов (F_{params}), значений идентификаторов процессов отправителя и получателя (F_{inst}), временных

ограничений (F_{time}) предложено использовать $F_{check} : \left(F_{params}, F_{inst}, F_{time}\right)$. В общем случае спецификация интерфейсов тестируемой системы и тестового окружения различна. Для их сопряжения предложено использовать функциональные преобразователи $F_{send} : I_{test}^{out} \to I_{sys}^{inp}$ и $F_{recv} : I_{sys}^{out} \to I_{test}^{inp}$, где I_{test}^{inp} и I_{test}^{out}, I_{sys}^{inp} и I_{sys}^{out} – множества входных и выходных интерфейсов тестового окружения и тестируемой системы. Таким образом, двусторонний функциональный преобразователь можно определить, как $F_{converter} : \left(F_{send}, F_{recv}\right)$. В общем виде разработанная модель определяется парой $S^{T*} = \left\langle S^T, EXT \right\rangle$, где S^T – система временных переходов, а EXT – расширение базовой модели, определяемое кортежем $\left\langle V, I_{V_{stat}}, P, F_{check}, F_{convert} \right\rangle$.

Основываясь на предложенной модели, сформулированы требования к технологии создания систем автоматизированного тестирования в заданной предметной области, которая должна обеспечить:

1. Платформо-независимое инвариантное описание тестовых сценариев.
2. Генерацию платформо-независимого представления тестов с возможностями конфигурации и управления процессом генерации.
3. Адаптивную генерацию платформо-зависимого целевого кода тестовых сценариев и тестового окружения с учетом задаваемой пользователем конфигурации.
4. Интеграцию сгенерированного набора тестов с тестируемой системой.

Для создания тестовых сценариев в работе предложено: формальное графическое описание поведенческого взаимодействия тестируемой системы и тестового окружения проводить на языках MSC/UML; первый этап генерации тестов осуществлять на платформо-независмый скриптовый язык; на втором этапе производить адаптивную интерпретацию абстрактных тестов в целевой платформо-зависимый код, содержание которой определяет эксперт в целевой области тестирования, чем достигается эффективность целевого кода тестового набора.

В **третьей главе** разработана концепция технологии создания систем автоматизированного тестирования встроенного Java программного обеспечения (рис. 2). В качестве основных этапов создания систем автоматизированного тестирования определены:

1. **Формализация поведенческого описания работы тестируемой системы** в виде тестовых сценариев в расширенной нотации MSC/UML (1).

2. **Предварительная обработка множества тестовых сценариев** с раскрытием статической параметризации, производимая макро препроцессором (2, 3, 4), который осуществляет преобразование $\left\langle S_{MSC}^{macro}, V_{stat}, D_{V_{sstat}}, I_{V_{sstat}} \right\rangle \to \left\langle S_{MSC} \right\rangle$, где: $S_{MSC}^{macro} : \left(s_{msc1}^{macro}, .., s_{mscN}^{macro}\right)$

– множество MSC сценариев; $V_{stat}:\left(v_1^{stat},\ldots,v_M^{stat}\right)$ – множество статических параметров;

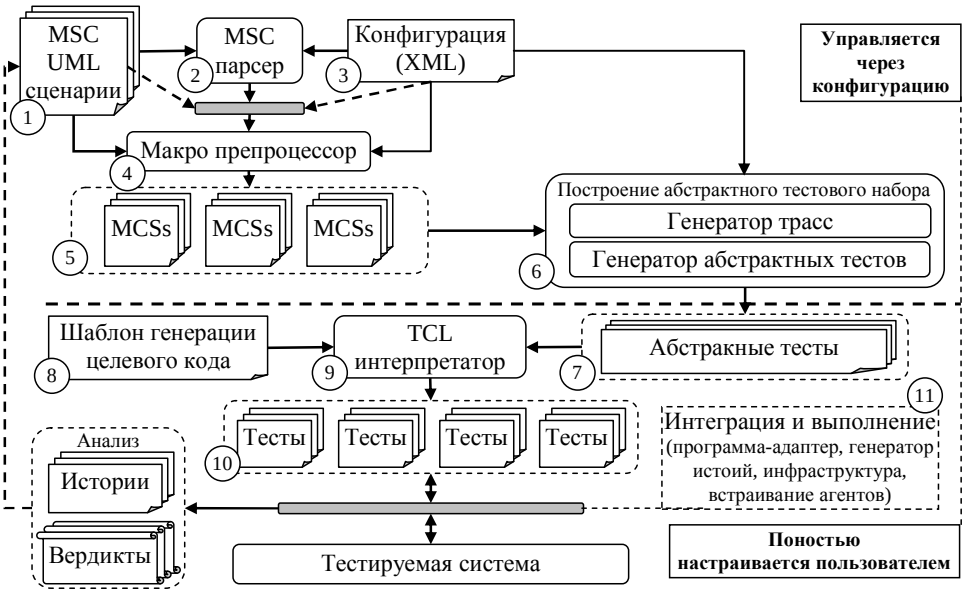

Рис. 2 Технологическая цепочка автоматизации тестирования

$D_{V_{stat}}$ – область их определения; $I_{V_{stat}}$ – описание перебора значений из $D_{V_{stat}}$, используемого в тестовых экспериментах; $S_{MSC}:\left(\left(s_{msc11},..,s_{msc1K1}\right),..,\left(s_{mscN1},..,s_{mscNK\,N}\right)\right)$ – генерируемое множество сценариев с развернутой статической параметризацией при $K:\left(K_1,..,K_N\right)$, где K_i – число переборов для каждого сценария s_{msci}^{macro}.

3. **Преобразование MSC сценариев в платформо-независимое представление**, которое предложено осуществлять генератором трасс и абстрактных тестовых наборов (6, 7) посредством преобразования вида: $\left\langle S_{MSC},I,P,V_{din},Ftr\right\rangle\rightarrow\left\langle Tr_{MSC}\,|\,Tst_{abstr}\right\rangle$ где: $S_{MSC}:\left(s_{msc1},\ldots,s_{mscN}\right)$ – множество MSC сценариев; $I:\left(I_{test}^{out},I_{sys}^{out},I_{test}^{inp},I_{sys}^{inp}\right)$ – множество интерфейсов; $P:\left(P_{sys},P_{env}\right)$ – множество процессов; $V_{din}:\left(v_1^{din},\ldots,v_M^{din}\right)$ – множество переменных; $Ftr:\left(ftr_1,\ldots,ftr_K\right)$ – множество фильтров; $Tr_{MSC}:\left(\left(tr_{msc11},\ldots,tr_{msc1L_1}\right),\ldots,\left(tr_{mscN1},\ldots,tr_{mscNL_N}\right)\right)$ – множества MSC трасс при $L:\left(L_1,\ldots,L_N\right)$, где L_i – число трасс для каждого сценария s_{msci}; $Tst_{abstr}:\left(tst_{abstr1},\ldots,tst_{abstrN}\right)$ – множество абстрактных платформо-независимых тестов (7).

4. **Адаптивная генерация целевого кода тестов**, которую предложено осуществлять из платформо-независимого представления посредством интерпретации абстрактных тестов с использованием механизма шаблонов генерации кода (8, 9, 10) в виде преобразователя: $F:\left(Tst_{abstr},MAP\left(DT_{abstr}:DT_{code}\right)\right)\xrightarrow{TCL}\left(Tst_{code},WRAP\right)$, где: $Tst_{abstr}:\left(tst_{abstr1},\ldots,tst_{abstrN}\right)$ – множество абстрактных тестов; $MAP\left(DT_{abstr}:DT_{code}\right)$ – множество пар соответствия абстрактных DT_{abstr} и целевых DT_{code} типов данных; $Tst_{code}:\left(tst_{code1},\ldots,tst_{codeN}\right)$ – множество

тестов на целевом языке; *WRAP* – оболочка для функционального преобразователя $F_{converter}$ с описанием интерфейсов тестового окружения I_{test}^{inp} и I_{test}^{out} (11).

5. **Сопряжение интерфейсов тестового окружения и тестируемой системы** с помощью функционального преобразователя $F_{converter}$ предложено осуществлять пользователю с применением I_{sys}^{inp} и I_{sys}^{out}, полученных из приложения (байт-код анализ) посредством преобразования $F:\left(C, c_0, A_{(priv,prot)}, MAP(C_{std}:C_{agt})\right)\rightarrow\left(C^*, c_0^*, A_{pub}\right)$, где: C – набор классов исходного приложения; c_0 – основной наследуемый класс; $A_{(priv,prot)}$ – *private* и *protected* атрибуты полей и методов классов, заменяемых на *public* A_{pub}; $MAP(C_{std}:C_{agt})$ – множества пар соответствия стандартных библиотечных классов C_{std} и классов-оболочек агента C_{agt}; C^* – множество модифицированных классов; c_0^* – модифицированный основной класс, который наследует исходный c_0.

Представленные программные реализации вошли в релиз инструментального комплекса и позволили обеспечить создание технологии автоматизации тестирования. Все модули оттестированы и использованы при пилотировании технологии в реальных программных проектах, представленных в пятой главе.

В четвертой главе на основе предложенной модели взаимодействия тестируемой системы и тестового окружения разработаны методологические основы в виде комплекса взаимосвязанных методик, реализующие концепцию создания специализированных систем автоматизированного тестирования, изложенную в третьей главе.

Методика разработки тестовых сценариев в формальной графической расширенной нотации MSC позволяет разработчику тестов формализовать поведенческие требования к тестируемой системе в виде множества MSC сценариев S_{MSC}^{macro}. Определение каждого сценария s_{msci}^{macro} предложено производить по следующим этапам:

- описание конечного множества процессов $P:(SYS, ENV)$ тестируемой системы и тестового окружения, между которыми происходит обмен сигналами;
- задание: конечного множества переходов (сигналов) $T\subseteq\{\Sigma\rightarrow\Sigma\}$, где Σ – множество состояний; множества нижних и верхних границ временных интервалов срабатывания переходов $L,U:(ABS, REL, LAB)$, задание которых предложено осуществлять абсолютным (ABS) или относительным значением, отсчитываемым от предыдущего (REL) события или некоторой базы (LAB); множества параметров $V:\left(V_{stat}, V_{din}\right)$;
- описание последовательности переходов с использованием: in-line операторов

$OP_{IN-LINE}:(Alt, Opt, Par, Exc, Loop)$, задающих альтернативное (Alt), опциональное (Opt), параллельное (Par), исключительное (Exc) и циклическое ($Loop$) поведения; дополнительно предложенных управляющих операторов ветвления (IF), условных циклов ($WHILE$), проверки условий ($ASSERT$), ссылок (REF) $OP_{CTRL}:(IF, WHILE, ASSERT, REF)$;

- задание конфигурации, в которой указываются области определения $D_{V_{stat}}$ и предложенные функционалы перебора $I_{V_{stat}}:(LIST, COND, FUNC)$, такие как: списки ($LIST$), описывающие набор значений; условные списки ($COND$); функции ($FUNC$), позволяющие реализовать пользовательскую подстановку параметров в библиотеке.

Над результирующим множеством и определенной конфигурацией производится формальное преобразование $\left\langle S_{MSC}^{macro}, V_{stat}, D_{V_{sstat}}, I_{V_{sstat}} \right\rangle \rightarrow \left\langle S_{MSC} \right\rangle$ в макро препроцессоре.

Методика генерации трасс и абстрактных тестов основывается на результате работы методики разработки тестовых сценариев и обеспечивает их преобразование в платформо-независимый тестовый набор посредством генерации линейных трасс и обобщенных сценариев $\left\langle S_{MSC}, I, P, V_{din}, Ftr \right\rangle \rightarrow \left\langle Tr_{MSC} \mid Tst_{abstr} \right\rangle$. Описание интерфейсов I, процессов P и переменных V_{din} поступает из конфигурации, множество фильтров $Ftr:(SYNC, LIM)$ в виде условий синхронизации $SYNC:(Sync_{SYS}^{P}, Sync_{ENV}^{P}, Sync_{NONE}^{P}, Sync_{OVERLAP})$ и ограничений на число ветвлений $LIM:(Lim_{ACTIVE}^{BRANCH}, Lim_{PASSIVE}^{BRANCH}, Lim_{(A:P)}^{PRIOR})$ задаются пользователем. Синхронизацию предложено определять выбором приоритета обработки событий на стороне процессов тестового окружения $Sync_{ENV}^{P}$, тестируемой системы $Sync_{SYS}^{P}$, без указания приоритета $Sync_{NONE}^{P}$ и режимом разделения событий посылки и приема сигнала $Sync_{OVERLAP}$. Ограничения на число ветвлений по типу событий в каждой точке нелинейного поведения предложено определять заданием числа возможных активных путей Lim_{ACTIVE}^{BRANCH}, пассивных путей $Lim_{PASSIVE}^{BRANCH}$ и их приоритетом выбора $Lim_{(A:P)}^{PRIOR}$.

Методика управления балансом выбора обобщенных сценариев и линейных трасс на основе заданного ограничения по доступной памяти позволяет сокращать время выполнения тестового набора за счет баланса объема линейных трасс и обобщенных сценариев в результирующем тестовом наборе. В работе показано, что объем памяти, занимаемый трассами, больше либо равен объему памяти, занимаемый обобщенными сценариями за счет повторения линейных участков: $MEM_{SC} \leq MEM_{TR}$. При этом время выполнения набора трасс меньше либо равно времени выполнения эквивалентного обобщенного сценария за счет дополнительных проверок для выбора пути: $T_{TR} \leq T_{SC}$. В методике предлагается следующий подход: 1) заменять сценарии множеством трасс до

достижения заданного ограничения на доступную память; 2) если памяти недостаточно, а время выполнения необходимо сокращать, переходить к пакетной загрузке. При этом необходимо учитывать время загрузки пакетов: $T_{прогона} = (T_{исполнения} + T_{загрузки}) * N_{пакетов}$. Тогда при увеличении (↑) объема сценариев растет время прогона и уменьшается (↓) время загрузки: $Sc \uparrow \Rightarrow \forall Pack_i : (T_{прогона} \uparrow \& T_{загрузки} \downarrow)$, а при увеличении объема трасс: $Tr \uparrow \Rightarrow \forall P_i : (T_{прогона} \downarrow \& T_{загрузки} \uparrow)$. Для пакетной обработки, чем значительнее время загрузки, тем большую долю сценариев необходимо обеспечить для каждого пакета.

Методика создания шаблонов адаптивной генерации целевого кода тестов обеспечивает построение функционального преобразователя для получения целевого кода тестов $F : (Tst_{abstr}, MAP(DT_{abstr} : DT_{code})) \xrightarrow{TCL} (Tst_{code}, WRAP)$ посредством интерпретации команд абстрактного теста. Каждый абстрактный тест $tst_{abstr\ i}$ представляет собой последовательность команд, описывающих взаимодействие тестируемой системы и окружения: $\langle StartTC, StopTC, (I_{test}^{inp}, I_{test}^{out}), Events : (Snd, Rcv, Cnd, Act, Rfr, SLoop, FLoop), States \rangle$.

Команды *StartTC* и *StopTC* описывают начало и окончание теста. *Events* описывают множество сигналов посылки (*Snd*) и приема (*Rcv*) с указанием интерфейсов I, процессов P, параметров V, временных ограничений (L, U) или операций: условие (*Cnd*), действие (*Act*), ссылка (*Ref*), начало (*SLoop*) и окончание (*FLoop*) цикла. Множество команд *States* описывает события и переходы каждого состояния. Из множества интерфейсов тестового окружения $(I_{test}^{inp}, I_{test}^{out})$ генерируются функции, образующие оболочку *WRAP* функционального преобразователя $F_{converter}$. В соответствие командам ставятся блоки целевого кода, обеспечивающие контроль значений параметров, принадлежности процессам и временных ограничений: $F_{check} : (F_{params}, F_{inst}, F_{time})$, и образующие $tst_{code\ i}$.

Предложено два подхода реализации механизма кодогенерирующего шаблона (КШ): специализированный – для использования в конкретном проекте и универсальный – для многоцелевого использования. Для разработки КШ выделены и обоснованы следующие этапы: 1) определение уровня абстрактности интерфейсов; 2) разработка правил преобразования команд абстрактного теста в целевой код; 3) определение структур и типов данных; 4) разработка структуры генерируемого кода; 5) организация протоколирования процесса тестирования; 6) проектирование служебной не генерируемой функциональности; 7) дизайн типовой программы-адаптера.

Методика встраивания тестовых агентов обеспечивает работу функционального

преобразователя $F:\left(C, c_0, A_{(priv,prot)}, MAP(C_{std}:C_{agt})\right)\rightarrow\left(C^*, c_0^*, A_{pub}\right)$ для получения программных интерфейсов встроенного приложения мобильного телефона. В ходе ее разработки решены следующие вопросы: 1) определено множество классов стандартных библиотек c_0, C_{std}, которые необходимо подменить для получения контроля над тестируемой системой; 2) разработана библиотека классов-оболочек с переопределением стандартных методов; 3) необходимые методы расширены путем дополнения функциональности воздействия и контроля реакций тестируемой системы $(I_{sys}^{inp}, I_{sys}^{out})$; 4)проведено сопряжение интерфейсов тестового окружения с полученными интерфейсами тестируемой системы через $F_{send}: I_{test}^{out}\rightarrow I_{sys}^{inp}$ и $F_{recv}: I_{sys}^{out}\rightarrow I_{test}^{inp}$ в виде $F_{converter}:\left(F_{send}, F_{recv}\right)$.

Таким образом, четвертая глава содержит методическую базу разрабатываемой технологии. Конструктивность и эффективность разработанных методик подтверждена при их реализации и пилотировании на реальных программных проектах.

В пятой главе представлены результаты проверки работоспособности разработанной технологии в реальных проектах по созданию Java программных интерфейсов, пользовательских и системных приложений. При этом для каждого проекта применялись методики, соответствующие постановке задачи.

Обобщенная схема применения технологии приведена на рис. 3.

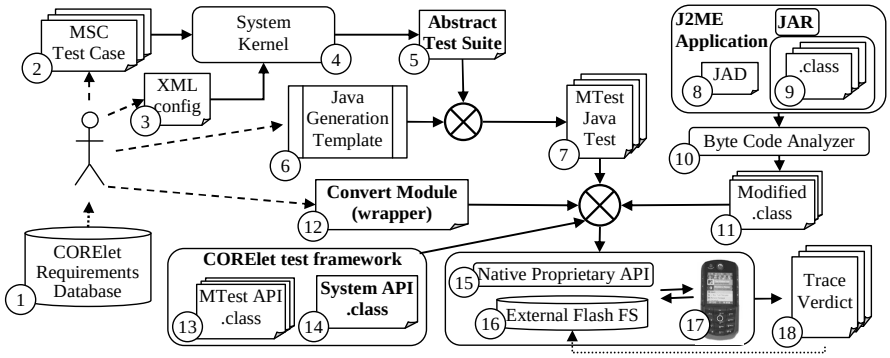

Рис. 3 Схема применения технологии автоматизации тестирования

Из структуры, показанной на рис. 3, видно, что по требованиям из базы данных (1) составляются формализованные тестовые сценарии с использованием MSC (2), которые совместно с конфигурацией (3) поступают в ядро инструментальной поддержки (4). Результатом работы ядра является абстрактный тестовый набор (5), при интерпретации которого с использованием шаблона (6) производится генерация целевого кода тестов (7). На основании применения методики встраивания тестовых агентов производится анализ и модификация (10) байт-кода Java-классов исходного приложения (8) для их интеграции (11) с пакетом классов-оболочек (13) совместно с набором сгенерированных тестов (7) и программой-адаптером (12). При тестировании Java системных приложений

используются дополнительные библиотеки (15) для получения доступа к ресурсам платформы (16, 17). Проверка корректности выполнения теста осуществляется в процессе выполнения (on-line), либо по его окончании (off-line) с помощью сравнения с эталонными образами экрана. Результатом работы системы являются трассы выполнения тестов (18).

Оценка разработанной технологии осуществлялась на 8 реальных проектах (табл. 1) различной сложности: три проекта связаны с тестированием Java программных интерфейсов (JSR-120, JSR-135, Sanity); два проекта связаны с тестированием пользовательских (Logic Game, Picture Viewer) и три проекта связаны с тестированием системных (Image Editor, Music Player; User App) Java приложений.

Таблица 1 Характеристики применения технологии в промышленных Java-проектах

Метрики / Проекты	Java API			MIDlets		CORElets		
	JSR120	JSR135	Sanity	Logic Game	Picture Viewer	Image Editor	Music Player	User App
Общее число требований	68	90	—	121	106	183	156	467
Общее число тестов	48	59	98	116	854	328	199	432
Число автоматизированных тестов	48	41	93	112	405	289	172	391
Разработка ручных тестов (ч/д)	40	30	25	7	56	45	25	86
Разработка MSC сценариев (ч/д)	42	35	25	8	55	44	30	90
Настройка: шаблон или адаптер (ч/д)	8	8	8	5	8	6	5	5
Прогон ручного тестового набора (ч/ч)	20	16	8	16	40	48	16	152
Прогон автомат. тестов: (м/ч)/(ч/ч)	7/1	6/1	4/0,5	4/0,5	16/2	16/2	8/1	40/8
Прогон неавт. остатка тест. набора (ч/ч)	0	4	1	1	4	6	1,5	16
Число дефектов, ручное тестирование	65	90	300	135	390	77	110	290
Число дефектов, автомат. тестирование	69	98	315	152	421	82	121	321
Число циклов тестирования	17	19	298	26	28	15	14	14
ч/ч – человеко-часы; ч/д – человеко-дни; м/ч – машино-часы								

Данные табл. 1 показывают, что для представленных проектов наблюдаются общие тенденции. При росте числа циклов тестирования наблюдается увеличение выигрыша от применения технологии автоматизации тестирования по трудозатратам и времени.

На рис. 4 (1) приведено сравнение роста трудозатрат и времени при увеличении проведенных циклов тестирования. Усредненная оценка по восьми проектам для четырех моделей мобильных телефонов производилась для четырнадцати тестовых циклов.

При анализе степени автоматизации представленных проектов можно выделить три основных типа: **Тип А** – проекты, поведение в которых кроме взаимодействия “пользователь – мобильный телефон” во многом определяется “клиент-серверной” архитектурой с взаимодействием пользователя с сервером без участия мобильного телефона (“Picture Viewer”), а также отображения видеоряда и проигрывания или записи звуковых файлов различных форматов (“JSR-135”). Наличие в общем объеме тестового набора значительной части ручных тестов по отношению к объему автоматизированных тестов позволяет оценить степень автоматизации проектов типа А на уровне 70%. **Тип Б** – проекты, в которых также присутствует “клиент-серверный” компонент, но при этом работа пользователя с сервером сконцентрирована на стороне мобильного телефона, а

также проекты, у которых анимация осуществляется на уровне приложений, а не на системном уровне вычислительной платформы (“Sanity”, “ Image Editor”, ”Music Player”, “User App”). Для проектов этой группы уровень автоматизации оценивается в 70% – 95%.

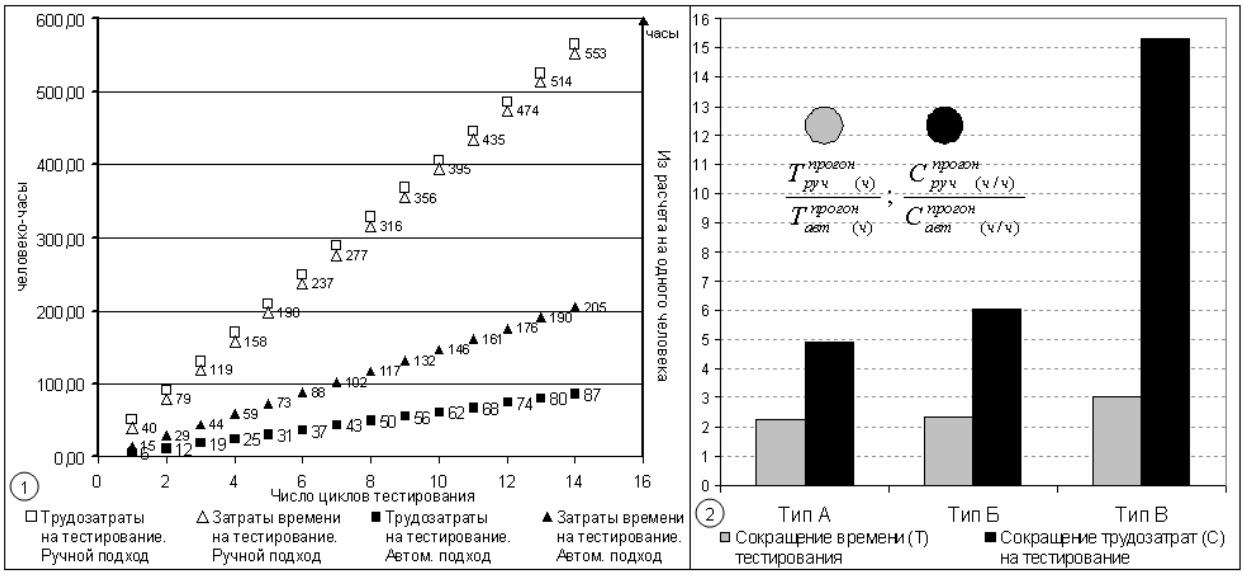

Рис. 4 Среднее значение трудозатрат и времени в зависимости от количества циклов (1) и сокращение трудозатрат и времени по типам проектов (2)

Тип В – проекты, в которых отсутствует необходимость отображения динамической графической информации, проигрывания звуковых файлов, когда взаимодействия осуществляются прямо на мобильном телефоне (“JSR-120” и “Logic Game”). Для группы проектов этого типа достигается коэффициент автоматизации до 100%. Результаты анализа приведены на рис. 4 (2), где показана зависимость сокращения трудозатрат и времени по типам проектов.

Диаграмма на рис. 4 (2) показывает эффективность применения разработанной технологи автоматизации тестирования. Сокращение необходимых ресурсов наблюдается по всем типам проектов. Ускорение по времени исполнения тестового цикла более чем в 2 раза достигается при 60-70% автоматизации тестового набора для проектов типа А, а трудозатраты сокращаются в 5 раз. При этом для проектов типа В по тем же параметрам достигаются улучшения в 3 и 15 раз соответственно, как показано на рис. 4 (2).

3. ОСНОВНЫЕ РЕЗУЛЬТАТЫ И ВЫВОДЫ

В диссертационной работе решена научная задача создания систем автоматизированного тестирования встроенных Java программных интерфейсов и приложений на основе формальных спецификаций, позволяющая существенно сократить трудозатраты и время на процесс тестирования.

Основными результатами диссертационной работы являются:

1. Модель описания поведенческого взаимодействия тестируемой системы и тестового

окружения, положенная в основу функционирования систем автоматизированного тестирования, и которая позволила описать процесс тестирования встроенного Java ПО, поддержать механизмы свертки тестовых сценариев и процедуры удовлетворения ресурсным ограничениям встраиваемых вычислительных платформ.

2. Методологические основы создания систем автоматизированного тестирования в виде комплекса взаимосвязанных методик, ориентированных на поддержку следующих этапов технологии автоматизации тестирования:

- разработка тестовых сценариев для обеспечения инвариантности тестовых сценариев относительно платформы тестирования с помощью формализации спецификации поведенческого взаимодействия тестируемой системы и тестового окружения в расширенной нотации MSC;
- генерация трасс и абстрактных тестов для преобразования графического описания тестового сценария на MSC в платформо-независимое представление на языке Tcl;
- преобразования графического представления тестового сценария в абстрактный платформо-независимый вид (Tcl скрипт);
- управление балансом выбора объема генерации линейных трасс и обобщенных сценариев для удовлетворения ограничениям памяти и времени выполнения тестов;
- создание шаблонов генерации целевого кода тестов для адаптивного, контролируемого пользователем, преобразования платформо-независимого представления тестовых сценариев в платформо-зависимый целевой код тестов;
- встраивание тестовых агентов в тестируемые Java приложения для сопряжения интерфейсов тестируемой системы и тестового окружения, что позволило обеспечить эмуляцию тестовых воздействий и контроль откликов.

3. Программный комплекс поддержки методик технологии автоматизации тестирования, обеспечивающий этапы создания систем автоматизированного тестирования.

4. Оценка эффективности разработанных методик и ПО на базе использования в программных проектах различной сложности в ЗАО “Motorola ЗАО”, ЗАО “Северо-Западная лаборатория”, НПО “Интелтех” позволила установить, что при увеличении затрат на разработку автоматизированного тестового набора на 25% по сравнению с традиционным ручным тестированием применение технологии обеспечивает выигрыш до 3 раз по затратам времени и до 15 раз по трудозатратам.

Результаты, полученные в процессе выполнения проектов с использованием разработанной технологии, позволили сделать выводы о работоспособности и эффективности методов и средств технологии автоматизации процесса тестирования для Java программных интерфейсов и приложений.

4. СПИСОК ПУБЛИКАЦИЙ ПО ТЕМЕ ДИССЕРТАЦИИ

1. Котляров В.П., Голубев А.А., Карпов А.Н., Автоматизация тестирования встроенных Java приложений с адаптацией к целевой платформе // Научно-технические ведомости СПбГТУ – СПб.: Изд-во Политехнического Университета, 2006, №5, с. 117-124.,
2. Vsevolod P. Kotlyarov, Alexey A. Golubev, Andrey N. Karpov, Testing automation for system core kJava applications // IEEE Tenth International Symposium on Consumer Electronics conference, 2006, pp.596-599.
3. Голубев А.А., Карпов А.Н., Котляров В.П. Автоматизация тестирования системных J2ME приложений // Технологии Microsoft в теории и практике программирования: Материалы межвузовского конкурса-конференции студентов, аспирантов и молодых ученых Северо-Запада – СПб.: Изд-во Политехнического Университета, 2006, с.28-30.
4. Vsevolod P. Kotlyarov, Alexey A. Golubev, Andrey N. Karpov, Testing automation for J2ME applications and API // IEEE Russia Northwest Section, 110 Anniversary of Radio Invention conference, Volume II, 2005, pp.98-103.
5. Мелькова Е.Н., Карпов А.Н., Котляров В.П., Применение формального языка спецификаций в целях автоматизации тестирования J2ME-приложений // XXXIII неделя науки: Материалы Всероссийской межвузовской научно-технической конференции студентов и аспирантов Ч.V – СПб.: Изд-во Политехн. ун-та, 2005с. 36-37.
6. Карпов А.Н., Котляров В.П. Применение автоматизированной технологии верификации и тестирования на основе формальных спецификаций в разработке телекоммуникационных систем // XXXII Неделя науки СПбГТУ: Материалы межвузовской научно-технической конференции. Ч.V. – СПб.: Изд-во СПбГПУ, 2004, с.12-13
7. Яковенко Н.М., Кривченко О.В., Карпов А.Н. Разработка Java-приложения «SMS connection API» для устройств беспроводной связи реализующего работу с SMS сообщениями // Конкурс конференция студенческих работ в области современных технологий компании Microsoft: Материалы межвузовского конкурса-конференции – СПб.: Изд-во СПбГПУ, 2002, с. 102
8. Никитин М.А., Югай Д.В. Карпов А.Н. Разработка программного модуля на PC, обеспечивающего взаимодействие мобильного телефона, подключенного к PC через последовательный интерфейс, с мобильными телефонами посредством SMS сообщений // Конкурс конференция студенческих работ в области современных технологий компании Microsoft – СПб.: Изд-во СПбГПУ, 2002, с. 68-69