форматов данных. Сейчас планируется хранение конфигурации каждого устройства в виде структур данных в энергонезависимой памяти и изменение ее соответствующими командами протокола. Также в протоколе предусматривается описание каждого передаваемого элемента данных.

Заложенная в архитектуре устройства гибкость, конфигурируемость, и переменные состав и характеристики узлов требуют также строгого следования модульному подходу при проектировании программного обеспечения, например опираясь на рекомендации [5]. В частности, целесообразно выделять модули – своего рода "драйверы", реализующие программные модели компонентов и узлов и обеспечивающие за унифицированное взаимодействие с ними. Это облегчит управление версиями программного обеспечения либо оперативную настройку на текущую конфигурацию, что сокращает затраты на сопровождение системы.

В настоящее время работы продолжаются в направлении уточнения конфигурации аппаратных средств и проектирования программного обеспечения как устройства, так и управляющей ЭВМ.

Литература

1. ZigBee specification. – ZigBee Aliance, 2006
2. XBee/XBee-PRO OEM RF Modules. Product Manual v1.xAx – MaxStream Inc., 2006.
3. SLAU049F: MSP430x1xx Family. User's Guide. – Texas Instruments Inc., 2006.
4. IEEE 1451.4: Transducer Electronic Datasheet. – IEEE Standard Association, 2004
5. SPRA 701: A Software Modularity Strategy for Digital Control Systems. – Texas Instruments Inc., 2001.

РЕАЛИЗАЦИЯ СИСТЕМЫ КОНТРОЛЯ ИСПОЛЬЗОВАНИЯ ПАМЯТИ В РАМКАХ ВИРТУАЛЬНОЙ МАШИНЫ JAVA

С.П. Маюк, А.А. Сокольский
Беларусь, г. Минск

На сегодняшний день ни одна из коммерчески успешных операционных систем общего назначения не предоставляет должного механизма управления ресурсами для программ, класса серверов приложений масштаба предприятия. В силу этого, сервер приложений не может ограничивать выделение ресурсов контролируемыми приложениями, что делает невозможным обеспечение необходимого уровня робастности и управления.

Для реального решения подобных задач, еще с начала 60-х годов фактически используется только один метод: запуск сервера в рамках ограничивающей виртуальной среды, а не предоставление серверу необходимых механизмов контроля со стороны операционной системы. Среда создавалась либо через полную виртуализацию (виртуализация оборудования) операционной системы (например, IBM CP/CMS и z/VM, VMWare, VirtualPC - как ее современное раз-

витие), либо, как дальнейшее развитие идеи, через паравиртуализацию (виртуализация нижнего уровня операционной системы, например, IBM CP/CMS, Virtuozzo, Xen). Кроме такого очевидного недостатка, как высокие накладные расходы (даже в случае паравиртуализации), данное решение обладает излишней сложностью и размывает предметную область (для решения прикладных задач нужно принимать во внимание системные и административные аспекты). Идеологически решение работает на слишком высоком уровне гранулярности, что препятствует более точному контролю над использованием ресурсов, оптимизации и усложняет такие задачи как динамическая перестройка системы.

При разработке JEE платформы, изначально предназначенной для создания серверов масштабируемых до уровня предприятия, проблема контроля над ресурсами учтена не была. Фактически, производители JEE-контейнеров, для решения проблемы предлагают решения, аналогичные виртуализации операционной системы (в терминологии JEE – вертикальная кластеризация), и, для частичного улучшения производительности, совместное использование классов (фактически – паравиртуализация).

Последущие попытки решения данной проблемы (JSR 121 и проект SUN microsystems, Barcelona, как его близкая реализация), в рамках виртуальной машины, вводят системы ограничений на выделение произвольных ресурсов, требуют коренной переработки существующих JEE контейнеров. Таким образом, на сегодняшний день реализация каких – либо решений рассматриваемой проблемы отсутствует.

Однако, для решения широкого спектра прикладных задач возможно более практичное решение рассматриваемой проблемы: достаточно наложить ограничение только на один ключевой ресурс - память.

В силу существующей спецификации платформы Java, приложения, выполняющиеся в рамках виртуальной машины Java, не могут быть изолированы один от другого, в результате чего некоторые ресурсы, в частности память, необходимые для корректного выполнения данных приложений, не могут быть контролируемы средствами платформы. Поэтому, учитывая отсутствие контроля над использованием памяти средствами платформы Java EE, при совместной работе нескольких приложений в рамках одной виртуальной машины Java возникает проблема невозможности управления использованием памяти этими приложениями.

Рассмотрим сервер приложений JEE, на котором развернуто и выполняется приложение. Согласно существующей реализации платформы Java, приложения, выполняющиеся на сервере JEE, и сам сервер работают в рамках одной виртуальной машины Java. При работе клиентского приложения на сервере отсутствует возможность управления использованием памяти клиентским приложением со стороны сервера. В результате чего клиентское приложение может задействовать всю свободную память, что не только приведет к невозможности полноценной работы сервера приложений, но и невозможности корректного завершения работы сервера.

Для решения данной проблемы предлагается расширить сервис управления памятью в рамках виртуальной машины Java понятием качества сервиса управления памятью.

Качество сервиса (английский термин Quality of Service, сокращенно QoS) — понятие которое включает в себя набор качественных требований к рассматриваемому сервису. На этапе регистрации клиент может затребовать у QoS – сервиса определенное качество сервиса. Это требование может быть отвергнуто на этапе регистрации, но, в случае успешной регистрации, клиенту будет гарантированно предоставлено оговоренное качество сервиса.

Выделим два вида критериев:

- По гарантированному времени выделения памяти.
- По гарантированному объему выделения памяти.

В контексте рассматриваемой проблемы контроля управлением выделения памяти в рамках виртуальной машины Java наибольший интерес представляет качество сервиса выделения памяти с критерием по гарантированному объему.

На этапе регистрации контекст загрузчика классов может затребовать у QoS – сервиса выделения памяти определенное качество сервиса. Если это требование не будет отвергнуто на этапе регистрации, то данному загрузчику классов будет гарантированно предоставлен оговоренный объем памяти.

В рамках модели памяти Java, расширенной понятием качества сервиса выделения памяти, обобщим произвольный алгоритм сборки мусора понятием качества сервиса выделения памяти.

1. Для каждого контекста выделения памяти заводится счетчик выделенной памяти.
2. Маркируется заголовок каждого объекта текущим контекстом выделения. (Логическая модель объекта состоит из заголовка и тела объекта. Расширим заголовок объекта полем, содержащим идентификатор контекста выделения памяти для данного объекта.)
3. Процессу выделения требуемого объема памяти для объекта предшествует оценка возможности выделения требуемого объема для данного объекта, происходящая по следующему алгоритму (см. рис. 1):
   a. Оценивается возможность выделения требуемого объема памяти для нового объекта согласно следующего правила: объем суммарного значения счетчика выделенной памяти текущего контекста и размера нового объекта не должно превышать ограничение для текущего контекста.
   b. Если объем суммарного значения счетчика выделенной памяти текущего контекста и размера нового объекта превышает ограничение для текущего контекста, запускаем механизм полной сборки мусора.
   c. После финальной фазы сборки мусора, пользуясь маркерами контекстов выделения, пересчитываем точный суммарный размер объектов для каждого контекста.

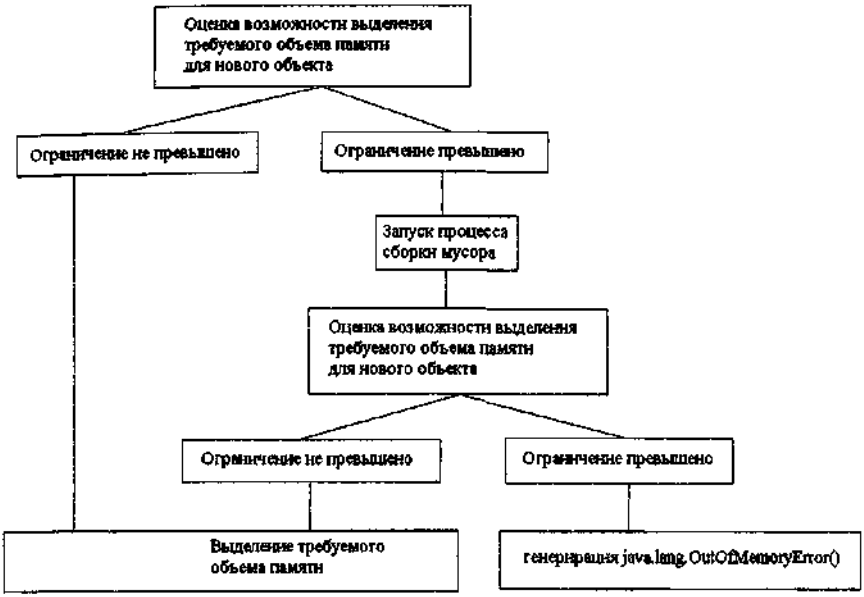

Рис. 1. Механизм обобщения произвольного алгоритма сборки мусора понятием качества сервиса выделения памяти

Повторяем процесс выделения требуемого объема памяти. Если выделение невозможно уведомляем об этом вышележащий уровень абстракци (в случае JVM – генерируем java.lang.OutOfMemoryError).

Заметим, что данный алгоритм контроля выделения памяти, является обобщением механизма работы произвольного сборщика мусора и независит от его конкретной реализации.

Особо следует отметить, что полученные результаты представляют интерес не только как решение частной задачи, но уже могут использовается в других прикладных областях и легко перенесен на другие платформы(.NET, real-time Java).

Литература

1. Jones R., Lins R. Garbage Collection: Algorithms for Automatic Dynamic Memory Management. – John Wiley & Sons, 1996. – 404 c.

2. Gosling J., Joy B., Steele G., Bracha G. The Java Language Specification. 3-rd edition. – Addison Wesley Professional, 2005. – 684c.

3. Java™ Virtual Machines. Sun Microsystems, Inc. http://java.sun.com/j2se/1.5.0/docs/guide/vm/index.html

4. Lindholm, Yellin. The Java Virtual Machine Specification, 1996 http://citeseer.ist.psu.edu/context/20581/0