СПИСОК ЛИТЕРАТУРЫ

1. Горно-экологический мониторинг на территории горного отвода Самотлорского месторождения нефти с учетом анализа выполненных работ при заложении наблюдательной геодезической сети. Трехгодичный цикл наблюдений 2005–2007 гг. Отчет за годовой цикл 2007 года. / Отчет по НИР. ЗСФ ИНГГ СО РАН. Тюмень. 2007. 156 с.

2. Филатов А. В. Обнаружение подвижек земной поверхности в зоне интенсивной нефтедобычи методами радарной интерферометрии // Вестник Югорского государственного университета. 2006. № 4. С. 103–109.

3. Евтюшкин А. В., Филатов А. В. Мониторинг сезонных деформаций земной поверхности методом радарной интерферометрии по данным ENVISAT\ASAR и ALOS\PALSAR // Обратные задачи и информационные технологии рационального природопользования: материалы IV Научно-практической конференции. Ханты-Мансийск: Полиграфист, 2008. 224 с. С. 195–201.

4. Евтюшкин А.В., Филатов А.В. Мониторинг деформаций земной поверхности методом радарной интерферометрии по данным ENVISAT\ASAR и ALOS\PALSAR // Контроль и реабилитация окружающей среды: Мат-лы симпоз. / Под общ. ред. М.В. Кабанова, А.А. Тихомирова. VI Международный симпозиум, Томск, 3-5 июля 2008 г. Томск: Аграф-Пресс, 2008. 384 с. С. 263–265.

5. Евтюшкин А. В., Филатов А. В. Интерферометрическая обработка радарных изображений ENVISAT, ALOS PALSAR и ERS-2 // Материалы одиннадцатой региональной конференции по математике «МАК-2008». Барнаул: Изд-во Алт. ун-та, 2008. 157 с. С. 45–47.

6. http://www.eorc.jaxa.jp/ALOS/obs/overview.htm

Суслов Н. В.

Формирование сетевой операционной системы для защищенного взаимодействия и распределенных вычислений

С момента зарождения компьютерных технологий программное обеспечение (ПО) развивается в соответствии с запросами аппаратной архитектуры, новых парадигм программирования, философских аспектов в области информационных технологий. Сам процесс развития ПО часто сводится к инкрементальному обновлению или модификации уже созданного программного кода путем наращивания функциональных возможностей и расширения области применения. Такое развитие объясняется необходимостью сохранения старых технологий и пользователей, сохранением затраченного технологами-программистами труда и средств работодателя, критикой несовершенности новых технологий. Но на самом деле эти объяснения представляют собой лишь оправдание внутреннего коренного несоответствия ранее созданного ПО новым технологиям, трудностям, связанным с переквалификацией разработчиков, коммерческими интересами и т. д. Преднамеренное лоббирование устаревших или персонализированных технологий в производственной и образовательной сферах ИТ лишает создателей ПО малейшей возможности качественной модернизации в соответствии с запросами современной науки и техники. Операционные системы являются на сегодня главными консервативными программными решениями, диктующими свои правила при создании прикладного программного обеспечения. В данной статье, на примере распределенного виртуального пространства для многопользовательского защищенного взаимодействия в реальном времени рассматриваются принципы и стратегии создания открытой операционной системы, а также возможные сценарии ее технического исполнения.

1. Виртуальное пространство для многопользовательского защищенного взаимодействия “Крестьянство” [1]

Цель проекта, поддержанного Российским фондом фундаментальных исследований, – создание виртуального пространства для многопользовательского защищенного взаимодействия, которое обеспечит синхронную работу над объектами определенной доменной области в реальном

времени автоматизированных автономных гетерогенных систем. В качестве таких систем могут выступать роботы, средства связи, компьютеры и другие вычислительные устройства.

Сформулируем основные принципы, которым должно отвечать создаваемое виртуальное пространство:

1. Синхронизация взаимодействий должна осуществляться с помощью единого моделируемого виртуального времени и механизмов репликации (Replication).
2. Безопасность взаимодействия должна быть на основе уровня возможностей и видимости объектов (Object-capability model).
3. Язык программирования и система в целом должны быть способны загружать, исполнять, изменять и поддерживать сами себя (Self-sustaining System).

Реализовать эти принципы в полной мере на существующих операционных системах невозможно. Поэтому требуется создание принципиально новой операционной системы. Методы создания также будут отличаться от существующих подходов. Для выхода на заявленные принципы используются промежуточные ступени (как при запуске космического корабля), которые после выполнения своих функций отбрасываются при переходе системы на новый качественный уровень. Рассмотрим более подробно каждый из принципов и ступени их формирования.

1.1. Моделирование виртуального времени и механизмы репликации(Replication)

Единицей репликации в виртуальном пространстве является Island (наподобии понятия Vat в языке программирования E). Island в объектно-ориентированном программировании можно рассматривать как мета-объект с расширенной моделью инкапсуляции, более защищенной, чем в традиционной модели. Каждый участник сетевого взаимодействия содержит реплику Island один в один совпадающую с другими участниками. Механизмы репликации должны строго гарантировать совпадение реплик по содержанию [2]. Рассмотрим механизм репликации более подробно (рис.1). Сообщения, посылаемые объектами, кроме собственного имени, аргументов, адресата включают в себя также штамп виртуального времени, когда это сообщение должно быть исполнено. Последовательность таких сообщений, идущих через маршрутизатор, можно рассматривать как глобальное виртуальное время для Island, так как эти сообщения являются внешними, атомическими, промаркированными счетчиком реального времени [3]. Маршрутизатор – единственный объект, который управляет сообщениями, сгенерированными вне Island. Он определяет, когда то или иное сообщение должно быть выполнено во времени, и отправляет его копии всем репликам Island через контроллер. Контроллер – это объект-интерфейс между определенной репликой и маршрутизатором. В его функции входит управление внешними сообщениями из реплики, перенаправление их маршрутизатору и обратно. Реплики Island имеют внутренние, готовые к выполнению очереди сообщений, управляемые контроллером, но они не могут быть выполнены без получения атомического сообщения от маршрутизатора, которое задает временную границу [4]. Принцип репликации может естественно реализовать мощь многоядерных и сетевых платформ.

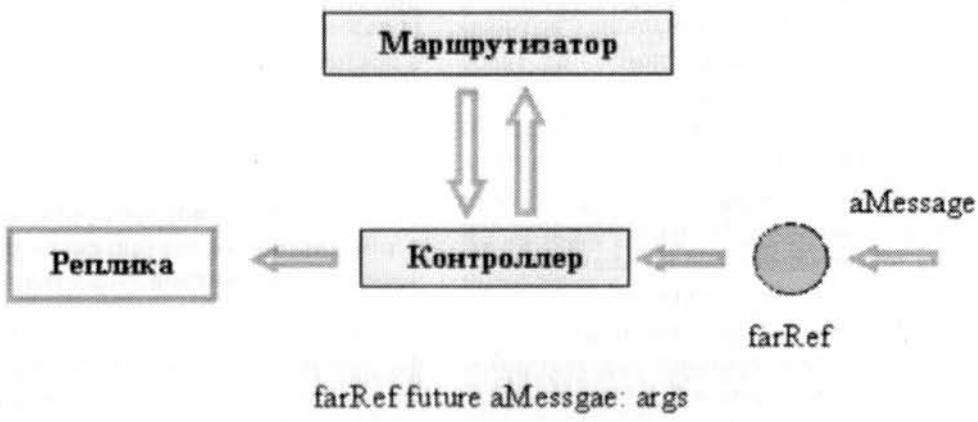

Рис. 1. Механизм репликации сообщения

1.2 Безопасность взаимодействия на основе уровня возможностей и видимости (Object-capability model)

Объекты внутри Island обладают теми же привилегиями взаимодействия, что и обычные объекты. Они могут посылать сообщения друг другу напрямую, поддерживать любые типы связей и т. д. Тем не менее эти объекты не могут посылать прямые сообщения другим за пределы видимости Island, так же как и в обратном напрвлении. Для того чтобы посылать сообщения за пределы видимости Island, имеется специальный объект FarRef, который существует вне Island и работает в качестве прокси-объекта, перенаправляя сообщения реальным объектам [5]. Именно такой механизм обеспечивает защиту взаимодействия объектов между различными Island, внешними интерфейсами. Важно отметить, что существующие системы защиты взаимодействия объектов базируются на принципах делегирования объектам прав доступа текущего пользователя. Проверка подлинности объектов осуществляется при помощи алгоритма шифрования открытым ключом при каждом атомарном взаимодействии. В данном же случае безопасность взаимодействия будет обеспечиваться областью видимости и возможностей объектов Island.

1.3. Язык программирования и система, способные загружать, исполнять, изменять и поддерживать сами себя (Self-sustaining System)

В такой системе программист, вычислительное устройство или конечный пользователь обладает полным контролем над всеми ее частями. Ни одна часть не является статичной, ни один аспект системы не является ранее связанным или жестко-определенным [6]. Программа, создающаяся для такой системы, становится лишь расширением ее исходной реализации. Система однородная, так что в ней нет искусственных различий между средой запуска, приложением или языком программирования. Таким образом, описанные выше возможности будут гарантировать созданной операционной системе или приложению возможность свершения коренного скачкообразного развития даже на этапе реальной эксплуатации. Единственное, что останется пока вне досягаемости для системы – это аппаратная платформа ("железо"). Данный принцип будет свидетельствовать о возможностях реализации национального или социокультурного аспекта в самопорождающемся языке программирования и операционной системе.

2. Техническая реализация ступеней проектирования

Рассмотрим сценарий, по которому ведется разработка ПО для создания распределенного виртуального пространства [7]. В качестве платформы используется виртуальная машина языка SmallTalk, диалект Tweak/Croquet/E-Squeak. Виртуальная машина доступна для множества аппаратных платформ, и концепция ее работы позволяет частично соответствовать вышеизложенным принципам в процессе разработки. Фактически виртуальная машина SmallTalk работает с динамическим образом операционной системы в оперативаной памяти, написанной полностью на языке SmallTalk. Общение с родительской операционной системой осуществляется посредством FFI запросов и примитивов, заранее откомпилированных библиотек. Виртуальная машина SmallTalk откомпилирована и не может модифицироваться в режиме работы. Тем не менее средств языка SmallTalk достаточно, чтобы определить в нем самом динамическую виртуальную машину и ее язык программирования, что позволяет вести эксперимент по заявленному третьему принципу о самопорождающихся системах. Осуществляется это с помощью объектно-ориентированного языка разметки языков OMeta [8]. Использование технологии SqueakNOS дает возможность загружать компьютер непосредственно с образа операционной системы на SmallTalk. Данные технологии позволили разработать прототип программного решения для распределенного виртуального пространства [7].

Для полного соответствия перечисленным принципам требуется перевод постоенного прототипа системы на объединенную объект-лямбда (COLA) или подобную (Self/Klein) архитектуру [6].

Одним из сценариев, например, по которому движется фирма Google и Sun Microsystems, является создание подобной операционной системы на основе виртуальной машины JavaScript, доступной сейчас практически на любой платформе через интернет-обозреватель. С использованием технологии Google Caja [9] (реализация концепций безопасности из языка E на JavaScript), языка OMeta на языке JavaScript и создается динамическая виртуальная машина Sun Lively Kernel [10]. Такая система, как безопасный JavaScript апплет, запускается без инсталляции в интернет-браузере и представляет собой полноценную платформу для ИТ экспериментов в реальном времени. Единственное, что еще остается нереализованным в

данном сценарии – это репликация. Важно что для полного соответствия перечисленным нами трем принципам потребуется также реализация виртуальной машины JavaScript на одной из самопорождающихся технологий.

Итак, приведем еще раз консервативные или устаревающие парадигмы существующих операционных систем:

– сетевое взаимодействие есть не что иное, как поочередное выполнение команд и передача данных между сетевыми узлами;

– вопросы безопасности решаются закрытием или открытием исходного программного кода и созданием программных средств защиты;

– операционная система является базовой недвижимой основой ("мертвым кодом" [7]) для создаваемого программного обеспечения.

Новые парадигмы:

– сетевое взаимодействие и выполнение команд происходит синхронно, обмен происходит скорее не данными, а исполняемым кодом и невозможен без виртуального времени;

– безопасность базируется на принципиально новой идее и механизме взаимодействия объектов (проверка "свой-чужой" при каждом взаимодействии), не зависит от закрытости или открытости программного кода и неуязвима для агрессивных атак;

– операционная система является "живым кодом" [7], динамически видоизменяется в ходе работы, стирая грань между собой и программным приложением, функционально обеспечивая лишь самопорождение любой его части.

Таким образом, происходит переосмысление самого понятия операционной системы, ее функций и назначений. Можно выделить минимальное ядро размером менее 20 Кбайт, которое содержит лишь реализацию механизма посылки атомарного, безопасного, с поддержкой виртуального времени сообщения, определение объекта, лямбда-вычислений и гарантирует самопорождение и развитие, определяемое функциями предметной области. Управление "железом" и прочие функции, приписываемые ранее операционным системам, порождаются в результате взаимодействия и не требуют жесткого определения в функционале ядра операционной системы, что принципиально важно, например, для создания нанороботов.

СПИСОК ЛИТЕРАТУРЫ

1. Проект Российского фонда фундаментальных исследований № 07-07-00332 Виртуальное обучающее пространство – "Крестьянство". 2007–2008 г. // Создание и развитие ИВТР для фундаментальных исследований.

2. **Суслов Н. В.** Репликация виртуального обучающего пространства как необходимое условие его существования. Принципы методологии "Крестьянство" // Вузовская наука региону. Мат. 5-й Всерос. НТК. В 2 т. Вологда: Изд-во ВоГТУ, 2007. Т.1. С. 357–359.

3. **David P.** Reed Naming and synchronizations in a decentralized computer system // Dissertation Massachusetts Institute of Technology. 1976.

4. **David A. Smith, Kay A., Raab A., David P. Reed.** Croquet – A Collaboration System Architecture. 2003.

5. **Miller M. S.** Robust Composition: Towards a Unified Approach to Access Control and Concurrency Control. // Dissertation. JHUniversity, 2006.

6. Ian Piumarta COLA white-paper // VPRI, 2005.

7. **Суслов Н. В.** Принципы построения виртуального обучающего пространства // Научно-технические ведомости СПбГПУ, №5 «Информатика. Телекоммуникации. Управление», Санкт-Петербург, 2008.

8. **Piumarta I., Warth A.** OMeta: an Object-Oriented Language for Pattern Matching // VPRI, 2007.

9. **Miller M. S., Samuel M., Laurie B., Awad I., Stay M.** Caja – Safe active content in sanitized JavaScript // Google technical report. 2008.

10. Taivalsaari A., Mikkonen T., Ingalls D., Palacz K. Web Browser as an Application Platform: The Lively Kernel Experience // Sun Microsystems technical report. 2008.