ПРОЕКТИРОВАНИЕ РАСПРЕДЕЛЕННЫХ СИСТЕМ В ГИБРИДНОЙ ОБЛАЧНОЙ ИНФРАСТРУКТУРЕ

О. И. Лукьянчиков
Московский государственный университет приборостроения и информатики

Сегодня наиболее распространенным является системное проектирование, где крупная система разбивается на части комплексы программ и компоненты. Это разделение происходит на основе функциональных назначений частей. Данный способ позволяет так же разделить и исполнителей, отдав каждую часть отдельным подразделениям или субподрядчикам. Однако преимущества системного проектирования на этом заканчиваются и особые проблемы уже возникают на этапе интеграции и отладке, поэтому данные этапы зачастую занимаю больше всего времени от разработки всей системы, а при разработке распределенных систем проблема интеграции и отладки возрастает в разы.

Структурное проектирование программных средств основано на модульном принципе[1]. Все трудности и проблемы, возникающие на этапах интеграции и тестирования, возникают из-за использования модульного подхода к проектированию программных средств. Каждый разработанный программный модуль может включаться в состав разных программ, если выполнены условия его использования, декларированные в документации по этому модулю. Таким образом, программный модуль может рассматриваться и как средство борьбы со сложностью программ, и как средство борьбы с дублированием в программировании (т.е. как средство накопления и многократного использования программистских знаний) [2]. Они сокращают время разработки программ, облегчают внедрение и обеспечивают гибкость. Однако всё взаимодействие модулей производится через их интерфейсы, не вдаваясь в подробности реализации. Однако, из-за специфичности реализаций интерфейсов, не все готовые реализации модулей удается внедрить в разрабатываемые программные средства, и поэтому требуется выполнять адаптацию под существующие модули или, «изобретая велосипед», разрабатывать аналоги существующих модулей. Связи с этим появись стандарты, такие как COM или CORBA, описывающие правила для интерфейсов.

Стандартизация в этом случае не решает в полном объеме проблему взаимодействия модулей. По этому, для решения данной проблемы необходимо изменить сам подход к разработке программных средств.

В идеи подхода проектирования программных средств лежит использование другой минимальной единицы актора, взамен модулю. Под актором будем понимать процесс, способный «общаться» подобными и выполняющую определенные функции, которые могут быть востребованы другими процессами. Данное определение не противоречит функциям акторов, описанных в актрорной модели Carl Hewitt [3], а именно создавать новые акторы, посылать свои сообщения, а также устанавливать, как следует реагировать на последующие сообщения.

При актроном подходе к построению программных средств вся система разбивается на акторы, тем самым образуя распределенную систему. У каждого актора определены сообщения, которые он может отправлять, и сообщения, которые он может принимать. Данные сообщения, по сути, являются входными и выходными данными для каждого актора. Каждый актор должен находиться на каком-нибудь вычислительном узле, который является адресом для обмена сообщения. Пример графического изображения системы из акторов изображен на рисунке 1.

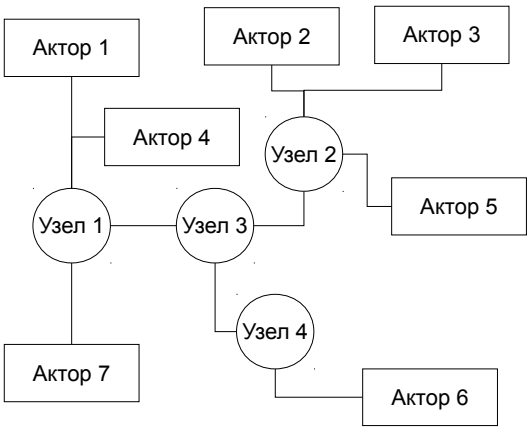

Рис. 1.

Применение акторного подхода к построению программных средств, позволит, во-первых, упростить процесс интеграции и поддержки, так как каждая программа является сервисом, предоставляющим свои услуги другим, отсылая и принимая некоторые сообщении. Требуется просто воспользоваться этими услугами, в отличие от модульного проектирования, где за весь импорт и экспорт отвечает интерфейс. Внесение изменений в него, что зачастую приходится делать в процессе интеграции и поддержки, влечет множество проблем, так как требуется вносить изменения в зависящие модули. Модули подгружались в процессе выполнения программы, размещаясь в том же адресном пространстве, что и основная программа. Независимость при этом довольно условная. Во-вторых, упростит процесс тестирования, так как каждый актор легко проверить по отдельности, послав ему сообщения и проверив какие сообщения он отправит в ответ. В третьих, возможность разделение задач на разных вычислительных узлах, увеличивая производительность за счет распараллеливания и распределения задач между вычислительными узлами.

Вычислительные узлы в системах различаются не только производительностью, но и размещением. Они могут находиться как в локальной сети так и в облаке, тем самым образуя гибридную облачную инфраструктуру, как показано на рисунке 2.

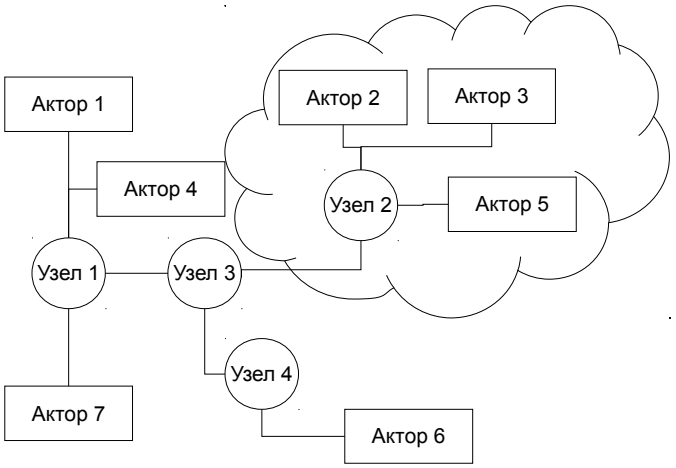

Рис. 2

В этом случае задача распределения акторов по вычислительным узлам становится нетривиальной, потому что в этом случае необходимо учитывать не только производительные возможности узла, но и пропускную способность каналов и количество клиентов, обращающихся к узлу. Для определения оптимального по производительности

размещения акторов необходимо проводить эксперименты по оценке разниц производительности в гибридных системах как в работах [4, 5]. Результаты данных экспериментов позволят на этапе проектирования системы определить лучшее размещение акторов или даже помогут разработать интеллектуальный модуль, отвечающий за миграцию акторов между узлами, с целью повышения производительности.

Литература

1. Липаев В.В. Прогрмммная инженерия. — Издательство «ТЕИС», 2006.
2. Бреслер И.Б., Семенов С.А., Корниенко В.В., Борисов В.В. Перспективный подход к организации программных комплексов // Радиопромышленность, 2009. — Вып. 1. — С. 72–88.
3. Hewitt C., Bishop P., Steiger R. A Universal Modular Actor Formalism for Artificial Intelligence. // IJCAI'73 Proceedings of the 3rd international joint conference on Artificial intelligence. — Morgan Kaufmann Publishers Inc., 1973. — P. 235–245.
4. Pluzhnik E.V., Nikulchev E.V. Use of Dynamical Systems Modeling to Hybrid Cloud Database // International Journal of Communications, Network and System Sciences, 2013.— V.6.— N. 12.— P. 505-512. (doi: 10.4236/ijcns.2013.612054)
5. Плужник Е. В., Никульчев Е. В., Паяин С. В. Лабораторный экспериментальный стенд облачных и сетевых технологий // Cloud of Science. 2014. Т. 1. № 1. С.78–87.