УДК 004.457

ПРОГРАММИРУЕМЫЙ АГЕНТ ВЗАИМОДЕЙСТВИЯ ПО ПРОТОКОЛУ ПЕРЕДАЧИ ГИПЕРТЕКСТА

© 2009 г. Ю. В. Власенко, А. В. Столяров

yuliavlasenko@intelib.org, avst@cs.msu.ru

Кафедра алгоритмических языков

1 Мотивация и постановка задачи

Комплекс сетевых служб, основанных на протоколе передачи гипертекста (HTTP), возникший в начале девяностых годов прошедшего века, за полтора десятилетия своего существования непрерывно и весьма интенсивно развивался; в своём современном виде комплекс используется для решения самых разнообразных информационно-технических задач, в том числе и таких, для которых протокол HTTP исходно не был предназначен.

Изначально HTTP задуман как протокол передачи информации, организованной в виде гипертекста и хранящейся на HTTP-серверах в виде наборов файлов, клиенту, представляющему собой пользовательскую программу визуализации гипертекста (браузер). В современных реалиях очень часто информация на сервере генерируется динамически непосредственно перед отправкой клиенту. Сама эта информация также может содержать не только гипертекст, но и информацию, представленную в практически произвольном формате — изображения, архивы, аудио- и видеофайлы и т.п. вплоть до бинарных исполняемых файлов. Многие гипертекстовые страницы включают в себя код на языках Java и JavaScript, предназначенный для выполнения в браузере клиента. Более того, часто на сайтах можно встретить файлы, предназначенные к исполнению специальными модулями расширения, встраиваемыми в браузеры[1].

Таким образом, как сервер, так и клиентская программа в нынешних реалиях оказываются скорее платформами для запуска программ, нежели традиционными средствами хранения и визуализации информации.

Часто возникают задачи, связанные с автоматизированным (без непосредственного участия человека) получением информации по протоколу HTTP. К таким задачам относятся, например, зеркалирование сайтов, автоматический поиск информации (data mining), построение баз данных поисковых серверов и т.п. Все эти случаи объединяются возникновением необходимости в клиентской программе, не предназначенной для непосредственной работы с пользователем; в частности, такой программе не нужна визуализирующая подсистема.

Для решения задач такого класса разработана масса программ, каждая из которых имеет весьма узкую область применения. Так, зачастую возможностей флагов командной строки программы *Wget* [8] не хватает для организации пакетной загрузки с сайта именно нужной пользователю информации; приходится либо загружать, наряду с нужными, много ненужных файлов, либо смиряться с необходимостью множества ручных запусков программы.

Это приводит к идее создания универсальной клиентской программы для пакетного взаимодействия по протоколу HTTP, поведение которой задавалось бы программой на встроенном алгоритмически полном языке программирования. Такая программа может быть полезна при решении практически любых задач, связанных с неинтерактивной (пакетной) загрузкой информации из Всемирной паутины.

В некоторых случаях, однако, для решения задачи может попросту не хватить одной только клиентской программы; так происходит, например, при недостатке понимания деталей взаимодействия между сервером и неким исполняемым кодом внутри браузера. Некоторые сайты

[1] Нельзя не отметить, что запуск любого исполняемого кода, полученного по сети, чреват серьёзными проблемами, связанными с безопасностью клиентской машины; практика показывает, что практически любой интерпретатор алгоритмически полного кода, сколь бы тщательно его авторы ни подходили к вопросам защиты, на поверку оказывается уязвим к взлому со стороны исполняемой им программы.

в сети Интернет намеренно организованы таким образом, чтобы затруднить получение информации с них иначе как в интерактивном режиме. Само по себе это вполне нормально, но в некоторых случаях владельцы сайтов вынуждают своих посетителей многократно скачивать одну и ту же информацию, весьма «тяжелую» по объему (чаще всего это видео- и аудиофайлы), порождая, таким образом, бессмысленную нагрузку на каналы передачи данных. Разобраться в происходящем можно, если «встроиться» во взаимодействие между браузером и сервером. Такое «встраивание» предусмотрено протоколом HTTP; соответствующие программы называются прокси-серверами. Прокси-сервер, получив запрос от клиентской программы, передаёт его (уже от своего имени) серверу, а полученный ответ, опять-таки, передаёт клиенту. Прокси-серверы используются во многих ситуациях, в частности, для кеширования, для защиты клиенских программ от нежелательной информации и т.п. Следует отметить, что и мощности конфигурационных файлов современных прокси-серверов часто не хватает для решения возникающих у пользователей специфических задач, и круг таких задач никоим образом не ограничивается анализом взаимодействия клиента и сервера.

Из вышесказанного можно сделать вывод о потенциальной полезности программы, способной играть роль прокси-сервера и управляемой, опять-таки, встроенным интерпретатором алгоритмически полного языка. Поскольку такая программа неизбежно должна уметь играть роль как сервера, так и клиента, возникает логичная идея объединить в одной программе как возможности программируемого клиента, упоминавшегося выше, так и возможности программируемого прокси-сервера. Поскольку с протокольной точки зрения прокси-сервер почти не отличается от обычного HTTP-сервера, в программе имеет смысл предусмотреть возможность использования её и в качестве конечного HTTP-сервера, поскольку это технически достигается сравнительно просто, а по своим возможностям может оказаться полезным (хотя бы для организации управления работой уже запущенной программы через браузер пользователя).

В рассматриваемой реализации роль встроенного интерпретатора играл интерактивный интерпретатор усечённого диалекта языка Лисп, входящий в библиотеку *InteLib* [1]. Это позволило использовать в качестве основного языка реализации язык C++.

2 Структурные элементы веб-агента и их свойства

2.1 Диспетчер взаимодействий

Центральный объект, осуществляющий управление всеми процессами в системе, будем называть диспетчером взаимодействий. Диспетчер взаимодействий осуществляет непосредственное управление клиентскими сессиями и слушающими серверами, отвечает за переключение задач клиентских сессий и за буферизацию данных об общем состоянии системы. Также в функции диспетчера входит приём запросов на вычисление от анализатора пользовательского ввода.

2.2 Клиентские сессии

В системе веб-агента различаются два типа клиентских сессий веб-агента: стандартные сессии и сессии с явным управлением функциями обратного вызова. В первом случае клиент получает задачу на загрузку страницы и выполняет эту задачу асинхронно относительно основного потока вычислений управляющей программы. При этом загруженные данные целиком сохраняются на клиенте вплоть до его удаления. По окончании выполнения задания такой клиент инициирует событие, сигнализирующее о готовности данных. Это событие буферизуется диспетчером до явного запроса события управляющей программой. После этого пользовательская программа может запросить у клиентской сессии загруженное HTTP-сообщение или отдельную его часть. Поскольку стандартные клиентские сессии сохраняют загруженные данные целиком, их не следует использовать для загрузки заведомо больших ресурсов.

Для более гибкого управления предусмотрен еще один тип клиентских сессий — это сессии, управляемые функциями обратного вызова. Такой клиентской сессии, кроме адреса загружаемого ресурса, передается функция для обработки получаемых данных. Эта функция будет вызываться веб-агентом каждый раз при получении клиентом порции данных. Значение, возвращаемое функцией обратного вызова, игнорируется, поэтому смысл самого вызова

заключается только в побочном эффекте. В качестве параметров функции обратного вызова используются объект клиентской сессии и полученная порция данных в виде строки или в виде неструктурированных данных (в последнем случае используется тип `SExpressionRawBuffer` библиотеки *InteLib*). Возможность передачи функции обратного вызова других параметров не предусмотрена, так как все необходимые значения можно поместить в лексический контекст функции, который будет доступен в момент вычисления. Клиентские сессии, управляемые функциями обратного вызова, по умолчанию сохраняют все получаемые данные, поэтому для них также определены функции получения загруженного HTTP-сообщения в структурированном виде. Однако предусмотрена возможность явного сброса буферов клиентской сессии управляющей программой. После вызова этой функции запросы управляющей программой целого HTTP-сообщения будут возвращать ошибку, но доступ к данным заголовка полученного сообщения по-прежнему будет возможен.

Очевидно, что возможности управляемых клиентских сессий включают в себя возможности стандартных сессий: стандартная сессия работает так, как управляемая сессия с «пустой» функцией обратного вызова. Стандартные сессии были введены для оптимизированного выполнения тривиальных задач клиентских сессий и для упрощения работы пользователя с такими задачами.

Ниже приведены простейшие примеры использования стандартных и управляемых клиентских сессий. В обоих случаях результатом программы является загрузка страницы и вывод ответа сервера (вместе с заголовочной частью) на экран.

В случае стандартного клиента управляющая программа получает доступ к данным только по окончании загрузки:

```
(let
    (
         (cl (create-client-by-url "http://someserver/shorttext.txt"))
    )
    (wait-for-event cl) ; блокируется только данный вычислительный поток
    (princ (get-raw-response cl))
    (remove-client cl)
)
```

В случае управляемого клиента можно выводить данные по частям, сразу при получении:

```
(let
    (
        (cl (create-specific-client-by-url
                "http://someserver/longtext.txt"
                #'(lambda (session data) (princ data))
                   ; функция будет вызываться при получении
                   ; каждой порции данных
            )
        )
    )
    (wait-for-event cl) ; блокируется только данный вычислительный поток
    (remove-client cl)
)
```

Для клиентских сессий обоих типов предусмотрена возможность явного задания управляющей программой HTTP-запроса.

2.3 Слушающие сервера

Еще одна часть функциональности веб-агента реализуется слушающими серверами. С каждым слушающим сервером веб-агента связывается функция языка Лисп. Слушающие сервера веб-агента принимают HTTP-запросы от внешних клиентов и запускают вычисление соответствующей Лисп-функции. При этом в случае получения заведомо некорректного запроса или при возникновении ошибки вычислений информирование клиента об ошибке становится ответственностью самого сервера, а не управляющей функции. В этих случаях по умолчанию сервер возвращает следующие коды ошибок:

- `400 - "Bad Request"`, если формат запроса не соответствует протоколу HTTP;
- `415 - "Unsupported Media Type"`, если содержимое запроса типа `POST` представляет собой бинарные данные;
- `505 - "HTTP Version Not Supported"`, если запрос соответствует версии протокола, отличной от HTTP/1.1;
- `500 - "Internal Server Error"`, если при вычислении функции произошла ошибка: исключительная ситуация библиотеки *InteLib* или веб-агента.

В системе предусмотрена возможность изменения логики действий слушающего сервера в случае возникновения таких ошибок. Для этого управляющая программа может определить для слушающего сервера функции обработки ошибок.

Функция, связываемая с сервером при его создании, должна принимать два аргумента. Это требование обосновано соглашением о передаче параметров: функции, вычисляемые слушающими серверами, в качестве параметров получают объект соответствующей сессии и HTTP-запрос. Даже если связываемая с сервером функция не использует эти значения, она должна принимать на вход соответствующие параметры.

2.4 События и ошибки в системе веб-агента

Помимо задачи перенаправления вызовов клиентам и серверам, в функции диспетчера также входит управление очередями событий веб-агента и передачей управляющей программе информации о возникающих ошибках. События и ошибки веб-клиента содержательно бывают очень похожи, но они имеют разное происхождение, на котором следует остановиться подробнее. События веб-агента — это объекты, сохраняющие информацию об изменениях в системе, которые могут быть интересны управляющей программе и которые произошли *асинхронно* относительно обращений пользовательской программы к веб-агенту. Событиями в системе могут быть, например, окончание загрузки требуемой страницы клиентом, или возникновение некорректных ситуаций при работе клиента или сервера. Ошибки же, напротив, всегда появляются *синхронно* с обращениями управляющей программы к веб-агенту и возвращаются в ответ на эти обращения. Ошибка может возникнуть, например, при попытке прикладной программы обратиться к клиенту, который был удален. Для событий веб-агента реализован механизм буферизации, для ошибок необходимости в буферизации нет.

2.5 HTTP-сообщения

В системе веб-агента предусмотрены средства для структурированного представления сообщений протокола HTTP. Как на клиентской, так и на серверной стороне производится проверка корректности формата получаемых сообщений и разбор сообщений на структурные составляющие. В языке управления веб-агентом предусмотрена возможность получения от клиентской сессии отдельных структурных составляющих сообщения — статуса, заголовочной части или отдельного поля заголовка, тела сообщения. При этом получение данных из заголовка возможно до окончания загрузки тела сообщения.

2.6 Модуль вычислений

Важнейшей частью веб-агента является модуль вычислений. С точки зрения диспетчера этот модуль является абстрактным вычислителем, и набор его функций не зависит от природы проводимых вычислений. Конкретная реализация этого модуля как Лисп-вычислителя использует механизмы, предоставляемые библиотекой *InteLib*.

Один и тот же модуль вычислений используется как для основного потока вычислений, так и для вычислений, связанных с запросами к слушающим серверам. При этом все вычисления ведутся в едином глобальном контексте.

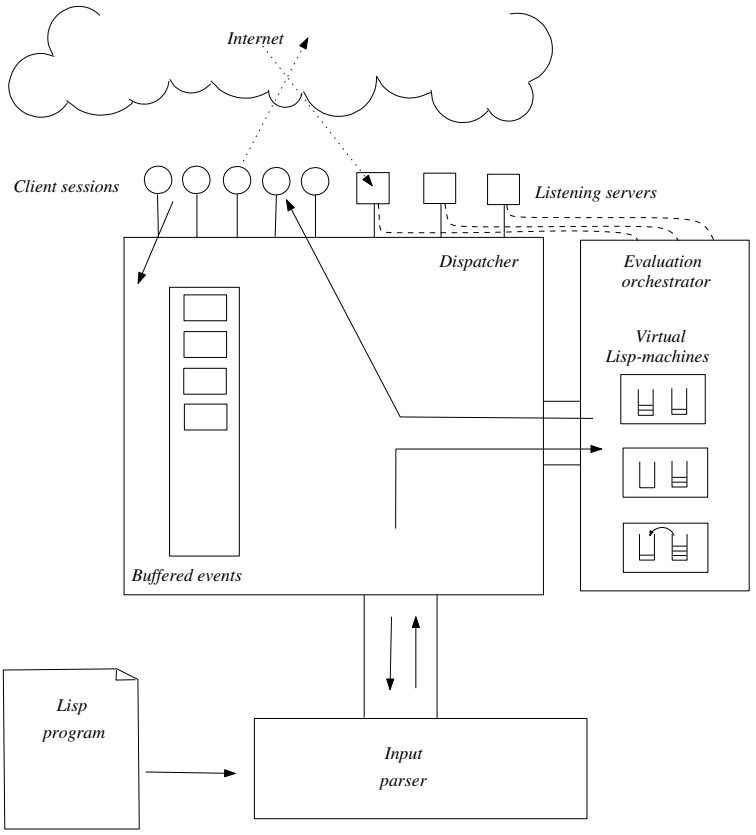

Рис. 1: Структурное представление веб-агента

2.7 Анализатор пользовательского ввода

Идея, на которой основан принцип функционирования веб-агента, подразумевает, что задержка поступления данных от анализатора пользовательского ввода не должна блокировать выполнение текущих задач веб-агента. Это условие достигается путем выделения анализатора ввода интерактивного интерпретатора в отдельный процесс. В таком случае в ответственность анализатора пользовательского ввода входит только поддержка возможностей редактирования вводимой строки, выделение в потоке ввода форм языка Лисп и передача полученных данных процессу, в котором работает диспетчер. Перевод же данных в структуры, передаваемые Лисп-машине, происходит уже в процессе диспетчера. При реализации этой возможности использовались средства библиотеки *InteLib* для интерактивного ввода программы на Лиспе, которые в свою очередь используют API библиотеки *GNU Realine*.

3 Логическая структура системы

Рассмотрим структурные связи между компонентами веб-агента. Центральным элементом системы веб-агента является диспетчер взаимодействий; именно он управляет информационными потоками между пользовательской программой и клиентскими сессиями и серверами, а также хранит информацию о происходящих событиях. У всех слушающих серверов есть прямые ссылки на модуль вычислений, поскольку процедура обращения сервера к вычислителю проста, и перегружать диспетчер этой задачей необходимости нет. Аналогичные ссылки могут быть и у клиентских сессий, если для них явно заданы функции реакции на приход данных. Схема логической структуры веб-агента показана на рисунке 1. В целом такая схема зависимостей соответствует общепринятым канонам объектно-ориентированного проектирования [4].

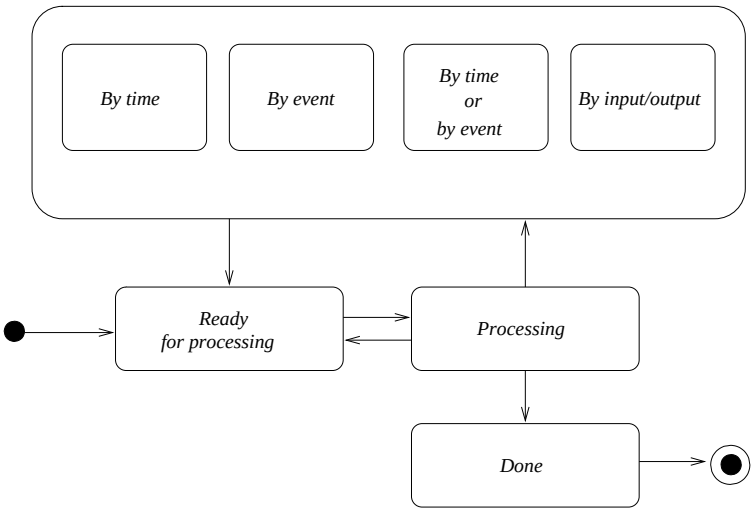

Рис. 2: Диаграмма состояний вычислительных потоков веб-агента

4 Принципы функционирования веб-агента

4.1 Потоки вычислений в системе веб-агента

Функционирование клиентских сессий и слушающих серверов происходит асинхронно относительно друг друга и относительно работы анализатора пользовательского ввода. При этом в системе одновременно сосуществуют несколько потоков Лисп-вычислений, выполняющихся псевдопараллельно. Можно выделить четыре типа вычислительных потоков:

- основной поток управляющей программы;
- потоки вычислений слушающих серверов;
- потоки вычислений функций реакции на приход данных клиентских сессий;
- потоки обработки ошибок слушающих серверов.

Для каждого из этих потоков определен набор состояний, в которых может находиться поток. Диаграмма состояний вычислительных потоков приведена на рисунке 2. Стрелками на диаграмме обозначены возможные переходы между состояниями. Отметим, что в состоянии вычисления в каждый момент времени может находиться не более чем один вычислительный поток.

4.2 Переключение вычислительных потоков

Переключение вычислительных потоков осуществляется в трех случаях:

- По окончании вычисления потока. Сразу по окончании вычисления поток удаляется.
- При вызове функций, переводящих поток в состояние блокировки. Это могут быть функции ожидания события, функции блокировки на заданный интервал времени или функции ввода-вывода.
- При превышении потоком количества шагов виртуальной машины, разрешенного для непрерывного вычисления. Это сделано во избежание дискриминации вычислительных потоков.

Дескрипторы сокетов и потоков ввода/вывода контролируются системным вызовом `select` в цикле работы диспетчера взаимодействий. На каждой итерации цикла диспетчер обрабатывает активизацию дескрипторов и передает модулю вычислений данные об изменении состояния системы. Это могут быть события, сигнализирующие об окончании загрузки данных клиентскими сессиями или о произошедших ошибках, а также активизация дескрипторов ожидающих потоков ввода/вывода. На основании этих данных модуль вычислений разблокирует соответствующие вычислительные потоки.

События на дескрипторах, контролируемых вызовом `select`, могут также вызывать создание новых вычислительных потоков. Это происходит, например, в случае окончания получения запроса слушающим сервером, когда запускается вычисление соответствующей серверу функции.

После обработки активизации дескрипторов, диспетчер вызывает метод `PerformStep` объекта модуля вычислений, запуская тем самым очередной шаг вычислений для готовых вычислительных потоков. Логика выполнения шага вычислений для каждого отдельного вычислительного потока зависит от конкретной реализации вычислительных потоков. В системе веб-агента, управляемого на языке Лисп, выполнение вычислительных потоков осуществляется виртуальными Лисп-машинами (более подробно см. в подразделе 4.3).

После выполнения вычислений диспетчер запрашивает у модуля вычислений информацию, необходимую для определения максимального времени ожидания вызова `select`. В случае если существуют вычислительные потоки, готовые к выполнению, это время устанавливается в ноль. Если готовых вычислительных потоков нет, но есть потоки, которые могут быть разблокированы по времени, передается минимальное время ожидания для таких потоков. Если вычисления по всем потокам завершены или заблокированы по событию, время ожидания вызова `select` не ограничивается.

4.3 Многопоточный Лисп-вычислитель

В реализации абстрактного модуля вычислений как многопоточного Лисп-вычислителя отдельным потокам соответствуют разные виртуальные Лисп-машины, работающие в едином глобальном вычислительном контексте. Эти Лисп-машины представляют собой объекты класса `WeblispContinuation`, наследуемого, с одной стороны, от класса `LispContinuation` библиотеки *InteLib*, а с другой — от абстрактного класса `VirtualMachine`, представляющего вычислительный поток. Таким образом, объекты этого класса способны производить вычисления в соответствии с моделью языка Лисп, и блокировать вычисления в соответствии с логикой модуля вычислений веб-агента.

Лисп-вычислитель библиотеки *InteLib* представляет собой виртуальную вычислительную машину, в которой вычисление формы языка Лисп разделяется на отдельные атомарные шаги. При этом вычисление может быть прервано после любого из таких атомарных шагов и в дальнейшем возобновлено без потери уже вычисленных данных. Это свойство позволяет программе переходить в состояние отложенных вычислений (так называемые *continuations* [5]). Возможность прерывания Лисп-вычислений используется и виртуальными Лисп-машинами веб-агента для ограничения количества шагов, которые может выполнить в непрерывном режиме один вычислительный поток.

Идея многопоточного Лисп-вычислителя привносит в реализацию дополнительные сложности, связанные с возможными взаимдействиями вычислительных потоков. В частности, в случае если несколько вычислительных потоков используют и модифицируют значение некоторой глобальной переменной, может возникнуть необходимость в атомарности таких операций чтения/записи. Для корректной обработки такого рода ситуаций необходимо предусмотреть механизм, позволяющий пользовательской программе устанавливать запрет переключения вычислительных потоков в некоторой секции. В нынешней реализации эта задача пока не решена.

5 Поддержка протокола HTTP

В рамках поддержки веб-агентом протокола HTTP версии 1.1 (стандарт RFC 2616 — [6]) при работе с HTTP-сообщениями поддерживаются следующие возможности:

- разделение сообщения на начальную строку, заголовок и содержимое путем выделения парных символов CR и LF[2] (согласно формату сообщений HTTP/1.1: см. [6], п. 4.1);
- обаботка ведущих и конечных пробелов в значениях заголовков, обработка переносов строк отдельного поля заголовка согласно [6], п. 4.2);
- определение длины сообщения по значению поля `Content-Length` в заголовке или на основе кодировки `"Transfer-Encoding: chunked"` ([6], п. 3.6.1); если эти поля не заданы, предполагается разрыв соединения сервером по окончании передачи сообщения;
- требование обязательного наличия в заголовке поля `Host` во всех запросах к слушающим серверам (согласно [6], п. 14.23).

Веб-агент позволяет пользовательской программе как явно задавать запросы, посылаемые клиентскими сессиями внешним серверам, так и использовать запросы стандартного вида. В заголовке запроса стандартного вида задаются поля `Host`, `Content-Length` и `User-Agent` (последний — со значением `Weblisp/0.1`).

Ответы слушающих серверов на запросы внешних клиентов имеют в заголовке следующие поля: `Content-Type`, `Content-Length`, `Date`, `Last-Modified` (с датой последнего обращения к Лисп-вычислителю), `Connection` (со значением `close`, так как сервер разрывает связь после ответа).

6 Задача перехвата видео- и аудиопотоков

Задача оптимизации Интернет-трафика при повторном обращении к ресурсам решается всеми современными браузерами путем кэширования веб-страниц, но гибкая настройка такого кэширования для конкретных задач на уровне браузера невозможна. Например, если существует потребность многократного просмотра некоторого видеофайла, получаемого из сети, то естественно сохранить этот файл на прокси-сервере вместе с HTTP-заголовком и при повторном обращении не запрашивать файл из сети, а доставать его из репозитория.

Такая задача естественным образом решается с помощью веб-агента. В качестве демонстрационного примера была написана программа на языке Лисп для управления работой прокси-сервера, основной задачей которого является организация репозитория видеофайлов, полученных с сайта *www.youtube.com*. В этой программе прокси-сервер анализирует запрос клиента и выполняет следующие действия:

- Если браузер осуществляет запрос по адресу *http://__repository/*, то в браузер передается список видеофайлов, сохраненных в репозитории.
- Если целью запроса не является видеофайл с сайта *www.youtube.com*, прокси-сервер просто передаёт запрос клиента серверу и возвращает клиенту получаемый ответ.
- Если целью запроса является видеофайл с сайта *www.youtube.com*, и данный файл есть в репозитории, клиенту возвращается HTTP-заголовок и содержимое видеофайла, найденное в репозитории.
- Если целью запроса является видеофайл с сайта *www.youtube.com*, и данного файла нет в репозитории, прокси-сервер передает запрос клиента на сервер, а получаемый ответ передает клиенту и сохраняет в репозитории.

[2] US-ASCII CR, carriage return (13) и US-ASCII LF, linefeed (10)

7 Перспективы дальнейшей работы

В качестве ближайших перспектив описываемой разработки можно выделить оптимизацию существующего аппарата управления потоками Лисп-вычислений, а также разработку минимальных средств взаимодействия для вычислительных потоков. Также для корректной работы веб-агента на сложных сценариях с совместным использованием файлов и перекрестными блокировками вычислительных потоков потребуется разработка механизма разрешения тупиков.

Для упрощения программирования веб-агента имеет смысл снабдить его набором базовых функций работы с документами в форматах HTML/XML, в частности, функциями выбора ссылок и представления XML-деревьев в виде S-выражений. В нынешней реализации всё это остается задачей пользовательской программы.

Список литературы

[1] И.Г. Головин, А.В. Столяров. Объектно-ориентированный подход к мультипарадигмальному программированию // Вестник Московского Университета, сер. 15 вычисл. матем. и киберн. 2002. N 1. С. 46–50.

[2] Хювёнен Э., Сеппянен Й. Мир Лиспа. В 2-х т. Т. 1: Введение в язык Лисп и функциональное программирование. М.: Мир, 1990. 458 с.

[3] W. Richard Stevens. UNIX Network Programming, Volume 1, Second Edition: Networking APIs: Sockets and XTI, Prentice Hall, 1998. 1240 p.

[4] Г. Буч. Объектно-ориентированный анализ и проектирование с примерами приложений на C++, второе изд. М.: Бином, СПб.: Невский диалект, 1998. 560 с.

[5] John C. Reynolds. The discoveries of continuations // Lisp and Symbolic Computation, 1993. N 6(3-4). P. 233–248.

[6] Hypertext Transfer Protocol – HTTP/1.1. Official Internet Protocol Standards: Standards Track: RFC 2616.

[7] The InteLib home page [HTML] (`http://www.intelib.org/`).

[8] GNU Wget Official Site [HTML] (`http://www.gnu.org/software/wget/`).