© 2012 г. В.А. ГАЛАТЕНКО,
К.А. КОСТЮХИН
(Научно-исследовательский институт системных исследований РАН, Москва)

САМОЛЕЧЕНИЕ СИСТЕМ В КОНТЕКСТЕ КОНТРОЛИРУЕМОГО ВЫПОЛНЕНИЯ

В данной работе рассматривается расширение концепции контролируемого выполнения на самолечение систем. Анализируются современные технологии разработки самостабилизирующихся систем, предлагаются новые решения, основанные на концепции контролируемого выполнения.

SELFHEALING OF COMPLEX SYSTEMS IN CONTROLLED EXECUTION PARADIGM / V.A. Galatenko (Scientific Research Institute for System Analysis, Russian Academy of Sciences, 36-1 Nakhimovskiy pr., Moscow 117218, Russia, E-mail: galat@niisi.msk.ru), K.A. Kostyukhin (Scientific Research Institute for System Analysis, Russian Academy of Sciences, 36-1 Nakhimovskiy pr., Moscow 117218, Russia, E-mail: kost@niisi.msk.ru). The present work considers new extension of controlled execution paradigm. This extension deals with selfhealing of complex systems. Authors made a brief analysis of modern selfhealing methods and offer new solutions, based on controlled execution paradigm.

1. Введение

Лечение может потребоваться аппаратуре, программам и данным. Самолечение могут производить только активные сущности, но данные, учитывая наличие корректирующих кодов, также можно считать активными.

Современные аппаратно-программные системы нельзя считать свободными от ошибок и отказов в силу их усложнения и уменьшения характерных размеров логических элементов аппаратных компонентов. Тем не менее, из несовершенной аппаратуры необходимо создавать устойчиво работающие системы с соблюдением требований энергопотребления, рассеиваемой мощности, производительности, площади кристаллов, стоимости проектирования и контроля.

Обычно применяются три подхода к обеспечению отказоустойчивости:

— модульная избыточность;

— оперативное тестирование и восстановление;

— корректирующие коды.

Модульная избыточность (обычно – троирование и голосование) обеспечивает отказоустойчивость, но не устраняет причины сбоев и отказов. Кроме того, становятся неверными основные предположения данного подхода:

— наличие единственного отказавшего компонента;

— независимость возникновения отказов.

Корректирующие коды имеют фиксированную область применения, их нельзя считать универсальным средством.

Наиболее перспективным представляется *оперативное тестирование и восстановление*. На аппаратном уровне оно реализуется путем введения дополнительных компонентов и соединений с минимальным влиянием на производительность системы и умеренными накладными расходами на занимаемую площадь кристалла. Если выявлен отказ, соответствующий компонент выводится из употребления, на его место накладывается заплата либо из резервных компонентов (см., например, [1]), либо из резерва в виде ПЛИС ([2]). Резервирование с помощью ПЛИС имеет то достоинство, что при отсутствии отказов резерв может быть использован для наращивания функциональности. На программном уровне существует ряд стандартов обеспечения надлежащего уровня отказоустойчивости операционных систем и приложений.

Реконфигурирование может выполняться при программной поддержке и аппаратно. Обычно при программной нейтрализации устранимых ошибок выполняется откатка к корректному состоянию и делается попытка повторения операций. Недостаток этого метода в том, что он требует заметного времени. Аппаратная реконфигурация может выполняться быстрее. Отдельную проблему составляет локализация причин отказов. Чтобы оперативное тестирование не оказывало значительного негативного воздействия на эффективность функционирования системы, тестированию подвергаются крупные сущности (компоненты). Если в компоненте обнаруживается отказ, может быть произведен переход к следующему уровню детализации с тестированием подкомпонентов. Тем самым достигается приемлемый компромисс между накладными расходами на тестирование и гранулярностью масштабов лечения.

Отказы можно не только фиксировать, но и предсказывать (см., например, [3]). Предсказанию путем измерения характеристик работающих компонентов поддаются отказы, являющиеся следствием несовершенства технологии изготовления микросхем и/или старения кристаллов. В качестве мер профилактики предсказанных отказов могут использоваться понижение тактовой частоты и/или повышение входного напряжения.

Для предсказания и выявления отказов в многоядерных конфигурациях можно использовать параллельное автономное самотестирование микросхем с использованием заранее подготовленных тестовых шаблонов, как предлагается, например, в [3].

Такие тесты могут сочетать граничное сканирование и функциональное тестирование. Наличие свободных ядер, а также больших объемов недорогой стабильной памяти позволяет достичь высокой степени покрытия тестами при минимальных накладных расходах.

Разработанная авторами концепция контролируемого выполнения направлена на то, чтобы обеспечить выполнение системой своей миссии, несмотря на внутренние ошибки, а также внешние случайные или умышленные деструктивные воздействия.

Контролируемое выполнение включает в себя широкий спектр механизмов информационно-управляющего воздействия, среди которых ключевая роль отводится средствам самоконтроля аппаратуры и программ, а также средствам самолечения.

Базой для выполнения технологических требований к средствам контролируемого выполнения служит использование стандартов и архитектурных подходов, разрабо-

танных для средств управления, интеграция средств информационной безопасности, отладки и управления.

2. Понятие самостабилизации

Теоретической основой проектирования и реализации средств самолечения может служить понятие самостабилизации, предложенное Э. Дийкстрой в 1974 году в кратком сообщении [4]. Дадим несколько определений.

Конфигурация системы называется *безопасной*, если система демонстрирует поведение, соответствующее предварительно сформулированным требованиям. Система называется *самостабилизирующейся*, если из любой конфигурации она за конечное время приходит в безопасную конфигурацию, а из безопасной всегда переходит в безопасную.

Чтобы аппаратно-программная система была самостабилизирующейся, необходимо, чтобы этим свойством обладали аппаратура, операционная система и программные приложения.

Чтобы самостабилизирующиеся программные системы можно было создавать с использованием языков высокого уровня, необходимо, чтобы компилятор сохранял свойство самостабилизации.

Возможные направления исследований самостабилизации в контексте самолечения и автономной обработки данных предложены в работе [5]. Следует учитывать, однако, что применимость понятия самостабилизации в контексте самолечения небесспорна.

Например, из-за возможных случайных сбоев нарушается свойство замкнутости безопасных конфигураций; из-за неизбежных аппаратных и программных ошибок множество безопасных конфигураций может оказаться пустым. В [5] в качестве прямолинейного решения предлагается внешний мониторинг с рестартом в случае выявления проблем, но это нельзя назвать самостабилизацией.

3. Самолечение путем реконфигурирования в сервис-ориентированных архитектурах

Обычным средством повышения надежности аппаратных комплексов является внесение избыточности в их конфигурацию и переход на резервные компоненты в случае выхода из строя основных. Аналогичный подход возможен и применительно к программным системам, построенным из компонентов в сервис-ориентированной архитектуре (см., например, [6]).

Может существовать несколько компонентов, по-разному реализующих одну и ту же функциональность или способных заменить друг друга приемлемым для пользователя образом (например, перенаправив вывод на экран в случае поломки подсистемы печати). Если в каком-либо компоненте обнаруживается программная ошибка, программная система может быть реконфигурирована с использованием других, предположительно, корректно реализованных компонентов.

— Каркас для самолечения путем реконфигурирования в сервис- ориентированных архитектурах обычно включает в себя следующие процессы:

— мониторинг, обнаруживающий отклонения от поведения, специфицированного требованиями к системе;

— диагностика, выявляющая первопричины аномального поведения;

— выработка плана реконфигурирования;

— нейтрализация последствий аномального поведения, реконфигурирование, запуск системы в новой конфигурации.

Подобный каркас применим к унаследованным системам, созданным без учета перспектив самолечения. Он является внешним по отношению к системам, но часть включенных в него процессов не являются универсальными. Например, реализация возврата к непротиворечивому состоянию (нейтрализация отказов) зависит от специфики системы.

Полезной эвристикой является минимизация масштабов реконфигурирования, что, в свою очередь, требует детальной диагностики первопричины аномального поведения (с точностью до программного компонента).

4. Самолечение в жизненном цикле систем

Можно распространить самолечение на все этапы жизненного цикла систем – от проектирования до эксплуатации ([7]).

Основой подхода служит проектирование на основе моделей. На ранних стадиях жизненного цикла процедура проверки моделей способна выявить аномалии поведения. На более поздних этапах для той же цели фактическое поведение может сопоставлять с модельным.

Помимо универсальных проблем, самолечению подлежат следующие виды дефектов:

— проблемы производительности;

— проблемы распараллеливания;

— функциональные проблемы.

Один из методов лечения проблем производительности – образование пула ресурсов и их повторное использование вместо выделения новых. Проблемы распараллеливания лечатся путем выявления как возможностей нарушения атомарности доступа к разделяемым объектам, так и «игры на опережение», и ликвидации этих проблем без угрозы возникновения тупиковых ситуаций.

Функциональные проблемы могут возникать из-за некорректного интерфейса с готовыми компонентами. В таком случае производится смена интерфейса и/или компонентов.

На разных этапах жизненного цикла выявление и лечение перечисленных проблем может производиться различными методами. Например, на этапе реализации целесообразно производить статический анализ исходных текстов, на этапе эксплуатации – мониторинг выполнения.

5. Самолечение сетевого программного обеспечения

Современные сетевые конфигурации становятся все более сложными, разнородными и динамичными, а требования к их доступности – все более жесткими. Ручное администрирование подобных сетей по многим причинам является неприемлемым. Сетевое программное обеспечение должно само справляться с изменчивостью конфигураций и с возникающими проблемами, такими, как нехватка каких-либо ресурсов (коммуникационных и/или вычислительных), сбои и отказы каналов связи и функциональных компонентов.

В работе[8] в качестве основных средств решения перечисленных проблем предлагается выделение в сетевом программном обеспечении уровня самоуправления и реализация на этом уровне распараллеливания и балансировки нагрузки. Такие средства можно отнести к средствам самолечения сетевой конфигурации в целом.

6. Рефлексия как основа самолечения

Чтобы система могла производить самолечение, она должна располагать достаточной информацией о себе. Вычислительная рефлексия (см., например, [9]) обеспечивает информацию такого рода, на основании которой строится модель системы, реализующей самолечение.

В рефлексии можно выделить три аспекта:

— самопредставление;

— рефлексивная обработка данных;

— разделение обязанностей.

Самопредставление – это формирование и поддержание в актуальном состоянии модели системы. Детализация самопредставления должна, вероятно, соответствовать разбиению системы на компоненты, поскольку именно на этом уровне производится самолечение.

Самопредставление может быть процедурным или декларативным. В первом случае оно является частью самой системы, во втором – отдельной сущностью, обрабатываемой универсальным образом. Безусловно, декларативную форму следует считать предпочтительной.

Рефлексивная обработка данных распадается на два типа:

— структурная;

— поведенческая.

В первом случае имеется в виду статическая структура системы, определения классов и методов, во втором – доступ к динамическим структурам, таким как состояния объектов, стек вызовов и т.п.

Разделение обязанностей в контексте рефлексии – это средство борьбы со сложностью. Вообще говоря, рефлексия увеличивает сложность системы. Чтобы сделать увеличение умеренным, целесообразно поддерживать разные модели для разных аспектов системы, возможно, ввести несколько уровней рефлексии с соответствующими моделями на каждом из них.

В области рефлексии имеется много нерешенных проблем. Так, языки описания архитектуры как средство самопредставления обладают недостаточной выразительной силой, особенно применительно к динамическим характеристикам, присущими самоадаптирующимся программным системам. Возникают сложности и с самопредставлением распределенных систем.

Поддержка расширяемости и компонентного подхода – еще одна проблема. Модели систем должны обладать для этого гибкостью и способностью к интеграции.

Рефлексивную обработку данных трудно сделать прозрачной для основной части системы. Более перспективной представляется идея применения дисциплины аспектно-ориентированного программирования, когда рефлексия служит одним из аспектов.

7. Место самолечения в концепции автономной обработки данных

Парадигма автономной обработки данных была предложена в 2001 году сотрудниками корпорации IBM (см., например, [10]). Автономность систем базируется на следующих четырех основных «само» свойствах:

— самоконфигурирование – способность системы выполнять конфигурирование в соответствии с предопределенными высокоуровневыми политиками и адаптироваться к изменениям, вызванным автоматическим реконфигурированием;

— самооптимизация – способность системы непрерывно отслеживать использование ресурсов и управлять ими с целью повышения производительности и/или эффективности;

— самолечение – способность системы автоматически выявлять, диагностировать и нейтрализовывать отказы;

— самозащита – способность системы с упреждением идентифицировать злоумышленные атаки и каскадные отказы, не нейтрализованные средствами самолечения, и защищать себя от подобных атак и отказов.

Таким образом, самолечение – это один из аспектов автономной обработки данных, в целом направленной на преодоление сложности администрирования крупномасштабных разнородных систем путем автоматизации самоадминистрирования.

8. Контейнерный каркас для самолечения программных систем

Средства самолечения для программных систем должны обнаруживать ненадлежащее функционирование приложений, транзакций и/или бизнес-процессов и инициировать корректирующие действия без прерывания работы пользователей. Подобные средства должны работать в крупномасштабной, распределенной, открытой среде для разнородных приложений со сложными зависимостями (см., например, [11]).

Логическим следствием компонентных архитектур является контейнерный подход к проектированию и реализации средств самолечения программных систем. Контейнерный элемент – это сущность, управляющая определенным системным элементом, предоставляя тому услуги мониторинга, выявления и нейтрализации ошибок. Системный элемент, нуждающийся в услугах контейнера, определяет контракт с контейнерным элементом. В контракте специфицируются среда выполнения, схемы и методы выявления ошибок, возможные корректирующие действия, уровни ошибок и зависимости от других приложений.

9. Омоложение программного обеспечения как универсальный механизм самолечения

Чем сложнее система, тем проще должны быть механизмы ее самолечения. В противном случае выявление первопричин проблем и детализированное самолечение потребуют чрезмерное количество времени и других ресурсов.

Омоложение (в простейшем случае – перезапуск) компонентов программных систем, приложений или операционных систем – один из простых, но действенных и универсальных механизмов самолечения (см., например, [12]). Цель омоложения – очистка внутренних структур данных, оказавшихся испорченными вследствие программных

и/или аппаратных ошибок, и восстановление корректного и эффективного функционирования. Более продвинутыми формами омоложения можно считать замещение испорченного образа резервной копией или «прочистку» критически важных структур данных во время выполнения. Омоложение помогает также бороться со старением программного обеспечения (точнее – состояния ПО), которое выражается в накоплении негативных воздействий, таких как утечки памяти, «забытые» ресурсы и/или ошибки округления.

Омоложение – это временное решение проблем, оно не отменяет необходимость в диагностировании ошибок, поиска и устранения их первопричины, однако этот метод можно считать практичным и привлекательным по крайней мере по следующим причинам:

— для многих сложных систем, включающих в себя унаследованные компоненты, исходные тексты которых недоступны, выявление первопричин проблем в эксплуатационной среде практически невозможно;

— диагностирование и исправление сложных систем оказывается слишком длительным и дорогостоящим;

— ряд ошибок трудно воспроизвести в тестовой среде в силу недетерминированного поведения компонентов системы, но и проявляются они редко, так что на практике решение, которое дается омоложением, можно считать удовлетворительным.

Очевидны проблемы, ассоциированные с применением механизма омоложения:

— Временная потеря доступности. Этот недостаток можно сгладить, если воспользоваться технологией виртуализации, иметь несколько копий виртуальной машины и адаптивным образом омолаживать эти копии.

— Потеря рабочих данных при перезапуске. Этот недостаток ограничивает применение механизма омоложения компонентами и приложениями без состояния. Это может быть, например, активное сетевое оборудование.

— Отсутствие поддержки в операционных системах и компонентных каркасах. Только Solaris допускает учет зависимостей между процессами и частичную перезагрузку.

В работе [12] предлагаются возможные методы решения перечисленных проблем. Это в первую очередь включение в операционные системы прикладного программного интерфейса для использования услуг омоложения, адаптивное тиражирование процессов, зеркалирование состояний, очистка состояний без перезапуска.

Отметим, что механизм омоложения способен стать составляющей проектирования и реализации самостабилизирующихся операционных систем (см. [13]).

10. Заключение

Современная технология разработки сложных аппаратно-программных систем не позволяет сделать их свободными от ошибок. Кроме того, при работе систем возможны сбои и отказы, а также вредоносные воздействия.

Контролируемое выполнение направлено на то, чтобы обеспечить выполнение системой своей миссии, несмотря на внутренние ошибки, а также внешние случайные или умышленные деструктивные воздействия.

Контролируемое выполнение включает в себя широкий спектр механизмов информационно-управляющего воздействия, среди которых ключевая роль отводится

средствам самоконтроля и самолечения программ. Контролируемое выполнение подразумевает приемлемый уровень вносимых возмущений, что дает возможность его использования на этапе производственной эксплуатации систем.

СПИСОК ЛИТЕРАТУРЫ

1. *Metra C., Ferrari A., Omalia M., Pagni A.* Hardware Reconfiguration Scheme for High Availability Systems. - Proceedings of the 10th IEEE International On-Line Testing Symposium (OLTS'04). - IEEE, 2004, 6 pp.

2. *Venisbetti S.K., Akoglu A., Karla R.* Hierarchical Built-in Self-testing and FPGA Based Healing Methodology for System-on-a-Chip. - Proceedings of the Second NASA/ESA Conference on Adaptive Hardware and Systems (AHS 2007). - IEEE, 2007, 8 pp.

3. *Mitra S.* Globally Optimized Robust Systems to Overcome Scaled CMOS Reliability Challenges. - Proceedings of the EDAA 2008, pp. 941-946. - EDAA, 2008, 6 pp.

4. *Dijkstra E.W.* Self-stabilizing systems in spite of distributed control. - Commucations of the ACM, vol. 17, November 1974, pp. 643-644.

5. *Brukman O., Dolev S., Haviv Y., Yagel R.* Self-Stabilization as a Foundation for Autonomic Computing. - Proceedings of the Second International Conference On Availability, Reliability and Security (ARES'07). - IEEE, 2007, 8 pp.

6. *Wang Y., Mylopoulos J.* Self-repair Through Reconfiguration: A Requirements Engineering Approach. - Proceedings of the 2009 IEEE/ACM International Conference On Automated Software Engineering, pp. 257-268. - IEEE, 2009, 10 pp.

7. *Jung G., Margaria T., Wagner C., Bakera M.* Fromalizing A Methodology for Design- and Runtime Self-healing. - Proceedings of the 2010 Seventh IEEE International Conference and Workshop On Engineering Of Autonomic and Autonomous Systems, pp. 106-115. - IEEE, 2010, 10 pp.

8. *Michiels S., Desmet L., Joosen W., Verbaeten P.* The DiPS+ Software Architecture for Selfhealing Protocol Stacks. - Proceedings of the Fourth Working IEEE/IFIP Conference on Software Architecture (WICSA'04). - IEEE, 2004, 10 pp.

9. *Andersson J., de Lemos R., Malek S., Weyns D.* Reflecting on Self-Adaptive Software Systems. - Proceedings of the SEAMS'09, May 18-19, 2009, Vancouver, Canada, pp. 38-47. - IEEE, 2009, 10 pp.

10. *Khalid A., Haye M.A., Khan M.J., Shamail S.* Survey of Frameworks, Architectures and Techniques in Autonomic Computing. - Proceedings of the 2009 Fifth International Conference On Autonomic and Autonomous Systems, pp. 220-225. - IEEE, 2009, 6 pp.

11. *Ravi R.K., Sathyanarayana V.* Container based framework for Self-Healing Software Systems. - Proceedings of the 10th IEEE International Workshop on Future Trends of Distributed Computing Systems (FTDCS'04). - IEEE, 2004, 5 pp.

12. *Andrzejak A.* Generic Self-Healing via Rejuvenation: Challenges, Status Quo and Solutions. - Proceedings of the 2010 Fourth IEEE International Conference on Self-Adaptive and Self- Organazing Systems Workshop, pp. 239-242. - IEEE, 2010, 4 pp.

13. *Dolev S., Yagel R.* Towards Self-Stabilizing Operating Systems. - IEEE Transactions on Software Engineering, vol. 34, no. 4, July/August 2008, pp. 564-576. - IEEE, 2008, 13 pp.