Завдяки таким характеристикам використовуваних інструментальних засобів роботи з інформаційними потоками, як оперативність, повнота і релевантність, а також наявності єдиного захищеного інтерфейсу технологія контент-моніторингу може сприяти значному підвищенню ефективності та якості інформаційно-аналітичної роботи.

1.Берко А. Ю. та ін. Системи електронної контент-комерції: моногр. / А. Ю. Берко, В. А. Висоцька, В. В. Пасічник. – Львів: Видавництво Львівської політехніки, 2009. – 612 с. 2. Додонов А. Г. Виявлення категорій і їх взаємозв'язків у рамках технології контент-моніторингу / Додонов А. Г., Ланде Д.В. // Вісник державної служби України. — 2006. — № 4. – С. 45–52 с. 3. Григорьев А., Ландэ Д.. Контент-мониторинг сетевых информационных потоков [Электронный ресурс] / Официальный ежемесячный www-регистр бизнес-ресурсов Украины и зарубежья — Режим доступа: http://infostream.com.ua/infostream/publ/cbr/index.shtml — Загол.с экрана. 4. Ланде Д.В. Програмно-апаратна система інформаційної підтримки прийняття рішень: наук.-метод. посіб. / Д.В. Ланде, В.М.Фурашев, О.М. Григор'єв. – Київ, 2006. – 48 с. 5. Леліков Г.І. Моніторинг діяльності органів виконавчої влади із застосуванням комп'ютерної системи контент-аналізу електронних ЗМІ / Г.І. Леліков, В.М. Сороко, О.М. Григор'єв, Д.В. Ланде // Вісник державної служби України. — 2002. — № 2. — С. 72–78 с. 6. Федорчук А. Г. Контент-мониторинг информационных потоков [Электронный ресурс] / Б-ки Нац. акад. наук: пробл. функционирования, тенденции развития. — К., 2005. — Вып. 3. — Режим доступа: http://www.nbuv.gov.ua/articles/2005/05fagmip.html. — Загол. с экрана. 7. Григорьев А.Н. InfoStream. Мониторинг новостей из Интернет: технология, система, сервис: научно-методическое пособие / Григорьев А.Н., Ландэ Д.В., Бороденков С.А., Мазуркевич Р.В., Пацьора В.Н. – К., ООО "Старт-98", 2007. – 40 с.

УДК 004.89

Є.В. Буров
Національний університет "Львівська політехніка",
кафедра інформаційних систем та мереж

ІНТЕЛЕКТУАЛЬНА СИСТЕМА АВТОМАТИЗОВАНОГО ТЕСТУВАННЯ ПРОГРАМНОГО ПРОДУКТУ З ВИКОРИСТАННЯМ АЛГОРИТМІЧНИХ МОДЕЛЕЙ

Розглянуто використання алгоритмічних моделей знань для автоматизації тестування програмних продуктів. Запропоновано архітектуру системи тестування, ієрархію моделей. Продемонстровано взаємодію різних типів моделей для розв'язування задачі автоматизованого тестування

Ключові слова: автоматизоване тестування, модель, керовані моделями системи.

Paper describes the use of algorithmic knowledge models for overnight testing automation. The arhitecture of intellectual testing system is described. Algorithmic models formalization is proposed. Knowledge model interaction and execution is presented for a real world test automation system.

Key words: model-driven systems, automated testing, knowledge model.

Постановка проблеми у загальному вигляді

Однією з найважливіших проблем галузі програмного забезпечення є високий рівень складності програмних систем і пов'язані з нею проблеми складності та високої вартості адміністрування, розроблення та модифікації, значний рівень дефектів у таких системах. За

оцінками Національного інституту стандартів та технологій (NIST), щорічний збиток від дефектів програмного забезпечення для економіки США оцінюється у 59,6 млрд доларів [1]. Для виявлення дефектів традиційно застосовують тестування продукту з використанням визначеного набору сценаріїв використання [2]. Вартість тестування сьогодні становить значну частину в загальній вартості виробництва продукту. Водночас, складність програмного забезпечення унеможливлює його вичерпне тестування [3]. З метою підвищення ефективності тестування та зменшення витрат застосовують автоматизоване тестування [4].

Виконання завдання автоматизованого тестування передбачає виконання різноманітних операцій, пов'язаних з підготовкою середовища для тестування, отриманням та встановленням програмних продуктів, налаштуванням операційної системи та інструментальних програмних засобів тестування. Це завдання зазвичай виконує експерт-фахівець з автоматизованого тестування і воно є складним завданням, тому що вимагає ретельного врахування великої кількості взаємозалежних факторів. Помилка, яка виникає через неправильну підготовку середовища тестування, коштує дорого, адже тоді результати тестування доводиться анулювати і час (інколи декілька годин), витрачений на таке помилкове тестування, втрачається. Часові обмеження є суттєвою вимогою щодо самого автоматизованого тестування, оскільки розробникам та працівникам відділу контролю якості треба отримати результати тестування якнайшвидше. У багатьох фірмах, що розробляють програмні продукти, прийнята практика нічних компонувань продукту. Зміни, які розробники внесли у код, наприкінці робочого дня інтегруються у нову версію продукту. Цей продукт тестується мінімальним набором тестів з метою виявлення порушень базової функціональності, спричинених новим кодом. На початок нового дня розробники отримують список дефектів, які потрібно усунути. Для такого нічного тестування, як правило, використовують автоматизовані набори тестів. Вимоги до системи тестування – висока надійність, максимальний рівень автоматизації, за можливості – повністю автоматичне виконання. З іншого боку, система автоматизованого тестування працює в умовах постійних змін як у самому тестованому продукті, так і у середовищі тестування. Фахівцям з тестування доводиться постійно адаптувати тестувальний код, переналаштовувати систему тестування в умовах жорстких часових обмежень.

Аналіз останніх досліджень та публікацій

Організація та виконання автоматизованого тестування вимагає врахування великої кількості взаємозалежних деталей та вимог. Невиконання навіть однієї вимоги або ж поява хоч би одного збою під час тестування призводить до зупинки та відкидання тестових результатів. Це ставить підвищені вимоги до інженера з автоматизації. Постійна зміна тестованого продукту, його інтерфейсу, поява нових рис та можливостей вимагають постійної зміни та перетестування самого тестового коду, щоб досягти відповідності тестованому продукту. При цьому існують жорсткі часові межі для виконання цих завдань.

Для вирішення зазначених вище проблем щодо побудови самого тестового коду запропоновано такі архітектурні вирішення, як використання бібліотеки тестів, побудова тестів на основі ключових слів, табличний підхід або тести, що керуються даними (data driven) [5], тести, основані на моделях [6, 7]. Водночас, відомі архітектурні вирішення не дають змоги автоматизувати вирішення завдання налаштування та підготовки середовища тестування і проведення самого тестування, яке є складним та виконується вручну.

Зазначені особливості автоматизованого тестування програмних систем роблять перспективним впровадження інтелектуальних, орієнтованих на знання методів для побудови системи тестування. При цьому враховують сутності та залежності предметної області тестування, подані у відповідній онтології. Центральне місце у таких системах займають алгоритмічні моделі, тому що сам процес тестування відбувається як послідовність операцій, поданих певним алгоритмом.

Цілі статті

Метою цієї статті є дослідження способів подання та використання алгоритмічних моделей на прикладі інтелектуальної системи автоматизованого тестування програмних продуктів.

Архітектура інтелектуальної системи автоматизованого тестування

У роботі [8] наведено архітектуру та принципи функціювання інтелектуальної системи, яка використовує виконувальні концептуальні моделі. Підтримка алгоритмічних моделей додає до запропонованої системи моделювання нові компоненти (рис. 1). Центральним елементом системи моделювання виступає онтологія предметної області, яка забезпечує семантичну інтерпретацію усіх фактів інформаційної бази. Моделі також будують на основі цієї онтології.

Інформаційна база постійно поповнюється новими фактами, що відображають стан предметної області. Такими фактами є відомості про тестований продукт та стан і конфігурацію середовища моделювання. Істотним фактом є протокол поточного тестування, в якому відображаються стан та результати виконання усіх проміжних результатів тестування.

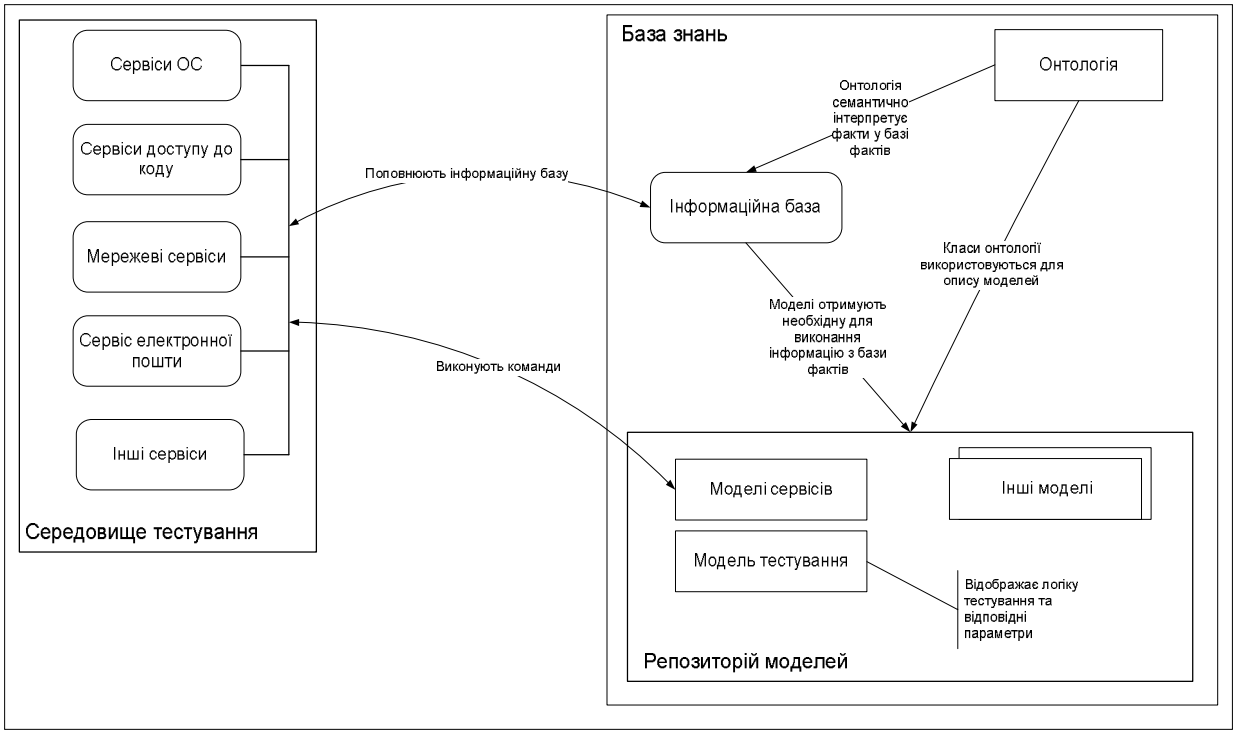

Рис. 1. Архітектура інтелектуальної системи автоматизованого тестування

Важливим елементом системи є зовнішні сервіси. Мета роботи системи досягається завдяки генеруванню системою тестування послідовності команд та їх виконання зовнішніми сервісами. Серед таких важливих зовнішніх сервісів назвемо сервіси команд операційної системи, сервіси завантаження файлів (ftp, http), сервіси отримання вихідного коду (VSS) тощо. Доступ до зовнішніх сервісів здійснюється за посередництвом моделей цих сервісів. Моделі сервісів інкапсулюють сутності, що описують параметри налаштування сервісу, команди, які виконує сервіс, стани сервісу, залежності між ними, обмеження та умови функціонування.

Формальна специфікація алгоритмічної моделі

Центральним компонентом інтелектуальної системи автоматизованого тестування є модель тестування, яка належить до класу алгоритмічних моделей. Завданням алгоритмічної моделі є подання послідовності операцій, так що виконання цієї моделі є виконанням цієї послідовності операцій. Така послідовність операцій описує алгоритм розв'язання певної задачі, що і пояснює назву моделі.

Формально алгоритмічна модель, як і будь-яка інша [8], складається зі схеми та реалізації:

$$AlgMd = (ScAlgMd, ReAlgMd).$$

Схема моделі відображає складові частини моделі та залежності між ними. Реалізація моделі забезпечує виконання моделі. Якщо змінюються умови, вимоги або середовище тестування, експерт

предметної області змінює схему моделі, залишаючи її реалізацію незмінною. Такий підхід забезпечує гнучкість та можливість швидкої адаптації моделі до змін.

Схема моделі містить секції метаданих, тіла моделі та деякі інші, допоміжні секції, які дають змогу ініціалізувати модель, верифікувати її, визначити необхідні передумови для її виконання тощо.

Тіло алгоритмічної моделі подамо невпорядкованою множиною моделей операцій та моделлю логіки виконання:

$$BodAlgMd = (M(OpMd), FlMd).$$

Модель логіки виконання FlMd залежно від поточного стану процесу тестування та тестового середовища активує моделі операцій, змінює значення в інформаційній базі або деактивує саму алгоритмічну модель. Ця модель є множиною ситуаційних моделей, кожна з яких відповідає певній ідентифікованій ситуації, та складається зі специфікацій сигнатури та дій:

$$FlMd = M(SitMd),$$

$$SitMd = (SgSit, AcSit).$$

Якщо сигнатура ситуації, задана набором умов, істинна, то виконуються специфіковані у моделі дії. Виконання кожної дії відповідає одному кроку алгоритму. Під час виконання дії активуються моделі операцій. Вважаємо, що кожна операція, після завершення, аналізує успішність свого виконання, оновлює статус тестування та заносить відповідні повідомлення у протокол тестування. Після закінчення операції знову виконується модель логіки виконання.

Реалізація моделі операції OpMd передбачає послідовне виконання таких завдань:

a) ініціалізації та перевірки передумов;
b) виконання операції (звертання до моделі або сервісу);
c) аналізу результатів та оновлення протоколу тестування

$$OpMd = (InOpMd, ExOpMd, AnOpMd).$$

Ініціалізація та перевірка передумов InOpMd специфікуються у секції ініціалізації моделі, а виконання операції ExOpMd та аналіз результатів AnOpMd – у тілі моделі операції.

Під час ініціалізації визначають усі факти, потрібні для виконання операції. Аналізують достатність фактів, необхідні передумови. Наприклад, під час операції скачування файла з віддаленого ftp-сервера перевіряють доступність цього сервера у мережі, наявність усіх даних, потрібних для встановлення сполучення. При цьому може використовують модель ftp-сервісу. Якщо передумови операції не виконуються, у протокол тестування заносять повідомлення про виявлену помилку і тестування припиняється або відкладається, залежно від виявленої помилки.

Виконання операції полягає у звертанні до зовнішнього сервісу за посередництвом моделі цього сервісу або ж у виконанні іншої моделі, яка опрацьовує дані в інформаційній базі, наприклад, здійснює класифікацію стану процесу тестування. В результаті виконання операції відбувається зміна у середовищі тестування.

На стадії аналізу результатів виконання операції аналізують зміни у середовищі тестування, наприклад наявність скачаних файлів, інстальованих програм, успіх (або неуспіх) у виконанні операції. Щоб отримати потрібні для аналізу дані, часто необхідно знову звернутися до зовнішніх сервісів із запитами на виконання операцій та відобразити їхні результати в протоколі тестування інформаційної бази.

Сутності та моделі інтелектуальної системи автоматизованого тестування

Розглянемо детальніше моделі та сутності онтології, які використовуються у системі автоматизованого тестування, їхні взаємозалежності та взаємодію.

Основні відомості про тестування зберігаються в екземплярі сутності *АвтоматизованеТестування* у вигляді атрибутів або посилань на інші сутності. До цих відомостей належить посилання на сутність (сервер, роль) тестового сервера, джерела інсталяторів. Тут також зберігається посилання на загальну алгоритмічну модель автоматизованого тестування.

Загальна алгоритмічна модель тестування подається як набір задач, які потрібно розв'язати у процесі автоматизованого тестування. Вона містить посилання на загальні моделі відповідних операцій, що відображають процес розв'язання кожної задачі. Такими операціями, наприклад, будуть: *ОтриматиІнсталятор*, *ІнсталюватиПродукт*, *ПровестиТестування*, *ДеінсталюватиПродукт*.

Кожна загальна модель, крім специфікації сутностей та операцій, містить визначення методів перевірки успішності виконання задачі. Наприклад, після закінчення задачі *ОтриматиІнсталятор*, модель передбачає перевірку наявності файла інсталятора у вказаному каталогу та, якщо він є, визначає результат виконання моделі як успішний. В протилежному випадку результат виконання моделі визначається як неуспішний з додатковою специфікацією причин помилки.

Крім загальних моделей операцій, загальна алгоритмічна модель містить посилання на модель визначення наступної операції та моделі опрацювання помилок. Модель визначення наступної операції залежно від результату виконання попередньої операції визначає наступну або одну із моделей опрацювання помилок. До того ж може використовуватися прийнята у системі класифікація помилок.

Розглянемо детальніше моделі операції *ОтриматиІнсталятор*, мета якої – отримати інсталятор у вказаному каталогу. Виконання моделі полягає у перевірці наявності та копіюванні файла-інсталятора з віддаленого сервера на локальний тестовий сервер із застосуванням певного сервісу копіювання файлів. Загальна модель такої операції (рис 2) містить такі сутності, як *ВіддаленеРозміщення*, *ЛокальнеРозміщення*, *СервісКопіювання*.

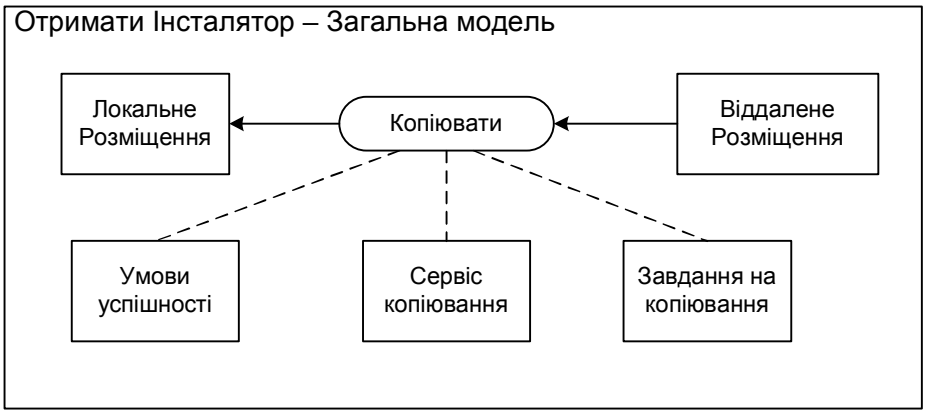

Рис. 2. Загальна модель задачі ОтриматиІнсталятор

Використання такої загальної моделі, яка подає поставлену задачу в найзагальнішому вигляді, надає системі необхідної гнучкості. З використанням такої загальної моделі поставлену задачу можна розв'язувати різними способами, просто визначаючи іншу конкретизацію загальної моделі. Наприклад, такій загальній моделі відповідають операції скачування інсталятора з використанням ftp, http протоколів, з VSS або з іншого комп'ютера локальної мережі. Для зміни способу розв'язання поставленої задачі достатньо просто замінити детальну модель.

Розглянемо як конкретизацію загальної моделі *ОтриматиІнсталятор* модель отримання файла з використанням протоколу ftp – модель *ОтриматиІнсталяторЧерезFtp* (рис 3). Сутності *ВіддаленеРозміщення* у цій моделі відповідає сутність *СерверFtp*, сутності *ЛокальнеРозміщення* – *ТестовийСервер*. Атрибутами сутності *СерверFtp* є мережева адреса (URL) сервера, параметри аутентифікації. Аналогічно, для тестового сервера встановлено його URL. Для обох сутностей визначено робочі каталоги. Сутності *СервісКопіювання* відповідає сутність *FtpСервіс*. З цією сутністю асоційовано обмеження, які відображають особливості використання цього сервісу для копіювання файлів. У моделі *ОтриматиІнсталяторЧерезFtp* визначають додаткові передумови виконання операції (наприклад, мережева доступність сервера ftp та тестового сервера) та додаткові процедури перевірки успішності виконання операції. Так, крім перевірки наявності файла на тестовому сервері, додатково перевіряють хеш-суму отриманого файла.

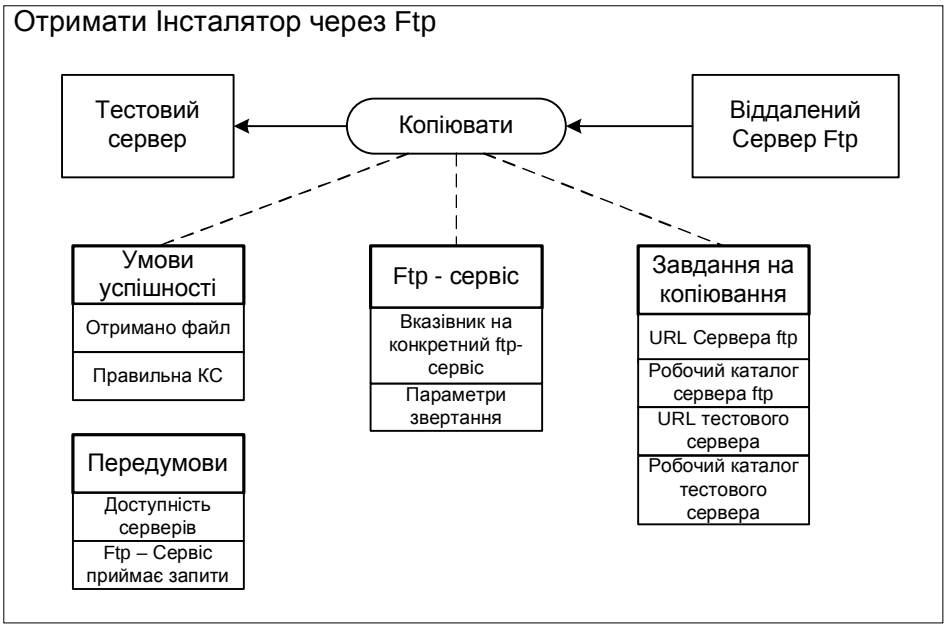

Рис. 3. Схема моделі Отримати ІнсталяторЧерезFtp

Сутність *FtpСервіс* містить посилання на модель сервісу ftp – *FtpМодель*. Ця модель відображає поточний стан конкретного ftp-сервісу, команди, які він приймає, та містить модель функціонування цього сервісу, яка використовується для організації взаємодії та прийняття рішень. Модель функціонування ftp-сервісу задамо скінченним автоматом з визначенням станів та переходів між ними. На основі знань, відображених у цій моделі, користувач сервісу може прийняти рішення очікувати звільнення сервісу, якщо цей сервіс у цей момент здійснює завантаження, або вирішити встановити сполучення, якщо воно не встановлене, або ж, навпаки, використати вже встановлене сполучення.

Подання алгоритмічної моделі мовою XML

Алгоритмічні моделі, моделі операцій та сервісів створюються за допомогою інструментального засобу "Редактор моделей" та зберігаються у форматі XML. Розглянемо приклади подання алгоритмічної моделі автоматизованого тестування та моделі операції отримання інсталятора через ftp-сервіс. Для простоти та наочності пропустимо не важливі для ілюстрації принципів роботи моделі секції.

```
<Model>
    <ModelMetadata>
        <GeneralInfo>
            <ModelId> id </ModelId>
            <ModelType> AlgorithmicModel </ModelType>
            <ModelName>OvernightAutomatedTestingModel</ModelName>
            <OntologyURI> www.acme.org/AutomatedTestingOntology</OntologyURI>

<ModelRepositoryURI>www.acme.org/ModelRepository</ModelRepositoryURI>
        </GeneralInfo>
        <ActivationInfo>
            <Condition>
                <ConditionBd> Was not active during</ConditionBd>
                <ConditionParameter>15 min<ConditionParameter>
            </Condition>
            <StartState>InstallerChecking</StartState>
        </ActivationInfo>
```

```
        </ModelMetadata>
        <ModelBody>
                <Operations>
                        <Operation>
                                <OperationName>CheckInstallerAvailabilty<OperationName>
                                <OperationModel>ModelId1<OperationModel>
                        </Operation>
                        <Operation>
                                <OperationName>GetInstaller<OperationName>
                                <OperationModel>ModelI2d<OperationModel>
                        </Operation>
                        <Operation>
                                <OperationName>Install<OperationName>
                                <OperationModel>ModelId3<OperationModel>
                        </Operation>
                        <Operation>
                                <OperationName>Test<OperationName>
                                <OperationModel>ModelId4<OperationModel>
                        </Operation>
                        <Operation>
                                <OperationName>UnInstall<OperationName>
                                <OperationModel>ModelId5<OperationModel>
                        </Operation>
                        <Operation>
                                <OperationName>FinishAndClean<OperationName>
                                <OperationModel>ModelId5<OperationModel>
                        </Operation>
                        <Operation>
                                <OperationName>InformByEmail<OperationName>
                                <OperationModel>ModelId6<OperationModel>
                        </Operation>
                </Operations>
                <ProcessFlow>
                        <Signature>
                                <Condition>
                                        <IB_Entity Type="Test_Status">ReadyForTesting<IB_Entity>
                                </Condition>
                                <Execute>
                                        <SetIB_InstanceValue
Type="Test_Status">CheckingInstallerAvailability<SetIB_InstanceValue>
                                        <ExecuteModel>CheckInstallerAvailabilty<ExecuteModel>
                                </Execute>
                        </Signature>
                .....
                </ProcessFlow>
        </ModelBody>
</Model>
```

Модель складається із секцій метаданих та тіла моделі. Секція метаданих містить інформацію про онтологію, ідентифікатор та назву моделі, адресу репозиторію моделей. Інформацію про активацію моделі використовує сервіс моделювання — *Менеджер запуску моделей* і визначає, що

модель автоматично активується через 15 хвилин після закінчення попередньої активації. У тілі моделі вказано операції, які виконуються в алгоритмічній моделі, та визначено модель логіки виконання. На початку перевіряють наявність інсталятора для завантаження. Якщо інсталятор не готовий, то модель деактивується. В іншому разі запускається процес тестування, зокрема наступна операція – отримання інсталятора. Перед деактивацією моделі виконується операція очищення середовища тестування.

Модель одержання інсталятора через ftp-сервіс виглядає так:

```
<Model>
        <ModelMetadata>
                <GeneralInfo>
                        <ModelId> id </ModelId>
                        <ModelType> OperationModel </ModelType>
                        <ModelName>GetInstallerFromFtp</ModelName>
                        <OntologyURI> www.acme.org/AutomatedTestingOntology</OntologyURI>

        <ModelRepositoryURI>www.acme.org/ModelRepository</ModelRepositoryURI>
                        <GenericModel>GetInstaller</GenericModel>
                </GeneralInfo>
                <ActivationInfo>
                        <Condition>
                                <ConditionBd> Server accessible</ConditionBd>
                                <ConditionParameter Name= "ServerAddress">Server Ip
address<ConditionParameter>
                                <ConditionParameter Reference=
"ServerAccessibilityCheckModel">Model Id<ConditionParameter>
                        </Condition>
                        <Condition>
                                <ConditionBd> Service available</ConditionBd>
                                <ConditionParameter Name= "Service
name">FtpService<ConditionParameter>
                                <ConditionParameter Reference= "Service model">FtpServiceModel
Id<ConditionParameter>
                        </Condition>
                </ActivationInfo>
        </ModelMetadata>
        <ModelBody>
                <Entities>
                        <Entity>
                                <EntityName>RemoteFtpServer</EntityName>
                                <EntityType>FtpServer</EntityType>
                                <GenericEntityRef>RemoteLocation</GenericEntityRef>
                                <DefineParametersFromEntity Name="InstallSource">
                                        ……
                                </DefineParametersFromEntity>
                        </Entity>
                        <Entity>
                                <EntityName>LocalTestServer</EntityName>
                                <EntityType>TestServer</EntityType>
                                <GenericEntityRef>LocalLocation</GenericEntityRef>

                        </Entity>
```

```
            </Entities>
            <Relations>
                    <RelationName>GetFromFtp</RelationName>
                    <RelationType>GenericRelation</RelationType>
                    <GenericRelationRef>CopyFile</GenericRelationRef>
                    <OperationRef>CopyFtpOperation</OperationRef>
            </Relations>
            <Operations>
              <Operation Name ="CopyFtpOperation">
                <ServiceRef>FtpServiceModelId</ServiceRef>
                <Command>GetFile</Command>
              </Operation>
            <Operations>
            <SuccessConditions>
              <Condition>
                <ConditionType>FileExists</Condition>
                <DefineParametersFromEntity Name="LocalTestServer">
                    <IpAddr>ipaddr<IpAddr>
                    <AccessCredentials>Credentials</AccessCredentials>
                    <FilePath>FilePath<FilePath>
                </DefineParametersFromEntity>
              </Condition>
            </SuccessConditions>
            <IB_Update>…</IB_Update>
        </ModelBody>
</Model>
```

У метаданих моделі вказано передумови її активації — перевірку мережевої доступності сервера та готовність сервісу ftp. У тілі моделі окремі секції визначають сутності, відношення, операції, умови успіху виконання та визначення наступної операції. Деякі атрибути сутностей містять посилання на атрибути інших сутностей. Кожна сутність, відношення містить посилання на відповідні елементи загальної моделі. Виконання моделі операції полягає у виконанні операції, специфікованої у секції операцій. Після здійснення операції перевіряють умови успіху, визначені у відповідній секції моделі. Операція вважається завершеною успішно, якщо усі умови успіху виконані. В останній секції тіла моделі оновлюють протокол тестування в інформаційній базі.

Створення, виконання та взаємодія алгоритмічних моделей

Алгоритмічні моделі створюють у програмному засобі “Редактор моделей”. Готові моделі в XML-форматі зберігаються у репозиторії моделей системи моделювання. У системі автоматизованого тестування за послідовністю запуску моделей стежить *Менеджер запуску*, який періодично перевіряє умови запуску і, якщо умови виконуються, активує модель.

Активовану алгоритмічну модель інтерпретує компонент системи моделювання — Інтерпретатор моделей. Реалізація такого інтерпретатора є достатньо простим завданням, він виконує синтаксичний розбір XML-файла моделі, перевіряє істинність сигнатур ситуацій та ініціалізує активацію відповідних моделей операцій.

Інтерпретатор моделі операції перевіряє передумови виконання операції і, якщо вони виконуються, ініціює виконання операції. Після її закінчення модель логіки виконання перевіряє статус процесу тестування і, залежно від нього, визначає та активує наступну операцію.

З опису роботи системи тестування видно, що загальна мета функціонування системи досягається у процесі взаємодії багатьох моделей. Так, алгоритмічна модель використовує ситуаційну модель для визначення логіки виконання. Ситуаційна модель, своєю чергою, активує моделі операцій. Кожна модель у процесі виконання відображає результати реалізації в

інформаційній базі, так, що стан інформаційної бази відображає статус процесу тестування. Використання ситуаційної моделі для специфікації логіки тестування дає змогу використати для перевірки умов загальний стан предметної області, а не тільки об'єкти, відображені у моделі. Порівняно з альтернативними, наприклад, скінченно-автоматним поданням логіки виконання, цей метод забезпечує гнучкість і простоту розширення списку ситуацій.

Програмна реалізація інтелектуальної системи автоматизованого тестування

З використанням описаного підходу створено, апробовано та автоматизовану систему тестування програмних продуктів, яка успішно функціонує. Для реалізації системи вибрана скриптова мова VBScript. Програмним засобом автоматизованого тестування слугує HP QuickTest Professional, який теж використовує VBScript як мову програмування.

Система автоматизованого тестування періодично перевіряє наявність нового файла-інсталятора на ftp-сайті розробника продукту. Якщо він наявний, файл інсталятора скачується, розархівовується у визначений тестовий каталог. Після цього запускається процес інсталяції в автоматичному режимі, який не вимагає, щоб інформацію вводив користувач. Після його завершення система перевіряє успішність інсталяції, конфігурує та перезапускає деякі системні сервіси.

Якщо продукт успішно інстальовано, запускають засіб автоматизованого тестування, що проводить вибрані тести. Результати тестування записують у протокол, який після завершення тестування пересилається електронною поштою представникам розробника. Працівників відділу контролю якості також інформують електронною поштою у разі виникнення збою процесу тестування. Після завершення тестування тестований програмний продукт деінсталюється, тестовий каталог очищується і середовище готується до нового тестування.

Розроблена система дала змогу систематично перетестовувати декілька значних за розміром (розмір файла інсталятора 500–700 Мбайт) програмних продуктів за одну ніч. Експлуатація системи тестування довела її високу надійність, гнучкість, простоту модифікації та розвитку.

Висновок

Використання алгоритмічних моделей для автоматизації тестування програмних продуктів дасть змогу створювати інтелектуальні системи тестування, які здатні швидко адаптуватися до змін у функціональності тестованого продукту, а також врахувати зміни в апаратно-програмному тестовому середовищі та адекватно реагувати на них.

1. Software Bugs Cost U.S. Economy $59.6 Billion Annually, RTI Study Finds [Electronic resource]. – Режим доступу: http://www.nist.gov/director/prog-ofc/report02-3.pdf. 2. Myers G. The art of software testing/Glenford Myers. – John Wiley&Sonc Inc, 2004. – 255 p. 3. Lewis W. Software testing and continuous quality improvement / William L. Lewis.-Auerbach publications. – NY.,2005. – 561 p. 4. Dustin E. Automated software testing: introduction, management, and performance/ Dustin E, Rashka J., Paul J. – Addison Wesley, 1999. – 575 p. 5. Kelly M. Choosing test automation framework/Kelly Michael.- Режим доступу: http://www.ibm.com/developerworks/rational/library/591.html 6. Baker P. Model-Driven Testing/ Paul Baker, Zhen Ru, Dai Jens, Øystein Haugen, Ina Schieferdecker. – Springer-Verlag Berlin Heidelberg, 2008. – 184 p. 7. Heckel R. Towards Model-Driven Testing. /Heckel R, Lohmann M/Electronic Notes in Theoretical Computer Science. – 2003, #6,v.92. – P. 33–43 8. Буров Є.В. Опрацювання знань у когнітивній інформаційній системі, керованій моделями / Буров Є.В. / Східно-Європейський журнал передових технологій. – № 6/7. – Харків: Технологічний центр, 2009. – С.40–49