ПРОГРАММНАЯ ИНЖЕНЕРИЯ
SOFTWARE ENGINEERING

УДК 004.4 DOI: 10.17587/it.25.11-19

А. М. Бершадский, д-р техн. наук, проф., e-mail: bam@pnzgu.ru,
А. С. Бождай, д-р техн. наук, проф., e-mail: bozhday@yandex.ru,
Ю. И. Евсеева, канд. техн. наук, доц., e-mail: shymoda@mail.ru,
А. А. Гудков, канд. техн. наук, доц., e-mail: alexei-ag@yandex.ru,
Пензенский государственный университет

Концепция рефлексивной самоадаптации прикладных программных систем[1]

Предложен новый подход к решению важной проблемы современной IT-индустрии — самоадаптации программных компонентов на основе концепции поведенческой рефлексии. Рассматриваются базовые принципы рефлексивной адаптации, а также идея использования инженерии линеек программных продуктов для создания адаптивных программ. Приводится обобщенное архитектурное решение для построения рефлексивных самоадаптивных систем, способных менять свои поведенческие характеристики непосредственно в ходе исполнения без перекомпиляции исходного кода. Отдельное внимание уделено вопросам формализованного представления и использования моделей изменчивости при реализации вариантов адаптивного поведения прикладных программ.

Ключевые слова: *адаптивные программные системы, рефлексивная самоадаптация, модель изменчивости, диаграмма характеристик, линейки программных продуктов, программная инженерия*

Введение

Обеспечение информационных потребностей современного общества зависит от использования огромного арсенала программно-аппаратных комплексов и систем. Интенсивность и динамика информационных процессов предъявляют жесткие требования к поведенческим свойствам программного обеспечения (ПО). Суть этих требований сводится к тому, что прикладная программа должна функционировать адекватно специфике и содержанию информационных потоков своей предметной области, т. е. своевременно адаптироваться к непрерывным изменениям внешней среды. Существует целый ряд прикладных задач, для которых остановка обслуживающего ПО (для внесения обновлений в математические и информационные модели, алгоритмы, исходный код, интерфейсные оболочки и т. п.) сопряжена с огромными трудностями или вовсе невозможна. И единственным выходом в этом случае является самоадаптация программы в режиме реального времени, без остановок на перепроектирование и перекомпиляцию. Такие программные компоненты должны уметь самостоятельно отслеживать критические изменения предметной области, адекватно приспосабливаясь к ним в фоновом режиме, не прекращая при этом выполнять свои основные полезные функции.

Таким образом, самоадаптация прикладного ПО — это одна из принципиально важных задач в сфере современной программной инженерии, имеющая как теоретическую важность, так и огромное практические значение. Особую актуальность она имеет и для российского IT-сектора, поскольку в ближайшие годы следует ожидать повышенного интереса со стороны правительственных структур, бизнеса, науки к вопросам интеграции социальных и программно-кибернетических процессов. Это связано с приоритетной программой перехода страны к моделям цифровой экономики. По прогнозам экспертов, наибольшего развития к 2025 г. должны достичь сферы киберфизических и информационных систем, работающих с большими данными (BigData).

На фоне новых вызовов не следует забывать и о традиционных проблемах IT-индустрии. Прежде всего, это возрастающая структурная

[1] Исследование выполнено при финансовой поддержке РФФИ в рамках научного проекта № 18-07-00408.

сложность и даже громоздкость современных программных систем прикладного назначения и, как следствие, увеличение времени и стоимости разработки, трудности сопровождения, сокращение длительности жизненного цикла, снижение отказоустойчивости, надежности, гибкости. Неспособность к быстрому самовосстановлению ПО в аварийных ситуациях может привести к потерям больших объемов данных, а в условиях динамично изменяющегося современного мира многие информационные системы (например, системы, обеспечивающие работу крупных предприятий, ERP-системы), лишенные способности саморазвития и оперативной адаптации, быстро устаревают.

Создание фундаментальных научных основ, практических приемов и универсальных механизмов программной самоадаптации — перспективный путь для решения большинства перечисленных трудностей и задач. Многие проблемы, связанные с вопросами адаптивного поведения искусственных систем (прежде всего технических), успешно решались в рамках теории автоматического управления и регулирования. Базовые понятия и принципы этой научной дисциплины можно и в настоящее время считать актуальными для большинства объектов и явлений окружающего мира, обладающих системными свойствами. Программные системы не являются исключением. Однако практический опыт последних лет показал, что механизмы обратной связи, адаптации и системной рефлексии в программном обеспечении до сих пор плохо формализованы, а методы реализации адаптивного поведения в технических системах, предлагаемые классической теорией управления, не актуальны для программных систем в силу их информационной природы. Большинство существующих работ, посвященных вопросам адаптивного поведения в программных системах, сводятся к созданию различных архитектурных и программно-технических решений, преимущественно частного или специфического характера [1—6]. Однако эта проблема имеет гораздо более общий характер и требует разработки фундаментальных принципов, которые будут положены в основу универсальных механизмов самоадаптации для широкого класса прикладного ПО.

Проблематика системного поведения на основе обратных связей, в том числе и вопросы адаптации, является традиционным предметом изучения кибернетики. За более чем полувековую историю развития этой науки накоплен богатый опыт концептуальных, математических, информационных и технических решений подобных задач. Поэтому не случайно, что в начале 2000-х годов появились работы, использующие традиционные кибернетические подходы для описания программного поведения [7]. Термин "программная кибернетика" (software cybernetics) был введен известным специалистом в сфере программной инженерии K. Cai в работе [8] и первоначально обозначал попытку применить методы классической кибернетики и теории управления к программным системам для оптимизации длительности жизненного цикла и сокращения затрат на разработку сложных программных систем.

За 15 лет существования программная кибернетика претерпела существенные изменения и в настоящее время включает в себя не только попытки реализации базовых кибернетических принципов применительно к структуре ПО, но и достаточно оригинальные концепции, связанные с вопросами распределенных и облачных вычислений, создания и эксплуатации киберфизических систем (в частности, сетевых систем и интернета вещей), процессов разработки, тестирования и сопровождения программ [9—11]. Но несмотря на столь высокий интерес к этому направлению, тема создания универсальных (иными словами, применимых для широкого класса прикладных программных систем) механизмов адаптации и самоадапатции практически не изучена. Существующие подходы, изложенные, в частности, в работах [12, 13], не позволяют решить весь класс проблем, связанных с адаптацией. Системы, построенные с их использованием, не являются достаточно гибкими и надежными [14].

Особенно перспективными нам видятся идеи самоадаптации программных систем, выстроенные в близкой аналогии с принципами адаптации высокоорганизованных биологических существ. Речь идет о рефлексивных типах адаптивного поведения, которые связаны с сознательной деятельностью живого организма: самоанализом, способностью ориентироваться в сложных ситуациях и планировать собственные действия, обучением посредством наблюдения за собственными состояниями и объектами внешнего мира. Подобный выбор связан с тем, что современные программные системы прикладного назначения, обрабатывающие большие объемы сложноструктурированной и неструктурированной информации, имеющие крайне сложную архитектуру, должны иметь соответствующую поведенческую сложность и разнообразие без нарушения заданной функциональной компактности.

1. Концепция рефлексии

Рефлексия — термин, появившийся изначально в философии и впоследствии ставший популярным в других областях знаний, в частности в психологии. Впервые феномен рефлексии был основательно исследован в работах английского философа Дж. Локка, который считал, что существуют два источника человеческих знаний: объекты внешнего мира и деятельность собственного ума. К деятельности ума Локк причислял мышление, рассуждения, сомнение и считал, что все это познается с помощью специального внутреннего чувства — рефлексии, "наблюдения, которому ум подвергает свою деятельность" [15].

В настоящее время рефлексию принято определять как мыслительный процесс, направленный на самоосознание, самопознание, анализ своих эмоций и чувств, состояний, способностей, целей, стратегий поведения. Данный процесс свойственен исключительно человеческому разуму и является неотъемлемым компонентом социально-психологической и психофизиологической адаптации личности. Без рефлексии невозможно полноценное развитие индивидуума [16].

Современные программные системы и соответствующие им предметные области таковы, что невозможно заранее, на этапе проектирования, учесть все возможные обстоятельства функционирования. Многие вещи становятся очевидными лишь в процессе эксплуатации, а некоторые так и остаются скрытыми. Это вынуждает разработчиков проявлять повышенное внимание к процессу сопровождения системы, регулярно выпускать обновления, не содержащие кардинально новой функциональности.

Решением проблемы недостаточности знаний может стать наделение систем способностью к самостоятельному анализу собственного состояния. В этом случае разработчику достаточно заложить в программу лишь базовые механизмы анализа и поиска закономерностей (тем самым предполагая, что какие-то моменты на этапе разработки остались неучтенными), а выявлением недостающих знаний система будет заниматься самостоятельно на протяжении своего жизненного цикла. Такой подход позволит не только снизить затраты на сопровождение системы, но и выявить те сведения о ее предметной области, которые могут быть неизвестны даже экспертам.

В этом отношении процесс самоанализа системы будет схож с процессом рефлексии человеческого разума. Объектом данного процесса будет собственная поведенческая продукция системы, а результатом — новые обратные связи, которые позволят системе в дальнейшем более эффективно функционировать. Как и в случае с человеческой рефлексией, программная рефлексия будет одним из основных компонентов высокоуровневой самоадаптации.

Перечислим базовые принципы концепции рефлексивной адаптации и требования к ее внедрению в сфере программной инженерии:

- инвариантность механизма самоадаптации к предметной области, типу и назначению программной системы;
- масштабируемость. В процедуре рефлексии должен учитываться тот факт, что уровень организации программной системы может быть различным: от простейшей утилиты до программного комплекса на основе нескольких систем. Также необходимо учесть, что адаптироваться может как система в целом, так и отдельные ее компоненты, при этом должны использоваться одни и те же методы адаптации;
- интеллектуальный анализ ретроспективных данных. Новые поведенческие обратные связи формируются путем выявления зависимостей и закономерностей в информации, отражающей работу программной системы на некотором временном промежутке в прошлом.

Рефлексия программных систем, безусловно, не является полноценным аналогом человеческой рефлексии, поскольку программы не обладают сознанием. Однако даже в упрощенной форме дополнительные рефлексивные контуры в структуре ПО позволяют решить многие задачи, связанные с недостаточностью знаний о предметной области системы и неполнотой процессов отладки и тестирования.

2. Архитектура адаптивной программной системы на основе рефлексивных компонентов

Рассмотренный выше принцип масштабируемости накладывает определенные требования на архитектурную организацию программной системы. Адаптируемая архитектура должна легко модифицироваться, причем речь идет не только о перестройке программы по результатам рефлексивного анализа, но и о модификации программы извне (например, самими разработчиками) и последующем учете изменений в процессе эксплуатации. Так, новый модуль, подключенный к системе, должен быть сразу же включен в общий процесс рефлексии, в резуль-

тате которого будет выявлена его связь с другими модулями, их взаимное влияние друг на друга. Помимо этого, определенные закономерности должны быть выявлены в рамках функционирования самого модуля.

Однако динамика современного мира такова, что модификацию программной системы коллективом (или коллективами) разработчиков часто не удается провести своевременно. Это может повлечь за собой возникновение аварийных или нестабильных ситуаций, сопровождаемых серьезными финансовыми потерями, техногенными и социально-экономическими катастрофами. Современный бизнес требует оперативных реакций как от людей и организаций, так и от программных продуктов, обеспечивающих их работу. Без сомнения, самоадаптивные системные компоненты окажутся незаменимым инструментом при решении этой непростой задачи.

Обобщая существующий опыт разработки систем с модульной архитектурой, можно сформулировать несколько важных принципов построения программных систем с рефлексивной самоадаптацией функционала:

- минимальными компонентами программной системы, подверженными адаптации, являются блоки изменчивости. Каждый блок изменчивости, входящий в состав системы, отвечает за возможную реализацию какого-либо атомарного компонента программы (функции, класса, модели объекта предметной области и т. п.);
- совокупность логически связанных блоков изменчивости, относящихся к одной предметной области и предназначенных для решения одной задачи, называется рефлексивным компонентом;
- рефлексивная программная система может состоять из одного или нескольких рефлексивных компонентов. Их число будет определять адаптивное разнообразие и рефлексивную сложность программы;
- рефлексивные компоненты объединены в единую систему посредством общего интерфейса и изначально могут не иметь данных друг о друге. Однако в процессе рефлексивного анализа будет происходить выявление степени и характера влияния одних компонентов на другие. Для этого необходимо предусмотреть каналы информационного обмена между компонентами.

Важно понимать, что рефлексивный компонент — это не обязательно программный модуль в классическом понимании. В ряде случаев выгодно иметь компоненты, реализованные в форме сервисов в рамках концепции сервис-ориентированной архитектуры (service-oriented architecture [17]). К преимуществам сервис-ориентированного рефлексивного компонента можно отнести:

- низкую связность. Сервисы, входящие в состав системы, могут быть реализованы независимо от других ее служб, а также могут входить в состав других систем. Единственное требование — знание соответствующих протоколов взаимодействия;
- принципиальную допустимость использования функций одного сервиса сторонними приложениями и системами. Если возникнет необходимость в модификации определенной функциональности, то достаточно будет изменить ее только в одном сервисе, а не в каждой использующей его системе;
- открытые стандарты взаимодействия, значительно уменьшающие время подключения нового сервиса к существующей системе.

Подводя итог, можно выделить три уровня рефлексии адаптивной программной системы на основе предложенной архитектуры:

- *уровень внутрикомпонентный.* Анализ поведенческой продукции отдельного компонента. Поиск зависимостей, формирование обратных связей;
- *уровень межкомпонентный.* Анализ поведенческой продукции нескольких компонентов, поиск зависимостей, оценка влияния компонентов друг на друга. Формирование обратных связей в нескольких компонентах;
- *уровень межсистемный.* Поиск зависимостей и оценка влияния компонентов одной системы на компоненты другой. Формирование межсистемных обратных связей.

3. Использование концепции инженерии линеек программных продуктов для создания адаптивных программных систем

Инженерия линеек программных продуктов (Software Product Lines Engineering, SPLE) — это концепция повторного использования компонентов программного обеспечения, помогающая разрабатывать семейства (линейки) продуктов с сокращением времени выхода на рынок и повышением качества [18].

Центральным понятием концепции SPLE выступает понятие модели изменчивости (variability model). Модель изменчивости — это некоторое формализованное описание множества возможных конфигураций программной системы [19]. Наиболее близким по смыслу

понятием в технических науках является понятие морфологического множества [20]. По сути модель изменчивости и представляет собой морфологическое множество программной системы, дополненное некоторыми ограничениями и правилами. Данные ограничения и правила касаются вопросов совместимости отдельных компонентов потенциальной системы друг с другом, однако могут иметь и иной смысл (в зависимости от типа используемой модели изменчивости).

В SPLE используются следующие типы моделей изменчивости [21]:

- *модели характеристик* — модели, которые наиболее часто используются на практике. Графически отображаются в форме модифицированного И/ИЛИ-дерева, называемого диаграммой характеристик (рис. 1), могут иметь различное формализованное представление (гиперграф, пропозициональная формула, алгебраическая нотация и др.);
- *ортогональные модели изменчивости.* Схожи с моделями характеристик, графически также отображаются в форме диаграмм. Основное отличие заключается в том, что ортогональные модели показывают только наличие изменчивости в рассматриваемой программной системе, в то время как модели характеристик предоставляют более конкретное описание как предметной области, так и точек изменчивости;
- *модели решений.* Модель решений включает в себя следующие компоненты [22]: вопросы из предметной области, на которые нужно получить ответы в процессе разработки программного продукта; множества возможных ответов на вопросы; ссылки на используемые компоненты (активы) или другие решения; описания последствий принятия решения (ответа на определенный вопрос или выбор определенного актива).

Обобщенная архитектура адаптивной программной системы, основанной на моделях изменчивости и уровнях рефлексии, приведенных в предыдущем разделе, представлена на рис. 2. Самоадаптивная программная система охватывается тремя уровнями рефлексивной обратной связи: уровнем отдельных компонентов, уровнем групп компонентов и общесистемным уровнем. Каждый контур рефлексии обеспечивается своей моделью изменчивости (МИ), отвечающей за выбор текущих конфигураций компонентного и системного уровней. Общая координация разноуровневой совокупности МИ и потоков поведенческих данных осуществляется модулем анализа поведенческой информации. Этот модуль не всегда является подсистемой системного уровня (как изображено на рисунке). Возможны и более гибкие архитектурные вариации, когда каждый отдельный компонент имеет свою собственную подсистему поведенческого анализа.

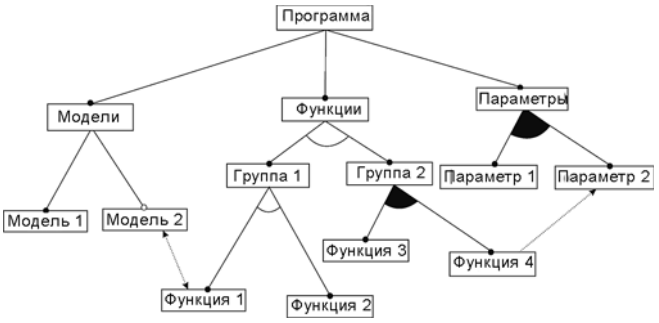


Рис. 1. Пример диаграммы характеристик

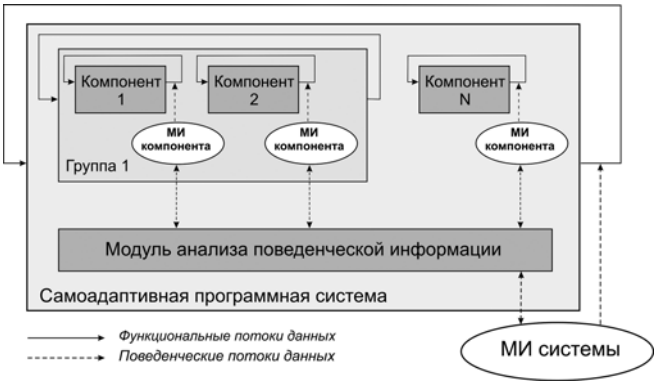


Рис. 2. Обобщенная архитектура самоадаптивной программной системы, основанной на моделях изменчивости

4. Пути формализации адаптивного поведения рефлексивной программной системы

Как уже было сказано выше, в рамках концепции SPLE модели изменчивости представляют собой аналог моделей морфологических множеств. Процесс создания оптимальной конфигурации программного продукта по модели изменчивости схож с процессом структурно-параметрического синтеза технических систем по морфологическому множеству, но гораздо менее формализован.

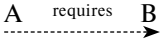
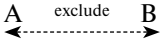
Концепция DSPLE [23] развивает идею SPLE: модели изменчивости динамических линеек программных продуктов (DSPL) используются для формирования оптимальной конфигурации программной системы в процессе ее выполнения. DSPL-системы можно считать адаптивными.

С нашей точки зрения, одним из перспективных подходов к реализации механизма программной рефлексии является интеграция

принципов DSPLE и технологии Data Mining [24]. При этом первым шагом к созданию универсальной технологии синтеза адаптивных систем должна стать разработка математической модели изменчивости, задающей формальные правила структурных и функциональных изменений для широкого класса ПО. Для этих целей предлагается использовать хорошо известную модель характеристик (feature model) — она обладает хорошей визуальной наглядностью диаграммы (по этой причине в научном обиходе часто используются другие термины — диаграмма характеристик, модель характеристик, feature diagram) и легко формализуема [25].

В таблице представлены основные типы отношений в моделях характеристик. Каждая из характеристик служит абстракцией определенного программного компонента (модуля, класса, 3D-модели, мультимедийного ресурса и т. д.). Отношения между характеристиками определяют потенциальный набор конфигураций системы в целом. Кроме того, с помощью отношений будет задаваться степень совместимости различных компонентов в рамках одного программного продукта.

Отношения в модели характеристик

Обозначение	Наименование	Описание
	Отношение обязательной дочерней характеристики	Если родительская характеристика включена в конфигурацию диаграммы, то дочерняя характеристика тоже должна быть включена
	Отношение опциональной дочерней характеристики	Включение дочерней характеристики в конфигурацию не является обязательным
	ИЛИ	Если родительская характеристика включена в конфигурацию, то также должно быть включено определенное число дочерних
	Исключающее ИЛИ	Только одна из представленных дочерних характеристик должна присутствовать в конечной конфигурации
	Отношение включения	Включение в конфигурацию характеристики A требует включения характеристики B
	Отношение исключения	Включение в конфигурацию характеристики A требует исключения характеристики B

Для формального описания зададим модель характеристик в виде ориентированного прямого гиперграфа (или F-графа):

$$F = (N, E, \Delta, \Psi), \quad (1)$$

где $N = \{N_1, N_2, ..., N_n\}$ — конечное множество вершин графа, каждая из которых представляет собой характеристику диаграммы; $E = \{E_1, E_2, ..., E_m\}$ — конечное множество гиперребер (отношений модели характеристик); $\Delta \in N$ — вершина, представляющая корневую характеристику диаграммы; Ψ: $E \to M$, $M \subset N \times N$ — функция маркировки, которая присваивает значение мощности $mv(E_i) = (\min, \max) \in M$, где M — конечное множество значений мощности модели характеристик; N — конечное множество вершин гиперграфа, каждому ориентированному гиперребру E_i, так что $\min, \max \in Z \wedge \min \geqslant 0$, $\max > 0$, $\min \leqslant \max \wedge \max \leqslant q_i$, где Z — множество целых чисел.

Значение мощности — это пара из минимального и максимального числа вершин головного множества гиперребра, которые могут быть включены в конфигурацию диаграммы характеристик, которой соответствует гиперграф.

С помощью отношений опциональной дочерней характеристики, исключающего ИЛИ и множественного ИЛИ реализуются блоки изменчивости — структурные единицы программной системы, затрагиваемые адаптивными процессами. Блок изменчивости может использоваться для определения множества состояний отдельного компонента программной системы, а также правила выбора того или иного состояния в каждом конкретном случае. Таким образом, можно описать, к примеру, ситуацию выбора оптимального алгоритма генерации изображения в зависимости от конфигурации аппаратной платформы или же настроить параметры какого-либо сервиса либо совокупности сервисов (системы).

В соответствии с принципом масштабируемости блок изменчивости может задавать стратегии адаптивного поведения отдельного компонента сервиса, всего сервиса или даже всей системы в целом. Отсюда следует, что блок изменчивости не атомарен, он может включать в себя другие блоки, может иметь иерархическую структуру.

Блок изменчивости описывается диаграммой характеристик:

$$VB = (SubF, BlockParams, BlockStates, Rule), \quad (2)$$

где $SubF$ — гиперграфовое (формула (1)) представление модели характеристик блока измен-

чивости; $BlockParams = \{BlockParam_1, BlockParam_2, ..., BlockParam_k\}$ — множество параметров, влияющих на состояние блока; $BlockStates = \{BlockState_1, BlockState_2, ..., BlockStates_z)$ — множество состояний блока, при этом $BlockState_i \subseteq SubF \ \forall \ i = 1, ..., z$; $Rule: BlockParams' \rightarrow BlockStates$ — функция, ставящая в соответствие каждому элементу множества $BlockParams'$ некоторый элемент множества *BlockStates*. Множество $BlockParams'$ представляет собой множество подмножеств *BlockParams*. Корневая вершина такой диаграммы называется точкой вариации.

Множество влияющих параметров *BlockParams* можно представить следующим образом:

$$BlockParams = ExtParams \cup InnerParams \cup TargetParams, \quad (3)$$

где $ExtParams = \{ExtParam_1, ExtParam_2, ..., ExtParam_{k1}\}$ — множество внешних по отношению к рассматриваемому блоку параметров;

$InnerParams = \{InnerParam_1, InnerParam_2, ..., InnerParam_{k2}\}$ — множество внутренних, варьируемых параметров блока;

$TargetParams = \{TargetParam_1, TargetParam_2, ..., TargetParam_{k3}\}$ — множество целевых параметров. Элементы множества *TargetParam*, как правило, связаны с некоторыми целевыми функциями и определяются (в отличие от других подмножеств множества *Param*) экспертным путем.

Рассмотрим простейший пример. Пусть имеется некоторое мобильное приложение, которое должно полноценно работать в автономном режиме (без подключения к сети Интернет). Основное назначение приложения — проведение теста на оценку пространственного мышления пользователя. Для этого приложение использует несколько алгоритмов генерации изображений: алгоритм на основе глубоких нейросетей, генетический алгоритм и алгоритм, основанный на случайном переборе и комбинации фрагментов изображения. Выбор алгоритма должен быть определен исходя из аппаратной конфигурации устройства. При этом предполагается, что в основном программном коде не заложено никаких сведений о конкретных конфигурациях мобильных устройств и не заданы условные операторы выбора. После инсталляции и запуска приложение должно самостоятельно установить нужные зависимости и провести выбор алгоритма.

Диаграмма характеристик блока изменчивости, отвечающего за выбор необходимого алгоритма, изображена на рис. 3. Вершина с именем "Algorithms" является точкой вариации, вершины "Neuro", "Genetic" и "Simple" представляют собой соответствующие алгоритмы.

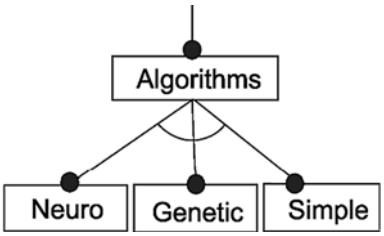


Рис. 3. Диаграмма характеристик блока изменчивости, отвечающего за выбор алгоритма генерации изображений

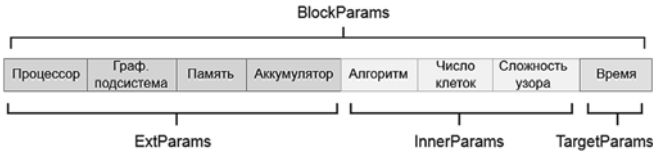


Рис. 4. Параметры блока изменчивости

На рис. 4 представлены параметры блока изменчивости *BlockParams*. Как видно, к внешним параметрам *ExtParams* относятся основные характеристики мобильного устройства. Следует отметить, что не все из перечисленных характеристик будут оказывать влияние на производительность устройства при выборе определенного алгоритма. Выявление того, является ли определенная характеристика влияющей на производительность, также является одной из задач рефлексии. К внутренним варьируемым параметрам (*InnerParams*) относится как сам алгоритм (нейросетевой, генетический или простой), так и его параметры: число клеток (фрагментов) изображения и его сложность. Целевые параметры (*TargetParams*) представляет время визуализации изображения. Оно должно быть минимально возможным.

Множество *BlockStates* в данном случае будет содержать в себе различные конфигурации диаграммы, представленной на рис. 3, т. е. пары из корневой характеристики ("Algorithms") и одного из возможных алгоритмов ("Neuro", "Genetic", "Simple").

Задача процедуры рефлексивной адаптации заключается в формировании функции *Rule*, устанавливающей соответствие между группой влияющих параметров блока изменчивости и элементами множества *BlockStates*. Методы, позволяющие достичь данной цели, будут рассмотрены в одной из ближайших статей.

Заключение

Проводимое исследование имеет ярко выраженный междисциплинарный характер. Про-

водятся аналогии между прикладными программными системами и сложноорганизованными живыми организмами с точки зрения их возможной самоадаптации к изменяющимся внешним условиям. Принципы работы аппарата самоадаптации ПО могут быть заимствованы от механизмов адаптации высших биологических существ, связанных с процессами самосознания и психики: рефлексии, планирования поведения в реальном времени и саморазвития на основе наблюдения за внешним миром. Наделение прикладной программной системы подобными адаптивными свойствами позволит ей осуществлять модификацию своей структуры на качественно ином уровне, что, в свою очередь, повысит ее надежность, отказоустойчивость и гибкость, снизит стоимость сопровождения и продлит срок эксплуатации. Такая система будет способна расширять класс решаемых задач на протяжении всего жизненного цикла, а также возьмет на себя выполнение тех операций, которые в настоящее время считаются частью обязанностей определенных специалистов (например, сможет выполнять функции самоадминистрирования).

Список литературы

1. **Carvalho M. B.** et al.The journey: a service-based adaptive serious game on probability // Serious games analytics / edited by C. S. Loh, Y. Sheng, D. Ifenthaler. Cham: Springer International Publishing, 2015. P. 97—106.

2. **Bernon C.** et al. Engineering self-organizing systems // Self-organizing software: from natural to artificial adaptation / edited by G. Di Marzo Serugendo, M. P. Gleizes, A. Karageorgos. Heidelberg: Springer Berlin Heidelberg, 2011. P. 283—312.

3. **Waignier G., Le Meur A. F., Duchien L.** A model-based framework to design and debug safe component-based autonomic sytems // Architecture for adaptive software systems / edited by R. Mirandola, I. Gorton, C. Hofmeister. Heidelberg: Springer Berlin Heidelberg, 2009. P. 1—17.

4. **Torbeyns J., Lehitinen E., Elen J.** Describing and studying domain-specific serious games. Cham: Springer International Publishing, 2015. 250 p.

5. **Colledanchise M., Ogren P.** How behavior trees modularize robustness and safety in hybrid systems // 2014 IEEE/RSJ International Conference on Intelligent Robots and Systems. Chicago. 2014. P. 53—62.

6. **Berinstein P.** et al. Game development tool essentials. Heidelberg: Springer Berlin Heidelberg, 2014. 467 p.

7. **Kenett R. S.** Future directions of software cybernetics: A position paper // 35th IEEE Annual Computer Software and Applications Conference Workshops. Washington: IEEE Computer Society, 2003. P. 43—44.

8. **Cai K. Y., Chen T. Y., Tse T. H.** Towards research on software cybernetics // 7th IEEE International Symposium on High Assurance Systems Engineering (HASE'02). Washington: IEEE Computer Society, 2002. P. 240.

9. **Ravindran K., Rabby M.** Software cybernetics to infuse adaptation intelligence in networked systems // IEEE International Conference on the Network of the Future (NOF). Washington: IEEE Computer Society, 2013. P. 1—6.

10. **Park J. S.** Essence-based, goal-driven adaptive software engineering // EEE/ACM 4th SEMAT Workshop on General Theory of Software Engineering (GTSE). Washington: IEEE Computer Society, 2015. P. 33—38.

11. **Liu C., Jiang C., Hu H., Cai K. Y., Huang D., Yau S. S.** Control-based approach to balance services performance and security for adaptive service based systems (ASBS) // 33rd Annual IEEE International Computer Software and Applications Conference (COMPSAC'09). Washington: IEEE Computer Society, 2009. P. 473—478.

12. **Ravindran K., Rabby M.** Software cybernetics to infuse adaptation intelligence in networked systems // Fourth IEEE International Conference on the Network of the Future (NOF). Washington: IEEE Computer Society, 2013. P. 1—6.

13. **Choi, T., Chan, H., Yue, X.** Recent development in big data analytics for business operations and risk management // IEEE Transactions on Cybernetics. Washington: IEEE Computer Society, 2016. P. 1—12.

14. **Mayer P., Klarl A., Hennicker R.** The autonomic cloud: a vision of voluntary, peer-2-peer cloud computing // 7th IEEE International Conference on Self-Adaptation and Self-Organizing Systems Workshops (SASOW). Washington: IEEE Computer Society, 2013. P. 89—94.

15. **Гиппенрейтер Ю. Б.** Введение в общую психологию: Курс лекций. М.: Юрайт, 2002. С. 89.

16. **Столяренко Л. Д.** Психология: учебник для вузов. СПб.: Питер, 2016. С. 124.

17. **Liu C., Jiang C., Hu H., Cai K. Y., Huang D., Yau S. S.** Control-based approach to balance services performance and security for adaptive service based systems (ASBS) // 33rd Annual IEEE International Computer Software and Applications Conference (COMPSAC'09). Washington: IEEE Computer Society, 2009. P. 473—478.

18. **Schobbens P. E., Heymans P., Trigaux J. C.** Feature diagrams: a survey and formal semantics // In 14th IEEE International Requirements Engineering Conference (RE'06) . Washington: IEEE Computer Society, 2011. P. 139—148.

19. **Kang K. C.** et al. Feature-oriented domain analysis (FODA): feasibility study. Pittsburgh: Software Engineering Institute, 1990. 161 p.

20. **Sinnema M., Deelstra S.** Classifying variability modeling techniques // Information and software technology. 2007. N. 7. P. 42—54.

21. **Liaskos S., Dinkelaker T.** et al. On goal—based variability acquisition and analysis // In 14th IEEE International Requirements Engineering Conference (RE'06). Washington: IEEE Computer Society, 2010. P. 77—85.

22. **Berger T.** Variability modeling in the real: an empirical journey from software product lines to software ecosystems. Leipzig: University of Leipzig, 2012. 225 p.

22. **Younis O., Ghoul S., Alomari M. H.** Systems variability modeling: a textual model mixing class and feature concepts // International Journal of Computer Science & Information Technology. 2013. N. 5. P. 127—139.

24. **Webber D. L., Gomaa H.** Modeling variability in software product lines with the variation point model // Science of Computer Programming. 2004. N. 3. P. 305—331.

25. **Losif-Lazar A. F., Schaef I., Wasowski A.** A Core Language for Separate Variability Modeling // Leveraging Applications of Formal Methods, Verification and Validation. Technologies for Mastering Change / edited by T. Margaria, B. Steffen. Heidelberg: Springer Berlin Heidelberg, 2014. P. 257—272.

A. M. Bershadsky, Dr. of Tech. Sciences, prof., e-mal: bam@pnzgu.ru,
A. S. Bozhday, Dr. Tech. Sciences, prof., e-mal: bozhday@yandex.ru,
Yu. I. Evseeva, Cand. of Tech. Sciences, Assoc. prof., e-mal: shymoda@mail.ru,
A. A. Gudkov, Cand. of Tech. Sciences, Assoc. prof., e-mal: alexei-ag@yandex.ru,
Penza State University

The Conception of Reflexive Self-Adaptation of Applied Software Systems

The article suggests a new approach to solving the important problem of the modern IT industry — self-adaptation of software components based on the concept of behavioral reflection. The basic principles of reflexive adaptation are considered, as well as the idea of using the engineering of software product lines to create adaptive programs. A generalized architectural solution for constructing reflexive self-adaptive systems capable of changing their behavioral characteristics directly during execution without recompiling the source code is given. Special attention is paid to the issues of formalized presentation and use of variability models for implementing adaptive behavior of application programs. The study was carried out with the financial support of the Russian Foundation for Basic Research in the framework of the scientific project No. 18-07-00408.

Keywords: *adaptive software systems, reflexive self-adaptation, variability model, characteristics diagram, software product lines, software engineering*

DOI: 10.17587/it.25.11-19

References

1. **Carvalho M. B.** et al.The journey: a service-based adaptive serious game on probability, *Serious games analytics*, edited by C. S. Loh, Y. Sheng, D. Ifenthaler, Cham, Springer International Publishing, 2015, pp. 97—106.

2. **Bernon C.** et al. Engineering self-organizing systems, *Self-organizing software: from natural to artificial adaptation*, edited by G. Di Marzo Serugendo, M. P. Gleizes, A. Karageorgos, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 283—312.

3. **Waignier G., Le Meur A. F., Duchien L.** A model-based framework to design and debug safe component-based autonomic sytems, *Architecture for adaptive software systems*, edited by R. Mirandola, I. Gorton, C. Hofmeister, Heidelberg, Springer Berlin Heidelberg, 2009, pp. 1—17.

4. **Torbeyns J., Lehitinen E., Elen J.** Describing and studying domain-specific serious games, Cham, Springer International Publishing, 2015, 250 p.

5. **Colledanchise M., Ogren P.** How behavior trees modularize robustness and safety in hybrid systems, *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Chicago, IEEE, 2014, pp. 53—62.

6. **Berinstein P.** et al. Game development tool essentials, Heidelberg, Springer Berlin Heidelberg, 2014, 467 p.

7. **Kenett R. S.** Future directions of software cybernetics: A position paper, *35th IEEE Annual Computer Software and Applications Conference Workshops,* Washington, IEEE Computer Society, 2003, pp. 43—44.

8. **Cai K. Y., Chen T. Y., Tse T. H.** Towards research on software cybernetics, *7th IEEE International Symposium on High Assurance Systems Engineering (HASE'02),* Washington, IEEE Computer Society, 2002, pp. 240.

9. **Ravindran K., Rabby M.** Software cybernetics to infuse adaptation intelligence in networked systems, *IEEE International Conference on the Network of the Future (NOF),* Washington, IEEE Computer Society, 2013, pp. 1—6.

10. **Park J. S.** Essence-based, goal-driven adaptive software engineering, *EEE/ACM 4th SEMAT Workshop on General Theory of Software Engineering (GTSE)*, Washington, IEEE Computer Society, 2015, pp. 33—38.

11. **Liu C., Jiang C., Hu H., Cai K. Y., Huang D., Yau S. S.** Control-based approach to balance services performance and security for adaptive service based systems (ASBS), *33rd Annual IEEE International Computer Software and Applications Conference (COMPSAC'09),* Washington, IEEE Computer Society, 2009, pp. 473—478.

12. **Ravindran K., Rabby M.** Software cybernetics to infuse adaptation intelligence in networked systems, *Fourth IEEE International Conference on the Network of the Future (NOF),* Washington, IEEE Computer Society, 2013, pp. 1—6.

13. **Choi T., Chan H., Yue X.** Recent development in big data analytics for business operations and risk management, *IEEE Transactions on Cybernetics*, Washington, IEEE Computer Society, 2016, pp. 1—12.

14. **Mayer P., Klarl A., Hennicker R.** The autonomic cloud: a vision of voluntary, peer-2-peer cloud computing, *7th IEEE International Conference on Self-Adaptation and Self-Organizing Systems Workshops (SASOW),* Washington, IEEE Computer Society, 2013, pp. 89—94.

15. **Gippenreiter Yu. B.** Introduction to general psychology. Lecture course, Moscow, Yurayt, 2002, pp. 89 (in Russian).

16. **Stolyarenko L. D.** Psychology: a textbook for high schools, St. Petersburg, Peter, 2016, pp.124.

17. **Liu C., Jiang C., Hu H., Cai K. Y., Huang D., Yau S. S.** Control-based approach to balance services performance and security for adaptive service based systems (ASBS), *33rd Annual IEEE International Computer Software and Applications Conference (COMPSAC'09),* Washington, IEEE Computer Society, 2009, pp. 473—478.

18. **Schobbens P. E., Heymans P., Trigaux J. C.** Feature diagrams: a survey and formal semantics, *In 14th IEEE International Requirements Engineering Conference (RE'06)*, Washington, IEEE Computer Society, 2011, pp. 139—148.

19. **Kang K. C.** et al. Feature-oriented domain analysis (FODA): feasibility study, Pittsburgh, Software Engineering Institute, 1990, 161 p.

20. **Sinnema M., Deelstra S.** Classifying variability modeling techniques, Information and software technology, 2007, no. 7, pp. 42—54.

21. **Liaskos S., Dinkelaker T.** et al. On goal—based variability acquisition and analysis, *In 14th IEEE International Requirements Engineering Conference (RE'06),* Washington, IEEE Computer Society, 2010, pp. 77—85.

22. **Berger T.** Variability modeling in the real: an empirical journey from software product lines to software ecosystems, Leipzig, University of Leipzig, 2012, 225 p.

23. **Younis O., Ghoul S., Alomari M. H.**Systems variability modeling: a textual model mixing class and feature concepts, *International Journal of Computer Science & Information Technology*, 2013, no. 5, pp. 127—139.

24. **Webber D. L., Gomaa H.** Modeling variability in software product lines with the variation point model, *Science of Computer Programming*, 2004, no.3, pp. 305—331.

25. **Losif-Lazar A. F., Schaef I., Wasowski A.** A Core Language for Separate Variability Modeling, Leveraging Applications of Formal Methods, Verification and Validation. Technologies for Mastering Change, edited by T. Margaria, B. Steffen, Heidelberg, Springer Berlin Heidelberg, 2014, no. 257—272.