CHALMERS

Complex data structure terminal interface

Master of Science Thesis

TOBIAS ENGVALL

Chalmers University of Technology
University of Gothenburg
Department of Computer Science and Engineering
Göteborg, Sweden, September 2013

Complex data structure terminal interface

TOBIAS ENGVALL
© Tobias Engvall, September 2013.

Examiner: Sven-Arne Andreasson

Chalmers University of Technology
University of Gothenburg
Department of Computer Science and Engineering
SE-412 96 Göteborg
Sweden
Telephone + 46 (0)31-772 1000

Department of Computer Science and Engineering
Göteborg, Sweden September 2013

Abstract

Although somewhat outdated, green screen terminals are still a very common sight in the industry sector.
This report describes a system, which handles request for accessing an advanced data model, which is used for storing booking data related to the travel industry.
The developed system has been integrated into Amadeus' existing systems with good results.

Preface

This report describes the work and result conducted by the author on designing and implementing a terminal interface for a new type of travel industry booking data structure.

Contents

Acknowledgments

I would like to take this opportunity to thank everyone who has helped me with my work over the last couple of months: Thanks to everyone in the SBR/SBC team at Amadeus: Gilles, Luis, Olivier, Ivano and Valerie. Many thanks to my supervisor Bruno, who has been an invaluable mainstay throughout the project.

I would also like to thank my dear friend Martin Andersson for helping me accommodate to a life in France. Thanks also for teaching me how to appreciate the, somewhat different, French way of doing things.

Tobias Engvall Chalmers, October 2013

Chapter 1

Introduction

1.1 Background

An increasingly more important travel industry trend is to be able to provide so-called cross-channel functionality: It is a good thing, if a reservation made on a website is accessible through many different channels.
One of the most used channels for accessing data related to the travel industry – be it booking data, airplane inventories, availability, etc. – is through a terminal interface called the "cryptic channel". Terminals connected to this cryptic channel are commonly found at classical travel agencies or at airports.

Classical travel-industry bookings are usually contained in a standardized data structure that is known as a Passenger Name Record (PNR).
Amadeus has a very mature and feature-rich system for accessing and manipulating these kind of bookings. However, the PNR has some limitations: Since it was originally designed for airplane bookings, the PNR is very centric to the air travel industry. More functionality has then later been added on to it, in order to support other products and services.

The Shopping Basket (SBK) is a new kind of data structure which offers more features and better flexibility than the Passenger Name Records. Since it is designed from the start to be a very flexible data structure, it can be used in many different fields.

1.2 Problem Description

The above central SBK component is currently only accessible through structured interfaces. In this case: XML or EDIFACT. Only "heavy" applications, i.e. J2EE application servers, can use these interfaces to access the Shopping Basket content. Support for the classic interfaces needs to be added.

My part in this project, and the topic of this thesis, was to design and implement support for Shopping Baskets in what is known as the cryptic channel. The cryptic channel is commonly defined as a terminal interface, which is usually accessed by using terminal emulators, which provides text interfaces to the

underlying system.
The clients are usually dumb – all generation and presentation are done server-side – and most of them cannot handle screen resolutions higher than 80x24 characters.

- In order to adapt to these display constraints, some very condensed screen layouts needed to be designed and implemented for the Shopping Basket content. The Shopping Basket has well over 2000 different attributes and many different layers of relations between elements.
  The goal is to define and implement a system, which is intuitive and easy to navigate.
- The Shopping Basket repository resides in a complex architecture with many nodes of load-balancing, different entry-points, routing to different back-ends, etc. The service must be able to maintain a stateful connection through this.
- The user must be able to locate interesting Shopping Baskets for further navigation using the cryptic channel. One part of the project will be to create a system, which given a list of search criteria displays a well formatted screen, containing the matching Shopping Baskets. The screen should be able to display a short summary of each Shopping Basket.
- A well thought out command system must be designed, the existing cryptic commands have a reputation of being a bit too cryptic sometimes. My task was to try to design a nice syntax for the retrieve and look-up commands.
- Given a resulting screen, be it look-up or retrieve, the system must be able to further translate a navigational request into the corresponding follow-up request. Some kind of logic for keeping a context between messages must be put in place, in order to support stateful connections.

1.3 Amadeus

The initiator of this project is a company called Amadeus. Amadeus is a ”solution provider to the travel industry in the management and distribution of travel services”, which was founded in 1987 by Air France, Lufthansa, Iberia and SAS. Amadeus is a large company, their systems handled 499 million travel related bookings in 2006[1].

This project has been conducted at the main Amadeus development site in Sophia Antipolis, France.

[1]Source: www.amadeus.com

Chapter 2

Analysis

2.1 Background

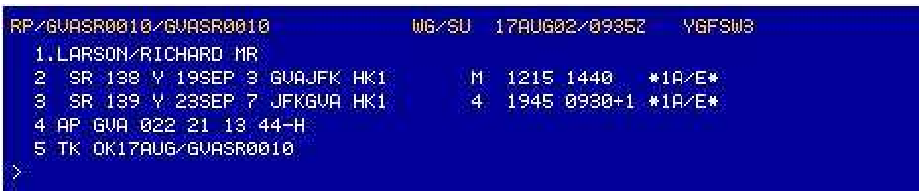
This chapter will present an in-depth background of the problem. In order to understand the decisions made by the author in the design phase of the project, a good understanding of both the historical as well as the functional background of the problem is necessary.

The beginning of the report will be used to describe the functionality of the terminal system used by the current travel industry data structure, which is known as a Passenger Name Record. Afterwards, a more advanced data structure will be described, this data structure is known as the Shopping Basket, it's differences compared to the PNR, the motivation behind it, and why my work on it is important.

2.1.1 Passenger Name Records

A *Passenger Name Record* (PNR) can be defined as a computerized travel industry data entity containing information related to a booking of some type. It is standardized by *the International Air Transport Association* (IATA) and was originally designed to allow the large reservation systems at the time to be able to exchange data with each other.
The term *PNR* may also refer to the nine-digit alphanumeric sequence called a Record Locator, which is used to uniquely reference a Passenger Name Record.

The Passenger Name Record can contain all sorts of travel-related information:



```
RP/GVASR0010/GVASR0010           WG/SU  17AUG02/0935Z    YGFSW3
  1.LARSON/RICHARD MR
  2  SR 138 Y 19SEP 3 GVAJFK HK1       M  1215 1440   *1A/E*
  3  SR 139 Y 23SEP 7 JFKGVA HK1       4  1945 0930+1 *1A/E*
  4 AP GVA 022 21 13 44-H
  5 TK OK17AUG/GVASR0010
>
```

Figure 2.1: An example of a Passenger Name Record

The itinerary, the passengers, the products, ticketing and fare details, contact information, frequent flyer information etc.
The PNR was originally designed for air bookings, which also is its primary area of use. It is a flat data structure, which can be represented as a list of key-value pairs.

In order for a proper Passenger Name Record booking to be stored in the system, at least five parts of the record must be set:

- The name of the person making the booking.
- All passengers.
- Itinerary for at least one travel-sector.
- Ticketing information, either a ticket number if it has been issued or a deadline for the issuing of a ticket.
- Contact details of the travel agent.

During the booking process, if all of the above restrictions are met, the travel agent may choose to issue what is known as a *End of Transaction* (EOT). If such a command is issued, the system will then assign a Record Locator to the PNR, causing it to be stored in the underlying system.

Passenger Name Records are contained in what is known as *Global Distribution Systems*[1] [2] (short: GDS). There are a few Global Distribution Systems on the market: Amadeus is currently one of the largest system, other large systems are *SABRE*, *Worldspan* and *Gallileo.*

2.1.2 The Shopping Basket

The Shopping Basket models a new way of storing travel-industry related information. It is not designed to completely replace the Passenger Name Records, at least not within the foreseeable future . Instead it may reference external information, i.e. data from a Passenger Name Record, at any place in the data structure.

Differences from a Passenger Name Record

The Shopping Basket data structure was designed to be a more flexible data structure than what the PNR currently is. Contrary to the Passenger Name Record, the Shopping Basket is not a flat data structure: It can store it's information grouped into what is known as *records.*
It also differs from the Passenger Name Record in that it can contain heterogeneous information without requiring itself to be split into multiple records, which the PNR requires. However, the SBK is also designed to be able to act as a standard PNR, if the situation requires it to.

[1]Ref: http://en.wikipedia.org/wiki/Computer_Reservation_System
[2]Another common name is *Computer Reservations System* (CRS)

The SBK is a very generic data structure. Where the PNR only can reference airline products as well as airline related products such as rental cars and hotels, the SBK offers the possibility to support more kinds of products.

- "Support for all known products, and also all the others" is something which is mentioned in the specification. This is a sentence, which sums up the whole idea of the SBK quite nicely.
- Third party users of the system should be able to store generic information in the SBK. E.g. a hotel business might want to be able to store data specific to their billing system in a SBK.
  This is made possible, by letting the third-party agents store their application specific data in raw data blobs, which are available throughout the data structure.

Records

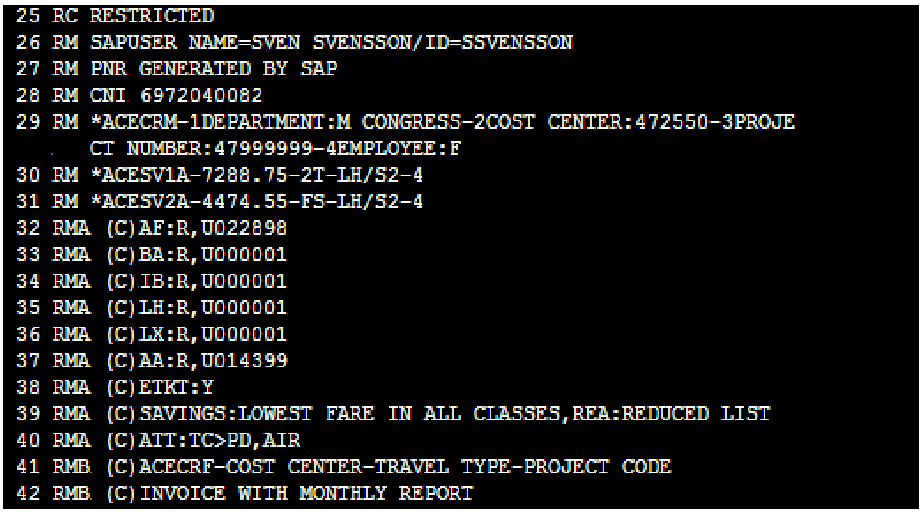
As mentioned earlier, a nice feature of the the Shopping Basket is that it supports the ability to group its different entities into something, which are known as records. As any other entity living in the SBK, records are standard, uniquely identified, SBK elements. This also means, that a grouping of elements can be the target of consuming relations, fares, fees, etc.

Since the records in a way can be seen as the graphical layout of the SBK, they are very important and central to this project. Throughout the terminal interface design, records play a central role in the navigation and visualizing of a SBK and its elements.

The motivation behind using records is two-folded. First and foremost, it is always a very useful thing to be able to group things up for better structure and general tidiness. Some applications of over-usage of the PNR (see figure 2.1.2) can be seen as a bad example of where limited grouping abilities is responsible for redundant information and clutter. Another nice feature of records is that they allow the SBK to fully model all steps of a booking process. This is something, which the PNR is unable to do.

A quite common use case scenario is, when a potential customer walks in to a travel agency and indicates interest in booking a trip to a country, e.g. Thailand. At this moment, the travel agent does not know any specific information about the trip, since the customer has not provided any information on possible places in Thailand to visit. He has neither expressed any preferences, on which airline he wants to fly with. At this point, the travel agent can create an empty shopping basket and specify these very generic wished as a remark in the SBK. He can then at a later point create a couple of possible travel packages, which he represents as a list of SBK records of type "offer". The customer can then later on decide on a package he likes, which the travel agent then converts into a full reservation.

A Shopping Basket record can be of many different types, each type represents a different booking related scenario. The most commonly used ones are:



```
25 RC RESTRICTED
26 RM SAPUSER NAME=SVEN SVENSSON/ID=SSVENSSON
27 RM PNR GENERATED BY SAP
28 RM CNI 6972040082
29 RM *ACECRM-1DEPARTMENT:M CONGRESS-2COST CENTER:472550-3PROJE
      CT NUMBER:47999999-4EMPLOYEE:F
30 RM *ACESV1A-7288.75-2T-LH/S2-4
31 RM *ACESV2A-4474.55-FS-LH/S2-4
32 RMA (C)AF:R,U022898
33 RMA (C)BA:R,U000001
34 RMA (C)IB:R,U000001
35 RMA (C)LH:R,U000001
36 RMA (C)LX:R,U000001
37 RMA (C)AA:R,U014399
38 RMA (C)ETKT:Y
39 RMA (C)SAVINGS:LOWEST FARE IN ALL CLASSES,REA:REDUCED LIST
40 RMA (C)ATT:TC>PD,AIR
41 RMB (C)ACECRF-COST CENTER-TRAVEL TYPE-PROJECT CODE
42 RMB (C)INVOICE WITH MONTHLY REPORT
```

Figure 2.2: PNR remarks containing structure data.

- *Reservation records*, which in a sense contains all the information needed to create a booking. This is the PNR equivalent of the SBK.
- *Offers* – Represents a virtual booking.
- Requirements – Used if a record requires something else in order to be valid.
- Views – Similar to views in databases, it might be useful to create a grouping of elements for display use.

Relations

Contrary to the Passenger Name Record, the Shopping Basket also offers a way to model different kinds of relations between elements. Relations can play a quite central role in a booking scenario. E.g. a normal flight can be modeled as a consuming relation between an air product an a customer. A fare can then be attached as a faring relation to the consuming relation.

Many different kinds of relations exists, among them we find:

- Possession - This relation indicates that an entity owns another entity.
  $RecordA = POSSESSES => CustomerB$
- Consumption - Models a consuming relation, e.g. a flight.
  $CustomerA = CONSUMES => FlightB$
- Extension - This is used to express a situation, when you want to extend an entity with something else.
  $SpecialServiceRequestA = EXTENDS => (CustomerA = CON => FlightB)$

2.1.3 Existing interfaces

At the time of the project start, the only way of accessing SBK:s was through structured XML-messages. A look-up request was sent as a dedicated XML lookup message, the lookup-results were calculated, and the response is sent back as an XML response message.

2.1.4 Passenger Name Record Displays

The Amadeus terminal booking system is very large and complex. Travel agents are able to access information about almost anything related to the booking industry through the terminal. Different subsystems exists for:

- Flight bookings.
- Rail related products.
- Rental car bookings.
- Hotel related information and bookings.
- Ticketing

and many more.

A flight booking through the classing Passenger Name Record interface is initiated when a travel agent performs the cryptic commands to add something to a PNR. The classic start command is to add a passenger:

```
NM1ENGVALL/TOBIAS MR
```

This commands adds one passenger. The *NM* stands for name and the digit expresses how many passengers will be added. The passenger's name and title follows.

The terminal system is based on the old IBM 3270 series of terminals. They commonly support screen sizes of 80x24 characters.
Although the old 3270 terminals may seem like very crude and outdated communication devices, they do have their advantages. One of the advantages of the cryptic terminal displays is, that they are blazingly fast to use. At least after you get past the initial difficulties of understanding how the commands are used and how to interpret the cryptic screens. Although the learning curve may be quite high, many users of 3270 systems regard them as superior to new, modern systems based on a graphical user interface.

However, some of the functionality in the cryptic channel has a gotten a reputation of being a bit too cryptic. The goal of my design is to be as self-explanatory as humanly possible, without lacking details or functionality.

2.2 The Target User

The targeted user for this project is travel agency professionals, airport employees, and other people related to the travel industry. What they all have in

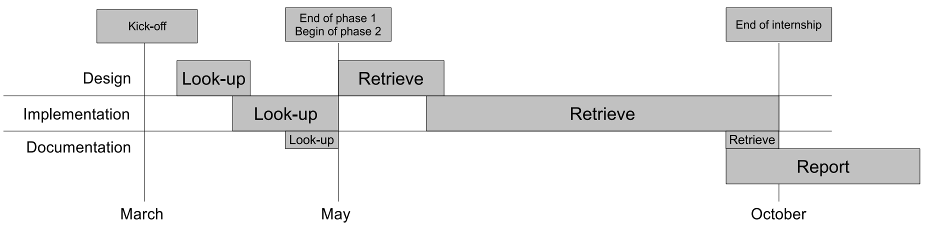


Figure 2.3: High level time plan

common, is that they have received proper training and they are well educated on the underlying systems, how they are used, and the meaning of the displayed attributes. This allowed me to focus more on designing a system, which is fast to use.

2.3 Time plan

An initial time plan was developed at the start of the project. Since the nature of the design naturally divides the problem into two parts, with the second part being more complex than the first part, my plan ended up with using the design and implementation of the first part, the look-up service, as a bit of a "hello world"-application in order to get familiar with the underlying application server and the internal libraries.

Chapter 3

Design & Implementation

3.1 Methodology

This project has been developed using an iterational development model: An initial implementation of a minimal service has then been further extended to include the features of the original specification.

A weekly status meeting has functioned as a way of tracking the progress of the work. It has also been used to define the upcoming tasks. Meetings has also been conducted with people from the product definition department at Amadeus. Their comments have been invaluable in the design phase of the project.

3.2 Information flow

In this chapter, I'm going to describe the information flow of the system in detail. The goal of the chapter is to provide a clear, detailed view of how a message is received and how the corresponding cryptic screen is generated.

The cryptic terminal system is built up by 7 subsystems:

1. *The service entry point*, which receives messages and orchestrates the underlying logic.
2. *The message interface*, which encodes/decodes messages.
3. *The parser*, which classifies the request.
4. *The context*, which stores data between messages of the same session.
5. *The screen manager*, which handles scrolling and clipping.
6. *The cryptic look-up service*, which performs a lookup and generates the look-up results screen.
7. *The cryptic retrieve service*, which handles the displaying of SBK entities.

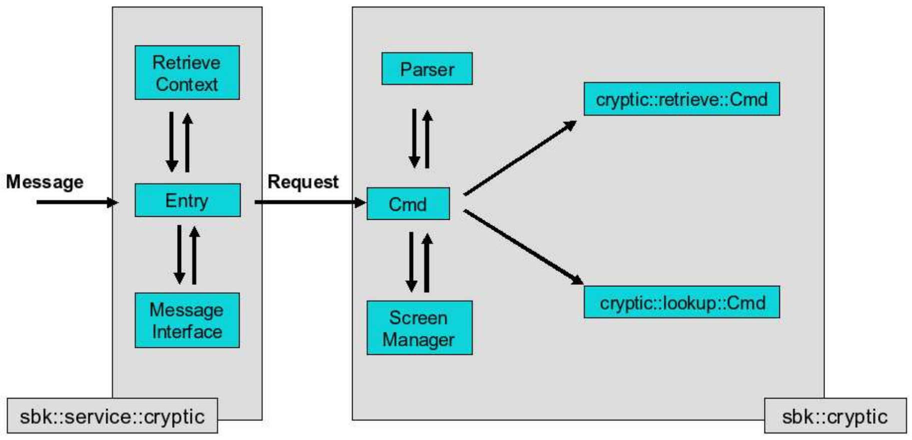


Figure 3.1: Cryptic terminal system overview

3.2.1 Entry Point

A terminal session begins with an EDIFACT request message, which is sent from a terminal to the SBK back-end through a *service integrator* (see: **??**).
The entry point then decodes the incoming request using the incoming message's corresponding message interface.
It then retrieves session data which is held in a context. This data, along with the incoming message, is then fed onwards to the cryptic screen generators.
After the response has been generated, the entry point sends the response back to the terminal and the generated screen is displayed to the user.

3.2.2 The Context

The cryptic service is a stateful service, which means that it is able to keep a memory with useful information between the messages in a session. Since the service can be spread out on many different back-ends, a separate *context server* is used to store the information. The memory, which I call the context, is created, when the first message of a session is received. On sub-sequential messages, the entry point retrieves the previous context from the context server before starting the message handling.

A session is defined as a series of messages, sent between the terminal and the service. Whenever the user presses enter on his/hers terminal, a message is sent to the service. A simple timeout mechanism prevents the context from being flooded with old connections.

In order to support the desired functionality, three parts needed to be stored between messages were identified:

1. The system stores a history – or command stack – of previously entered commands to the service. Navigational commands used for scrolling like MD or MD are not stored in the context, it only deals with full cryptic commands.

2. In order to support the above mentioned scrolling functionality, the system stores the previous position of the screen in the context.

3. The main purpose of the context is to enable support for *follow-up requests*. This is solved by storing a mapping between *local ids* and full cryptic requests.

   Example: $DK/1$ is translated into a full cryptic retrieve request:
   $DKSBKID1234/T1005CUS$

The Message Interface

The message interface is responsible for retrieving and filling data from and into the messages, which are used by the service.

The service is able to receive input from two different kinds of EDIFACT messages (see section 3.4.2). The two messages offers the same functionality, but through different channels:

- The *Hybrid Screen Format Request/Response* EDIFACT messages (short: HSFREQ / HSFRES) is a set of quite simple messages. The incoming cryptic request is stored as plain ASCII in the message. The response message also uses the same simple formatting.

- The *Terminal 3270 Request/Response* messages (short: TRMREQ / TRMRES) are very similar to the *HSFREQ/S* set of messages. The main difference is that the message data is stored as a 3270 stream the EBCDIC encoding.

The messages also contains information about the message origin. Such as requesting office, requesting agent, preferred language, and preferred currency. They also contain information about the requesting terminal: Screen width and height.

3.2.3 Parsing and pre-processing

After the request has been decoded by the message interface and the context of the previous context has been fetched from the context server, the parser is run on the cryptic request.
Since the same entry point is used for all kinds of cryptic request, the parser must be able to distinguish between and classify different kinds of requests. This also means, that my grammar must be non-ambiguous.

In the scope of this project, support for four different kinds of requests were included:

- Look-up requests: Given a list of search criteria, query the database and find the matching SBK:s.

- Full retrieve requests: A full request which specifies a displayable entity in the SBK.

- Navigational requests: My service also handles scrolling and clipping of screens. The user may issue commands to move up and down in a screen.
- Follow-up requests: This is when the request references another full request which is stored in the context.
  E.g. $dk/1$ translates into $dk/SBKID1234/T1005CUS$.

The current version of the parser is implemented using boost's regular expression library. Although an initial version was developed using a *LL parser* [1], which was deducted from a *EBNF* [2] grammar, the faster, more simple approach using the regex version, was eventually decided upon.

3.2.4 Screen generation

After the initial phase of decoding and pre-processing, the cryptic service calls one of the underlying screen generators. At the moment, two types of generators exists: One which handles everything related to look-ups and one which handles the displaying of any entity or calculated summary information related to a SBK.

[1]See http://en.wikipedia.org/wiki/LL_parser for more information.
[2]See http://en.wikipedia.org/wiki/Extended_Backus-Naur_form.

The screen generators are structured as follows:

- Look-ups
  - Look-up Screen
- Retrieve
  - Summary Screen
  - Record Screen
  - Product Screen
  - Customer Screen
  - Fare&Fee Screen
  - Forms of Payment Screen
  - Document Screen
  - Remark Screen

The look-up generator takes a SBK look-up result as input. Each retrieve generator takes an entire SBK or an element residing in a SBK as inputs.
Full cryptic displays – which are allowed to be longer than 24 lines – are generated and returned.

The Look-up Screen Generator

The look-up subsystem is a quite small and simple system. It can be split up into two major parts:

- A wrapper which calls the underlying SBK lookup logic.
- The screen generator which generates a cryptic screen given a look-up result.

The look-up logic receives the parse results along with the current context from the cryptic command class. It builds a list of search criteria from the parse results, which it then uses to call the SBK look-up service. The SBK look-up results are then fed into the look-up screen generator, which generates a cryptic screen.

The Retrieve Screen Generator

Because of the complexity and size of the SBK data structure, the retrieve system turned out to be the largest system of the ones developed in the project. It is able to generate a cryptic representation of any SBK entity worth displaying. It is also able to calculate and display summary information of a SBK.

The retrieve system is built up by the following subsystems:

- A *command class* which orchestrates the screen generation. This class takes a parse input and a context as input. It then calls the SBK retrieve service in order to receive an SBK. Depending on the type of the request, it then either feeds the entire SBK or a SBK element into one of the many screen generators.

- Given an input, a set of screen generators which generates full cryptic screens.

Since the system is based upon navigation using local IDs, the screen generation had to be performed in three passes:

1. The first pass identifies and fills a local representation of the screen. i.e. references to full SBK elements are placed in lists as they are going to appear in the final screen.

2. Local ids are assigned to each element which is capable of being further navigated. If an item references another item by, e.g. a consuming relation. A representation of the consuming relation is generated using local ids:
   Example: $Customer1 = CON => Product3$ yields:

   ```
   C1>P3
   ```

3. The full cryptic display is generated in standard ASCII.

The retrieve service uses a simple library called the *panel toolkit*, which provides a simple way to position text in a screen buffer.

Records

The record screen uses a table based approach: All elements contained in the record are displayed as one-line entries in its corresponding table. I have tried to keep the table display as homogeneous as possible. Common information like relations between elements are displayed at the same place regardless of the type of element.

A quite comprehensive set of helper classes has been developed, in order to ease the design of the record and summary screens. The most commonly used helper classes, are the cryptic lists. SBK elements may be added to these cryptic lists. The programmer may then specify the position of the list, and then add it to his cryptic screen. The lists will then generate a cryptic representation of a list of SBK elements.

Leaf entities

Leaf entities are entities which does not contain other entities. A prime example of a leaf entity would be a document. These screens all follow the same design principle: Display the information in a structured way and display the most important information at the top of the screen.

These entities may contain lots of information, structured in a complex way. An example of this would be, that a fare may have numerous associated fees and taxes.

3.2.5 Navigation

The navigation in the system is handled by keeping a stack containing the latest whereabouts of the user. In addition to navigating into something which is already displayed in the current screen, the user may choose to navigate to previously displayed screen. This navigation stack is kept in the context which is stored in the context server.

3.2.6 Scrolling & Clipping

Since the screen generators returns full cryptic screens, a class which handles clipping and scrolling needed to be implemented. Since the previous screen position is stored in the context. This is a quite trivial class.

Since the screen generators returns full cryptic screens, support for clipping and scrolling was implemented as a small set of quite trivial classes. These classes makes use of the previous screen position, which is stored in the context.

It currently handles:

- *Scrolling* – Whenever a navigational request is received, the last known full cryptic request is fetched from the context, parsed, and executed. The screen manager is then fed the full cryptic screen along with the previous screen position, which is available through the context. It then updates the current position given the resolution of the client terminal.

- *Wrapping* – The terminal system is designed to fit well into terminals that supports a maximum column resolution of 80 characters. However, there still exists systems, which only supports resolutions less than 80 characters. Thus, the screen manager wraps lines longer than 80 characters.

- *Clipping* – Until this point of execution, the system has only worked with full cryptic displays, i.e. displays, which may be larger than the maximum resolution of the terminal. The screen manager clips the screen to fit the terminal's resolution.

The system then returns a plain ASCII representation of the scrolled, wrapped and clipped screen.

3.3 Tools used

This sections will be used to describe some of the tools, which were used througout the development of the project.

3.3.1 The C++ Programming Language

Since the architecture around my service is developed using C++, the choice of C++ as the implementation language felt natural.

3.3.2 Boost::spirit

An initial version of the cryptic service used a LL-parser[3] based system to handle the requests. Although the idea was later discarded in favor of a more simple regex-based solution, I still think it is a more elegant solution to the problem.

The boost::spirit library allows you to create a parser using a EBNF grammar. The main difference between boost::spirit and other EBNF based parser systems is, that boost::spirit lets you define the parser as EBNF in C++. This is accomplished by heavy use of templates and operator overloading. Other EBNF parser systems often requires you to *generate* a parser from a source EBNF file.

3.3.3 Boost::regex

The second version of the cryptic command parser was written as a series of regular expressions using the boost::regex library.
Although this version of the parser turned out to be quicker and more simple than the boost::spirit version. It also turned out to be less extensible.

3.3.4 Valgrind

Valgrind is a profiling/debugging tool which proved to be an invaluable asset during the implementation phase of the project. It's main purpose is to help finding bugs related to memory-management, such as memory leaks and stack corruptions.

3.4 Architecture

This section will describe the underlying software and technologies used by the service.

3.4.1 Open Transaction Framework

The service is built on-top of an, internally developed, application server called OTF. Besides usual application server functionality, it also provides support for sending and receiving EDIFACT and XML messages.

3.4.2 EDIFACT

The *United Nations/Electronic Data Interchange For Administration, Commerce, and Transport*(UN/EDIFACT) is an electronic data interchange standard developed by United Nations. It is also standardized by the *International Organization for Standardization*[4] as ISO 9735.

EDIFACT is a hierarchical structure. A set of *elements* builds up a *composite*. A repetition of composites builds up a *segment*, which is what a *message* consists of. These entities also have a set of associated restrictions: Entities may be mandatory or conditional, repeatable or non-repeatable.

[3]http://en.wikipedia.org/wiki/LL_parser
[4]http://www.iso.org

The following is an example of an EDIFACT message handling flight availability requests:

```
UNB+IATB:1+6XPPC+LHPPC+940101:0950+1'
UNH+1+PAORES:93:1:IA'
MSG+1:45'
IFT+3+XYZCOMPANY AVAILABILITY'
ERC+A7V:1:AMD'
IFT+3+NO MORE FLIGHTS'
ODI'
TVL+240493:1000::1220+FRA+JFK+DL+400+C'
PDI++C:3+Y::3+F::1'
APD+74C:0:::6++++++6X'
TVL+240493:1740::2030+JFK+MIA+DL+081+C'
PDI++C:4'
APD+EM2:0:1630::6+++++++DAä
UNT+13+1'
UNZ+1+1'
```

As you can see, the message carries significantly less overhead than what an XML equivalent of it would.

3.4.3 EBCDIC and the tn3270 Communication Protocol

IBM 3270 is a series of terminals developed by IBM in the 1970's. The classic terminals used green as a standard color, which also led to it's commonly used nickname: "*green screens*".
Communication to and from the cryptic service is done through the tn3270 protocol, which is a slight modification of the standard telnet protocol. It communicates through a structured data stream. The contents is inter-weaved with different kinds of control characters, in order to support text and cursor placement.

Tn3270 uses *Extended Binary Coded Decimal Interface Code* (short: EBCDIC) as its character coding. The *Jargon file 4.4.7* has the following to say about it:

> EBCDIC: /eb´s@·dik/, /eb´see'dik/, /eb´k@·dik/, n. [abbreviation, Extended Binary Coded Decimal Interchange Code]
>
> An alleged character set used on IBM dinosaurs. It exists in at least six mutually incompatible versions, all featuring such delights as non-contiguous letter sequences and the absence of several ASCII punctuation characters fairly important for modern computer languages (exactly which characters are absent varies according to which version of EBCDIC you're looking at). IBM adapted EBCDIC from punched card code in the early 1960s and promulgated it as a customer-control tactic (see connector conspiracy), spurning the already established ASCII standard. Today, IBM claims to be an open-systems company, but IBM's own description of the EBCDIC variants and how to convert between them is still internally classified

top-secret, burn-before-reading. Hackers blanch at the very name of EBCDIC and consider it a manifestation of purest evil.

3.5 Problems Encountered

3.5.1 First version of the screens

The first draft of the cryptic screen design were based on a design which relied heavily on indentation.
Although the design were able to give the user with a very nice overview of the SBK, the design lacked scaling, and the results were quite horrid for large SBK:s.

```
RESERVATION 1:
     EASYJET FL12345  NCE CDG  05MAY07:
          ENGVALL/TOBIAS MR
```

3.5.2 Setting up a bank account in France

On of the most challenging tasks I faced during my stay in France was setting up a bank account.

In order to succeed in this vital task, I had to provide the correct authorities with the following:

1. Two copies of my ”internship placement agreement”.
2. Three copies of my passport.
3. Two copies of my insurance policy.
4. Two copies of my cohabitant’s most recent electricity bill.
5. Two copies of a signed document explaining that I was in fact living in the same flat as my cohabitant.

After providing the documents above, together with a two-week-long mail exchange using recommended letters, I finally received my account details.

3.6 Testing

A set of regression scenarios which verified the system were also developed. This was done with help from a script which injected XML or EDIFACT messages to the service. A test scenario consisted of a set of messages to be injected to the service – the cryptic commands – and a set of corresponding expected replies.

To ease the development of these test cases, the functional test scenarios were written using the HSFREQ/S set of EDIFACT messages which contains plaintext ASCII data.
The testing of the conversion between ASCII and EBCDIC was handled by a set of special test cases.

Chapter 4

Results

This section is meant to provide a "tour" of the cryptic Shopping Basket interface. It is not meant to provide a complete specification of the screens, rather to provide an example of a normal use case scenario.

The sections are ordered in a similar way, that an end user would encounter the screens during normal usage.

4.1 Lookups

As mentioned in the time plan, the first sub-project I began working with after familiarizing myself with the environment, was to enable cryptic support for the Shopping Basket Look-ups.

4.1.1 Syntax

A cryptic look-up query is built up by a set of look-up criteria. A criteria is specified by encapsulating the attribute of the criteria within parentheses. The type of the criteria is defined by pre-pending the encapsulated text with a *criteria label.*

```
CL(criteria attributes)
```

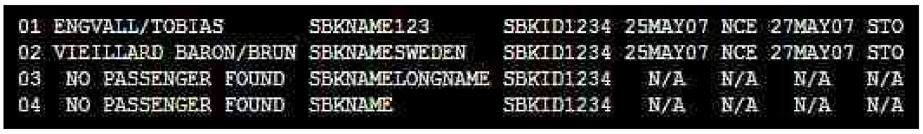
This allows the user to enter complex look-up queries in a quite nice and structured way. An example of a more complex query could look something like this:

```
RK/LN(ENGVALL)FN(TOBIAS)BP(NCE)DD(05MAY07)
```

This line tells the system to look for all SBK:s containing a customer named Tobias Engvall, it should depart from Nice on the 5th may 2007.

4.1.2 Presentation

The look-up results are then presented using a simple, table based approach. From left to right, a row contains of:



```
01 ENGVALL/TOBIAS       SBKNAME123       SBKID1234 25MAY07 NCE 27MAY07 STO
02 VIEILLARD BARON/BRUN SBKNAMESWEDEN    SBKID1234 25MAY07 NCE 27MAY07 STO
03  NO PASSENGER FOUND  SBKNAMELONGNAME  SBKID1234   N/A    N/A   N/A   N/A
04  NO PASSENGER FOUND  SBKNAME          SBKID1234   N/A    N/A   N/A   N/A
```

Figure 4.1: Results from a cryptic look-up query.

- `1  ENGVALL/TOBIAS`

  The local id of the match and the main passenger's name.

- `SBKNAME123      SBKID1234`

  A truncated version of the SBK:s name and the full SBK id are then shown.

  `25MAY07 NCE 25MAY07 STO`

  Finally, a short summary of the start date and end destination, and their dates are shown.

4.2 Retrieve

4.2.1 Syntax

Contrary to the look-ups, the retrieve system doesn't need to offer a complex syntax. It only needs to identify a whole SBK or an element living in a SBK.

SBK elements are uniquely identified by their identification number together with their type.

`DKSBKID1234/T1004CUS`

This query tries to fetch the customer element identified with an id of 1004. However, the retrieve system is meant to be used by issuing *follow-up queries*. The user is never supposed to enter a query like this.

4.2.2 The Summary Screen

The Summary Screen is the screen the user is first presented with, when a retrieve command is issued. It is meant to function as an entry-point to the rest of the SBK.
The purpose of this screen is to give the user a quick overview of the SBK:s topology and contents.

`T87JR29X1:Auftragsname Drucken tes    02OCT07 HAJ 09OCT07 FUE`

The first line shows the calculated begin and end dates of the SBK, the name of the SBK, the record locator of the SBK, and the departure and destination codes. Under the status line, we find the description of the SBK and the SBK:s associated customers.

After the header line, the customers referenced in the SBK are shown. Long names are truncated.

Figure 4.2: The Summary Screen

```
 1.Hofmann/Ulrike  2.HOFMANN/ULRIKE  3.HOFMANN/ULRIKE  4.HOFMANN/ULRIK ST..  5
 HOFMANN/ULRIKE  6.HOFMANN/ULRIK ST..  7.HOFMANN/ULRIKE  8.HOFMANN/ULRIK ST..
```

The topology of the SBK is shown by a table displaying the records contained in the SBK:

```
  9. HEADER       C1-8
 10. REQUIREMENT  C1
 11. RESERVATION  C3,5      2AIR 1CAR     02MAY07 NCE AMS  R12
```

From left to right, the row consists of:

- Type of record. (RESERVATION)
- Associated customers, if applicable [C1]
- Product summary, if applicable [2AIR 1CAR]
- Calculated begin date, boarding point, and off point, if applicable [02MAY07 NCE AMS]
- List of records contained in this record, if applicable [R12]

The Record Screen

The purpose of this screen is to display details about the all the entities contained in one record. And since records are meant to be the main "storage-container" of the SBK, most of the browsing will be done by viewing records.

As we recall, a record can contain:

- *Customers* – All customers, which are referenced at any point in a record, are displayed in a short list.

  ```
  1.VIEILLARD BARON/.. 2.TEULIERE/OLIVIER 3.ENGVALL/TOBIAS 4.PARADA/LUIS MIGU..
  ```

- *Products* of all kinds: Flights, Rental cars, Cruises, Insurances, Trains, etc.

  ```
  3. AIR:AF112345678  C1-2            02MAY07 NCE 02MAY07 AMS
  ```

  This line displays an air product. It is a flight with Air France from Nice to Amsterdam. We can tell from the association column, that only passengers 1 and 2 are attending the flight.

- Information related to *fares, fees and taxes.*

  ```
  7. FEE   100 EUR    C1>P3
  ```

  Item number 7 tells us that Customer one's flight on product three, which is the above mentioned flight, has a fee of 100 euros associated to it.

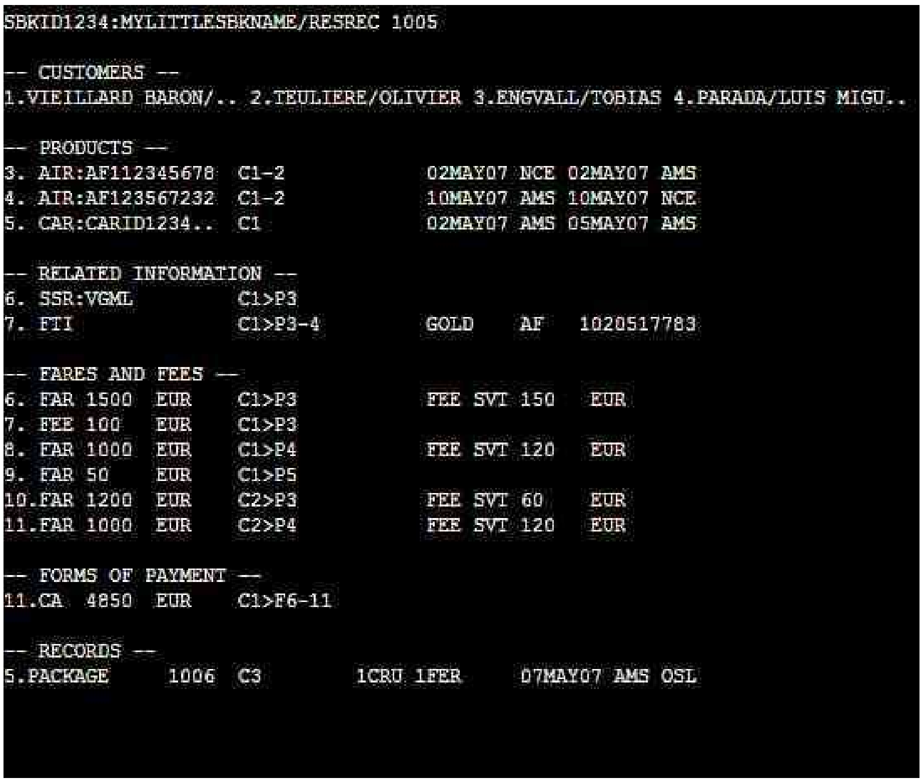
- *Modes of payment* for said fares:

  ```
  11.CA   4850 EUR    C1>F6-11
  ```

  A small summary of the form-of-payment is displayed. In this case: Customer number one is paying for all products in the record.

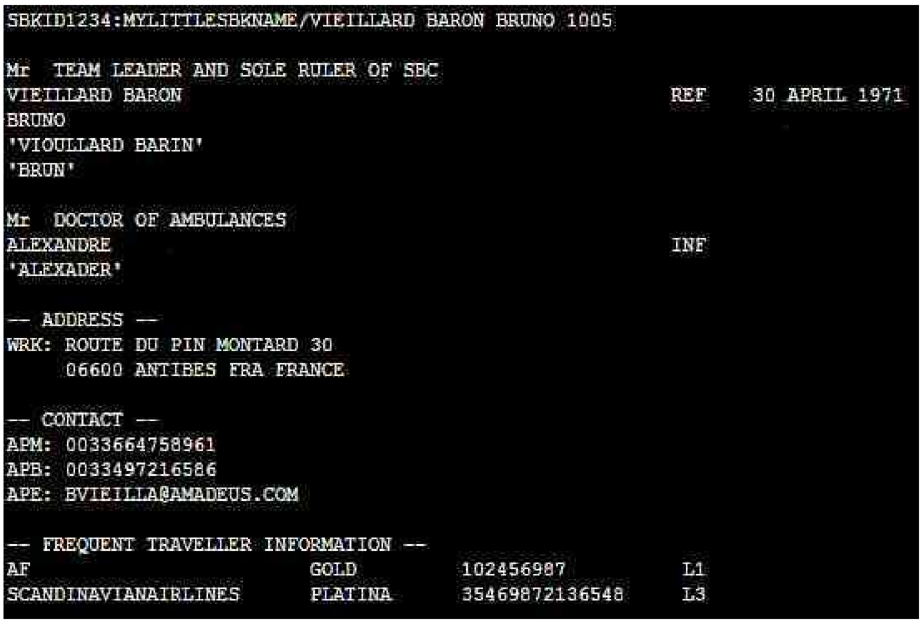
- *External references* – Any references to external data, e.g. a booking in another reservation system is also displayed in a short summarized view.

Other entities, such as documents and remarks, are also displayed in a similar fashion.



```
SBKID1234:MYLITTLESBKNAME/RESREC 1005

-- CUSTOMERS --
1.VIEILLARD BARON/.. 2.TEULIERE/OLIVIER 3.ENGVALL/TOBIAS 4.PARADA/LUIS MIGU..

-- PRODUCTS --
3. AIR:AF112345678  C1-2            02MAY07 NCE 02MAY07 AMS
4. AIR:AF123567232  C1-2            10MAY07 AMS 10MAY07 NCE
5. CAR:CARID1234..  C1              02MAY07 AMS 05MAY07 AMS

-- RELATED INFORMATION --
6. SSR:VGML         C1>P3
7. FTI              C1>P3-4         GOLD    AF   1020517783

-- FARES AND FEES --
6. FAR 1500  EUR    C1>P3           FEE SVT 150   EUR
7. FEE 100   EUR    C1>P3
8. FAR 1000  EUR    C1>P4           FEE SVT 120   EUR
9. FAR 50    EUR    C1>P5
10.FAR 1200  EUR    C2>P3           FEE SVT 60    EUR
11.FAR 1000  EUR    C2>P4           FEE SVT 120   EUR

-- FORMS OF PAYMENT --
11.CA  4850  EUR    C1>F6-11

-- RECORDS --
5.PACKAGE     1006  C3        1CRU 1FER     07MAY07 AMS OSL
```

Figure 4.3: Viewing a reservation record.



```
SBKID1234:MYLITTLESBKNAME/VIEILLARD BARON BRUNO 1005

Mr  TEAM LEADER AND SOLE RULER OF SBC
VIEILLARD BARON                                    REF    30 APRIL 1971
BRUNO
'VIOULLARD BARIN'
'BRUN'

Mr  DOCTOR OF AMBULANCES
ALEXANDRE                                          INF
'ALEXADER'

-- ADDRESS --
WRK: ROUTE DU PIN MONTARD 30
     06600 ANTIBES FRA FRANCE

-- CONTACT --
APM: 0033664758961
APB: 0033497216586
APE: BVIEILLA@AMADEUS.COM

-- FREQUENT TRAVELLER INFORMATION --
AF                     GOLD         102456987        L1
SCANDINAVIANAIRLINES   PLATINA      35469872136548   L3
```

Figure 4.4: Viewing a customer

4.2.3 The Customer Screen

A customer is always represented in the same way, regardless of whether the customer is buying a flight, cruise or anything else.

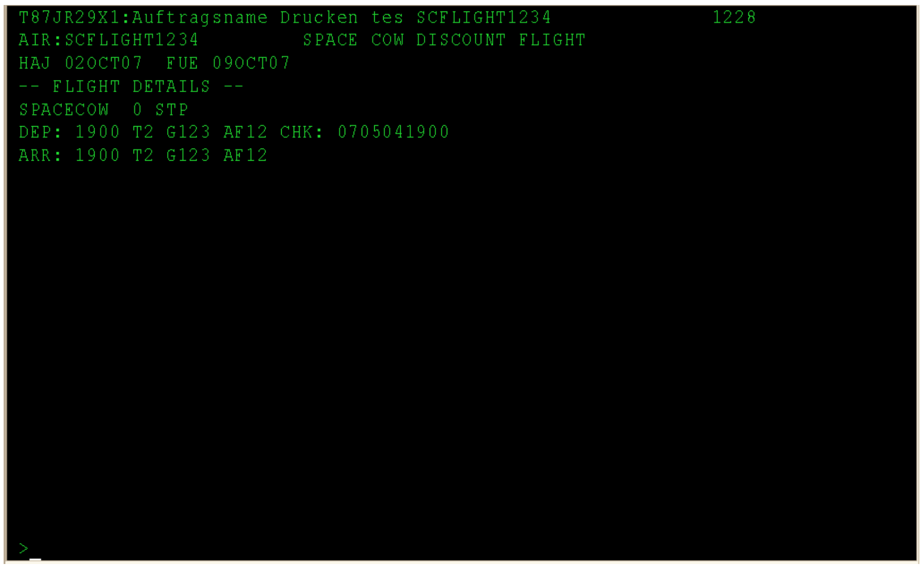
```
VIEILLARD BARON                                     REF 30 APR 1971
BRUNO
```

The first line displays the names and birth date of the customer. It is displayed in a simple way. The SBK has a feature, which allows displaying the customers *poor name*. The poor name is just a conversion between the real customer name, which may contain any kinds of special characters, into a terminal friendly representation.

After the passenger's name, any accompanied infants are displayed. Along with the specified address and contact information of the customer.

```
AF                    GOLD         102456987       L1
```

Any kind of related frequent traveler information is also displayed. A customer may have many different memberships, which all are displayed in a table.



```
T87JR29X1:Auftragsname Drucken tes SCFLIGHT1234           1228
AIR:SCFLIGHT1234         SPACE COW DISCOUNT FLIGHT
HAJ 02OCT07  FUE 09OCT07
-- FLIGHT DETAILS --
SPACECOW  0 STP
DEP: 1900 T2 G123 AF12 CHK: 0705041900
ARR: 1900 T2 G123 AF12









>_
```

Figure 4.5: A flight with a space cow, now that's something!

4.2.4 The Product Screen

All of the product screens follows the same general layout. The products common characteristics - Begin and end locations, dates, etc. are shown at the top of the screen. Under the general information, product specific information is found, such as: luggage information, information about boarding for an air product, and the type of car for rental car products.

The above screen describes a fictive air product:

```
AIR:SCFLIGHT1234             SPACE COW DISCOUNT FLIGHT
```

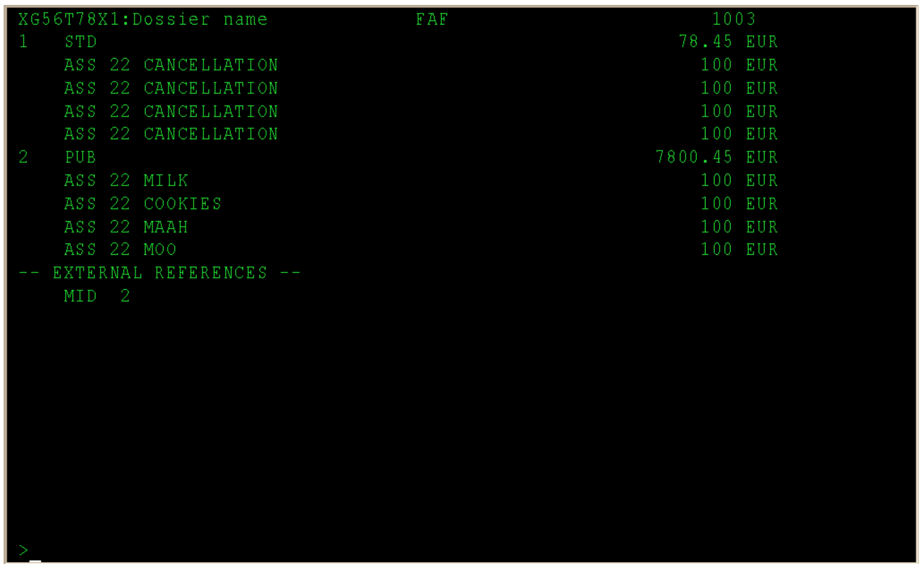
The first line gives us information about the type of the product. After the product type, a product identifier is shown which in this case is a flight number. If this was a car product, the car reservation number would be shown. This line is common for all kinds of products.

```
HAJ 02OCT07  FUE 09OCT07
```

The boarding and off points are then shown along with the begin and return dates. This information is also common for all types of products.
This specific line tells us that the flight leaves from Hannover on the 2:nd of October 2007 and that the return flight from Fuerteventura leaves on the 9th of October.

```
-- FLIGHT DETAILS --
SPACECOW  0 STP
DEP: 1900 T2 G123 AF12  CHK:0705041900
ARR: 1900 T2 G123 AF12
```

The product details section contains information about the check-in, departure and arrival times together with more related information such as luggage details, number of stops, etc.



```
XG56T78X1:Dossier name             FAF                      1003
1   STD                                                  78.45 EUR
    ASS 22 CANCELLATION                                    100 EUR
    ASS 22 CANCELLATION                                    100 EUR
    ASS 22 CANCELLATION                                    100 EUR
    ASS 22 CANCELLATION                                    100 EUR
2   PUB                                                7800.45 EUR
    ASS 22 MILK                                            100 EUR
    ASS 22 COOKIES                                         100 EUR
    ASS 22 MAAH                                            100 EUR
    ASS 22 MOO                                             100 EUR
-- EXTERNAL REFERENCES --
    MID  2







>_
```

Figure 4.6: Viewing a fare.

4.2.5 The Fare and Fee Screen

Fares and fees can be quite complex entities by themselves. Fares can have any number of fees and taxes associated to them. Fares, fees, and taxes also have types associated to them.

The above screen shows a fictive test fee.

```
1  STD                                   78.75 EUR
```

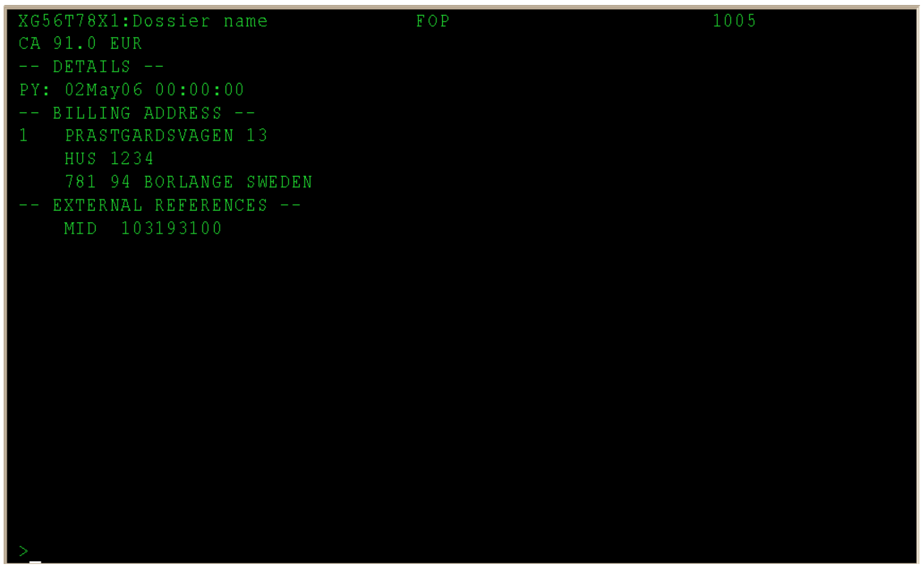
The first line shows a fee of 78.75 euros. The *STD* flag tells us, that this is a standard fee.

```
   ASS 22 CANCELLATION                   100 EUR
```

Under the first line is a number of associated fees. The *ASS* flag signals that the fee is associated to the first standard fee. If any remarks about the fee are present, they would be displayed between the flag and the amount.

```
   ASS 22 MILK                           100 EUR
```

The fare and fee structure can become quite complex, with lots of different fees and taxes associated to them.



```
XG56T78X1:Dossier name                FOP                           1005
CA 91.0 EUR
-- DETAILS --
PY: 02May06 00:00:00
-- BILLING ADDRESS --
1   PRASTGARDSVAGEN 13
    HUS 1234
    781 94 BORLANGE SWEDEN
-- EXTERNAL REFERENCES --
    MID  103193100










>_
```

Figure 4.7: A cryptic representation of a Form of Payment

4.2.6 The Form of Payment Screen

This screen displays information about a form of payment. It displays information about how something is payed.

```
CA 91.0 EUR
```

The first line tells us that the customer chose to pay by cash, as indicated by the *CA* flag. If the customer had chosen to use his credit card instead, the flag would have been *CC*.

```
PY: 05May06 00:00:00
```

It is followed by a line indicating when the payment was done. If the customer had chosen to pay by invoice instead, it would also had shown the deadline for the payment.

```
-- BILLING ADDRESS --
1   PRASTGARDSVAGEN 13
    HUS 1234
```

Another address than the customer's main address may be used as a billing address. Additional information like this is displayed after the payment details.

Chapter 5

Conclusion

What surprised me most while I was writing this project, was lack of previous work done on the subject. Since green-screen terminals are very common and widespread in the airplane industry, I was expecting to see large studies made on the subject. Sadly, I was unable to find any studies of interest.

Green-screen terminals will probably continue to be widely used in the future. Although somewhat crude and outdated, they still offer features that many modern interfaces lack, most notably: speed and simplicity.

5.1 Future work

The time frame of the project didn't allow for designing and implementing cryptic support for all the features of the shopping basket system. The two most significant improvements to the system would have been adding support for updating SBK:s, and finding a way of displaying information stored in externally referenced booking systems, such as PNR databases.

5.1.1 Update functionality

The current state of the cryptic SBK system does not allow the user to modify the contents of a SBK. In order to support this, the parser would have needed to be greatly extended in order to support all commands for adding, updating, and deleting information. In addition to this, a more advanced version of the context would also have been developed, along with a basic "end-of-transaction" mechanism in order to support commits of the changes made to the SBK.

5.1.2 PNR Bridge

The SBK system allows the user to reference data in external booking systems, most notably the Amadeus PNR database. Another possible extension of the system would have been to find a nice way of displaying the information contained in these systems.
This would involve creating a quite huge mapping between the corresponding attributes of the different systems as well as finding a nice way of letting the different systems communicate with each other.

Bibliography

[1] *Wikipedia article on 3270*
http://en.wikipedia.org/wiki/3270
2007-11-01

[2] *Wikipedia article on EBCDIC*
http://en.wikipedia.org/wiki/Ebcdic
2007-10-20

[3] *Wikipedia article on EDIFACT*
http://en.wikipedia.org/wiki/EDIFACT
2007-11-01

[4] *United Nations Directories for Electronic Data Interchange for Administration, Commerce and Transport*
http://www.unece.org/trade/untdid/welcome.htm
2007-10-27

[5] Eric S. Raymond
The Jargon File
http://catb.org/jargon/
2007-10-27

Chapter 6

Appendix

6.1 Abbreviations

1A	Amadeus
3270	The IBM 3270 class of terminals
tn3270	Modified version of the telnet protocol
ATE	Amadeus Terminal Emulator
SBK	Shopping Basket
PNR	Passenger Name Record
ORG	Originator
SI	Service Integrator
HSFREQ	Hybrid Screen Format Request Message
HSFRES	Hybrid Screen Format Response Message
TRMREQ	Process 3270 Query Message
TRMRES	Process 3270 Response Message
EBNF	Extended Backus Naur Form
Regex	Regular expressions
GCC	The GNU Compiler Collection
G++	The GNU C++ Compiler
XML	Extended Markup Language
EDIFACT	United Nations/Electronic Data Interchange For Administration, Commerce, and Transport

6.2 Overview

SBK Cryptic Message Interface (draft) – 04 Oct -07 rev5

Introduction

This document will describe the mechanics of the SBK cryptic interface. In the initial version of the cryptic interface, functionality for searches (lookups) and retrieving of SBKs will be supported

Lookup

Introduction

This command will implement searching for a group of SBKs given a search criterion. The idea is to parse a search request entered via the cryptic channel, wrap it into a **SSBLRR** message and pass it onto the SBK services. The response then needs to be parsed and presented in a nice way.

Cryptic commands

- Must not be ambiguous

Proposal of a syntax

"`DK`" = Placeholder for a SBK query.

Syntax:

- Encapsulate search attributes.
- Possibly with the most common criteria type not encapsulated. The examples below show the passenger name without encapsulation.
- Use a simple label for encapsulation, e.g. `ER()` for external reference.
- Encapsulation with parenthesises, e.g. `DD(27MAY07)` for departure date 2007-05-27
- Missing a leading slash after the `DK` indicates a search for a SBK ID. This criteria is not encapsulated.

Lookup-attributes:

- *LN* Passenger Last name
- *FN* Passenger First name
- *DD* Date of Departure
- *DR* Date of Return
- *OP* Off Point: NCE
- *BP* Boarding Point: STO
- *SN* Shopping basket name: MYSHOPPINGBASKET
- *ER* External reference: ID/Type

Example queries:

```
DK/DD(27MAY05)BP(NCE)LN(ENGVALL)FN(TOB*)
```
Search for all SBKs departing from Nice the 27th of May where the passenger's last name is "Engvall" and the given name starts with "Tob".

```
DK/ER(ASD1234/PNR)
```
Search for all SBKs having an external reference to a PNR with the ID "ASD1234".

```
DKSBKID1234
```
Retrieve the SBK with the SBK id "SBKID1234"

Screens

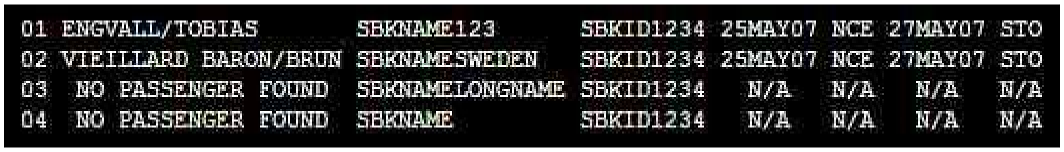
While designing the initial version of the output format, these premises were taken in to account:

- Maximum of 50 returned results by the SBK lookup. Thus two characters used for local IDs.
- SBK IDs are of 9 characters length.

The compact display of an SBK in the output list includes:

- **Local ID** [2 characters]
- **Passenger name** [20 characters]
  - *EDIFACT: Basket->shoppingBasketCustomers->customerInformation->... Check for a main customer. If none, use the first result*
- **SBK name** [15 characters]
  - *EDIFACT: Basket->shoppingBasketName->name*
- **SBK unique ID** [9 characters]
  - *EDIFACT: Basket->shoppingBasketIdentifier->controlNumber*
- **Departure and return date and location** 2*[7+3 characters]
  - *EDIFACT Basket->shoppingBasketSummaryInfo...*

Note that the SBK and passenger names will be truncated if their size exceeds the predefined boundaries.
Mock-up of output:



```
01 ENGVALL/TOBIAS        SBKNAME123       SBKID1234 25MAY07 NCE 27MAY07 STO
02 VIEILLARD BARON/BRUN SBKNAMESWEDEN    SBKID1234 25MAY07 NCE 27MAY07 STO
03  NO PASSENGER FOUND  SBKNAMELONGNAME SBKID1234   N/A    N/A    N/A   N/A
04  NO PASSENGER FOUND  SBKNAME          SBKID1234   N/A    N/A    N/A   N/A
```

Legend:
[Local ID] [Passenger name] [SBK Name] [SBK Id] [Date of departure] [Departure loc.] [Date of return] [Return loc.]

Error messages

To be defined.

Retrieve

Introduction

The complexity of the SBK data structure makes displaying the contents of a SBK on very limited screen estate a bit tricky. Although the SBK data structure has well over 2000 attributes associated to it, the interface for retrieving and displaying a SBK must be user-oriented without lacking information about details.
Since the SBK contains very much information, especially in the product and fares&fees sections, the initial work will focus on the most commonly used attributes.

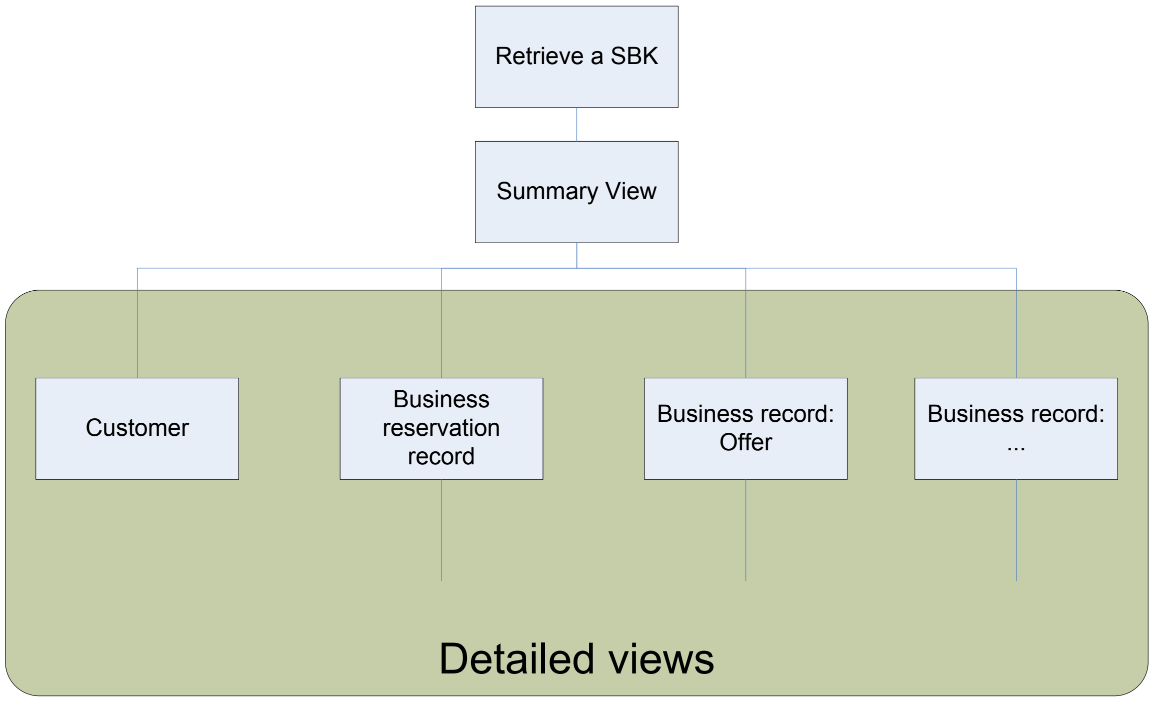
Some commonly used composites of elements are often reused throughout the SBK, an example being the date/time composite. This is a very good thing that can and should be taken advantage of.

Premises

- Standard screens are 79x24 characters.

The cryptic displays – overview

The first screen the user is presented with when retrieving a SBK is the summary/records screen. Its purpose is to show the structure of the SBK as well as some calculated summary information along with the customers of the SBK.
By using this as an entry point, the user can then choose to view information about the different entities: Customers, Records etc.



Navigation

The SBK is a quite complex data structure. With that in mind, having some kind of navigation stack would be a great asset.
The current plan is to have the current navigation context along with some navigational data stored in the context server by OTF. This data is used to navigate the SBK through local IDs.
Enumerating all entities capable of being browsed gives the user quick access to continued exploration. Instead of saying: “Show me the type of fees associated to this specific flight”, a user could just type in: “Show me more information about field 3” while viewing details on a flight.

```
[3.AIR:FLIGHTASD123 05MAY07 0700 NCE STO EJ1233321   ]
```

The “Show me details of 3” command would take you to the detailed display of the air product.

The general approach described in this document uses a “flattened” style for displaying things. Instead of showing all stand-alone, top-level business objects as entry points, all the business records contained in the SBK are presented in a structured table. Information about the relations between records can be found in one of the columns of the table.

Relations

The business and reservation records consist of atomic elements and their associated relations. Since these are the records which in a way defines the whole meaning of the SBK, this is what is important to display in a nice way. The SBK way of life document describes relations of order two as the highest type of a relation. An example of such a relation would be a SSR having the *EXT* relation relating to a *CON* relation: A passenger’s consuming relation is extended with a product. Having this as a premise, relations can be presented quite nicely as shown in the examples given later on.

Cryptic commands

Support for:

- Navigation.
- Filtering – not initially.
- Customizing the view – not initially.

`RK` = Placeholder Retrieve.

`RKSBKID12345`
Retrieve the SBK with id “SBKID12345”

`RK/DESC`
Show the description of the SBK, another way of displaying this would be to assign it as a browsable item in the summary screen.

`RK/REC`
Show the records in the SBK.

`RK/15`
Navigate to element number 15.

`RK/T1005`
Navigate to the entity with tattoo 1005.

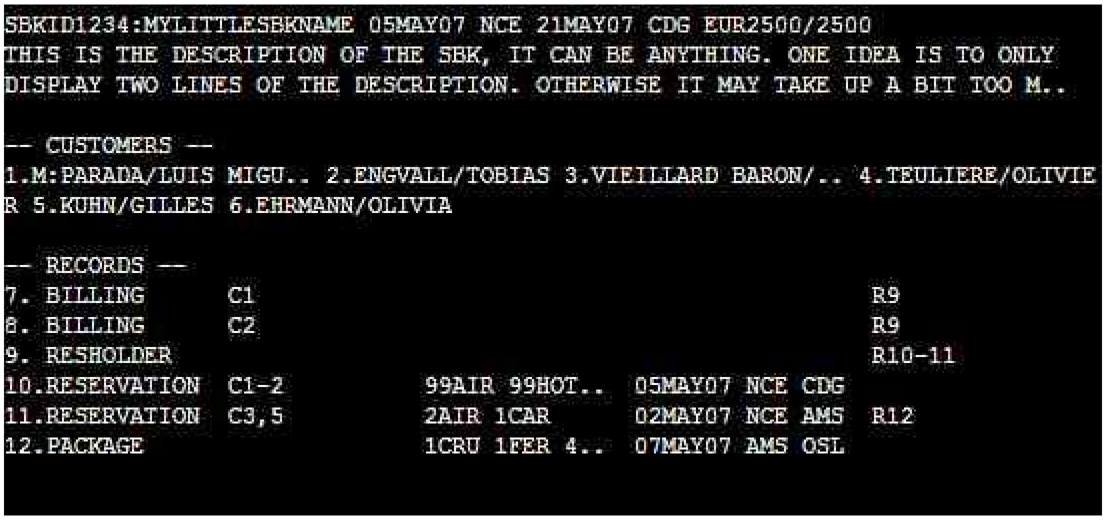
Support for filtering of data and custom displays will be explored in later versions.

Labels/Captions

Having a caption or a label which explains the entities is probably necessary, especially in sections which are not frequently used.

The summary screen:

This is the screen the user is first presented with when a retrieve is performed. The purpose of this screen is to give the user a quick overview of the SBK's contents and topology.



```
SBKID1234:MYLITTLESBKNAME 05MAY07 NCE 21MAY07 CDG EUR2500/2500
THIS IS THE DESCRIPTION OF THE SBK, IT CAN BE ANYTHING. ONE IDEA IS TO ONLY
DISPLAY TWO LINES OF THE DESCRIPTION. OTHERWISE IT MAY TAKE UP A BIT TOO M..

-- CUSTOMERS --
1.M:PARADA/LUIS MIGU.. 2.ENGVALL/TOBIAS 3.VIEILLARD BARON/.. 4.TEULIERE/OLIVIE
R 5.KUHN/GILLES 6.EHRMANN/OLIVIA

-- RECORDS --
7. BILLING      C1                                         R9
8. BILLING      C2                                         R9
9. RESHOLDER                                               R10-11
10.RESERVATION  C1-2          99AIR 99HOT..  05MAY07 NCE CDG
11.RESERVATION  C3,5          2AIR 1CAR      02MAY07 NCE AMS  R12
12.PACKAGE                    1CRU 1FER 4..  07MAY07 AMS OSL
```

Example 1: The SBK summary screen.

The first line shows the calculated begin and end dates of the SBK, the departure and destination codes and the calculated monetary summary. Under the status lines we find the description of the SBK and the SBK's associated customers.
The topology of the SBK is shown by a table displaying the records contained in the SBK:

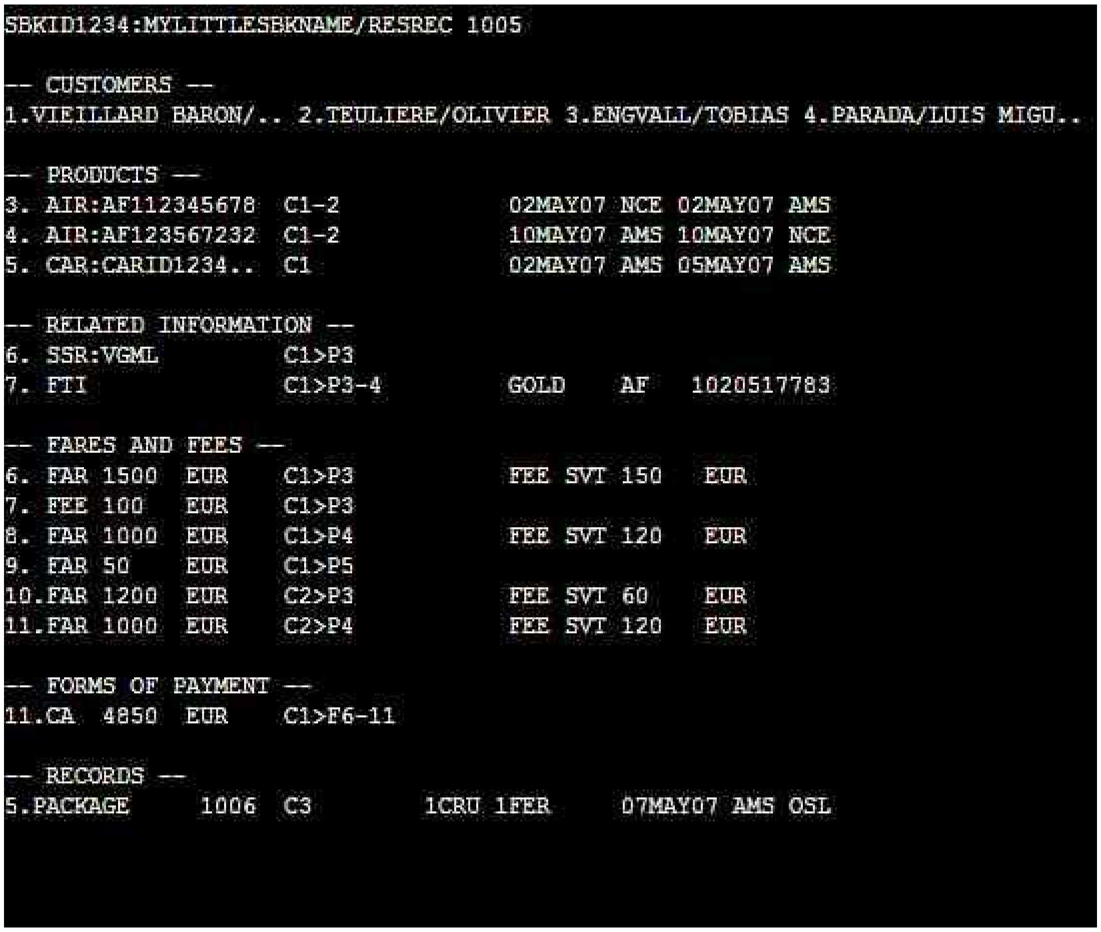
[11.RESERVATION   C3,5   2AIR 1CAR   02MAY07 NCE AMS   R12   ]

From left to right, the row consists of:

• Type of record.	[RESREC]
• Associated customers.	[C3,5]
• Product summary.	[2AIR 1CAR]
• Calculated begin date, boarding point and off point.	[02MAY07 NCE AMS]
• List of records contained in this record.	[R12]

Viewing one record

Access to the underlying records is gained from further navigation from the topology screen. In this case: the reservation record with tattoo 1005 is shown. The screen uses a table-based approach and the different sections of the record are separated by a header.



```
SBKID1234:MYLITTLESBKNAME/RESREC 1005

-- CUSTOMERS --
1.VIEILLARD BARON/.. 2.TEULIERE/OLIVIER 3.ENGVALL/TOBIAS 4.PARADA/LUIS MIGU..

-- PRODUCTS --
3. AIR:AF112345678  C1-2            02MAY07 NCE 02MAY07 AMS
4. AIR:AF123567232  C1-2            10MAY07 AMS 10MAY07 NCE
5. CAR:CARID1234..  C1              02MAY07 AMS 05MAY07 AMS

-- RELATED INFORMATION --
6. SSR:VGML         C1>P3
7. FTI              C1>P3-4         GOLD    AF   1020517783

-- FARES AND FEES --
6. FAR 1500  EUR    C1>P3           FEE SVT 150   EUR
7. FEE 100   EUR    C1>P3
8. FAR 1000  EUR    C1>P4           FEE SVT 120   EUR
9. FAR 50    EUR    C1>P5
10.FAR 1200  EUR    C2>P3           FEE SVT 60    EUR
11.FAR 1000  EUR    C2>P4           FEE SVT 120   EUR

-- FORMS OF PAYMENT --
11.CA  4850  EUR    C1>F6-11

-- RECORDS --
5.PACKAGE     1006  C3        1CRU 1FER     07MAY07 AMS OSL
```

Example 2: The single record view.

This screen's purpose is to show details about the entities contained in ***one*** record. In this case, the record contains two passengers and three products.

- A product is displayed by its common characteristics, as shown by the below example.

  [`3. AIR:AF112345678              02MAY07 NCE 02MAY07 AMS`   ]

  From left to right, the row consists of:
  - The type of the product [`AIR`]
  - The product identification, Flight number for air product, Hotel code for hotel product etc. [`AF112345678`]
  - The entities consuming this product. [`C1-2`]

- A short summary of the begin/end date and locations. [`02MAY07 NCE 02MAY07 AMS`]

- After the product summary, information related to the SBK is shown. In this case, we have a special service request for a vegetarian meal on Customer 1's consumption of Product 3 and some information about Customer 3's frequent traveller data.

  [`6. SSR:VGML         C1>P3                                      `]

  This row shows the vegetarian meal special service request and how it is related to Customer 1's flight with AF112345678. As a placeholder, the `'>'` sign is used to indicate a relation.

- The fares section describes monetary information about the record. The below example shows a fare of 1000€ related to Customer 1's consumption of Product 4, it also has a fee of 120€ associated to it.

  [`8. FAR 1000  EUR    C1>P4           FEE SVT 120   EUR     `]

  The row consists of:

• Type: fare or fee.	[`FAR`], [`FEE`]
• Type of fee.	[`SVT`]
• Amount and currency.	[`1000  EUR`], [`120   EUR`]
• Consumptions.	[`C1>P4`]

- Payment is defined after the fares in the Forms of Payment-section. In the above scenario, Customer 1 pays for the entire record in cash.

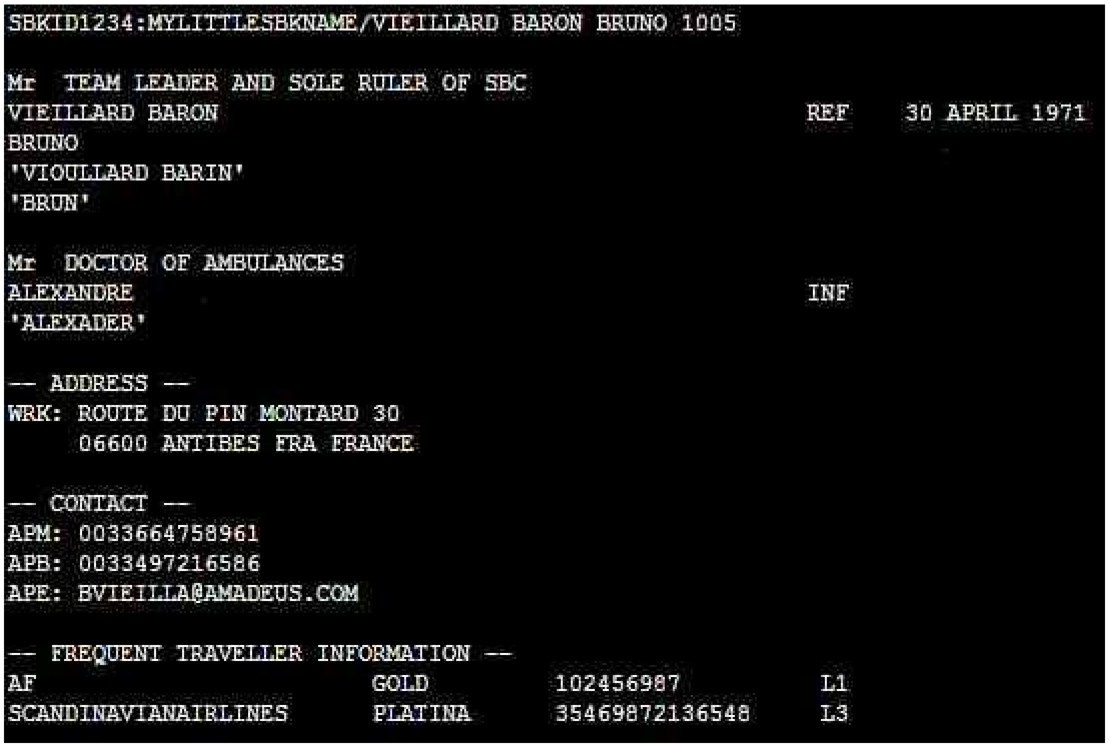
  [`11.CA  4850  EUR    C1>F6-11                               `]

  The row consists of:

• Type: Cash, Credit card, etc.	[`CA`]
• Amount and currency.	[`4850  EUR`]
• Relations. (Who is paying for what?)	[`C1>F6-11`]

- The last entry shows the underlying records of this record. It is displayed in the same way as it is in the summary screen.

Viewing one entity: A customer



```
SBKID1234:MYLITTLESBKNAME/VIEILLARD BARON BRUNO 1005

Mr  TEAM LEADER AND SOLE RULER OF SBC
VIEILLARD BARON                                           REF    30 APRIL 1971
BRUNO
'VIOULLARD BARIN'
'BRUN'

Mr  DOCTOR OF AMBULANCES
ALEXANDRE                                                 INF
'ALEXADER'

-- ADDRESS --
WRK: ROUTE DU PIN MONTARD 30
     06600 ANTIBES FRA FRANCE

-- CONTACT --
APM: 0033664758961
APB: 0033497216586
APE: BVIEILLA@AMADEUS.COM

-- FREQUENT TRAVELLER INFORMATION --
AF                         GOLD         102456987          L1
SCANDINAVIANAIRLINES       PLATINA      35469872136548     L3
```

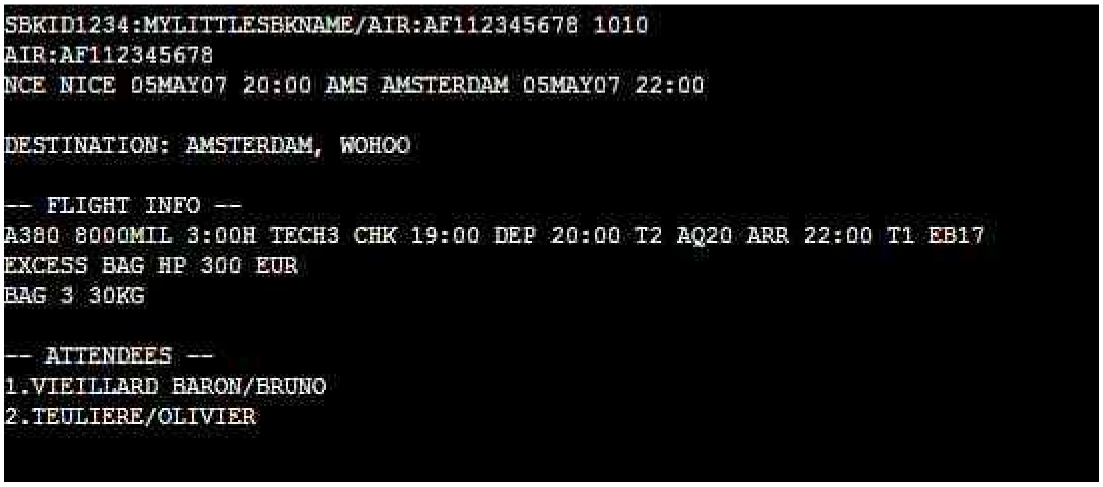
Example 3: Viewing a customer.

This is an example of a view of a single entity. The screen displays the entity and the attributes associated to it. In an initial phase, no information about participation or other information about relations with other products will be shown.

The above example shows a passenger accompanied by an infant. One address, two phone numbers and an email address are shown along with some frequent flyer information.

A feature that was left out to future versions was having a list of all items referencing to this particular customer.

Viewing one entity: Product



```
SBKID1234:MYLITTLESBKNAME/AIR:AF112345678 1010
AIR:AF112345678
NCE NICE 05MAY07 20:00 AMS AMSTERDAM 05MAY07 22:00

DESTINATION: AMSTERDAM, WOHOO

-- FLIGHT INFO --
A380 8000MIL 3:00H TECH3 CHK 19:00 DEP 20:00 T2 AQ20 ARR 22:00 T1 EB17
EXCESS BAG HP 300 EUR
BAG 3 30KG

-- ATTENDEES --
1.VIEILLARD BARON/BRUNO
2.TEULIERE/OLIVIER
```

Example 3: Viewing an air product.

This initial version of the screen shows an air product. Since the record screen uses limited space for displaying the consumers of a product, it is also necessary to show the consumers of the flight on this screen.

The flight info is just displayed as a concatenated list of the attributes.