Siberian Trucking System

Software Architecture Description

Lima

athas@sigkill.dk, shantanubala@gmail.com,
jesper_tved@hotmail.com

November 4, 2012

Version history

Version	Date	Author	Comments
1	2012-08-17	KMH	Initial template based on (Rozanski and Woods, 2011)
3	2012-09-17	SHB and JTM	added chapter 3
4	2012-09-19	TRH, SHB and JTM	Revised 3.1, clarified truck transmitter concept
5	2012-09-24	TRH, SHB and JTM	added chapter 4 and 5.2
6	2012-10-08	TRH, SHB and JTM	added chapter 5
7	2012-10-10	TRH, SHB and JTM	Added sections 6 and 6.1
8	2012-10-24	TRH, SHB and JTM	Added sections 6.3 and 6.4
9	2012-11-04	TRH, SHB and JTM	Added appendix A and B

Contents

Chapter 1

Introduction

1.1 Purpose and scope

Russia, as the worlds largest country by land area, has an extensive raw materials industry. Since the fall of the Soviet Union, the Russian trucking industry has undergone a dramatic growth, and government initiatives aim to maintain this growth into the future. Since many of the industrial installations are located in remote areas, logistics companies face great difficulties in managing their truck fleets, particularly as transportation times are often unpredictable due to weather, accidents and the generally poor condition of the Russian road network. As an additional complication, the distant locations and low population densities prevents the use of many common communication technologies (eg. no cell phone network).

We propose the creation of a system, the Siberian Tracking System (STS), for tracking a large fleet of trucks operating in far-flung regions of Russia, owned by the (fictive) Siberian Trucking Company (STC). A central server installation will be informed of the location and state of every truck in the fleet, permitting decision making systems to have complete knowledge of the state of the company assets. Each truck is responsible for tracking its own progress, then periodically relaying it back to the main servers. The project does not involve creation of any new ground stations; trucks will use standard wireless communication methods whenever in range of appropriate networks. The transmitters on the trucks will only use one way communication to the server, with data of the trucks current position and the speed of the truck. The transmitters are supposed to be low-cost hardware, so communication from the server to the trucks and additional measuring equipment, for instance of oil and gasoline levels have been deselected.

The STS is solely concerned with the state of the trucking fleet, and does not do freight tracking or make any kind of business logic decisions on its own. While it provides information allowing such decisions to be made, it is purely a data collection and dissemination infrastructure. In particular,

it is a one-way communication system. Other means must be employed to contact the trucks on the road. Also, STS is by itself not concerned with doing data mining or presenting a sophisticated user interface to its data. Instead, a data interchange mechanism will be defined that allows other systems to receive information from STS.

1.2 Audience

The intended audience of this document, and the reason for their inclusion, are as follows.

- Business logic decision makers of STC, who must determine whether the information provided by STS is sufficient.
- Truck maintenance department representatives, to determine whether the additional equipment needed on trucks is realistic.
- The developers who have to implement the suggested architecture and design.
- Finally, whoever approves the financing of the project.

1.3 Status

The basic requirements and design of STS has been determined, but the actual implementation has not yet begun.

1.4 Architectural design approach

We are designing a system with little user interaction and no actual stakeholders to guide the design process. We will therefore focus the architecture on the technical difficulties of the system. The architecture consists of the following views:

- Functional view describes the elements needed in the program.
- Information view describes how data is generated, sent thrrough the system and how it is stored.
- Concurrency view describes how the elements work together in a runtime enviroment.
- Deployment view describes in which enviroment the system is deployed and executed

- Development view decribes different programming design choises, such as code standard and testing approaches.
- Operational view describes how the system is deployed and who handles administration and support of the system.

It is expected that some parts of the architecture will be redefined during the project period, as more insight in the problem to be solved is gathered.

Chapter 2

Glossary

Term	Definition
Canonical truck position	An externally registered position of a specific truck at a specific time. Used to check consistency with measured positions.
Mesh network	An ad-hoc (usually short-range) network formed by autonomous units within range of each other
STS	Siberian Tracking System. Refers both to the project as a whole, and to the software running on stationary servers (ie. excluding the software on truck transmitters).
STC	Siberian Tracking Company
Truck transmitter	The hardware unit on a truck. Consists of a GPS receiver and an antenna for communications.
AWS	Amazon Web Services - the cloud computing platform chosen for the STS.

Chapter 3

System stakeholders and requirements

3.1 Stakeholders

- Acquirers: the Siberian Trucking Company (STC) will be paying for the development of the system to aid in business logic. The users of the system will be members of the STC's business administration.
- Communicators: the technical writers who will create documentation regarding the operation of the system, while the business administration of the STC will be responsible for the training of the end users of the system.
- Developers: the STC has contracted a group from the University of Copenhagen to develop the system.
- Maintainers: the STC has a team of developers responsible for maintaining and evolving the STS system after it is completed.
- Production Engineers: the STS system will be deployed onto the Amazon EC2 platform, outsourcing the deployment environment to Amazon's engineering staff.
- Suppliers: servers will be provided by Amazon's EC2 platform, while the GPS units are assumed to already be installed on the STC's trucks.
- Support Staff: it is assumed that the STC has the appropriate IT staff for helping the end users of the STS in accomplishing their appropriate business administration tasks. This staff also installs the physical equipment in the trucks.

- System Administrators: Amazon provides the appropriate hardware administration while the STC has a team of developers responsible for updating and maintaining the software environment of the EC2 instances.
- Testers: The STC has a team of developers responsible for testing and ensuring the STS works effectively.
- Users: members of the STC's business administration team who will use and analyze data provided by the STS to make informed business decisions.

3.2 Overview of requirements

Reference	Requirement description
R1	The system must provide business administrators with a detailed history of the location of every truck in the STC fleet.
R2	The system must be able to receive and store 1,000 GPS datapoints per second.
R3	The server interface for storing the trucks' location data must have an availability of at least 99 percent uptime.
R4	The tracking units present on each of the trucks must have a fallback when data cannot be sent in real-time due to bad network coverage.
R5	The server interface must be capable of receiving an individual data point (the truck's location) or a series of data points (the truck's location history over a period of time).

3.3 System scenarios

3.3.1 Functional scenarios

Scenario reference	FS1.
Overview	How truck information is sent to the server
System state	The truck is fitted with a truck transmitter and is currently driving
System environment	The system is operating normally
External stimulus	The truck transmitter has a position that should be sent to the server
Required system response	If the truck is in range of a network, the position is sent to the server and a confirmation is received. If the truck is out of range, the position is stored in the truck transmitter and will be sent when the truck is in range of a network again

Scenario reference	FS2.
Overview	How truck positions are queried by a user
System state	Positions from many different trucks have been sent to the server with timestamps
System environment	The system is operating normally
External stimulus	The user queries a specific truck, trucks within an area, and trucks that have not yet reached their deignated targets on time
Required system response	The system sends the requested data as a list via the API.

Scenario reference	FS3.
Overview	New trucks are imported to the system
System state	New trucks are listed in the truck store and a transmitter have been fitted into the new truck
System environment	STS, the truck transmitter and the truck store are operating normally
External stimulus	An employee from the support staff have registered the truckID with the trucks transmitterID
Required system response	STS can now be queried for the new trucks ID

Scenario reference	FS4.
Overview	How transmittion network are chosen
System state	The truck is fitted with a truck transmitter and is currently driving
System environment	The truck transmitter and the datastore is operating normally, the truck is driving in an area without any connection
External stimulus	The truck transmitter has a new position that should be sent to the server
Required system response	The truck transmitter first tries to send via GSM mobile network, after this the long range meshnetwork is tried. If neither of these worked, the position is stored to be sent at a later point.

3.3.2 System quality scenarios

Scenario reference	QS1.
Overview	If the number of trucks in the fleet increases, or if the business administrators need to collect data at a greater interval, the capabilities of the system ought to be appropriately scalable.
System environment	This process will be handled by Amazon's Elastic Load Balancing service in conjunction with Amazon EC2.
Environment changes	For example, if 10 small EC2 instances can receive and process 1,000 data points per second, the system will allocate the equivalent of 50 small EC2 instances to process an increased workload of 5,000 data points per second.
Required system behavior	When a tracking device in a truck sends data to the STS server, the data will be placed in a queue. Based on the size of the queue, an appropriate number of EC2 instances will be started or shut down to process the queue.

Scenario reference	QS2.
Overview	If the cellular network is not available, the transmitter on the truck will fall back to using a short-distance mesh network established between the trucks.
System environment	This process will be handled by the transmitters that are installed on every truck in the STC fleet.
Environment changes	For example, if a truck leaves cell network range, it will attempt to connect to a nearby truck, which will also be connected to other trucks. As long as a single truck in the mesh has cell coverage, all of the trucks will be able to transmit their data.
Required system behavior	The trucks' transmitters will have a queue for sending data to the STS server, and will process this queue using the fastest available connection.

Scenario reference	QS3.
Overview	If part of the Amazon EC2 cluster fails, the error must not propagate to the entire system.
System environment	This robustness must be built into the software running on the EC2 cluster. Additionally, the truck transmitters must still be able to transmit data.
Environment changes	If a node in the cluster loses contact with another node, it must ensure that the overall system still has redundant data, such that further node failures can be handled. The trucks must be able to connect to another node if the usual recipient for their data fails to respond. If the system is sufficiently heavily damaged that data integrity is lost, it must still be possible for trucks to transmit data, but any queries in the data should have their result marked as incomplete.
Required system behavior	Heavy data redundancy must be built into the system. Whenever a truck reports back, it may also receive an updated list of communication endpoints.

Scenario reference	QS4.
Overview	If the transmitter on a truck fails, this failure must be noticed and rectified.
System environment	The system must have logic to detect when a truck “should” have transmitted information, but has not. We assume that the fabrication and installation of new transmitters is done externally of our system.
Environment changes	If a truck fails to report back, the maintenance department must be notified that a truck has a faulty transmitter, and that it must be replaced.
Required system behavior	We receive information from the truck register whenever the truck reaches some central locations (the *canonical truck position*). If the truck itself does not send the same information, its transmitter will be assumed defective, and called in for repair. As long as every truck is guaranteed to eventually stop at such a destination, any failure will eventually be discovered.

Chapter 4

Architectural forces

4.1 Goals

Business driver: Trucking in Russia is mostly done using old and inefficient trucks with bad road conditions. It is not feasible to replace the vehicles or the road system, so in order to remain competetive, the STC must optimise its logistics instead.

Project goal: The STC wishes to optimise its logistics by keeping a detailed log of truck movement. By analysing the information in the log, more efficient transportation routes may be designed.

Project goal: In order to minimise truck idle time, the STC desires real-time information on truck locations, such that availability for further use can be easily predicted.

4.2 Constraints

- The company has very little in-house IT capacity, and does not wish to expand it much. In particular, it does not want to maintain its own servers.
- Since the servers must consequently be outsourced, the STS has to run in a generic, non-customised environment (such as a standard Linux server).
- The truck transmitters are very restricted, embedded hardware, that cannot run large and complicated software.
- Trucking in Russia is at an unusually high risk of hijacking. In order to not leak information about the locations of trucks, all communications has to be protected from eavesdropping, and all queries in the database must be authorised.

- As another safety-related concern, it must not be possible for a third party to falsify truck information.

4.3 Architectural principles

Principle reference	P1.
Principle statement	Redundancy of components
Rationale	The STS will be of critical importance in making real-time business decisions, so it is important that it is always available.
Implications	• System distributed across multiple nodes (for example, using multiple different Amazon EC2 instances across multiple availability zones). • Comprehensive error handling in all levels of the system. • Load balancing to handle node failures. • Redundancy at the data level, for example through replication.

Principle reference	P2.
Principle statement	Use of open source components
Rationale	Open source components will be used where available, in order to reduce development effort and cost.
Implications	• Able to modify external components, if necessary.

Principle reference	P3.
Principle statement	Encryption of communications
Rationale	To prevent outsiders from gaining knowledge about truck locations, all communications across external networks must be encrypted
Implications	• An encryption scheme must be decided upon. • We don't have to worry about security of the link layers.

Principle reference	P4.
Principle statement	Cryptographic signing of truck communications
Rationale	To prevent third parties from impersonating truck transmitters, all communications from trucks must be signed using assymetric cryptography.
Implications	• The cryptographic keys must be managed. • If a key is leaked, there must be a procedure for changing them.

Principle reference	P5.
Principle statement	The information retrieval API is REST-oriented
Rationale	REST APIs are easy to interface with existing HTTP protocol stacks.
Implications	• We need visible HTTP servers. • A protocol for how to represent the data in textual form must be decided.

Principle reference	P6.
Principle statement	Mesh networking by trucks
Rationale	In order to extend range beyond cell phone networks, short-range mesh networks between truck transmitters is used to indirectly transmit to STS.
Implications	• A truck can relay information about any number of other trucks. • Every piece of truck communication must be identified by the original sender, not the truck that managed to contact the STS.

Chapter 5

Architectural views

5.1 Context view

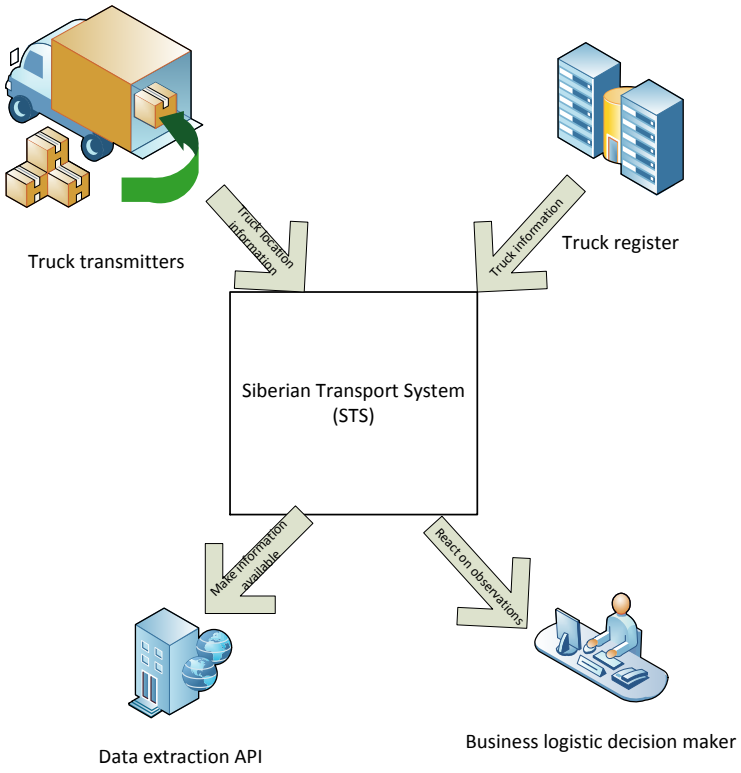

Figure 1. Context diagram of the Siberian Transport System (STS)

5.1.1 Context diagram

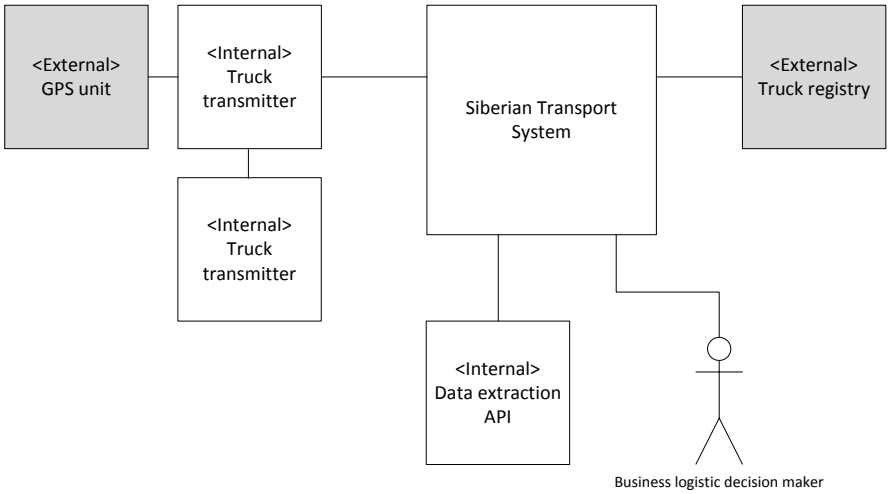

Figure 2. Context diagram with information of externat and internal parts

- The truck transmitters consist of a GPS unit, that gathers the trucks coordinates, and a sender unit, which will exchange truck positions with other truck transmitters through a mesh-network, or send the coordinates to our servers through mobile phone network. The hardware (including firmware for doing the actual GPS readings) is a commodity-bought external system, but the software on the sender unit is internal.
- The truck register is a system in which the Siberian trucking company's trucks are registered. All the trucks will already be registered in this system, so data is imported from here to STS. In the scope of our project, the truck register is an external system. The trucks register sends a message to STS whenever a truck is added to the fleet, removed from the fleet, or reaches one of STC's central depots.

5.1.2 Interaction scenarios

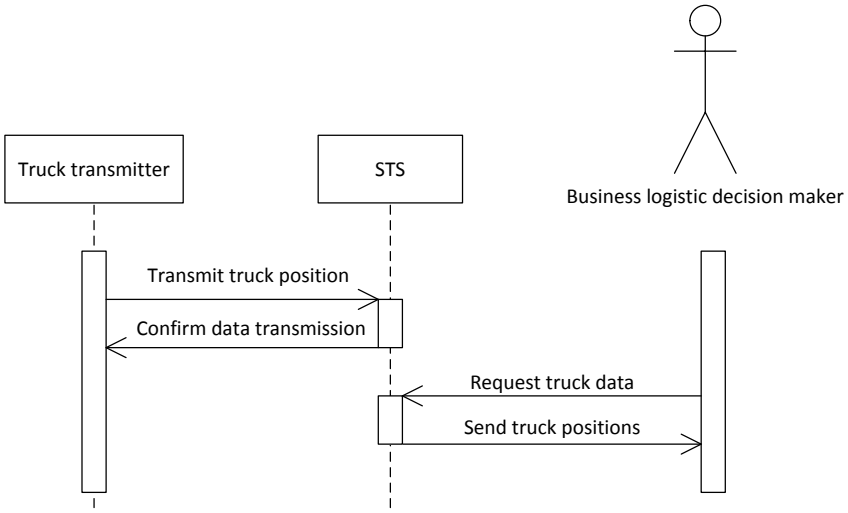

Figure 3. Truck transmitters send positions to the server. Later on, a user can make a query to see specific trucks or to see all trucks that fall behind schedule

5.2 Functional view

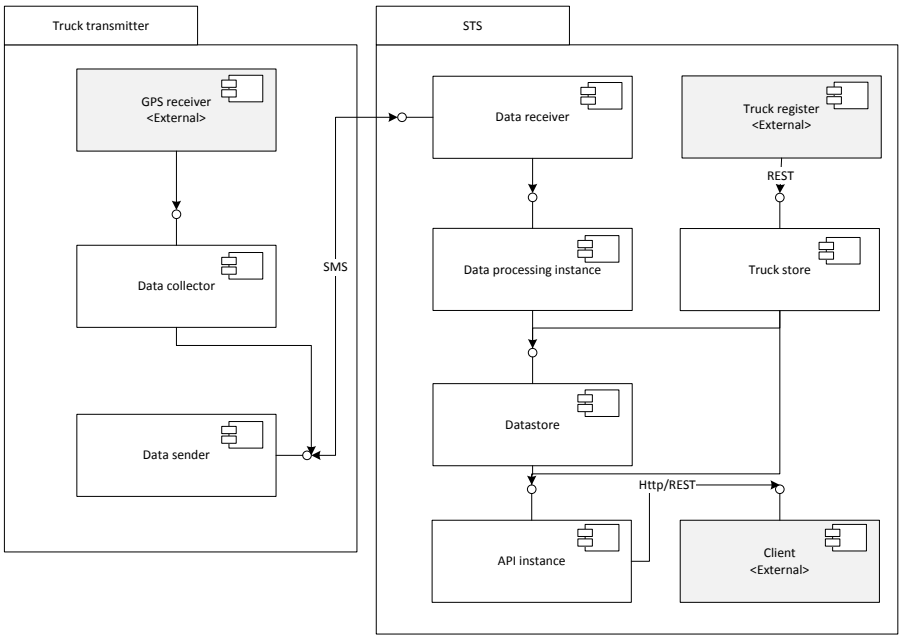

Figure 4. The figure shows the functional view for the truck transmitter and the Siberian Trucking System (STS)

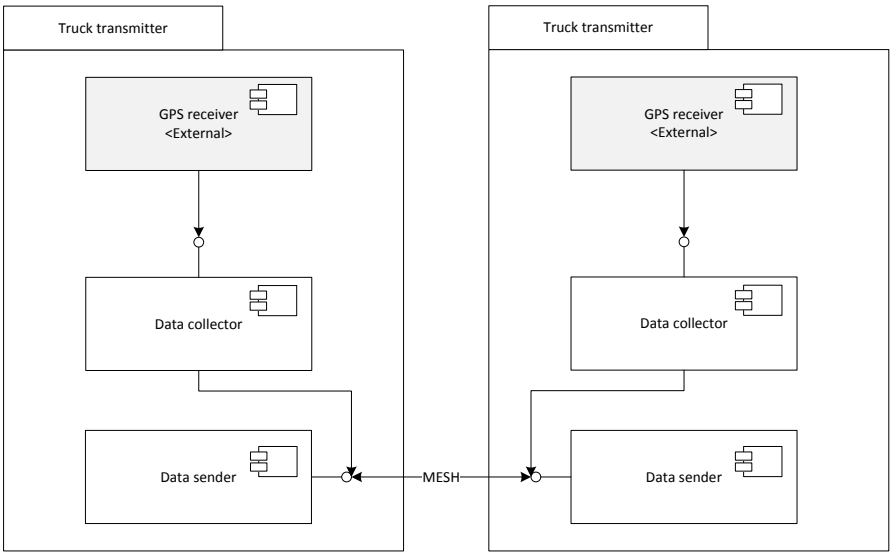

Figure 5. The figure shows the functional view for data transfer between two truck transmitters

5.2.1 Functional elements

Element name	GPS receiver
Responsibilities	acquire the current position of the truck
Interfaces – inbound	none
Interfaces – outbound	Data collector

Element name	Data collector
Responsibilities	Gather those positions that should be sent to the server
Interfaces – inbound	GPS receiver
Interfaces – outbound	Data sender

Element name	Data sender
Responsibilities	send truck positions encrypted via SMS to the server and get confirmation from the Data Receiver
Interfaces – inbound	Data Collector, Data Sender (MESH), Data Receiver (SMS)
Interfaces – outbound	Data Sender (MESH), Data Receiver (SMS)

Element name	Data Receiver
Responsibilities	Gather the positions from all the trucks, decrypt the messages and send confirmations to the trucks, when the data is received
Interfaces – inbound	Data sender
Interfaces – outbound	Data sender and Data processing instance

Element name	Data processing instance
Responsibilities	Send data to the datastore
Interfaces – inbound	Data Receiver
Interfaces – outbound	Datastore and Truck store

Element name	Datastore
Responsibilities	Store all relevant data from the trucks
Interfaces – inbound	Data processing interface, Truck register
Interfaces – outbound	API instance

Element name	Truck store
Responsibilities	Cache of the trucks
Interfaces – inbound	Truck register
Interfaces – outbound	API instance

Element name	API instance
Responsibilities	Make data accessible for other external applications
Interfaces – inbound	Datastore
Interfaces – outbound	External client

5.2.2 Functional scenarios

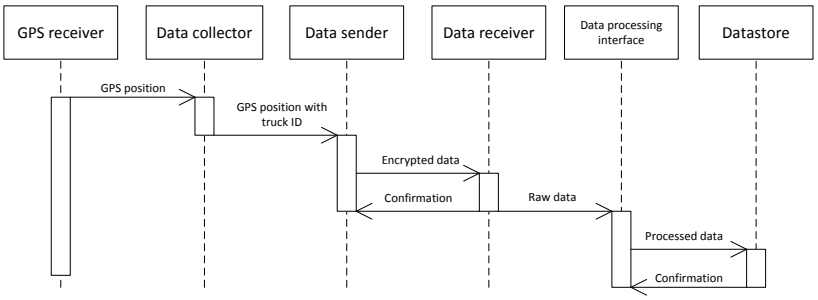

Figure 6. Truck transmitters send positions to the server and gets a confirmation when the data is received

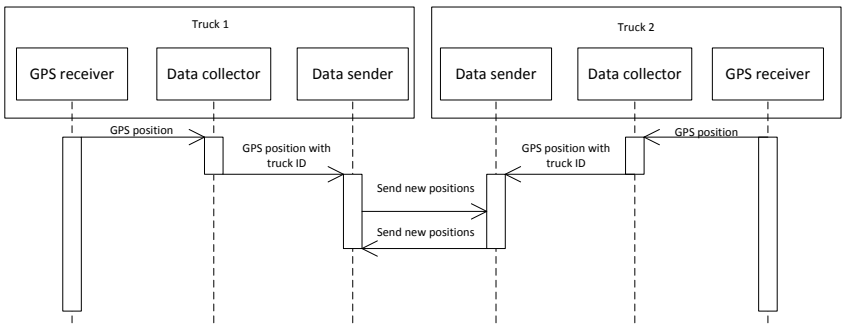

Figure 7. Truck transmitters exchange positions with other truck transmitters via the mesh-network

5.2.3 System-wide processing

The transmitters will often be out of reach for some network, therefore it is essential to be able to store and send the gathered data at a later point, when the truck again is within range of a network. When the transmitter sends data, it awaits a confirmation from the server before the data is registered as sent in the truck transmitter. If a transmitter hasn't sent a position in a considerable amount of time, when the truck is expected to be at a depot within transmission range, then a notification is raised in the system, to make a technician aware of a possible problem with the trucks transmitter.

5.3 Information view

The GPS receiver transmits GPS coordinates along with a timestamp. This information is transmitted to the data collector, which filters and aggregates the information, as well as tagging it with a truck ID and signing it cryptographically (see the security viewpoint in 6.2). These aggregations are sent to other truck transmitters in range, or to the STS via the data sender if possible. When the data has been successfully transmitted to the STS, it is deleted from the truck transmitter.

When truck position data is received by the STS, it becomes the property of the system. If it is to be saved (that is, if cryptographic checks affirm its validity and the truck ID is of a known truck), it will be stored in the data store component. Data is never deleted once stored.

When a truck is added or removed via the truck register, the truck ID is stored or deleted from the truck store. This data is always accessible to the data processor and must be up to date. Data never flows from the data processor to the truck store.

The user-facing API uses data from the data store and provides an aggregated view to its callers. That is, data flows from the data store to the API, never the other way. Hence, the API does not allow modification of data, it is read-only.

The data store is an unstructured NoSQL database (SimpleDB) provided by the Amazon S3 platform. The truck store is a relational database using Amazon RDS.

Every HTTP request to the API is logged by the API provider. Every component failure is logged by the redundancy infrastructure mechanism.

5.3.1 Data structure

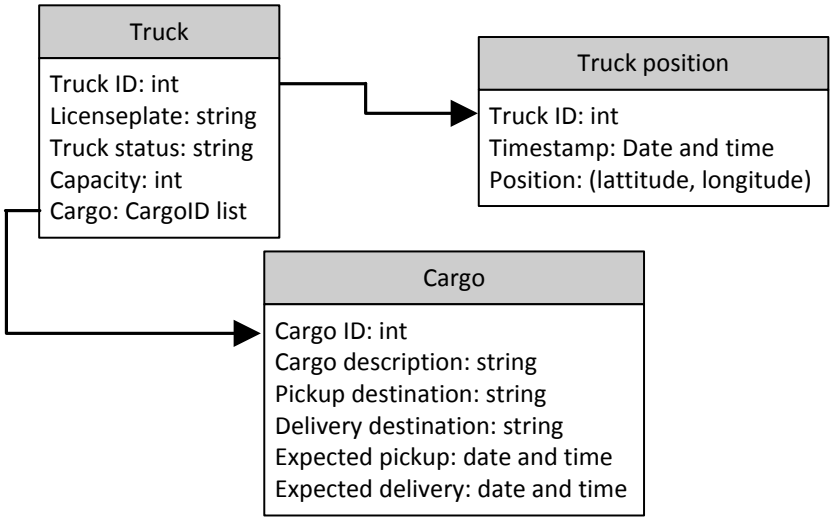

Figure 8. System data structure

The data store contains an unstructured set of tuples. Each tuple consists of a truck ID along with a timestamp and the trucks location at that time. The truck store contains the truck data the system uses, which is gathered from the truck registry.

5.3.2 Data flow

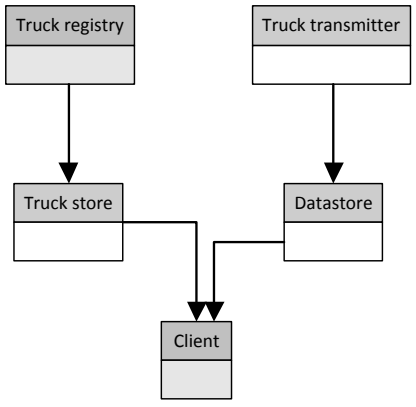

Figure 9. System data flow

Whenever a truck notification event is received from the truck register (for example, that a truck ID is added or removed), the truck store is updated

accordingly. Transmitted positions are kept in the datastore, and a client can access both via the API.

5.3.3 Data ownership

System	Truck transmitter	Data processor	API	Truck register	Data store	Truck store
Raw truck data	writer	updater	reader	none	master	none
Truck location	none	creator	reader	none	master	none
Currently known trucks	none	reader	none	writer	none	master

5.3.4 Information lifecycles

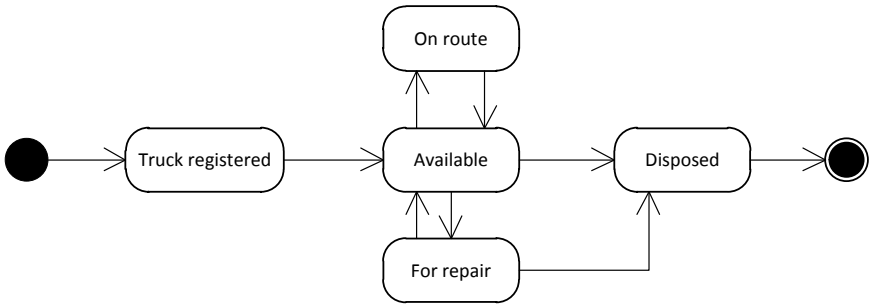

Figure 10. State diagram for a truck in the system

Truck information is gathered from the truck registry, which means that this diagram isn't directly a part of our system, but a part of the external "truck registry". These state transitions will however impact on our system too, as changes to the truck in the truck registry will propagate to the "truck store".

5.3.5 Timeliness and latency

There are few explicit timeliness or latency requirements. The unavoidable latency involved in communicating from trucks to the STS will dominate any other delays.

The only exception is the case where a new truck is added to the truck register. In that case, the knowledge that a truck has been added must be

propagated before that truck starts sending location information, or it will be discarded. This should not require any particular engineering effort in the system: the users can add the truck ID via the truck register several days before the truck is actually put into service.

5.3.6 Archive and retention

Records are never removed from the data store. No explicit archiving is necessary or performed, beyond that which is done automatically by the database provider (Amazon SimpleDB). Backups must still be performed, however, in order to fulfill availability scenario 4 (see section 6.3).

5.4 Concurrency view

Each physical truck runs a single truck transmitter instance, which contains three tasks: GPS reader, data collector and data sender.

The data and truck stores is distributed across several different processes in order to provide redundancy and scability.

There is a 1:1 relationship between trucks and Data Processor tasks. Each Data Processor is responsible for processing data from a single truck. Note that a “task” in this sense is not necessarily a physical operating systems thread (or process), but may be implemented as a lightweight (“green”) thread. If a Data Processor is unavailable (it may be crashed or the network may be down), the data receiver is responsible for starting up a replacement task.

The Data Receiver consists of a small (fixed) number of processes that merely route truck data to the proper Data Processors. If no task processor is running for the given truck, it must be started. If it fails, it must be restarted. See section 6.3 for more on availability.

Each API request results in a logical task responsible for carrying out the request.

5.4.1 Concurrency model

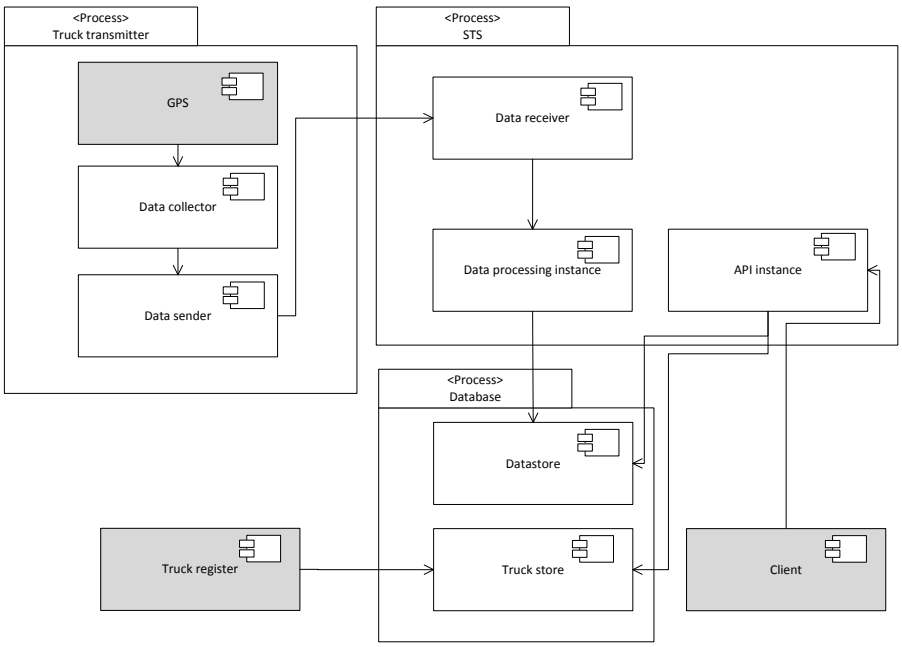

Figure 11. Concurrency model

The truck transmitter is one process, that consists of the Data collector and Data sender, which is placed on a physical device in the truck. Data is sent from the Data sender to the Data receiver. The Data receiver, Data processing instance and API instance all run on the server as individual processes. The Datastore and Truck store exists in a database process.

5.4.2 State model

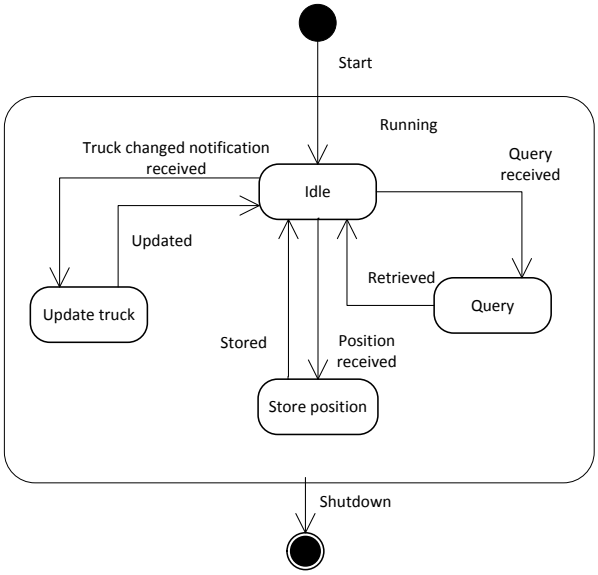

Figure 12. State model

When the system is running, truck changes will be registered in the truck store, positions can be written to the database and queries can be processed.

5.5 Deployment view

The STS will utilize Amazon EC2 for its data aggregation and processing, and the datastore will be implemented using Amazon's SimpleDB nonrelational database technology. The truck register in the system will utilize the Amazon RDS service to access its database. To simplify deployment, the STS will leverage the Amazon Elastic Beanstalk, an automated deployment utility that leverages Amazon's AWS services to provide an automatically load-balanced and scalable platform for development. The deployment will occur using a centralized Git repository hosted by Amazon, and new versions of the software will be automatically deployed when changes are pushed to this repository. Note that this is not the same as the *development* Git repository. Amazon's software and hardware infrastructure will ensure the correct deployment and scalability of the STS, as long as the code does not contain critical errors or bugs.

The STS will use Ubuntu Server 12.04 images on Amazon EC2, and all server-side code will be written in Python. The standard CPython Python interpreter will be used to run the STS, and all HTTP-based communications

will be handled using the Apache webserver. Since the STS will leverage Amazon's SimpleDB and RDS database platforms, no additional database software will be required on the EC2 instances.

5.5.1 Runtime platform model

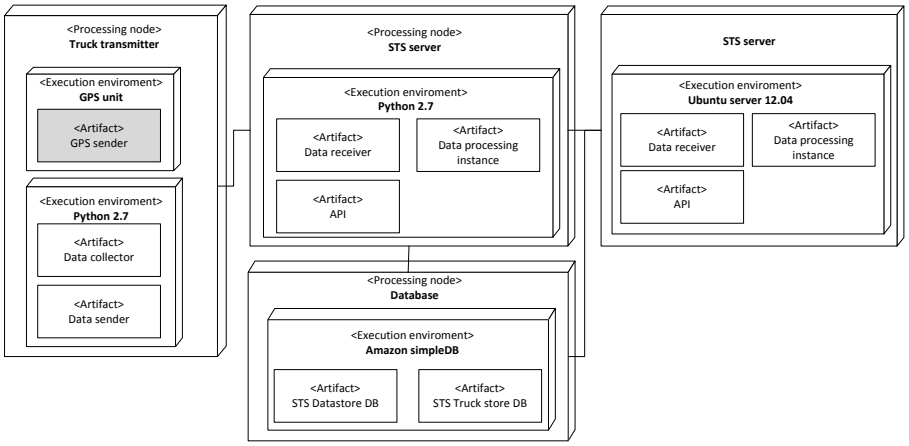

Figure 13. Deployment model

The truck transmitter gets data from an external GPS unit, and sends data to the server via SMS or MESH network. The server consists of an Amazon EC2 server running Ubuntu 12.04 and a Amazon simpleDB for data storage.

5.5.2 Software dependencies

Dependency	Version	Depends on
Ubuntu	12.04	
Python	2.7	
Boto (interface to AWS)	2.6.0	Python 2.6 or 2.7
PyCrypto	2.6	Python 2.x or 3.x

Additionally, various non-runtime systems will be needed for deployment and development, such as Git and Pip. The specifics of these are left at the discretion of the developers.

5.5.3 Network model

The physical network model is automatically managed by the Amazon WS platform. This implies that we cannot provide any guarantees beyond the

service level agreement (see section 6.3.1), and further robustness must be implemented at the software level.

Communication between trucks and the STS server is done through the mobile telephone network by sending SMS messages.

Communication between trucks is done through a short-range radio transmitter, forming a mesh network.

5.6 Development view

5.6.1 Module structure

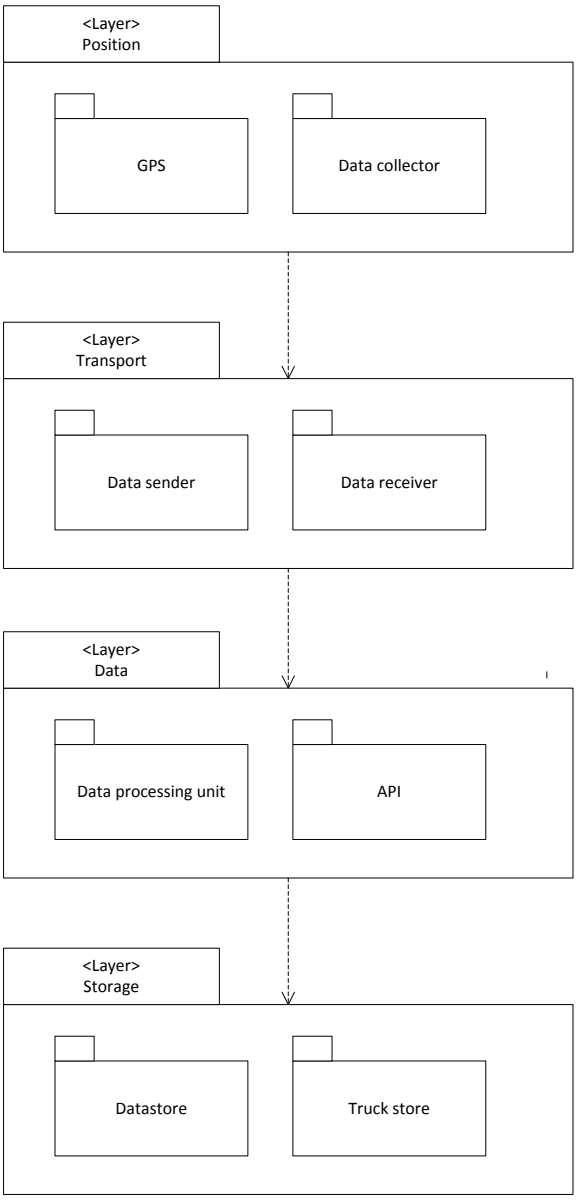

Figure 14. Module structure diagram

5.6.2 Common design

Logging and error-handling will be tracked using the standard Python logging module. The logging module on EC2 will be configured such that error logs from the STS will be saved to the Amazon SimpleDB service for review. Similarly, traces in the event of an exception will also be logged to SimpleDB. All data encryption within the STS will be handled through the PyCrypto library.

5.6.3 Standards for design, code, and test

The STS will be developed using the Python programming language, and each of the functionalities contained within the system will be separated into separate Python modules. The code will follow the guidelines set in PEP-8, the coding style recommended by the Python developer community. Every public API contained within a module of the STS must contained an appropriate unit test either embedded within its documentation using the doctest framework, or in a separate Python file using the unittest module provided with the Python programming language.

5.6.4 Codeline organization

Dependencies on third-party modules and libraries will be monitored using PIP, the Python package manager, and each discrete module in the STS must have a file named `requirements.txt` with an appropriately formatted list of modules and the minimum version number required for the proper functioning of the STS software. Deployment and version control will be managed through Git, and the deployment to Amazon EC2 will be handled through a centralized Git repository managed using Amazon Elastic Beanstalk. The STS modules will be separated into four directories: `static`, `src`" `docs`" and `libs`. The `static` directory will contain any static files related to the STS system, the "src" directory will hold the Python modules of the STS (each module will be contained within its own subdirectory with an initialization file), the "docs" directory will contain all documentation associated with the STS, and the "libs" directory will contain all of the third-party Python modules necessary to run the STS.

5.7 Operational view

5.7.1 Installation and migration

The installation of the STS will be handled through the Amazon Elastic Beanstalk platform. To install the STS, a new project must be created using

the AWS control panel. Once Amazon creates a central Git repository for the project, the STS code will need to be configured with the appropriate authentication details for the AWS account being used, including the SimpleDB and RDS databases being used. After the STS code is configured, it can be installed by pushing to the master branch of the Elastic Beanstalk Git repository. All further deployments of the STS may be handled through Git.

To separate the ongoing version control of the STS with its deployment, the STS development team will be required to push changes to two separate Git repositories. The first repository, hosted externally on Github will contain potentially-unstable development code. Once the code in this repository matures and is thoroughly tested, the changes can be easily pushed to the second repository hosted by Amazon. Due to the simplified branching and merging available within Git, bug fixes can be quickly and efficiently deployed to EC2, while larger changes that require more testing can be re-integrated later.

5.7.2 Operational configuration management

The configuration for the STS will be related to the implementation of AWS API's used by the system, including RDS, SimpleDB, EC2, and Elastic Beanstalk. The Python library Boto provides a simple interface for interacting with and configuring the AWS API's, so the configuration for the STS will be housed in a Python module named "configuration." This module should contain the relevant AWS API access keys and connection details for the SimpleDB and RDS database systems.

5.7.3 System administration

Amazon will provide all hardware-related systems administration, and EC2 provides a 99.95 percent uptime guarantee. Although hardware failure alone is unlikely, programmatic errors are possible, so systems administrators must maintain at minimum a daily backup of the STS data in the event of accidental data deletion or corruption. To ensure the optimal functioning of the STS, systems administrators must also regularly monitor the health of the system by using automated monitoring tools, such as New Relic.

5.7.4 Provision of support

The STC will have a team of IT professionals responsible for providing basic support for the STS. Any problems related to the general use of the system will be handled by this IT department. If the problem is found to be an issue with the STS itself, the issue will be escalated to the in-house development

team of the STC, which is in charge of maintaining and administering the STS system. If the problem is then found to be related to the infrastructure of the system, the development team will contact Amazon AWS support staff. In general, in accordance with the first constraint in section 4.2, the STS is intended to be self-managing and require little manual intervention.

Chapter 6

System qualities

6.1 Performance and scalability

In this section we present a table of the performance requirements for the expected scale of the system and reason by what means the requirements can be met and why these requirements are possible to uphold.

Requirement	How met
Average response time for the STS API should be less than 100ms under a load of up to 20 requests per second	Amazon Elastic Beanstalk will automatically start and stop additional EC2 instances according to the level of load experienced by the STS. The configuration of Elastic Beanstalk will ensure that the response time remains low.
Average response time for the Data Receiver should be less than 50ms under a load of up to 200 requests per second	the implementation of Amazon Elastic Beanstalk will mimic the implementation of the API, but Elastic Beanstalk will be configured to suit the Data Receiver and produce a sub-50ms response time.
The datastore should respond to queries for up to 10,000 separate data points in under 1,000ms, regardless of the total size of the database	the datstore will be implemented on top of SimpleDB, which has predictable performance characteristics that do not change based on the total amount of data in the database.

Verifying the existence of a truck in the truck store should require less than 15ms of time	the truck store will act as a cache for the truck register, and the truck store will be implemented on top of the Amazon RDS platform with aggressive and distributed in-memory caching.
Network latency between the datastore, truck store, data receiver, and API endpoints must be less than 5ms	AWS has latencies of approximately 1ms between different availability zones and less than 1ms when communicating within an individual availability zone.
Although the STS will be ready occasionally delayed SMS delivery, SMS messages from the trucks should ideally arrive within 5 seconds of being sent	the mobile operator or virtual network provider must guarantee a low latency for SMS messages. It is assumed that the STC can negotiate an appropriate contract.
Although the STS can appropriately handle the event in which an SMS is not delivered from a truck (no data will be recorded), SMS messages should ideally have a minimum of a 99 percent delivery rate	it is assumed that the STC can negotiate an appropriate contract with a network provider.
The server infrastructure of the STS must have at least 99.9 percent uptime	Amazon's AWS infrastructure guarantees 99.95 percent uptime.

6.2 Security

Much of the internal security of the STS server is delegated to our deployment infrastructure (Amazon WS). This section is mostly concerned with issues arising from the parts of STS using unsecured networks or interacing with the external world. The following table summarises our security requirements, we provide motivation for our requirements in subsections 6.2.1 and 6.2.2.

Requirement	How met
Truck position information must be kept confidential.	Position information must be encrypted whenever transmitted across unsafe networks (wireless meshes or the SMS network).
Truck communication must not be forgeable.	Every truck possesses its own unique private key. All positions generated by the truck will be encrypted with that key. The STS server will reject any truck position not encrypted with a known key.
Only authorised users may obtain truck information via the API.	An API key must be used to authenticate any requests.
Canonical truck position information must not be forgeable.	The truck register must prove its validity with a cryptographic signature.

6.2.1 Access control policy

Much of the system is automated, or only accessed by other systems, so the principals are generally not persons. Since we store very few kinds of data, and have only one user class, our access control policy is particularly simple.

Principal	Truck positions	Truck existence	Canonical truck location
API user	Read-only operations	None	None
Truck register	None	Write-only operations	Write-only operations
Truck	Write-only operations	None	None

6.2.2 Threat identification

Goal: Determine the location of a truck.

1. Extract information from the STS server itself.

1.1. Access the data store directly.

1.1.1. Guess database password.

1.1.2. Exploit software vulnerability.

1.2. Access data through API via obtained API key.

1.2.1. Crack API key.

1.2.2. Obtain API key from valid user.

1.3. Use social engineering to coerce a valid user to provide the information.

2. Intercept truck communication containing the position of another truck.

3. If you know the position history truck T_1, and communications from truck T_1 contains truck positions from truck T_2, the attacker can deduce that they must have met recently.

Handling: The truck IDs in truck positions are encrypted using assymetric key cryptography. Each truck has a unique keypair, the transmitter containing only the public key. Every truck position is encrypted with this key, and decrypted with the private key by the STS. Without the private key, you can tell neither which truck it is, nor where it has been.

We assume that the standard cryptographic algorithms will not be broken.

Goal: Forge location of a truck.

1. Obtain the private key of the truck and use it to send properly signed truck positions.

1.1. Gain physical access to the truck transmitter itself.

2. Access the data store directly.

Handling: Obtaining physical access to a hardware device is not easy. The data store uses a secure (standard) authentication mechanism.

Goal: Overload the mesh network.

1. Flood trucks with forged positions from other, fictitious trucks.

Handling: A truck transmitter must only store truck positions signed by a central key known to all trucks. This key is relatively easily obtained, by hijacking any truck in the fleet, and takes a long time to replace, since every truck transmitter must be modified. This is still acceptable, as this is only a mild denial-of-service attack.

6.3 Availability and resilience

There are three types of service, with associated service levels.

API requests: normal, no service.

Truck store: normal, read-only, no service.

Transmitting data from trucks: normal, STS down (trucks can still send to each other).

The services do not always function independently of each other. If the truck store is down, data cannot be received from trucks to the STS, as the validity of truck positions cannot be verified. On the other hand, if we take care that data, once stored, cannot be lost, and let truck transmitters retain data until the STS is able to store it, we will not suffer information loss.

The API requests only need to be under normal operation during business hours of the STC (8am - 4pm on business days) with an availability of 99.9 percent. The truck store must maintain normal operation at least 95 percent of the time during business hours of the STC, and must maintain at least read-only access 99.9 percent of the time during business hours. The STS server must be able to receive data from any active STC trucks with a 99.9 percent availability between 5am and 8pm, as it is expected, that most trucks will be operational in this period of time.

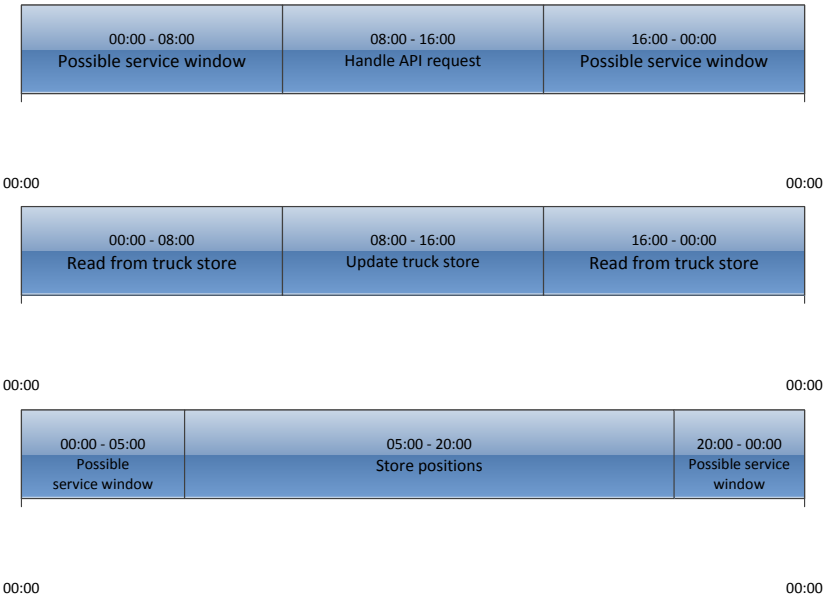

Figure 15. Availability of STS

The overall system defines the following service levels.

Normal operation: Data can be received from the trucks, new trucks can be added.

Autonomic operation: As in normal operation, but the API is not accessible and the truck store is read-only.

Read-only operation: Only the API is available. No new data is stored, but old data can still be queried.

No service: No part of the system is reachable or functional.

6.3.1 Platform availability

The STS is deployed on Amazon WS, using Amazon S3 for virtual machine instancs and Amazon EasyDB for data storage. According to the Service Level Agreement[1], we can expect 99.95% uptime per availability zone. This does not cover the reliability of an individual instance, but only our ability to start new ones in case an existing instance crashes. As a consequence, the STS must be able to automatically restart components running on virtual machine instances that become unreachable.

6.3.2 Availability scenarios

Scenario reference	AS1
Overview	Availability of truck position processing.
System environment	The system is operating normally.
Environment change	A data processor task fails.
Required system response	A new data processor task must be started.

Scenario reference	AS2
Overview	Resilience in the face of truck store failure
System environment	The truck store is operating normally
Environment change	The truck store becomes unreachable
Required system response	Data processors for already known trucks must keep processing data received from truck transmitters.

Scenario reference	AS3
Overview	Resilience in the face of STS failure.
System environment	The STS is operating normally.
Environment change	The STS becomes inaccessible to the truck transmitters.
Required system response	Truck transmitters must store data points locally, intelligently remove data points close in time, to conserve limited space. As soon as the STS is once again accessible, the data must be transmitted.

[1] `http://aws.amazon.com/ec2-sla/`

Scenario reference	AS4
Overview	Able to restore data store from backup.
System environment	The data store is operating normally.
Environment change	The contents of the data store are lost due to a technical fault at Amazon.
Required system response	The data store must be manually recreated using an externally stored (and recent) backup. Such a backup must hence exist.

Scenario reference	AS5
Overview	Able to recreate truck store from truck register.
System environment	The truck store is operating normally.
Environment change	The contents of the truck store are lost due to a technical fault at Amazon.
Required system response	The truck store is manually repopulated from the truck register.

6.4 Evolution

The STS is designed to meet an immediate, static business need of the company. As the STS is mostly a data aggregator, most new features will be implemented as new systems that consume data generated by the STS. Hence, we present few evolutionary perspectives.

Requirement	How met
1. It must be possible to add extra data processing and aggregation algorithms to the system based on the needs of the STC in the future	Data processing functionality will be separated into loosely coupled Python modules such that additional modules may be easily integrated.
2. It must be possible to add new ways of querying the data should the need arise	The datastore will have an index that can be updated to support more complex queries as necessary. Updating the index may take a substantial amount of time.
3. It must be possible to add other communication networks between trucks and the server	To do this, another type of communication device has to be physical installed in the truck and the data receiver should be expanded to handle this new type of communication. One way would be to have a class for the connection type, that could easily be replaced.
4. The system must be extended for marketing as a generic truck tracking system, usable by other trucking companies as well.	The STS needs to be system independent, so other companies can run the STS without having the same ”Truck registry” as STC. Some company specific setup will probably still be required.
5. Support for non-Amazon deployment platforms must be added.	Many of the response-time and uptime reqirements will be difficult to uphold. The architecture needs to handle loadbalancing and multiple servers, to be able to uphold the requirements.

Requirement	Type	Magnitude	Likelihood	Timescale
It must be possible to add extra data processing and aggregation algorithms to the system based on the needs of the STC in the future	Functional	M	H	H
It must be possible to add new ways of querying the data should the need arise	Functional	L	H	H
It must be possible to add other communication networks between trucks and server.	Functional	M	M	L
The system must be extended for marketing as a generic truck tracking system, usable by other trucking companies as well.	Integration	H	M	L
Support for non-Amazon deployment platforms must be added.	Integration	H	L	L

The STS has a very simple architecture, meaning that extending its functionality is easy. This was not an aim of the original design, however, but a fortunate consequence of its simplicity. On the other hand, we explicitly decided to make heavy use of facilities provided by Amazon's cloud platform, making future porting efforts very difficult.

6.5 Other qualities

6.5.1 Internationalisation

As our system is designed for a Siberian trucking company, we do not have any internationalisation requirements to meet. We do however have to be sure, that our data transfer method is compatible with the mobile phone network in the area. We use normal GSM phone network standards and send the gathered data via sms to the server.

6.5.2 Location

Our system is built for transportation in locations with very limited data connections. We have chosen Siberia as an example, but the system could be used in other remote locations, where truck transport is common and cellular network is limited. this could for instance be Alaska, India, China, Australia, etc. The only requirement the system has, is that the GSM cellular network is accessible in some parts of the trucks route in the given country.

6.5.3 Regulation

Although we have not included this as a requirement for the system, the STS can be used to check if drivers comply to ”driving/resting time” regulations for truck drivers in a given country. These data could be queried using the API and warnings could be given to drivers not complying with the regulation.

Appendix A

Architecture evaluation

The architecture evaluation is based on the requirements and scenarios from G2. We have made a priority of our scenarios and compared them to our requirements. This is shown in the following table. As we can see, R5 is never handled in the scenarios, which means, that we should probably have one more scenario for showing how one or more positions would be sent to the server. R5 is however still covered by the architecture, as data is always sent as a list, and a list can contain just one position.

Scenario	Priority	R1	R2	R3	R4	R5
FS1	High	Yes	No	No	Yes	No
FS2	Low	Yes	No	No	No	No
FS3	Medium	Yes	No	No	No	No
FS4	Medium	Yes	No	No	Yes	No
QS1	Medium	No	Yes	No	No	No
QS2	Low	No	No	No	Yes	No
QS3	High	No	No	Yes	No	No
QS4	Low	Yes	No	No	No	No

Scenario	Evaluation
FS1	The scenario upholds the architecture. It is however not described, that positions will still be stored in the truck transmitter, if STS is down.
FS2	It should be clarified, that we can only find trucks in an area, if the trucks can send their positions. Other truck might be within the area, but are out of reach of our communication networks.
FS3	This scenario upholds the described architecture.
FS4	This scenario upholds the described architecture.
QS1	This scenario upholds the described architecture.
QS2	This scenario has changed, as trucks exchange position via the mesh-network and sends the positions when one of the trucks enters a cellular network.
QS3	The scenario is described in the architecture
QS4	The scenario works in the described architecture, it is however not specified how the information is sent or who recieves it.

Appendix B

Architecture prototype

We have implemented a protype in Python, consisting of a prototype STS and a small program for simulating a fleet of trucks that move around randomly and occasionally meet and exchange information. We do not prototype the use of the cell phone network to exchange data, instead, the truck simulator uses an ad-hoc HTTP interface to send information to the STS. Our prototype also does not include logic related to the truck register, nor does it employ cryptographic verification of data. Our prototype models how information flows from trucks, into the data store, and is then accessible via an API. The prototype also includes an example implementation of the SimpleDB service as a backend for data storage, and the API is implemented on top of the Flask framework. As a result, the architectural prototype can also be used to evaluate the efficacy of the Amazon EC2 and SimpleDB infrastructure for the STS.

B.1 Running the prototype

Execute the following commands to run the STS prototype.

```
python src/runserver.py
python src/trucksim.py
```

API queries can then be made using curl, for example to list all truck positions within ten distance units of point (0,0):

```
curl localhost:5000/api/location/0.0/0.0/10.0
```

You may have to wait about thirty seconds for the truck simulation to send data.

Bibliography

Rozanski, N. and Woods, E. (2011). *Software Systems Architecture: Working with Stakeholders using Viewpoints and Perspectives*. Addison-Wesley Professional, second edition.