УДК 629.39

В.И. БОЖКО, О.Н. ОДАРУЩЕНКО

Полтавский военный институт связи, Украина

ИТЕРАЦИОННАЯ ПРОЦЕДУРА ДЕКОМПОЗИЦИИ ПРОЕКТА ПРОГРАММНОЙ ПОДСИСТЕМЫ РАСПРЕДЕЛЕННОЙ ИНФОРМАЦИОННОЙ СИСТЕМЫ КРИТИЧЕСКОГО ПРИМЕНЕНИЯ

Предлагается итерационная процедура декомпозиции проекта программной подсистемы распределенной информационной системы критического применения на архитектурные элементы с учетом рисков реализации механизмов межпроцессного взаимодействия.

проектирование распределенных систем, распределенная информационная система критического применения, механизмы межпроцессного взаимодействия

Постановка проблемы и ее связь с научно-практическими задачами

Одной из основных проблем при создании информационных систем (ИС) критического применения (КП) есть обеспечение высокого уровня функциональной безопасности (ФБ) ее программной подсистемы (ПП). Отдельный подкласс таких ИС составляют распределенные информационные системы критического применения (РИС КП). При анализе характеристик функциональной безопасности выделяют два класса систем и их ПП. Первый класс представляют системы, которые имеют встроенные комплексы программ жесткого режима реального времени, которые руководят внешними объектами или процессами в автоматизированном режиме (например: средства вооружения, транспорта, атомной энергетики). Системы второго класса применяются для управления процессами и обработки деловой информации из внешней среды, в которых активно участвуют специалисты-операторы (военные штабные, административные, банковские системы) [1]. Допустимое время реакции на опасные отказы в этих системах может составлять минуты, а операции по восстановлению работоспособности могут быть доверены специалистам-администраторам по обеспечению безопасности. Для современных систем второго класса характерной есть распределенная структура, которая определяет необходимость отказоустойчивой реализации механизмов межпроцессного взаимодействия и нуждается в автоматизации процесса восстановления функциональности системы при соответствующем уровне оперативности. К резкому снижению уровня ФБ в распределенных информационных системах приводят следующие проблемы межпроцессного взаимодействия: координации, „гонки” данных, бесконечной отсрочки, взаимоблокировки, ненадежности сети обмена данными [2]. Эти проблемы обуславливаются гетерогенностью сетей, и даже если РИС не является гетерогенной, то остается проблема взаимодействия между несколькими процессами или потоками.

Таким образом, актуальной является задача обеспечения заданного (гарантированного) уровня ФБ ПП в РИС КП. Для решения этой задачи на разных этапах жизненного цикла ПП применяются традиционные методы повышения ФБ: проверка проекта высококвалифицированными специалистами; применение средств проектирования программ, профилактическое сопровождение; защита ПП от ошибок. Этап проектирования ПП есть определяющим при создании РИС КП, поэтому применение известных механизмов межпроцессного взаимодействия [1], должно гарантировать требуемый уровень ее ФБ.

Кроме того, анализ современных платформ разработки ПП РИС, таких как J2EE, .NET, показывает, что реализация некоторых свойств отказоустойчивости не обеспечивается, и если в них возникает необходимость, то разработчик должен сам реализовывать эти механизмы (либо за счет специфиче-

ского проектирования ПП) либо пользоваться готовыми решениями за рамками платформы.

Сложность, а потому и неопределенность, применения известных механизмов, адекватны сложности системы, для которой она разрабатывается, поэтому целесообразным видится решение научной задачи на основе разработки итерационных процедур декомпозиции проекта ПП на архитектурные элементы (АЭ) с учетом рисков реализации механизмов межпроцессного взаимодействия.

Постановка задачи исследования

Задачу по декомпозиции проекта программной подсистемы РИС КП представим задачей выбора оптимальной версии архитектуры проекта $V_i, i = 1,2,..., N$ по критерию минимизации типовых рисков реализации механизмов межпроцессного взаимодействия.

Предлагаются следующие этапы решения задачи декомпозиции проекта ПП РИС КП:

1) определить требования к архитектурным элементам проекта программной подсистемы РИС КП;

2) определить логические критерии безопасного функционирования механизмов межпроцессного взаимодействия ПП и выделенных архитектурных элементов;

3) провести декомпозицию проекта ПП на конечное число H архитектурных элементов i=1,2,...,H и построить граф функциональной безопасности (ГФБ) проекта;

4) построить логическую функцию безопасного функционирования (ЛФБФ) ПП, позволяющую аналитически строго, определить все комбинации состояний архитектурных элементов, в которых обеспечивается требуемый уровень безопасного функционирования механизмов межпроцессного взаимодействия.

Решение задачи декомпозиции проекта программной подсистемы

1. *Определение требований к архитектурным элементам программной подсистемы РИС КП.* При определении требований к архитектурным элементам ПП, каждый из которых представляется в модели проекта простым (бинарным) событием x_i с двумя возможными состояниямb, предлагаются следующие типовые допущения и ограничения**:**

– бинарность, т.е. возможность АЭ находиться только в двух несовместных состояниях x_i, случайных событий, с помощью которых представляются (моделируются) элементы $i = 1,2,...,H$ ПП;

– взаимная независимость всех выделенных бинарных событий;

– неограниченность процессов восстановления элементов в моделях.

Анализ современных технологий разработки ПП РИС показывает актуальность использования понятия программного компонента (software component), которое является одним из ключевых в современной инженерии программного обеспечения (ПО). Этим термином обозначают несколько различных понятий. Если речь идет об архитектуре ПО, под компонентом имеется в виду то же, что часто называется программным модулем. Это, достаточно произвольный и абстрактный элемент структуры системы, определенным образом выделенный среди окружения, решающий некоторые подзадачи в рамках общих задач системы и взаимодействующий с окружением через определенный интерфейс (архитектурный компонент). На диаграммах компонентов в языке UML изображаются компоненты, являющиеся единицами сборки и конфигурационного управления, - файлы с кодом на каком-то языке, бинарные файлы, какие-либо документы, входящие в состав системы. Иногда там же появляются компоненты, представляющие собой единицы развертывания системы, — это компоненты уже в третьем, следующем смысле. Компоненты развертывания являются блоками, из которых строится компонентное программное обеспечение. Эти же компоненты имеются в виду, когда говорят о компонентных технологиях, компонентной или компонентно-ориентированной (component based) разработке ПО, компонентах JavaBeans, EJB, CORBA, ActiveX, VBA, COM, DCOM, .Net, Web-службы (web services), а также о компонентном подходе. Согласно [3], такой компонент представляет собой структурную единицу программной системы, обладающую четко определенным интерфейсом, который полностью описывает его ответственность перед окружением. Такой компонент мо-

жет быть независимо поставлен или не поставлен, добавлен в состав некоторой системы или удален из нее, в том числе, может включаться в состав систем других поставщиков. Компонент, единица развертывания системы в этом смысле — выделенная структурная единица с четко определенным интерфейсом. Один компонент может также иметь несколько интерфейсов, играя несколько разных ролей в системе. Становится важным и то, какие другие компоненты он может задействовать при работе, а также каким ограничениям должны удовлетворять входные данные операций и какие свойства выполняются для результатов их работы. Эти ограничения являются так называемым интерфейсным контрактом или программным контрактом компонента. Это требует наличия определенной инфраструктуры, которая позволяет компонентам находить друг друга и взаимодействовать по определенным правилам. Набор правил определения интерфейсов компонентов и их реализаций, а также правил, по которым компоненты работают в системе и взаимодействуют друг с другом, принято объединять под именем компонентной модели (component model) [3]. В компонентную модель входят правила, регламентирующие жизненный цикл компонента, т.е. то, через какие состояния он проходит при своем существовании в рамках некоторой системы - компонентной среды (или компонентного каркаса, component framework). Примеры известных компонентных сред — различные реализации J2EE, .NET, CORBA.

Компоненты отличаются от классов объектно-ориентированных языков. Класс определяет не только набор реализуемых интерфейсов, но и саму их реализацию, поскольку он содержит код определяемых операций. Обычно компонент является более крупной структурной единицей, чем класс. Реализация компонента часто состоит из нескольких тесно связанных друг с другом классов. Понятие компонента является более узким, чем понятие программного модуля. Понятие компонента добавляет атомарность развертывания, а также возможность поставки или удаления компонента отдельно от всей остальной системы.

Таким образом, предлагается архитектурным элементом ПП считать компонент в соответствии с рассмотренными выше принципами компонентного подхода в технологиях программирования.

2. *Определение логического критерия безопасного функционирования механизмов межпроцессного взаимодействия программной подсистемы и выделенных архитектурных элементов.*

Для определения логического критерия безопасного функционирования механизмов межпроцессного взаимодействия ($у_{БФ}$) в V_i версии архитектуры ПП был проведен анализ статистики реализации типовых рисков, который показал, что интегрегативной функцией в ПП может быть реализация угрозы отказа в обслуживании (ОВО) ее компонентов. В [4] отмечено, что в понятие ОВО должно быть включено время, основанное на том, что каждая функция элемента ПП должна быть связана с некоторым периодом времени, называемым максимальным временем ожидания (MWT - maximal wait time) ее реализации. Для некоторого элемента MWT определяется как длина промежутка времени после запроса на реализацию некоторой его функции, в течение которого считается приемлемым ее предоставление.

Дав определение MWT, мы можем теперь ввести точное определение угрозы ОВО, а именно, будем говорить, что имеет место угроза ОВО всякий раз, когда функция компонента архитектуры ПП с соответствующим MWT запрашивается зарегистрированным компонентом ПП в момент времени *t* и не предоставляется этому компоненту к моменту времени t+MWT. При более тщательном рассмотрении угрозы ОВО и понятия MWT возникают некоторые вопросы. Например, угрозы ОВО можно полностью избежать, если определить MWT для всех элементов равным бесконечности. Однако этот подход не применим для тех случаев, когда величина MWT определяется некоторой значимой операционной характеристикой системы КП, в интересах которой функционирует РИС. Кроме того, значение MWT можно определить только для конкретного набора функций ПП РИС КП, для которых оно необходимо, то есть можно определить некоторое подмножество активов системы как особенно критическое; тогда значения MWT будут соответствовать функциям, связанным с этими критическими активами.

Таким образом, предлагается в качестве критерия безопасного функционирования механизмов межпроцессного взаимодействия в версии архитектуры ПП V_i принять следующий логический критерий:

$y_{БФ}$ = I (истина), если $MWT_{V_i} \le MWT_{РИСКП}$.

3. *Декомпозиция ПП на конечное число H архитектурных элементов i=1,2,..., H и построение ГФБ проекта.* При выделении в проектируемой ПП конечного числа компонентов, следует учитывать следующие рекомендации [4]:

1. Главный критерий описания и построения ГФБ заключается в том, чтобы логически определить и графическими средствами ГФБ изобразить все условия реализации в системе выходных функций каждого выделенного компонента архитектуры ПП.

2. Каждый выделенный компонент до двух раз увеличивает расчетную модель системы. Вместе с тем, недостаточное число компонентов может привести к недопустимому снижению точности модели и проектной оценки ФБ системы.

3. Все выделенные компоненты должны иметь наименования, соответствующие реальным объектам проектируемой ПП.

4. Для учета в моделях ФБ многофункциональных компонентов выделяются группы размноженных функциональных вершин ГФБ.

5. Для представления сложных логических отношений и для повышения наглядности, в ГФБ могут вводиться фиктивные вершины, на которых осуществляется необходимая группировка логических условий.

6. Проверка полноты и правильности построения ГФБ должна выполняться постоянно в процессе проектирования ПП. Она предусматривает многократное, порой итерационное повторения всех этапов построения ГФБ, в целях его уточнения, исправления, расширения и проверки.

По аналогии с [5] ГФБ можно представить одним обобщенным структурным фрагментом $G_F(X,Y)$ (на рис. 1) и двумя базовыми аналитическими соотношениями (логическими уравнениями):

$$\tilde{y}_d = \{y_d, \bar{y}_d\}; \quad \tilde{y}_k = \{y_k, \bar{y}_k\}.$$

На рис. 1 в обобщенном виде представлены все типовые элементы ГФБ. Обеспечивающие вершины разделены на две группы – дизъюнктивную и конъюнктивную. Номера d обеспечивающих вершин дизъюнктивной группы составляют множество D_i, а номера k обеспечивающих вершин конъюнктивной группы составляют множество K_i. С помощью знака «тильда» на схеме обозначена возможность использования в ГФБ любого из двух возможных выходов обеспечивающих вершин – прямого или инверсного. Знаком тильды отмечены и выходные интегративные функции обеспечивающих вершин, которые также могут быть прямыми или инверсными.

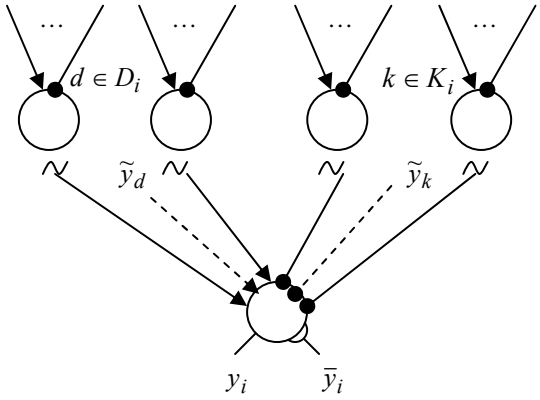

Рис. 1. Схема обобщенного фрагмента ГФБ

С учетом принятых обозначений интегративные функции прямого и инверсного выходов вершины i обобщенного фрагмента ГФБ определяются следующими универсальными, базовыми логическими уравнениями.

Для прямой выходной интегративной функции:

$$y_i = i \cdot (\bigcup_{d \in D_i} \tilde{y}_d) \cdot (\bigcap_{k \in K_i} \tilde{y}_k),$$

где i – функциональная вершина;

$$y_i = (\bigcup_{d \in D_i} \tilde{y}_d) \cdot (\bigcap_{k \in K_i} \tilde{y}_k),$$

где i – фиктивная вершина.

Для инверсной выходной интегративной функции:

$$\bar{y}_i = \bar{i} \cup (\bigcap_{d \in D_i} \bar{\tilde{y}}_d) \cup (\bigcup_{k \in K_i} \bar{\tilde{y}}_k),$$

где i – функциональная вершина;

$$\bar{y}_i = (\bigcap_{d \in D_i} \bar{\tilde{y}}_d) \cup (\bigcup_{k \in K_i} \bar{\tilde{y}}_k),$$

где i – фиктивная вершина.

4. *Построение ЛФБФ версии ПП, позволяющей аналитически строго, определить все комбинации*

состояний архитектурных элементов, в которых обеспечивается требуемый уровень функционирования механизмов межпроцессного взаимодействия.

Определим понятие ЛФБФ как логическую форму представления тех состояний системы, в которых реализуется заданный критерий $y_{БФ}$ ее функционирования. Исходными данными для построения ЛФБФ есть ГФБ версии архитектуры ПП и логические критерии функционирования ПП. Строится ЛФБФ при помощи специальных процедур решения систем логических уравнений для выходных функций каждой из вершин ГФБ. ЛФБФ должна быть получена для каждого заданного логического критерия безопасного функционирования системы. Для небольших и структурно не очень сложных архитектур может быть применен метод прямой аналитической подстановки, который предусматривает последовательную замену в критерии $y_{БФ}$, всех интегративных функций y_i их уравнениями, которые выбираются из системы уравнений для каждого из y_i. Такая подстановка выполняется до тех пор, пока в полученном выражении не останется ни одного не раскрытого обозначения функции y_i, т.е. все они будут заменены простыми логическими переменными x_i. Раскрывая скобки и преобразуя выражение по правилам алгебры логики, окончательно получаем искомую ЛФБФ для заданного критерия $y_{БФ}$.

Далее определяются все комбинации состояний архитектурных элементов, в которых обеспечивается реализация критерия $y_{БФ}$ и делается вывод о продолжении или прекращении итерационного процесса декомпозиции архитектуры программной подсистемы РИС КП.

Выводы и перспективы дальнейших исследований

Таким образом, предлагаемая итерационная процедура декомпозиции проекта программной подсистемы РИС КП на архитектурные элементы с учетом рисков реализации механизмов межпроцессного взаимодействия включает следующие этапы:

1. Определение архитектурных элементов в версии проекта программной подсистемы РИС КП в соответствии с принципами компонентного программирования.

2. Определение логического критерия безопасного функционирования механизмов межпроцессного взаимодействия компонентов, как реализацию угрозы отказа в их обслуживании.

3. Декомпозиция версии проекта ПП на конечное число *H* архитектурных элементов i=1,...,*H* и построение графа функциональной безопасности проекта.

4. Построение логической функции безопасного функционирования для версии проекта ПП.

5. Проверка реализации требуемого уровня безопасного функционирования механизмов межпроцессного взаимодействия в соответствии с определенным критерием.

6. Вывод о продолжении или прекращении итерационного процесса декомпозиции архитектуры программной подсистемы.

Предлагаемая итерационная процедура позволяет провести оценку декомпозиции архитектуры проекта ПП с точки зрения безопасности ее функционирования в РИС КП. Развитие предлагаемой итерационной процедуры планируется в направлении совершенствования инструментальных средств построения ЛФБФ для структурно-сложных ПП.

Литература

1. Липаев В.В. Функциональная безопасность программных средств. – М: СИНТЕГ, 2004. – 348 с.

2. Таненбаум Э., Стеен М. Распределенные системы. – СПб.: Питер, 2003. – 877 с.

3. F.Bachmann, L.Bass, C.Buhman, Vol. II: Technical Concepts of Component-Based Software Engineering, 2nd Edition/ Technical Report CMU/SEI-2000-TR-008 [Электрон. ресурс]. – Режим доступа: http://www.sei.cmu.edu/pub/documents/00.reports/pdf/00tr008.pdf.

4. Зегжда П.Д., Корт С.С., Ростовцев А.Г. Математические основы информационной безопасности / А.П. Баранов и др. – Орел: ВИПС, 1997. – 388 с.

Поступила в редакцию 22.02.2007

Рецензент: д-р техн. наук, проф. В.С. Харченко, Национальный аэрокосмический университет им. Н.Е. Жуковского «ХАИ», Харьков.