Статья опубликована в журнале «Информатизация и связь». 2008. № 2, с.20–23.

МОДЕЛИРОВАНИЕ ЖИЗНЕННОГО ЦИКЛА КОМПОНЕНТЫ ПРОГРАММНОГО КОМПЛЕКСА С ИСПОЛЬЗОВАНИЕМ ДИАГРАММ СОСТОЯНИЙ

Маврин Павел Юрьевич
магистрант СПбГУ ИТМО
email: mavrin@rain.ifmo.ru

Корнеев Георгий Александрович
ассистент СПбГУ ИТМО
email: kgeorgiy@rain.ifmo.ru

Станкевич Андрей Сергеевич
ассистент СПбГУ ИТМО
email: stankev@rain.ifmo.ru

Шалыто Анатолий Абрамович
доктор технических наук
профессор СПбГУ ИТМО
email: shalyto@mail.ifmo.ru

Аннотация

Излагается подход к решению задачи моделирования жизненного цикла компонент программных комплексов, основанный на использовании диаграмм состояний в качестве базовой модели. Предлагаемый подход позволяет описывать сложные жизненные циклы таких компонент, что, в свою очередь, позволяет реализовать сложную логику их поведения.

***Ключевые слова**: компонентная архитектура, диаграммы состояний.*

1. Введение

Большинство крупных программ строятся с использованием компонентной архитектуры. При этом функциональность распределяется между независимыми компонентами, взаимодействующими между собой через узкие интерфейсы, называемые сервисами.

В большинстве случаев, несмотря на логическое разделение, после сборки, компоненты функционируют как единое целое и могут запускаться и останавливаться только все вместе.

В некоторых случаях этого бывает недостаточно, и требуется, чтобы компоненты имели возможность запускаться и останавливаться независимо, а также могли изменять конфигурацию «на лету» — без перезапуска всей системы. Программирование таких систем является сложным, так как приходиться решать дополнительные задачи. Например, требуется выполнить моделирование жизненных циклов компонент, построить механизмы, отслеживающие потоки, порожденные компонентами и т. д.

Оригинальное решение одной из этих задач — задачи моделирования жизненного цикла компоненты было реализовано в ядре «Taiga» компонентных систем, созданных студентами и аспирантами СПбГУ ИТМО для проведения олимпиад по программированию различных уровней.

2. Постановка задачи

Изложим несколько упрощенную версию жизненного цикла компоненты в ядре «Taiga».

Запуск компоненты в этом ядре происходит поэтапно. Компонента регистрирует сервисы, которые ей необходимы, и сервисы, которые она может предоставить. После этого компонента ожидает, пока будут активированы все необходимые ей сервисы, затем запускается, активирует предоставляемые сервисы и начинает работать.

При возникновении ошибки, отключении необходимого сервиса или изменении конфигурации компонента останавливается, ее потоки обрываются, ее сервисы отключаются, и компонента перезапускается.

Возможны дополнительные ситуации, например, если конфигурация компоненты была изменена до ее запуска, то компонента сразу же возвращается в начало цикла.

Поэтому весьма актуально решение задачи моделирования жизненного цикла компоненты и его реализации в точном соответствии со спецификацией.

3. Формирование автомата

В описанном жизненном цикле могут быть выделены состояния, в которых может находиться компонента: ожидание сервисов, запуск, работа, остановка и т. д. Если между состояниями указать переходы, то получим диаграмму переходов конечного автомата, моделирующего поведение компоненты. Упрощенная диаграмма переходов автомата изображена на рис. 1 (расшифровка сокращений, используемых на диаграммах, приведена в Приложении).

Автомат, поведение которого описано на рис. 1, не учитывает всей сложности жизненного цикла компоненты.

Например, в ядре «Taiga» компоненты запускаются, только если предоставляемый ими сервис кому-то необходим. Начало жизненного цикла для этого случая изображено на рис. 2.

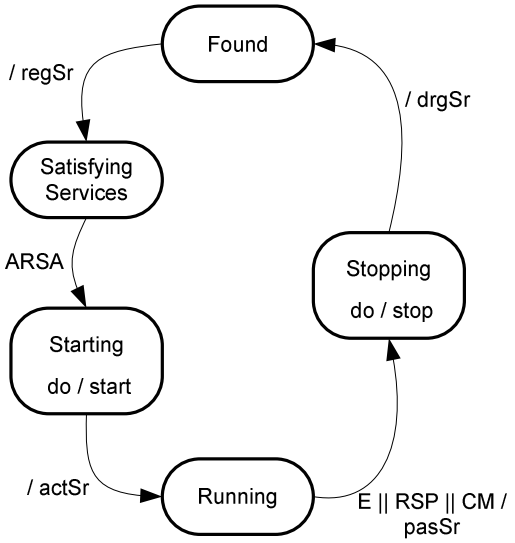

Рис. 1. Упрощенная версия жизненного цикла

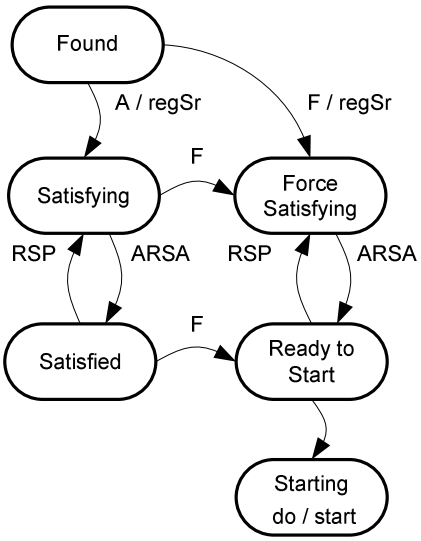

Рис. 2. Усложнение начала цикла

Кроме изложенного, необходимо учесть, что после принудительной остановки, компонента в зависимости от причины остановки, может вести себя по-разному. Возможно, также введение дополнительных состояний для неправильно сформированных компонент и т. д.

Автомат, учитывающий все эти нюансы, содержит более 20 состояний и более 50 переходов (например, из каждого состояния есть переход по событию «configuration modified»).

Таким образом, автомат получается большим и запутанным. Поэтому для моделирования жизненного цикла была выбрана более мощная модель – диаграмма состояний (*StateChart*) [1].

Применение диаграмм состояний вместо классических диаграмм переходов позволило уменьшить число состояний до 15, а число переходов — до 20.

4. Преимущества использования диаграмм состояний

Рассмотрим подробнее возможности диаграмм состояний, за счет которых достигается сокращение числа состояний и переходов по сравнению с классическими диаграммами переходов.

4.1. OR-декомпозиция состояний

OR-декомпозиция — это выделение подсостояний и надсостояний. Она позволяет, во-первых, создать удобную и понятную иерархию состояний, а, во-вторых, заменить несколько одинаковых переходов на один общий. Например, если на рис. 2 из каждого из состояний: «Satisfying», «Force Satisfying», «Satisfied» и «Ready to Start» добавить переход в состояние «Found» по событию «configuration modified», то граф переходов резко усложнится. Вместо этого можно добавить новое состояние «Preparing», объединяющее эти четыре состояния и сделать переход из него (рис. 3).

4.2. AND-декомпозиция состояний

AND-декомпозиция — это выделение состояний, в которых система может находиться одновременно.

Например, на рис. 3 у состояния «Preparing» выделяются две части, существующие одновременно. Выделив их явно, получим конструкцию, изображенную на рис. 4.

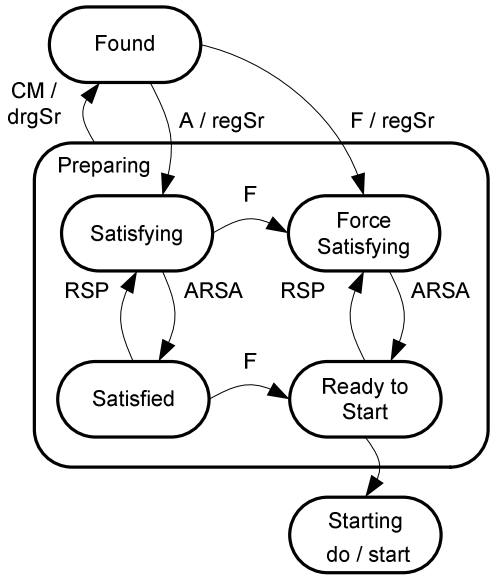

Рис. 3. OR-декомпозиция состояний

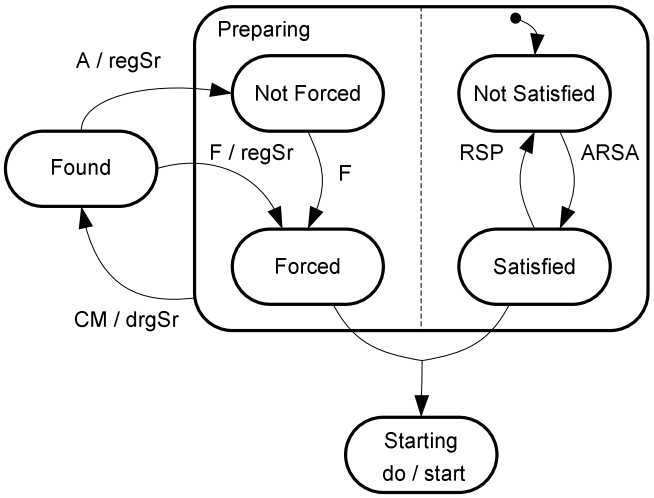

Рис. 4. AND-декомпозиция состояний

4.3. Действия при входе и при выходе

Действия при входе и при выходе, приписанные к некоторым состояниям — одни из ключевых элементов диаграмм состояний. Совместно с OR- и AND-декомпозицией состояний, они обеспечивают диаграммам состояний огромную выразительную силу.

Особенность действия при входе состоит в том, что оно выполняется как при прямом, так и при косвенном входе в состояние. На рис. 6 во всех трех случаях при выполнении выделенного перехода формируется действие при входе в состояние.

То же верно и для действия при выходе. Реализация той же логики с использованием только действий при переходах невозможна без увеличения числа переходов.

Возвращаясь к основной задаче этой работы, можно отметить, например, что в предлагаемой модели требуется следить за своевременной регистрацией и дерегистрацией сервисов. С использованием действий при входе и при выходе это можно сделать следующим способом. Добавим специальное состояние «Services Registered», поместим внутрь него все состояния, в которых сервисы должны бить зарегистрированы и добавим ему соответствующие действия при входе и при выходе (рис. 5).

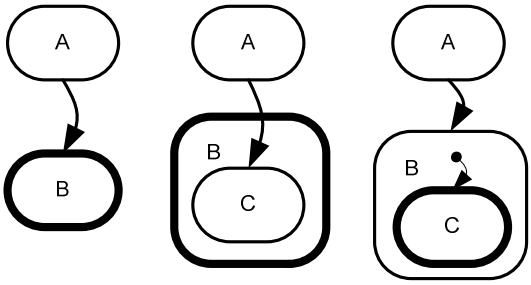

Рис. 6. Различные варианты входа в состояние

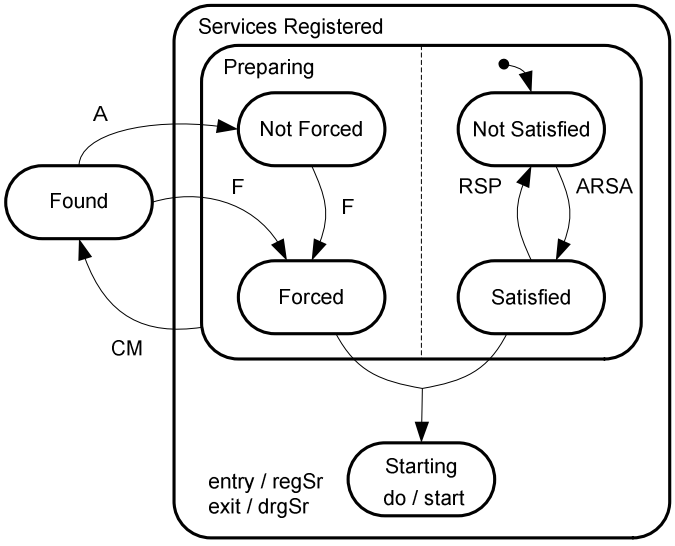

Рис. 5. Использование действий при входе и при выходе

Заключение

Немного упрощенный вариант диаграммы состояний, описывающей жизненный цикл компоненты в ядре «Taiga» представлен на рис. 7. В этой диаграмме применяются C-вершины, позволяющие сделать диаграмму более наглядной. Диаграмма получилась достаточно простой, однако она формально описывает поведение компоненты во всех выбранных ситуациях. Благодаря этому, а также отсутствию повторяющихся элементов, эта модель позволяет легко вносить изменения в логику компоненты и добавлять новые функциональные возможности.

В заключение отметим, что реализация диаграммы состояний представляет отдельную задачу, поскольку выбранная семантика и способ, принятый для ее программирования должны корректно обрабатывать AND- и OR- декомпозицию состояний, а также действия при входе и при выходе. При разработке ядра «Taiga» для этого была разработана семантика диаграмм состояний [2] и написана библиотека, позволяющая реализовать поведение, описанное диаграммой состояний.

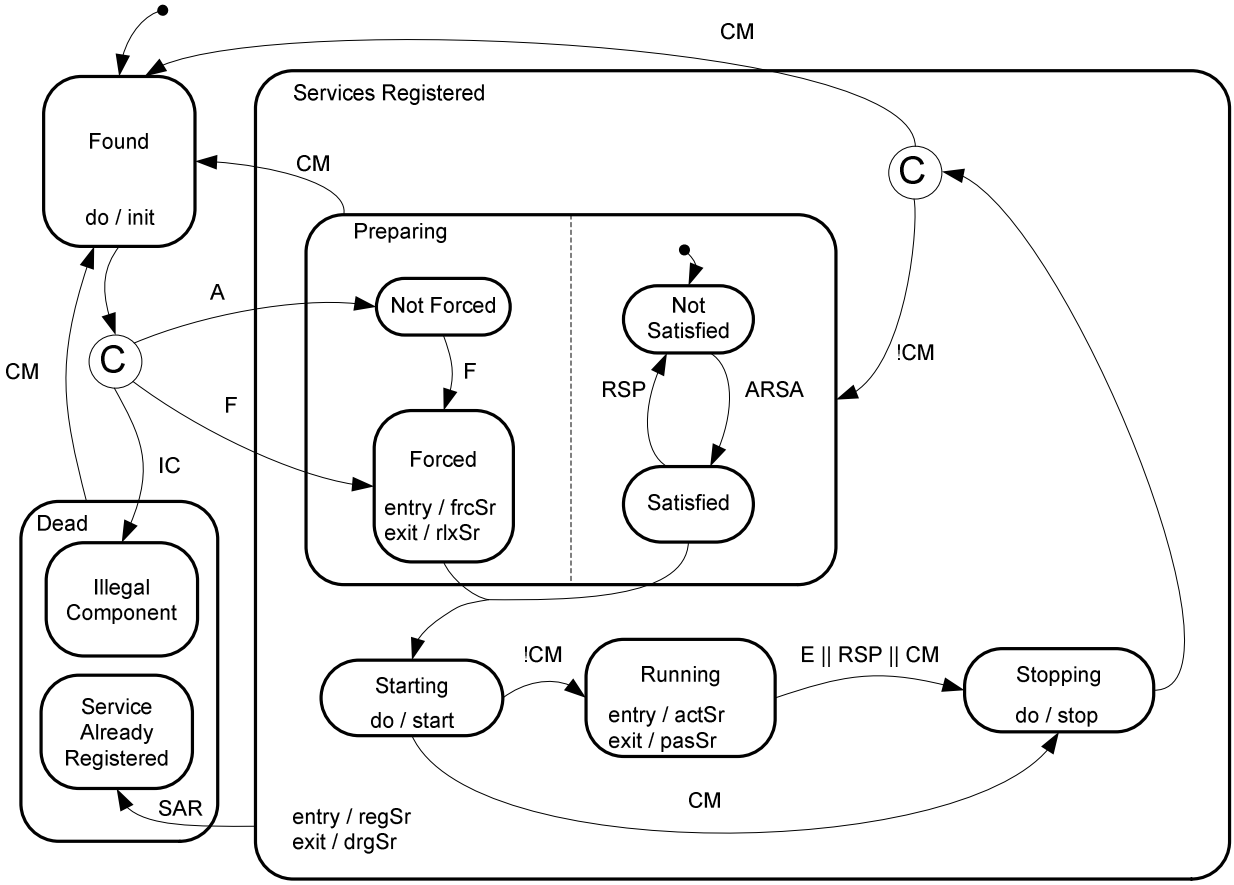

Рис. 7. Жизненный цикл компоненты в ядре «Taiga»

Приложение. Расшифровка сокращений, используемых на диаграммах

Идентификатор	Расшифровка	Описание
actSr	activate services	Активация сервисов, предоставляемых компонентой
drgSr	disregister services	Дерегистрация сервисов, предоставляемых и используемых компонентой
frcSr	force services	Форсирование запуска необходимых компонент
pasSr	passivate services	Отключение сервисов, предоставляемых компонентой
regSr	register services	Регистрация сервисов, предоставляемых и используемых компонентой
rlxSr	relax services	Прекращение форсированного запуска необходимых компонент
A	auto	Необходимости в запуске компоненты пока нет
ARSA	all required services activated	Активированы все сервисы, необходимые компоненте
CM	configuration modified	Изменилась конфигурация компоненты
E	error	Произошла неожиданная ошибка, требующая остановки компоненты
F	forced	Компоненту необходимо запустить
RSP	required service passivated	Отключен сервис, необходимый для работы компоненты
SAR	service already registered	Предоставляемый сервис уже зарегистрирован другой компонентой

Литература

[1] Harel D. Statecharts: A Visual Formalism for Complex Systems //Science of Computer Programming. 1987. June, pp. 231–274.

[2] **Маврин П. Ю., Корнеев Г. А., Шалыто А. А. Формальная семантика диаграмм состояний, удобная для практического применения /Материалы конференции «Software Engineering Conference (Russia) – SEC(R)2006», с. 43–46.**