

ASSIGNMENT 3

Domain Analysis and System Design, 1st Iteration

apt-got

<http://www.apt-got.com/>

- E01 -

TA: Ted Huffmire

Jamie Fitz-Gerald	jamiefitzgerald@umail.ucsb.edu
Jonas Jägerhök	jonas_jagerhok@umail.ucsb.edu
Tobias Hertkorn	t_hertkorn@umail.ucsb.edu

PROJECT OUTLINE

Packages from the Debian-Distribution are obtained from public http-mirrors. To save bandwidth our program will function like a drop-in, stand-alone proxy for the internal network. But in addition it will store already requested packages locally. Packages that are not yet stored locally are fetched upon client-request from the parent server transparently.

VISION

Problem Statement

To write a program that improves the performance of the Linux Debian package update and install application named apt-get.

Key high-level goals

- Reduce bandwidth usage
- Reduce download time

System features

- Handle client request
- Respond to request from client
- Request files from remote server
- Handle server responses (e.g. file not found – 404 etc)
- Cache files locally
- Respect link structure on remote Debian Server
- If multiple clients request a specific file simultaneously, download only once from remote server
- Handle disk usage (purge old files)

Other requirements and constraints

- Watch out for “full disk”
 - Prevent “denial of service” attacks by limiting number of threads
 - Prevent other files than those in the mirror from being read.
-

USE CASE

Provide file

Main Actor:

Client wants to have a specific file. Requests it via internet (http-protocol).

Supporting Actor:

Remote Debian Server holds all available files in a directory structure. Files can be accessed via internet (http-protocol).

Basic Flow (or main success scenario):

1. Client requests file from apt-got server.
2. apt-got searches local cache for file. (Does not find it)
3. apt-got requests file from remote Debian Server.
4. Remote Debian Server sends file to apt-got.
5. apt-got stores file locally.
6. apt-got sends file to Client.

Extensions (or alternative flow):

*a. At any time – System fails.

1. apt-got signals error.
2. Send “Internal Server error” to Client.
 - 2a. No connection to Client
 1. Do nothing.

3-6a. file already stored in local cache.

1. apt-got sends local file to Client.

3-6b. No connection to server

1. apt-got sends “file not found” to Client.

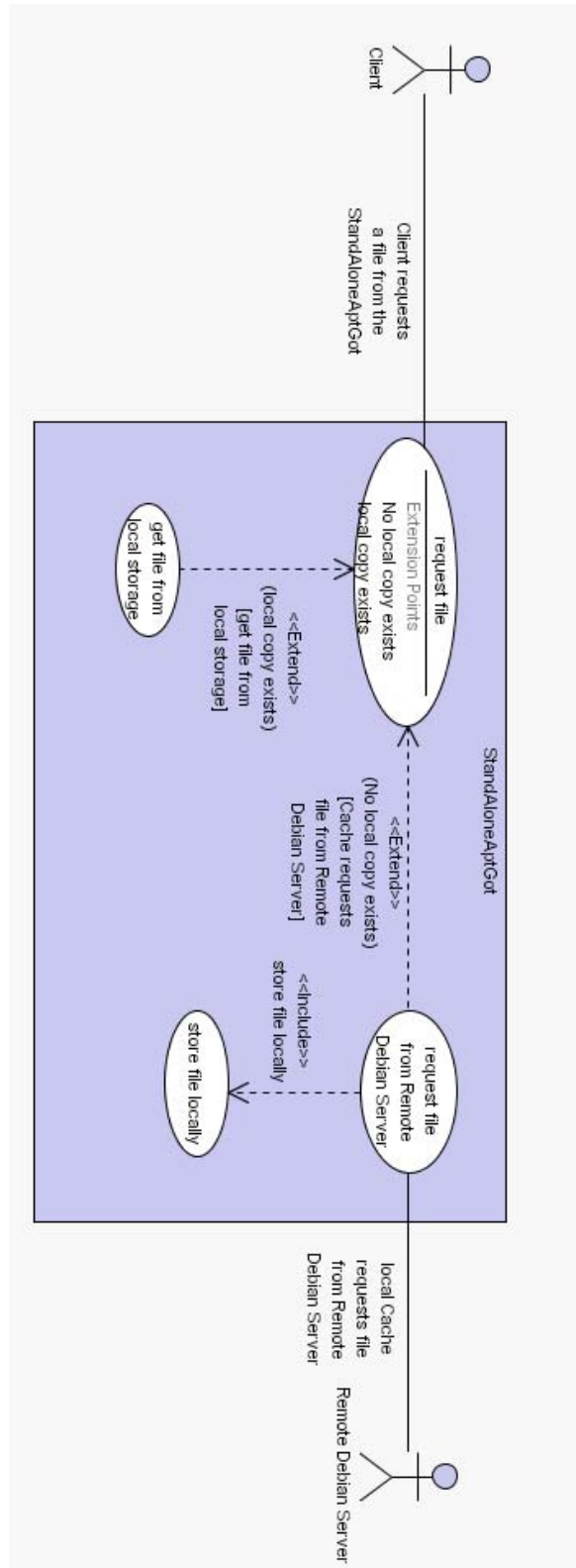
4-6c. File not Found

1. Remote Debian Server sends “file not found” to apt-got.
2. apt-got sends “file not found” to Client.

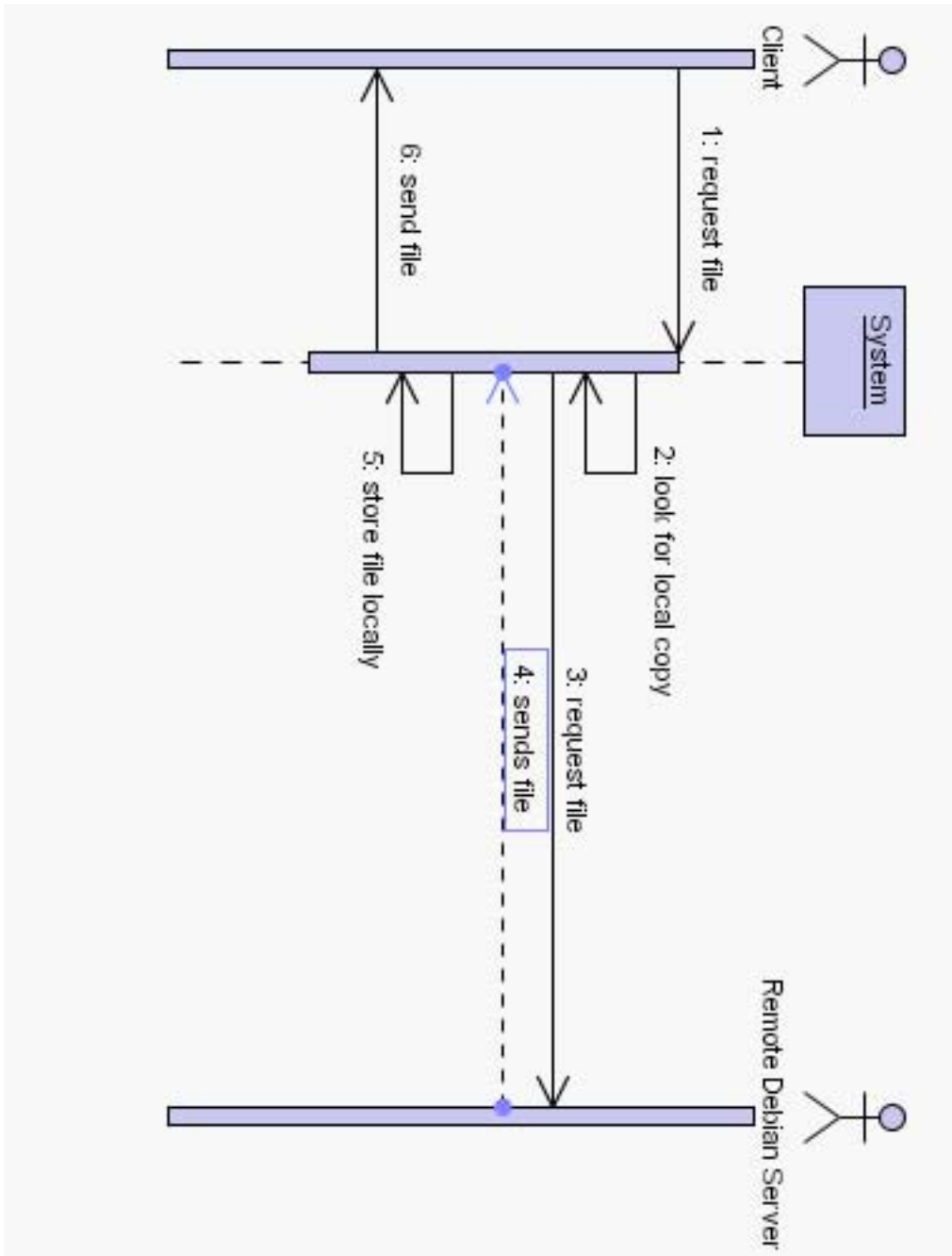
6d. No connection to client

1. Do nothing
-

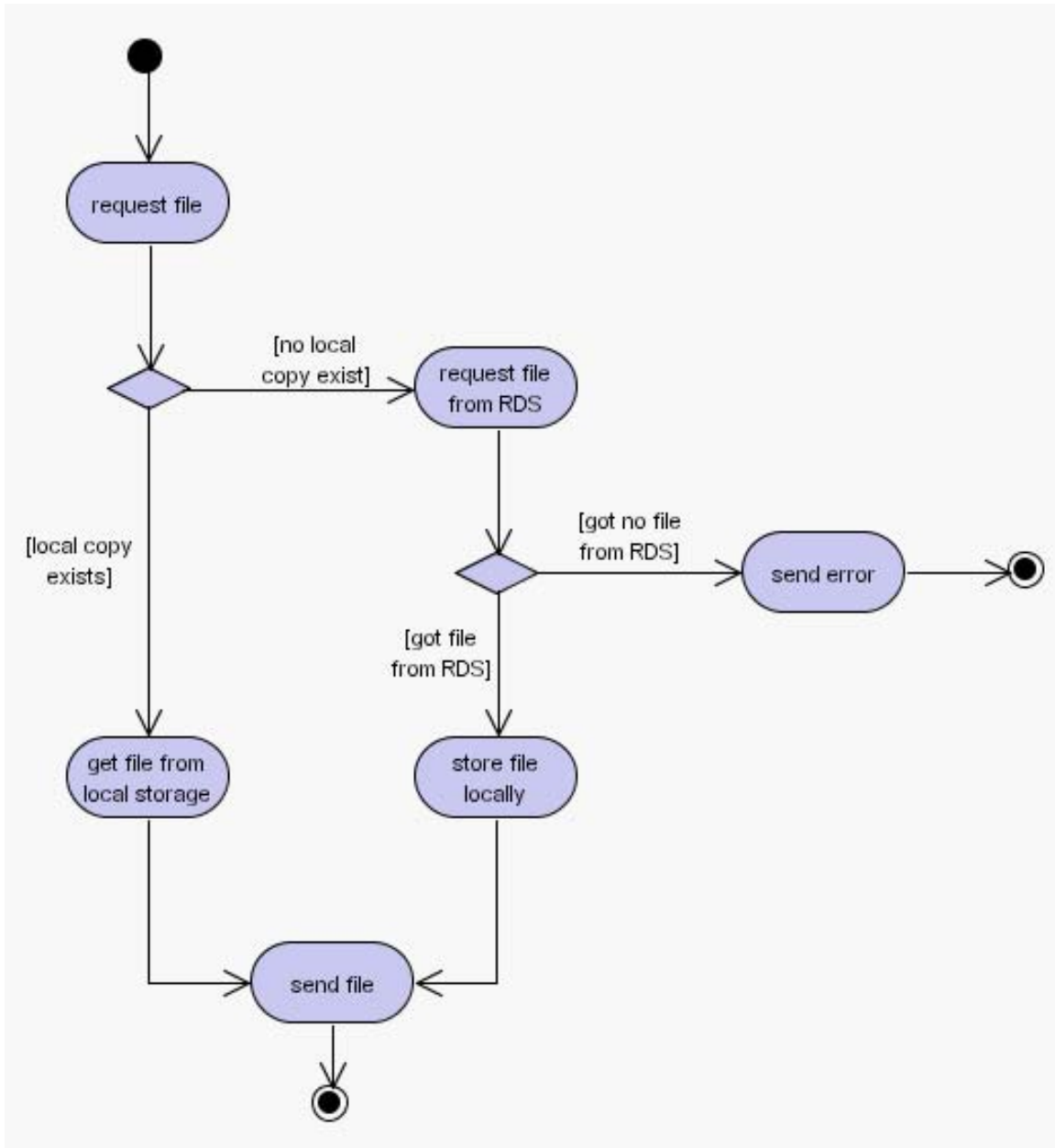
USE CASE DIAGRAM



SYSTEM SEQUENCE DIAGRAM



ACTIVITY DIAGRAM



SYSTEM OPERATION CONTRACTS

Operation	requestFile(fileName: String)
Cross Reference	Request File
Preconditions	Connection to client is established
Postconditions	A http request header from the client was received and the requested file was identified Response received

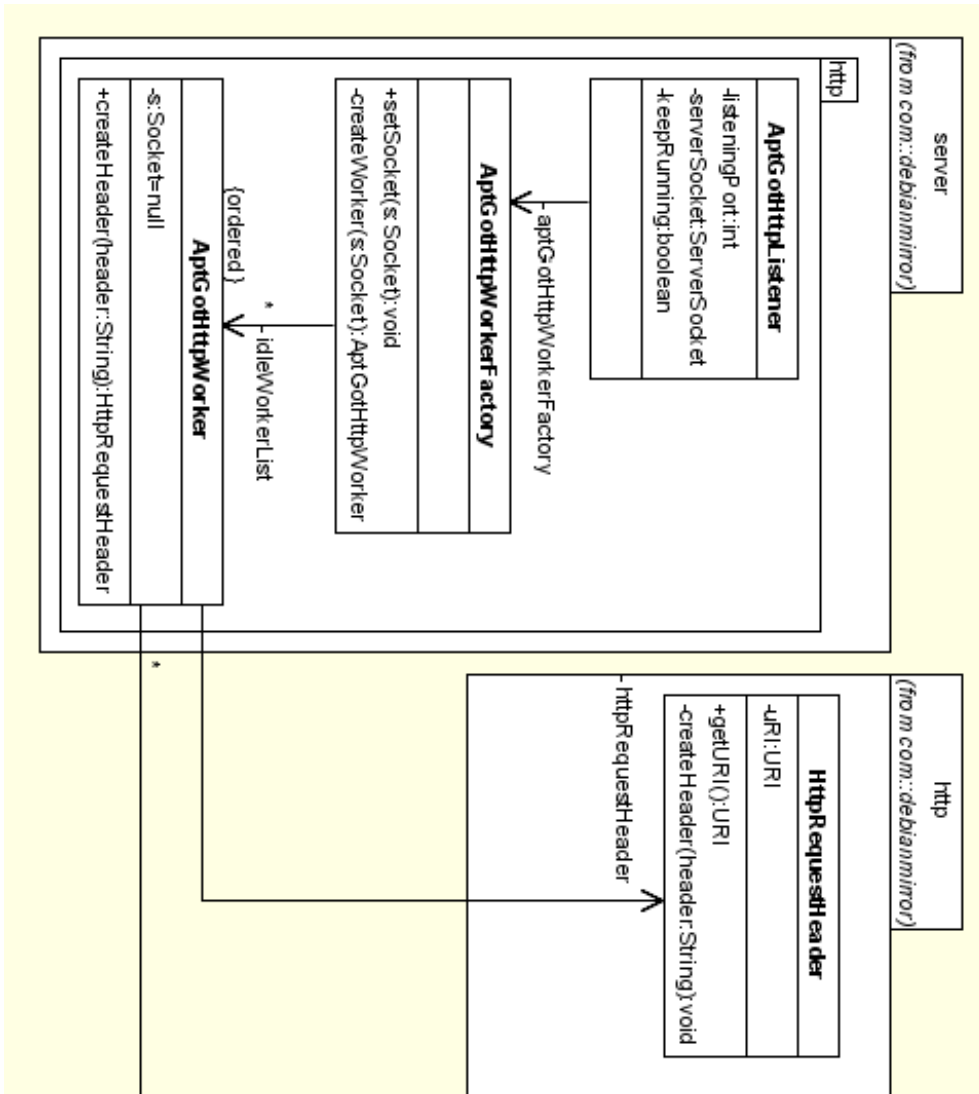
Operation	lookForFile(fileName: String)
Cross Reference	Request File
Preconditions	A valid filename was received from client
Postconditions	Response received

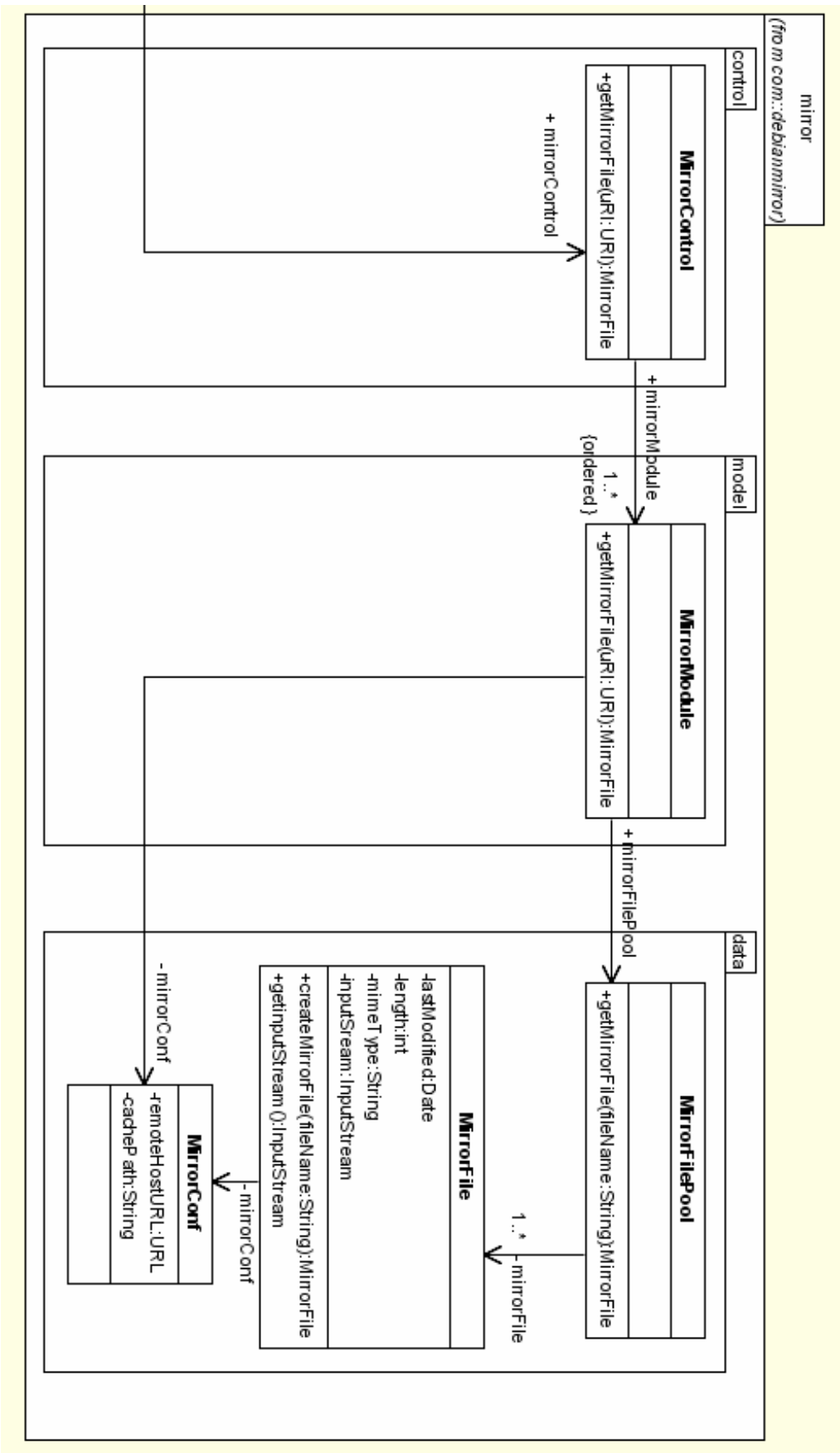
Operation	retrieveFileFromServer(fileName: String)
Cross Reference	Request File
Preconditions	Connection to remote server is established
Postconditions	Response received Connection to remote server closed

Operation	storeFile(file: File)
Cross Reference	Request File
Preconditions	File Found Enough disk space available
Postconditions	File Object created with information about the retrieved file File saved locally

Operation	sendFile(file: File)
Cross Reference	Request File
Preconditions	Connection to client is established A file or a message was received from the control
Postconditions	File or other message was sent to the client Connection to client closed

STATIC CLASS AND PACKAGE DIAGRAMS





CLASS SPECIFICATIONS/SOURCE CODE

```
/** Java class "HttpRequestHeader.java"
 * Purpose
 * (c) Apt-Got Group, Tobias Hertkorn,
 *                                     Jonas J&auml;gerh&ouml;k
 *                                     Jamie Fitz-Gerald
 */
package com.debianmirror.http;

import java.lang.String;
import java.net.URI;
import java.util.*;

/**
 * HttpRequestHeader parses a given String and breaks
 * it into Key-Value pairs.
 * It contains get-methods for the most common used keys
 */
public class HttpRequestHeader {

    //////////////////////////////////////
    // attributes

    /**
     * uRI represents the requested file URI
     * found in the header
     */
    private URI uRI;

    //////////////////////////////////////
    // operations

    /**
     * Returns the value of the key URI in the
     * Header
     *
     * @return the URI related to this request
     */
    public URI getURI() {
        return this.uRI;
    } // end getURI

    /**
     * Parses a given string for value key pairs.
     * Stores the important values in local key variables
     *
     * @param header is a String representing the whole header
     */
    private void createHeader(String header) {
        // your code here
    }
}
```

```
    } // end createHeader
} // end HttpRequestHeader
```

```
/** Java class "MirrorControl.java"
 * Purpose
 * (c) Apt-Got Group, Tobias Hertkorn,
 *                                     Jonas J&auml;lgerh&ouml;k
 *                                     Jamie Fitz-Gerald
 */
package com.debianmirror.mirror.control;

import com.debianmirror.mirror.data.MirrorFile;
import com.debianmirror.mirror.model.MirrorModule;
import java.net.URI;
import java.util.*;

/**
 * Represents the interface to the mirror part of the project
 * it contains a list of all configured mirrormodules and will
 * decide which module to use for the given URI
 */
class MirrorControl {

    ////////////////////////////////////////////////////
    // associations

    /**
     * The list of all configured mirrormodules
     */
    public Collection mirrorModule = new ArrayList(); // of type
MirrorModule

    ////////////////////////////////////////////////////
    // operations

    /**
     * Returns the MirrorFile associated with the given URI
     *
     * @return the MirrorFile represented by the uRI
     * @param uRI the URI to the requested file
     */
    public MirrorFile getMirrorFile(URI uRI) {
        // your code here
        return null;
    } // end getMirrorFile

} // end MirrorControl
```

```
/** Java class "MirrorConf.java"
 * Purpose
```

```

* (c) Apt-Got Group, Tobias Hertkorn,
*                               Jonas J&auml;gerh&ouml;k
*                               Jamie Fitz-Gerald
*/
package com.debianmirror.mirror.data;

import java.lang.String;
import java.net.URL;
import java.util.*;

/**
 * Represents all configuration data needed to run the mirror
 */
public class MirrorConf {

    // attributes

    /**
     * Represents the URL to the root dir of the RemoteDebianServer archive
     */
    private URL remoteHostURL;

    /**
     * Represents the local path to the root of the cache
     */
    private String cachePath;
} // end MirrorConf

```

```

/** Java class "MirrorFile.java"
 * Purpose
 * (c) Apt-Got Group, Tobias Hertkorn,
 *                               Jonas J&auml;gerh&ouml;k
 *                               Jamie Fitz-Gerald
 */
package com.debianmirror.mirror.data;

import java.io.InputStream;
import java.lang.String;
import java.util.*;
import java.util.Date;

/**
 * The MirrorFile represents a File that is either locally mirrored or
 * is currently requested from the RemoteDebianServer.
 */
public class MirrorFile {

    // attributes

    /**

```

```

    * Represents the last modified value of the File
    */
    private Date lastModified;

/**
 * Represents the length of the file
 */
    private int length;

/**
 * Represents the mime type of the file
 */
    private String mimeType;

/**
 * Represents the actual InputStream that can be used
 * to read the data of the file
 */
    private InputStream inputStream;

    //////////////////////////////////////
    // associations

/**
 * contains all information needed to translate the fileName to a local
store path
 * and a remote request URL
 */
    private MirrorConf mirrorConf;

    //////////////////////////////////////
    // operations

/**
 * Translates the given fileName into a local store path
 * if the file is not found there it creates the remote request URL
 * and requests the file from the RemoteDebianServer
 *
 * @param fileName the file name representing the requested file
 * @return the MirrorFile associated with the given file name
 */
    public MirrorFile createMirrorFile(String fileName) {
        // your code here
        return null;
    } // end createMirrorFile

/**
 * Returns the InputStream of this File
 *
 * @return an InputStream of this File
 */
    public InputStream getInputStream() {
        // your code here
        return null;
    }

```

```
    } // end getInputStream
} // end MirrorFile
```

```
/** Java class "MirrorFilePool.java"
 * Purpose
 * (c) Apt-Got Group, Tobias Hertkorn,
 *                                     Jonas J&auml;lgerh&ouml;k
 *                                     Jamie Fitz-Gerald
 */
package com.debianmirror.mirror.data;

import java.lang.String;
import java.util.*;

/**
 * Represents all files known to the mirrormodule
 */
public class MirrorFilePool {

    //////////////////////////////////////
    // associations

    /**
     * Stores the currently used MirrorFiles
     */
    private Collection mirrorFile = new TreeSet(); // of type
MirrorFile

    //////////////////////////////////////
    // operations

    /**
     * Will search the mirrorFile collection for the file.
     * if non is found it initiates a new MirrorFile object
     * represented by this fileName
     *
     * @param fileName what file to search for
     * @return the MirrorFile
     */
    public MirrorFile getMirrorFile(String fileName) {
        // your code here
        return null;
    } // end getMirrorFile

} // end MirrorFilePool
```

```
/** Java class "MirrorModule.java"
 * Purpose
 * (c) Apt-Got Group, Tobias Hertkorn,
 *                                     Jonas J&auml;lgerh&ouml;k
```

```

*                               Jamie Fitz-Gerald
*/
package com.debianmirror.mirror.model;

import com.debianmirror.mirror.data.MirrorConf;
import com.debianmirror.mirror.data.MirrorFile;
import com.debianmirror.mirror.data.MirrorFilePool;
import java.net.URI;
import java.util.*;

/**
 * The MirrorModule represents one configured module
 * The basic idea is to have one module per RemoteDebianServer.
 * This will get extended, so that all known mirrors of this
RemoteDebianServer
 * will be represented by this module
 */
public class MirrorModule {

    ////////////////////////////////////////////////////
    // associations

/**
 * contains the configuration of this mirrorModule
 */
    private MirrorConf mirrorConf;
/**
 * represents the FilePool used to request the MirrorFile from
 */
    public MirrorFilePool mirrorFilePool;

    ////////////////////////////////////////////////////
    // operations

/**
 * Asks the FilePool for the MirrorFile represented by the URI
 *
 *
 * @param uRI The variable contains the information about which file
was requested
 * @return the MirrorFile that was requested.
 */
    public MirrorFile getMirrorFile(URI uRI) {
        // your code here
        return null;
    } // end getMirrorFile

} // end MirrorModule

```

```

/** Java class "AptGotHttpListener.java"
 * Purpose
 * (c) Apt-Got Group, Tobias Hertkorn,
 * Jonas J&auml;lgerh&ouml;k

```

```

*                               Jamie Fitz-Gerald
*/
package com.debianmirror.server.http;

import java.net.ServerSocket;
import java.util.*;

/**
 * AptGotHttpListener will listen on the assigned
 * listeningPort for incoming client requests.
 * These client requests will be dispatched to
 * the AptGotHttpWorkerFactory.
 * After that it will return to listen for the next
 * request.
 */
public class AptGotHttpListener {

    // attributes

    /**
     * Represents the port on which the AptGotHttpListener will
     * listen for client connections.
     * Needed to create the ServerSocket
     */
    private int listeningPort;

    /**
     * Represents the ServerSocket. The Listener will use the
     * accept() method to listen on it in blocking mode.
     */
    private ServerSocket serverSocket;

    /**
     * The Listener (and therefor the Server) will run, as long
     * as this variable is set to true.
     */
    private boolean keepRunning;

    // associations

    /**
     * The listener dispatches the socket to the AptGotHttpWorkerFactory
     */
    private AptGotHttpWorkerFactory aptGotHttpWorkerFactory;

} // end AptGotHttpListener

```

```

/** Java class "AptGotHttpWorker.java"
 * Purpose
 * (c) Apt-Got Group, Tobias Hertkorn,
 * Jonas J&auml;lgerh&ouml;k
 * Jamie Fitz-Gerald

```

```

*/
package com.debianmirror.server.http;

import com.debianmirror.http.HttpRequestHeader;
import com.debianmirror.mirror.control.MirrorControl;
import java.lang.String;
import java.net.Socket;
import java.util.*;

/**
 * The AptGotHttpWorker is responsible for talking to the client.
 * This includes receiving the request header and sending the file
 * or any given error back to the client
 */
public class AptGotHttpWorker {

    //////////////////////////////////////
    // attributes

    /**
     * The Socket represents the connection to the client.
     * As long as there is no work assigned this variable must
     * be equal to null
     */
    private Socket s = null;

    //////////////////////////////////////
    // associations

    /**
     * The HttpRequestHeader will contain all information associated
     * with the client connection. Is equal to null when there is no
     * work assigned
     */
    private HttpRequestHeader httpRequestHeader;

    /**
     * The MirrorControl represents the layer below this worker
     * The worker will request the file from MirrorControl.
     */
    public MirrorControl mirrorControl;

    //////////////////////////////////////
    // operations

    /**
     * creates a new HttpRequestHeader by giving it the given
     * header String
     *
     * @return a new HttpRequestHeader representing the given header
     * @param header a String containing all unparsed header information
     */
    private HttpRequestHeader createHeader(String header) {
        // your code here
        return null;
    }
}

```

```

    } // end createHeader
} // end AptGotHttpWorker

```

```

/** Java class "AptGotHttpWorkerFactory.java"
 * Purpose
 * (c) Apt-Got Group, Tobias Hertkorn,
 *                                     Jonas J&auml;lgerh&ouml;k
 *                                     Jamie Fitz-Gerald
 */
package com.debianmirror.server.http;

import java.net.Socket;
import java.util.*;

/**
 * This factory produces or reuses one Worker per
 * request and assigns a socket to this Worker
 *
 * @author Tobias Hertkorn
 */
public class AptGotHttpWorkerFactory {

    ////////////////////////////////////////////////////
    // associations

    /**
     * <p>Contains a list of available idle workers</p>
     *
     */
    private Collection idleWorkerList = new ArrayList(); // of type
    AptGotHttpWorker

    ////////////////////////////////////////////////////
    // operations

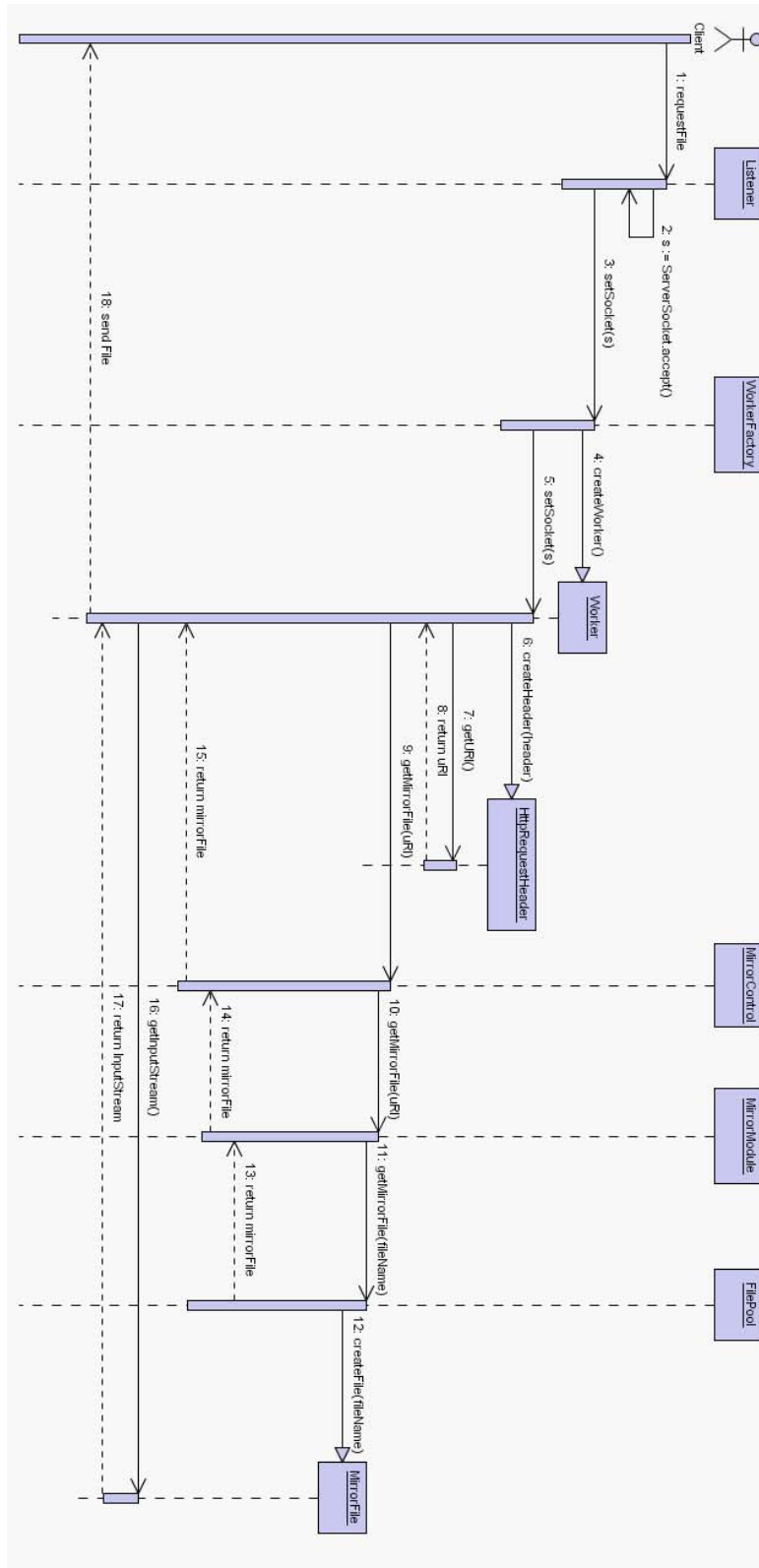
    /**
     * AptGotHttpWorkerFactory receives a socket by the use
     * of this method. This Socket will be handed to the
     * Worker assigned for this connection
     *
     * @param s
     */
    public void setSocket(Socket s) {
        // your code here
    } // end setSocket

    /**
     * Will create a new AptGotHttpWorker and assigns the Socket to it
     * so it will start working on this socket right away.
     *
     * @return a new AptGotHttpWorker with the Socket s already assigned
     * @param s The socket to be assigned to the new
     */

```

```
private AptGotHttpWorker createWorker(Socket s) {  
    // your code here  
    return null;  
} // end createWorker  
  
} // end AptGotHttpWorkerFactory
```

INTERACTION DIAGRAMS



ADDITIONAL INFORMATIONS

CLASS/PACKAGE RESPONSIBILITY

View

//provide frontend

com.debianmirror.StandAloneAptGot

Main method

Starts the stand alone server

com.debianmirror.StandAloneAptGotConf

Parses the StandAloneAptGot.props file for

- info about the class name of the mirror configuration data
- the listenerPort
- the nrOfWorkers

com.debianmirror.server.http.AptGotHttpServer

Starts the HttpListener

com.debianmirror.server.http.HttpListener

Listens for http requests on assigned port

Hands new connections to AptGotHttpWorkerFactory

com.debianmirror.server.http.AptGotHttpWorkerFactory

Initializes new AptGotHttpWorker

Keeps a list of running AptGotHttpWorker

Assigns the connection to an AptGotHttpWorker

com.debianmirror.server.http.AptGotHttpWorker

Parses the request header by the use of HttpRequestHeader

Requests file from Control

Create the response header by the use of HttpResponseHeader

Sends response header by the use of HttpResponseHeader

Sends file data if applicable

Control

//decide which module handles the request

com.debianmirror.mirror.control.MirrorControl

Is interface for View to communicate with Control

com.debianmirror.mirror.control.MirrorControlImpl
Implements MirrorControl
Requests list of configured MirrorModule from MirrorControlConf
Holds a list of all the configured MirrorModule
Breaks down file URL string so it can dispatch it to the right MirrorModule
Requests file from that MirrorModule
Gives file or error to AptGotHttpWorker

Model

//all the logic for the file handling

com.debianmirror.mirror.model.MirrorModule

Is interface for Control to communicate with Model

com.debianmirror.mirror.model.debian.DebianMirrorModule

Implements MirrorModule
Initializes FilePool
Gets file URL from Control
Decide if Debian package/regular file or Debian package list
1.) Debian package/regular file
Translates links
Generate
- relative location to find/store file with information stored in DebianMirrorModuleConf
- list of locations to download file if not found locally with information stored in DebianMirrorModuleConf
Requests file by passing these information on to FilePool
Gives file or error to MirrorControlImpl
2.) Debian package list
for now – handle as if regular Debian package
later: Generate/update package list if appropriate

Data

//data storage and data maintenance (especially what to purge and what to re-request)

com.debianmirror.mirror.data.MirrorFilePool

Is interface to Control to communicate to Data

com.debianmirror.mirror.data.debian.DebianMirrorFilePool

Has a WeakHashMap of MirrorFiles
Decides what and when to purge (not what to re-request)
Translate relative storage path into an absolute path

com.debianmirror.mirror.data.MirrorFile

Interface for Pool

com.debianmirror.mirror.data.SimpleMirrorFile

takes absolute storage path and remote mirror list

requests file from mirrors if not locally stored or modified on server

stores new file locally

holds `FileInputStream` (must be synchronized on! So we can switch association from

`URLConnection` to `FileInputStream`. How do we handle concurrent reads on the

`InputStream`? How does it react... Always use `read(byte[] b, int off, int len)`!!!)

holds `fileLength`

holds `mimeType`

creates a unique hash for hash map. Must create different hash for different files (not only filenames) and same hash for same file!!!

com.debianmirror.mirror.data.DebianMirrorConf

implements `MirrorControlConf` and `DebianMirrorModuleConf`

holds information