Adapa Stockwatcher (AS)
Software Architecture Description

Group Adapa

1. Version History

Version	*Date*	*Author*	*Comments*
1	November 13	Ludvig Gee	Initial version of SAD. Establish template.
2	December 16	Ludvig Gee	Completion of SAD.
3	December 20	Ludvig Gee	Review of final SAD

Table of Contents

2. Introduction

2.1 *Purpose and Scope*

The purpose of this document is to provide the reader with an overview and explain the architecture of the Adapa Stockwatcher (AS). Several different architectural views are presented both diagrammatically and textually to depict different parts of the system. The intention of this is to capture and convey significant architectural decisions which have been made on the overall system.

The system itself can be divided into three sections: Backend, middleware and front-end applications. The main focus of this document will be the backend and overall system architecture. Due to varying platforms and programming languages of the front-end applications the architecture of these will be generalized into a single part of the system at some points in this document. This is also due to the fact that they ultimately are supposed to have the same user functionality independent of platform.

2.2 *Audience*

The intended audience of this document are:

Björn Olsson – Course manager

David Gregory – Supervisor

Group Adapa – Development Team

Ludvig Gee – Software Architect

2.3 *Architectural Design Approach*

The structure of this document and the architectural design approach is based on the SEI model view of architecture. There are three different viewtypes that will allow the different stakeholders and intended audience to gain accurate insight of the architecture of AS.

Module views: Describes how the system is structured as a set of implementation/code units.

Component-and-connector (C&C) views: Describes how the system is to be structured as a set of interacting runtime elements.

Allocation views: Describes how the system relates to non-software structures in its environment.

Each viewtype will in turn consist of these four key parts:

Primary presentation: A diagrammatical representation of the view showing elements and the relationship amongst them.

Element catalogue: Details and describes the elements and relationships of the diagram used in the primary presentation.

Context diagram: Shows how the view depicted (if it is a subset of a view) relates and interacts with external components and the rest of the system.

Architecture background: Provides a rationale explaining why a particular design, pattern or style has been applied to the view.

3. System Stakeholders and Requirements

3.1 Stakeholders

Stakeholder	Relevant views	Definition
Acquirer: Björn Olsson	All	Commissioner and ultimate evaluator of the system and documentation.
Developer	All	
Architect	All	Defines, documents and conveys the architecture
User	Some details	Person who makes use of the front-end applications (web, app or desktop) when the system is running.
Maintainer	All	Manages the evolution and eventual fixes of the system once it is running.
Quality manager	Module Views	Defines tests for the system and requirements. Reports eventual errors in code and requirements.
Project Manager	Some details	

3.2 Overview of Requirements

Reference	Requirement Description
R1.	User shall be able to analyse market index details by chart and numbers.
R2.	User shall be able to analyse stock details by chart and numbers.
R3.	User shall be able to view stock in three different views.
R4.	User shall be able to search and filter stocks from selected market.
R5.	System displays news about each available stock.
R6.	System shall supply user with RSS news feeds for available markets
R7.	User shall be able to follow desired stocks in all applications.
R8.	Data collected from sources shall be accessible in applications maximum five minutes after being fetched.
R9.	Applications shall only receive stock data from backend.
R10.	One view shall be candlestick chart.
R11.	ETL shall be implemented in Erlang.
R12.	User shall be able to access data via three platforms (mobile, desktop, web).
R13.	System shall use at least four different data sources.
R14.	Database shall only consist of well formatted data.
R15.	Database shall contain data up to five years old.

R16.	Database shall not use any SQL language.
R17.	ETL shall discard data not up to standard.
R18.	ETL shall be packaged and run as an OTP application.
R19.	ETL shall be automated, meaning that once started, no more interaction is necessary.
R20.	ETL shall set up database properties necessary to run.

3.3 Use cases

3.3.1 Fetch Stock (Server side)

Brief Description
The system collects stock data from source.

Actors
1-Primary actor is the ETL.
2- Secondary actor is the source.
3- Third actor is the database

Flow of events
Basic Flow:

1. ETL checks if database exist.
2. ETL sends request to server.
3. Server gives a symbol back.
4. ETL checks if symbol is correctly formatted.
5. ETL stores data into database.

Alternativ Flow:

1. At one, if database does not exist, database is created.
2. At four, if symbol is not correctly formatted it will be discharged, logy stored in server_logger.

3.3.2 Fetch Stock (Client side)

Brief Description
The client collects stock data from database.

Actors
1-Primary actor is the Client.
2- Secondary actor is the database.

Flow of events
Basic Flow:

1. Client constructs a URL string.
2. Client sends http request to database with said URL.
3. Database responds with a json object.

4. Client receives and parses json object.
5. Client stores json object locally for usage.

Alternativ Flow:

1. At three, if the url request is not correctly formatted the database will respond with an error message.
2. At four, if json object is not properly formatted then an exception will be thrown.

3.3.3 Install ETL

Brief Description

Setting up the backend system as an OTP application.

Actors

1- Administrator
2- Application

Prerequisites:

- Erlang 14.0b is installed on the system.
- CouchDB is installed and running on the system.
- CouchDB's URL and port have been entered into the definitions.hrl file.

Flow of events

Basic Flow:

1. User starts a windows Powershell or command prompt.
2. User switches to the correct directory where the emakefile is located.
3. User starts an Erlang shell at this location, including the ebin file as external files.
4. The windows user runs the command to compile all Erlang files.
5. User starts the application by typing application:start(stocksadapa).
6. Application checks everything is running and reports results to the user.
7. Application runs until stopped.

Alternative Flow:

1- **Basic Flow: 3.**
   a. User is on a Unix system, the user runs the makefile to compile and move all the files into the correct locations.
   b. The basic flow resumes at point 4.

2- **Basic Flow 6.**
   a. Application reports that not all checks passed and does not start the application
   b. User corrects errors reported by the Application
   c. The basic flow resumes at point 5.

3.3.4 Start ETL

Brief Description
Start the ETL

Actors
1- ETL
2- Database

Flow of events
Basic Flow:
1. Start head supervisor.
2. Sends messages o Database.
3. Must pass 12 hardcoded checks
4. Creates main supervisor
5. Checks so that the called databases exists
6. OTP supervisors start the fetchers.

4. Architectural Forces

4.1 Goals

Develop a stock trading analysis system with a backend data warehouse that serves at least three front end applications.

- High Availability (Fault tolerant)
- High Performance
- Scalability
- Distributed
- Interoperability

4.2 Constraints

- The three front end applications must consist of web, desktop and mobile
- The backend must be programmed in Erlang.
- Datacrawler must collect data from at least 3 different data sources
- Backend must utilize Extract, Transform and Load (ETL)
- The system must be complete before 2013-12-20 23:59

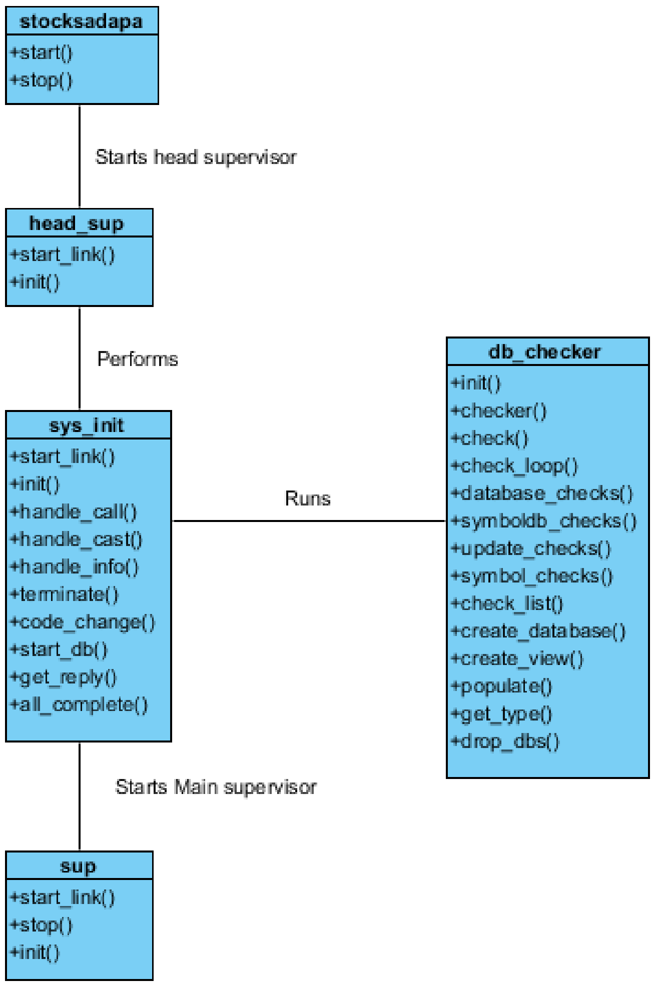
5. Architectural Views

5.1 *Module views*

5.1.1 *ETL*

5.1.1.1 Start-up

5.1.1.1.1 Primary presentation



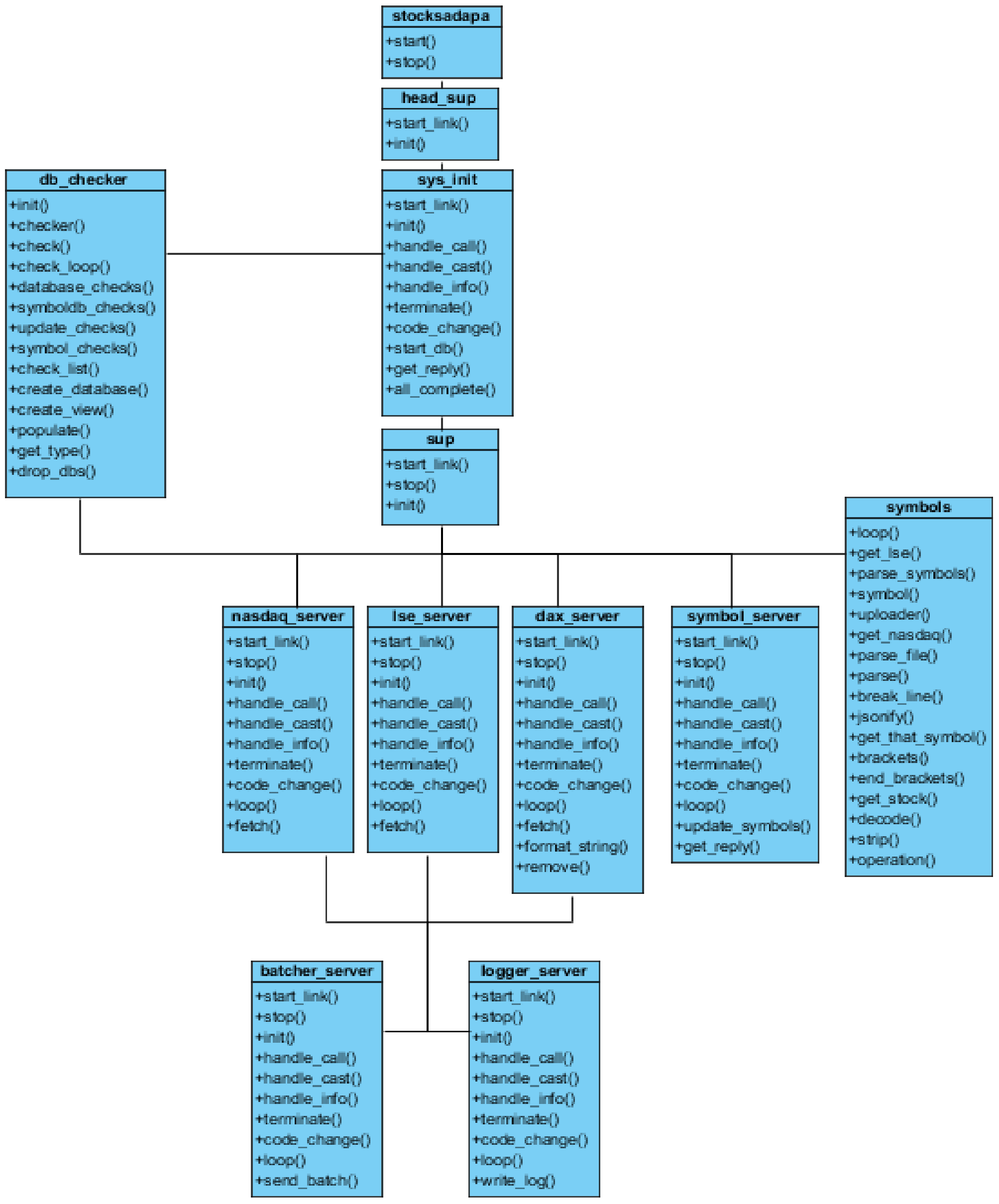
5.1. *El e m en t*

catalogue

- Stocksadapa – Erlang .app module. Deals with the creation and start of the app. It starts the head supervisor.
- Head_sup – The head supervisor. Establishes sys_init as a child and runs it. It is an OTP supervisor.

- Sys_init – Starts the database checks, it must pass the hardcoded value of 12 checks to move to the next stage. It sends messages to the db_checker process which performs the checks. Once all have finished it moves on to creating the main supervisor.
- Db_checker – Db_checker listens for instructions which are sent by the calling process. It performs checks on the database to ensure that all the required databases actually exist and that they can be communicated with. It sends HTTP requests to the expected views or databases and acts upon their return values.
- Sup – an OTP supervisor which starts the fetchers described in the next section. It has 6 children with a one-to-one restart mode.

5.1.1.1.3 Context diagram

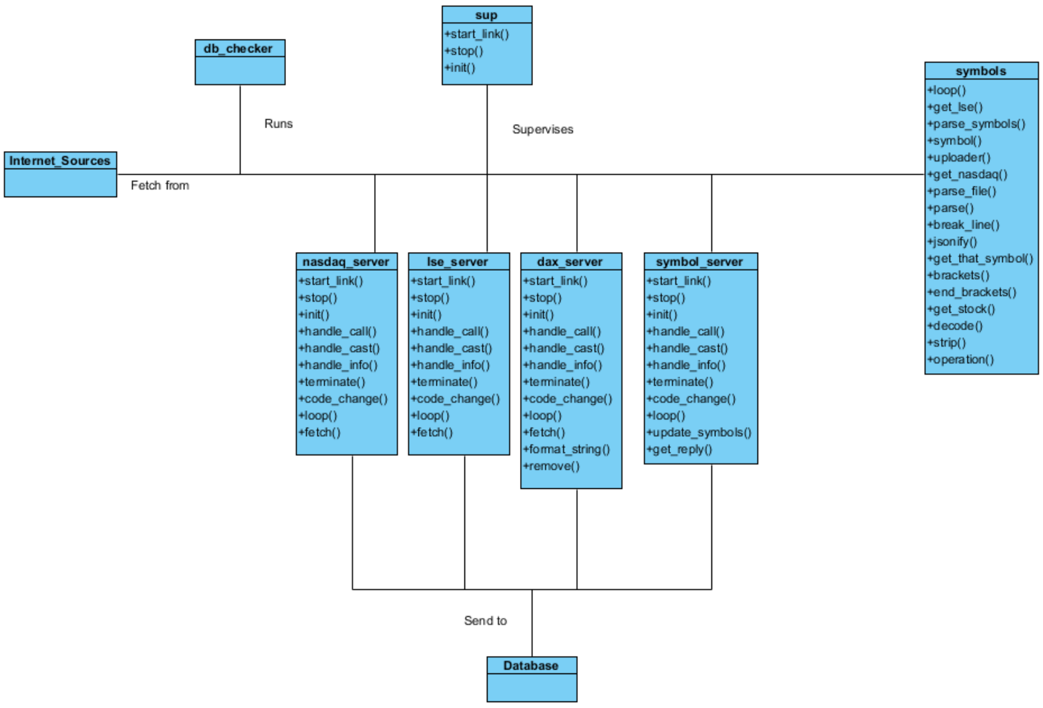


5.1.1.1.4 Architecture background

The start-up section of the ETL deals mainly with running the application as a service while ensuring that the database application it is configured for has the correct views and database elements. This solved the problem of having to manually create the database each time a significant change was made and also allows the application to be quickly and easily deployed on new systems.

5.1.1.2 Extract and Transform

5.1.1.2.1 *Primary presentation*



5.1.1.2.2 *Element catalogue*

- Sup – main supervisor controls the fetchers, symbol servers, the logger and batcher servers.
- Internet sources – GAS/Yahoo Finance/Markitondemand. Communicated to via HTTP requests from the servers. They send data back to the servers as follows:
  GAS – JSON
  Yahoo – CSV – This is parsed into JSON to be sent to the database.
  Markitondemand – JSON
- Nasdaq_server – One of the 3 OTP fetcher servers. Schedules and fetches data based on the stock markets opening times.
- LSE_server – Another of the 3 OTP fetcher servers. Schedules and fetches data based on the stock markets opening times.
- Dax_Server –The final of the 3 OTP fetcher servers. Schedules and fetches data based on the stock markets opening times.
- Database – It runs CouchDB and is communicated to via HTTP requests. In this instance it takes the place of what happens when data is sent to the database, normally it’s sent to the batcher_Server which is sitting and waiting for messages. This is covered further in the Load section.

5.1.1.2.3 *Context diagram*

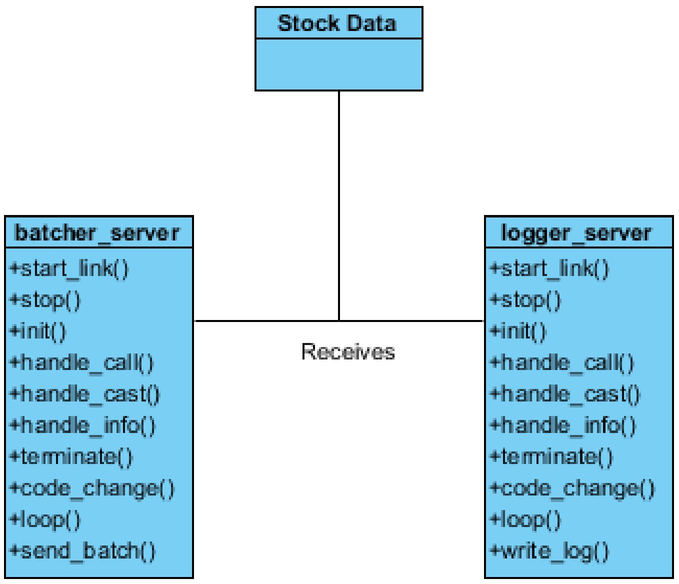
See Context diagram 7.1.1.1.3.

5.1.1.2.4 *Architecture background*

The extract and transform section fetches data from the internet sources via the fetchers; nasdaq_server, lse_server and dax_server. They perform a time check to see if their stock markets are open and if so perform requests. The information for this is defined in the definitions.hrl file. The fetchers do not perform fetches when their markets are closed. If they are, then the fetchers wait for a set period of time. The symbols module deals with collecting and parsing the symbols for the markets. An exception to this is Dax which is hardcoded because there are only 30 stocks available. The symbols are refreshed each day by the symbol server which in turns calls the symbols module to re-upload the symbols. Also included in the symbols module are methods which allow the servers to fetch symbols from the database and to parse them back into normal strings or lists for use in their queries.

5.1.1.3 **Load**

5.1.1.3.1 Primary presentation



5.1.1.3.2 Element catalogue

- Stock data - in this case it’s just the data being included in the message sent to the server by the fetcher servers.
- Batcher_server – started by the supervisor is also OTP. Waits to receive messages from the fetcher servers. Communicates with the database via HTTP requests.
- Logger_server – started by the supervisor and is also OTP. Waits for messages from the fetcher servers upon successful data requests or errors. Writes to a log on errors.

5.1.1.3.3 Context diagram

See Context diagram 7.1.1.1.3.

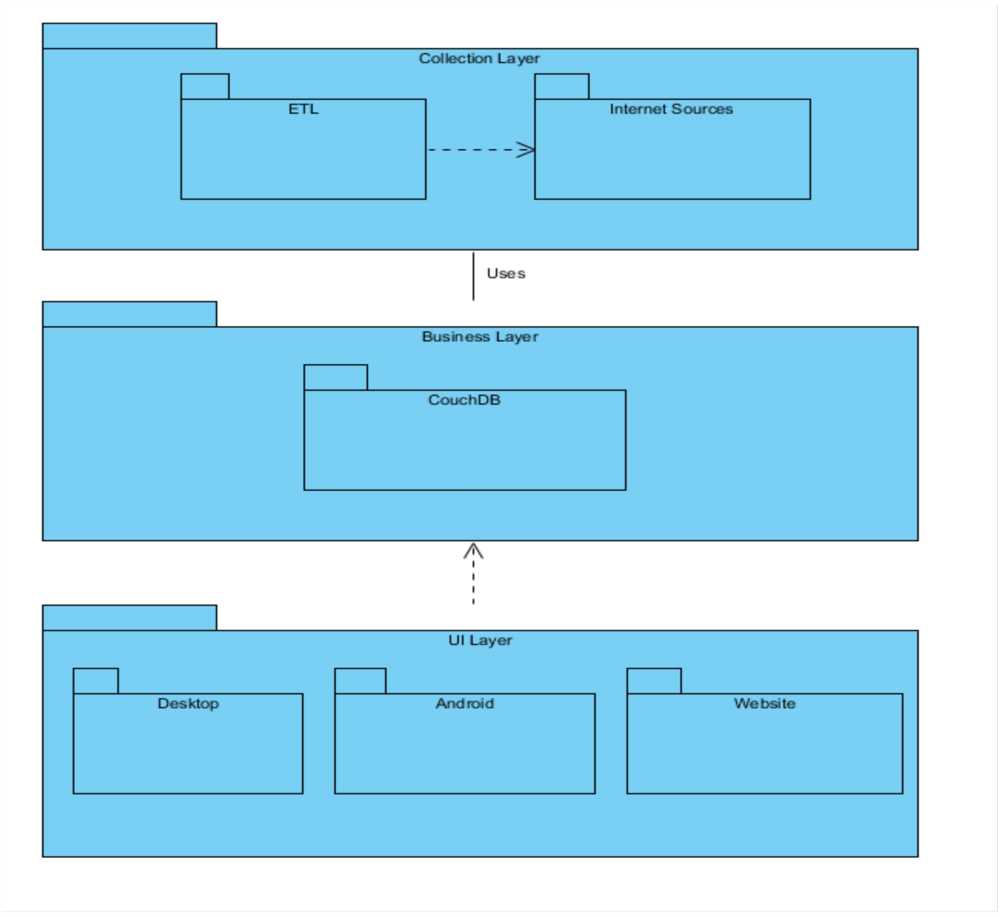
5.1.1.3.4 Architecture background

Loading happens after data has been received from a HTTP request. The data is sent to the batcher_server (which is started and maintained by the supervisor, so is the logger_server). They sit waiting for messages from the fetcher servers. If the batcher receives a message it finds out who sent it and depending on which database that sent it, it sends the information on to that database via a HTTP request. The logger is notified of successes and failures from the data requests being made but only writes to the log file in the case of errors. It provides a small error message as to what went wrong.

The reason for the batcher was to consolidate the database connection code into one module. We can better find errors and maintain it properly if the code is separate from the fetchers. This way we stay DRY (Don't repeat yourself). The logger was created to help keep track of failing or bad data. There are roughly 500/700 stocks on nasdaq alone that fail because they have bad data. We were getting lots of errors before and this has helped to cut down or at least figure out what was causing the issues.

5.1.2 *Layer Diagram*

5.1.2.1 Presentation Layer



5.1.2.2 Element Catalogue

- Collection Layer – The layer describing collection of data. ETL is dependent upon Internet Sources.

- Business Layer – The collection layer uses the business layer to store data in CouchDB
- UI Layer – Dependent upon the Business Layer. Desktop, Android and Website all require data from the database.

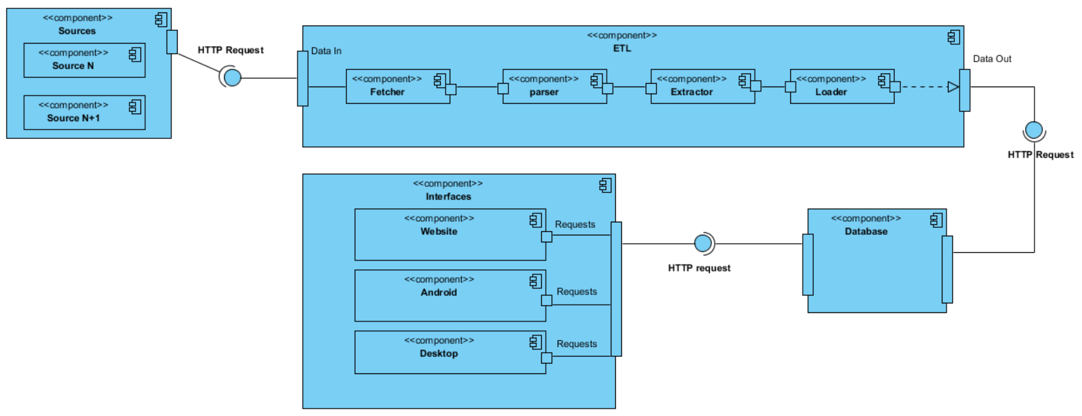
5.1.2.3 Architecture Background

This layering was chosen as it closely reflects that of a client server style. It has a clear distinction between the clients, server and the data that they collect.

5.2 *Component and Connector views*

5.2.1 *System Components*

5.2.1.1 Presentation Layer



5.2.1.2 Element Catalogue

- Sources – Represents the sources from which we gather the data. Communicated to via HTTP requests from the ETL.
- ETL – Consists of fetchers, parsers, extractors and loaders. It performs all these actions on the data and sends it out as a HTTP request to the database.
- Database – Stores the information and handles HTTP requests from both the ETL and the interfaces. Serving or storing data as request.
- Interfaces – Request information from the database via HTTP requests. They then present the data to the users. Little to no data crunching should be done here.

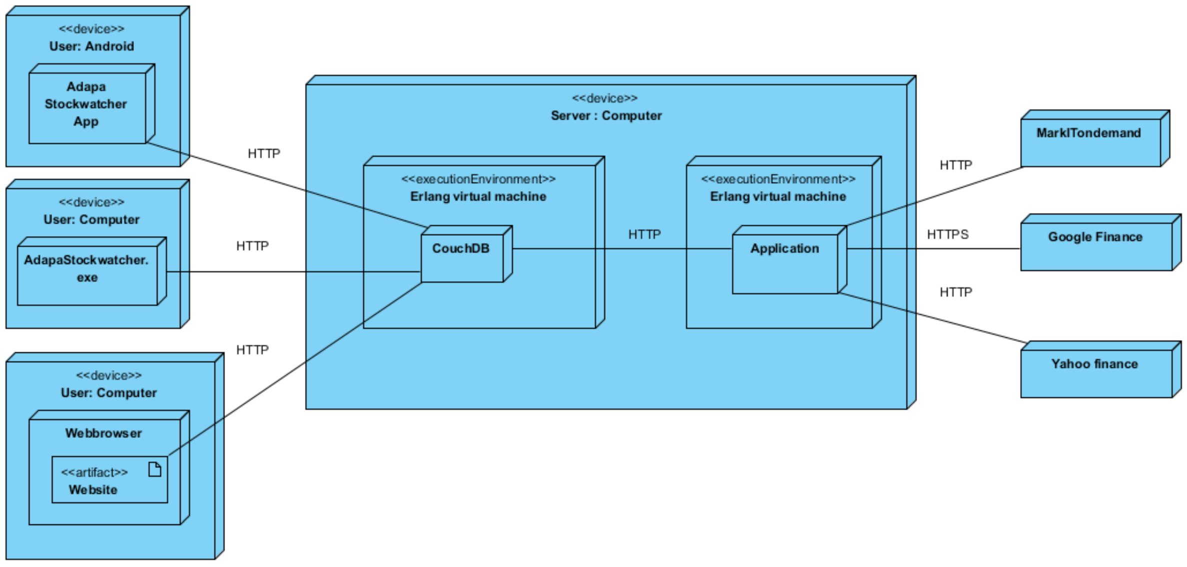
5.2.1.3 Architecture Background

This component and connector view shows a little more detail in the tasks that the system shall perform. It is intended to display what the individual components do while showing how they communicate with each other. Elements of pipe and filter and client server are visible in this diagram.

5.3 *Allocation view*

5.3.1 *Deployment*

5.3.1.1 Presentation Layer



5.3.1.2 Element Catalogue

- User: Android – A user’s mobile device that acts as the interface and runs the mobile application.
- Adapa Stockwatcher app – The mobile application that connects to CouchDB via HTTP request/response.
- User: Computer – A user’s computer that acts as the interface and runs the desktop application.
- AdapaStockwatcher.exe – The desktop application running on the user’s computer.
- Webbrowser- Undetermined webbrowser that the user has access to on his/her computer.
- Website – The website application run on the webbrowser that connects to couchDB via HTTP request/response.
- Server: Computer – A laptop running Win OS 7.
- Erlang virtual machine – Two seperate execution environments or shells on which the Application is running and CouchDB (Database).
- Application – Erlang ETL application that connects to the different external websites and CouchDB through http and https request/response.
- CouchDB – NoSql Database that connects to the application on the server and external devices through http request/response.
- MarkITondemand / Google Finance / Yahoo finance – External websites where data is fetched from.

5.3.1.3 Architecture Background

This view shows how the system is mapped onto physical nodes and more detailed aspect of the separation of concerns in the client-server style architecture. The Erlang VM runs as one OS process. By default it runs one OS thread per core to achieve maximum utilisation of the machine. The number of threads and on which cores they run can be set when the VM is started. This emphasizes the scalability of the system.

6. Appendices

6.1 Appendix: References

[1]. *Software Systems Architecture: Working with Stakeholders Using Viewpoints and Perspectives, 2nd Edition*. Nick Rozanski and Eoin Woods, Addison Wesley, 2011.

[2]. http://www.sei.cmu.edu/reports/04tr008.pdf

[3]. *Software Architecture in Practice,* 2nd Edition. Len bass, Paul Clements and Rick Kazman, Addison Wesley, 2011.