Системы управления и обработки информации. 2009. Вып. 18, с. 82–87.

УДК 681.322:681.5

А.О. РЕМИЗОВ,
А.А. ШАЛЫТО, д-р техн. наук, проф.

ПРИМЕНЕНИЕ АВТОМАТНОГО ПОДХОДА ПРИ СОЗДАНИИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ БИУС

В настоящее время основная функциональность задач, решаемых боевыми информационно-управляющими системами (БИУС) надводных кораблей и подводных лодок, реализуются с помощью программного обеспечения.

В соответствии с классификацией Д. Харела [1] программные системы можно разделить на следующие три типа.

- **Трансформирующие системы.** Преобразуют входные данные и завершают работу. К таким системам относятся компиляторы и архиваторы.
- **Интерактивные системы.** Взаимодействие с окружающей средой в режиме диалога. Например, текстовые редакторы.
- **Реактивные системы.** Взаимодействуют с окружающей средой посредством обмена сообщениями в темпе, задаваемом окружающей средой. Примерами являются системы управления техническими средствами, автоматизированные системы боевого управления и т. п.

БИУС являются классическим образцом реактивных систем. В настоящей работе на примере задачи начальной загрузки одного из модулей системы будут рассмотрены различные аспекты построения программного обеспечения систем этого класса.

ОСОБЕННОСТИ ПОСТРОЕНИЯ МОДУЛЯ

Рассматриваемый модуль является реактивной системой. Он представляет собой «черный ящик», взаимодействующий с внешней средой (другими модулями) за счет обмена сообщениями. Модуль находится в состоянии ожидания внешнего воздействия (поступления сообщения), с приходом которого он выполняет действия, определяемые алгоритмом управления.

ПОСТРОЕНИЕ МОДЕЛИ

На этапе проектирования программного обеспечения **заказчиком было предложено использовать** два типа диаграмм *UML*: диаграммы классов и диаграммы последовательности. Для построения

диаграмм применялось инструментальное средство *IBM Rational Rose RT*.

Диаграммы классов предназначены для проектирования статической структуры модуля и отображают взаимосвязь между экземплярами классов. Диаграмма классов модуля приведена на рис. 1.

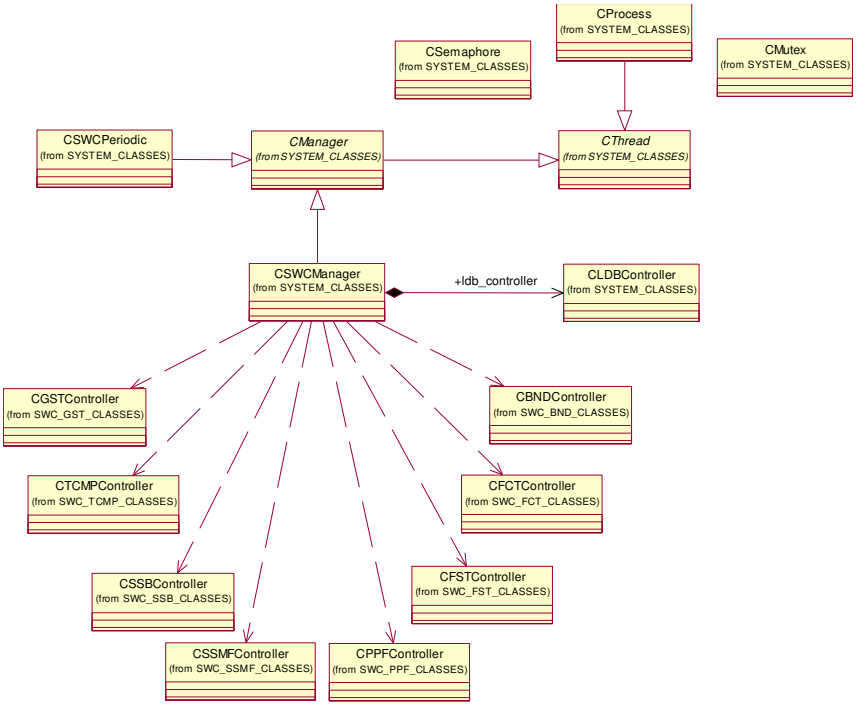

Рис.1. Диаграмма классов модуля

Диаграммы классов целесообразно использовать при моделировании статики по следующим причинам:

- структура программы при таком задании понятна всем участникам проекта (архитектору, разработчику, тестеру и т. д.);
- диаграммы могут применяться в проектной документации;
- по диаграмме может быть изоморфно сгенерирован «скелет» программного кода на априори заданном языке программирования.

Для моделирования динамической составляющей модуля, как отмечено выше, **заказчиком было предложено** использовать диаграммы последовательности, которые отображают взаимодействие между объектами классов при различных входных воздействиях. Пример диаграммы последовательности для одного из внешних воздействий представлен на рис. 2. Диаграммы последовательности:

- в качестве сценария поведения системы понятны всем участникам проекта (архитектору, разработчику, тестеру и т. д.);
- могут использоваться в проектной документации.

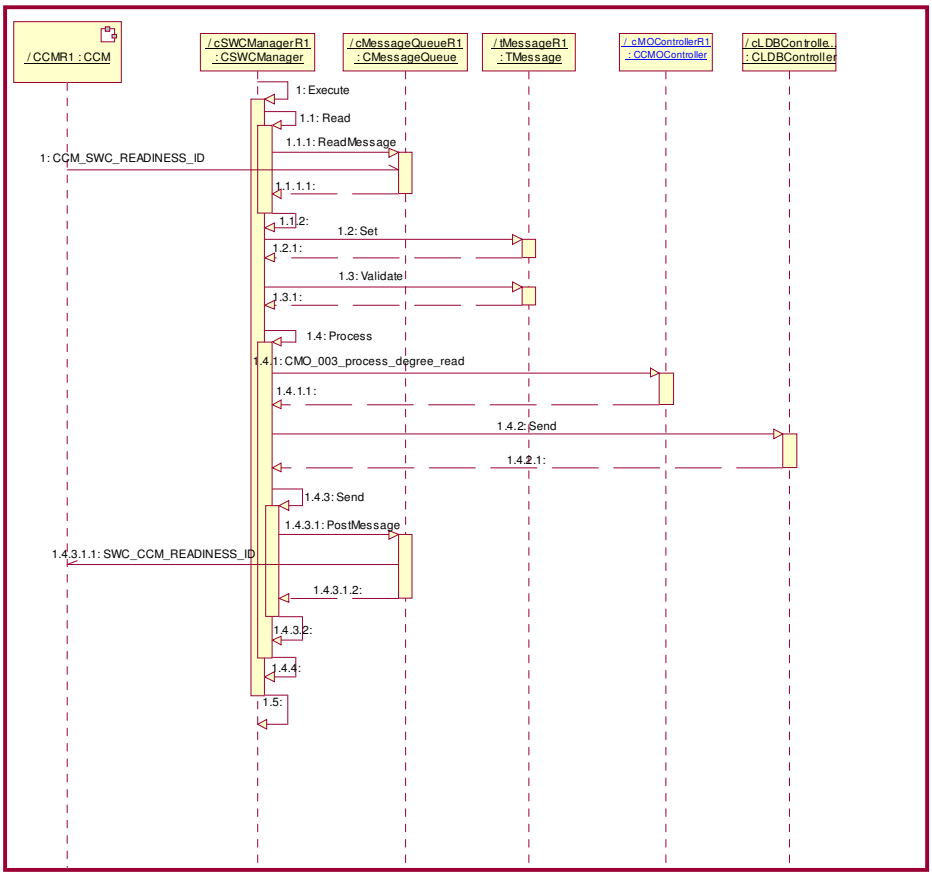

Рис. 2. Диаграмма последовательности для одного из внешних воздействий

Недостатками этого типа диаграмм является:

- фрагментарное отображение поведения объекта;
- нецелесообразность генерации по ним кода, так как они отражают поведение системы только в рамках одного сценария.

ПРИМЕНЕНИЕ АВТОМАТНОГО ПОДХОДА ПРИ МОДЕЛИРОВАНИИ ДИНАМИКИ МОДУЛЯ

В работах [2–4] показано, что для моделирования реактивных систем целесообразно использовать «диаграммы состояний», как расширение графов переходов конечных автоматов, так как они хорошо описывают реакцию системы на события. Поэтому для по-

строения программы для решения задачи начальной загрузки модуля **авторами было предложено использовать** для описания его поведения не диаграммы последовательности, а диаграммы связей и состояний, которые применяются в автоматном программировании [5].

На рис. 3 изображена схема связей автомата, которая позволяет использовать в диаграммах состояний только символы переменных.

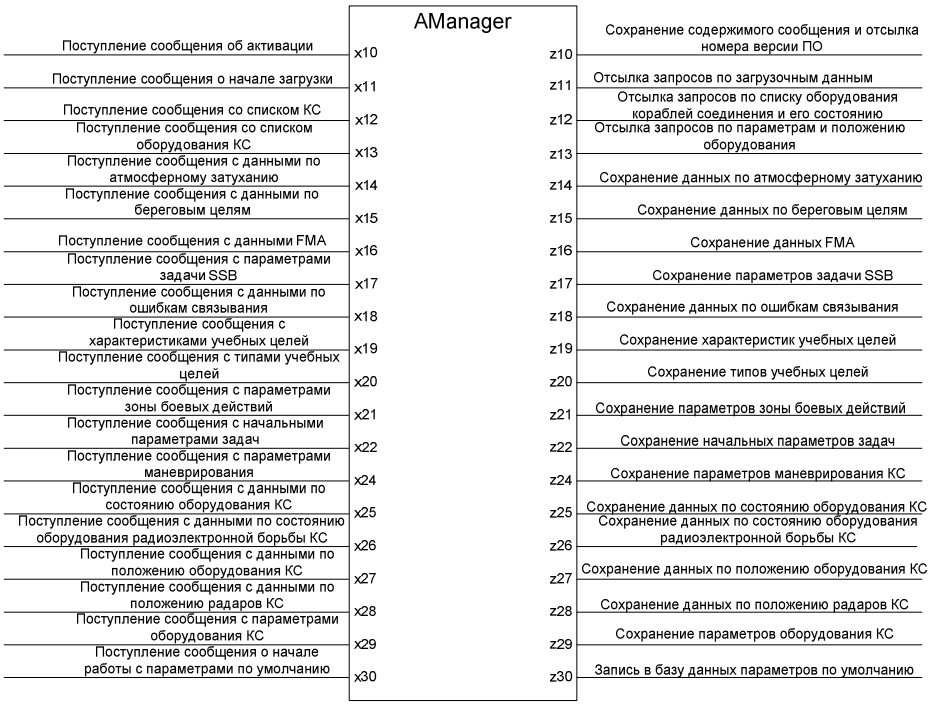

Рис. 3. Диаграмма связей автомата задачи начальной загрузки

Использование этого класса диаграмм, отсутствующих в *UML*, позволяет строить диаграммы состояний компактными и обозримыми, что позволяет человеку «охватить» одним взглядом достаточно сложные диаграммы состояний.

Диаграмма состояний автомата для задачи начальной загрузки модуля изображена на рис. 4.

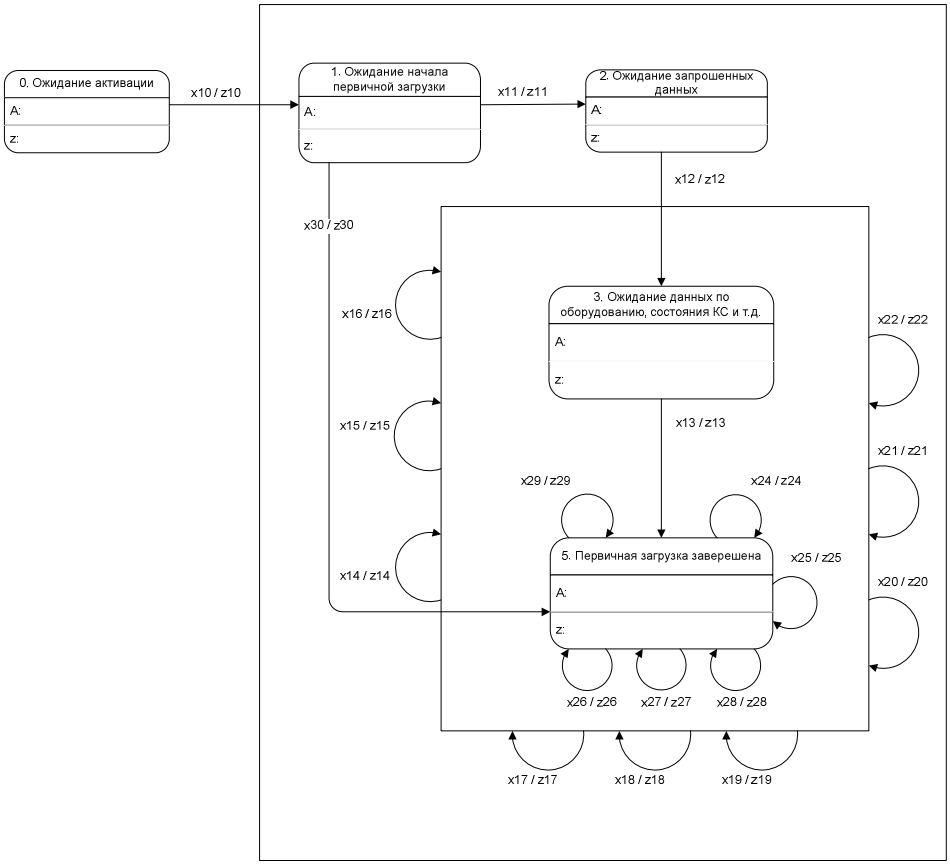

Рис. 4. Диаграмма состояний автомата задачи начальной загрузки

Достоинствами использования описанных диаграмм является то, что:

- поведение программы в этом случае понятно всем участникам проекта (архитектору, разработчику, тестеру и т. д.);
- они могут использоваться в проектной документации;
- по диаграммам состояний может быть изоморфно сгенерирован программный код на априори заданном языке программирования;
- все сценарии поведения задачи могут быть получены из одного автомата (системы взаимосвязанных автоматов), что уменьшает число используемых диаграмм и делает процесс разработки более простым и понятным;
- существенно облегчается процесс отладки и тестирования программного обеспечения благодаря возможности отслеживания состояний программы;
- применение диаграмм состояний для описания поведения программ позволяют повысить уровень автоматизации про-

цесса верификации программ по сравнению с другими подходами к программированию.

ВЫВОДЫ

Из рассмотренного примера следует, что применение для описания поведения программ диаграмм связей и состояний лишено недостатков, присущих его описанию с использованием только диаграмм, входящих в состав *UML*. При этом отметим, что в отличие от диаграмм последовательности, демонстрирующей, по сути, порядок вызовов методов экземпляров различных классов, автоматы позволяют описывать поведение объекта при различных сценариях.

Автомат управления, используемый в задаче начальной загрузки, был спроектирован, реализован и продемонстрирован во время проведения рабочей встречи с Заказчиком в августе 2009 года, который отметил эффективность использования диаграмм связей и состояний для описания поведения подобных систем.

СПИСОК ЛИТЕРАТУРЫ

1. *Harel D., Pnueli A.* On the development of reactive systems //In «Logic and Models of Concurrent Systems». NATO Advanced Study Institute on Logic and Models for Verification and Specification of Concurrent Systems. Springer Verlag. 1985, pp. 477–498.
2. *Harel D., Politi M.* Modelling Reactive Systems with Statecharts. NY: McGraw-Hill, 1998.
3. *Harel D.* Statecharts: A Visual Formalizm for Complex Systems //Science of Computer Programming. 1987. № 8, pp. 231–274.
4. *Harel D. et al.* STATEMATE: A Working Environment for the Development of Complex Reactive Systems //IEEE Transactions on Software Engineering. 1990. № 4, pp. 234 – 252. http://csdl.computer.org/comp/trans/ts/1990/04/e0403abs.htm
5. *Поликарпова Н. И., Шалыто А. А.* Автоматное программирование. СПб: Питер, 2009.