Interfacing heterogeneous databases with a formal specification of the attribute mappings

BY

M.W. Mak

March 1996

Abstract

This paper introduces a flexible and easy way to integrate views of heterogeneous databases into an application. The purpose of this project is to develop an interface between external databases and an application called Crisis Response Prototype (CRESP). CRESP is a prototype developed by SHAPE Technical Centre (STC) to support military situation monitoring. This interface enables the usage of data from different external databases within CRESP. Two solutions will be examined, firstly the integration of all databases into one global scheme, secondly the use of a federation of databases. Finally a combination of those solutions will lead to the best way to solve this particular problem. The interface will use a formal specification of the mappings between the attributes to implement the data transfer in a declarative style.

Table of contents:

1. Introduction

CRESP is a package developed at STC that supports monitoring crisis response operations. It supports interoperability and serves as a common database system for all NATO Commands involved in crisis operations. It includes an interface to foreign databases, and it allows to import data from these different databases. CRESP will exchange data with other CRESP's or display the data. The display can be tabular or on a map. CRESP is distributed over a wide area network. The foreign databases used by CRESP are mostly relational (e.g. Oracle databases). Their conceptual schemes are different from each other. To be able to compare the entities in these foreign databases to each other in CRESP there has to be a standard format for the data. Because the databases are part of existing external systems it is not possible to change them. CRESP only uses predefined views on the databases. At this moment it is only necessary to import data into CRESP. However, exporting data to foreign databases will be required in the future.

2. Current interface description

Currently one interface, the GOB1, has been implemented, which supports only one view of the THISTLE database for CRESP. THISTLE is an Army Tactical Prototype System developed by Cranfield University (UK).

2.1 The GOB1 interface

2.1.1 General interface description

The *GOB1 interface* exports data from a THISTLE database to CRESP. The export can be incremental, that is it transfers only new and updated data. Otherwise the export is full and it deletes all pre-existing data concerning the units in the new export files before it imports all data from text files into dBaseIV tables. This export/import action is demand driven. The programming language of the interface is Access Basic. It uses for each table of *GOB1* an ASCII file, two MS Access tables and a dBaseIV table. The first MS Access table is used to store the foreign data in and the second to store the converted data temporarily. The interface has the following structure:

- The interface gets flat ASCII files from the foreign export module.
- The interface transfers the data from the ASCII files to MS Access tables. These tables are part of the interface and contain data in the format of the foreign database. Each table corresponds to an ASCII file. The interface definition specifies its attribute names.
- Some conversion on the data takes place to get the data in the format specified by the destination database. Look-up tables accomplish this. A function in the interface maps every element of the foreign tables onto an element of the corresponding destination tables. These are temporary tables to store the new data in until one can check whether it handles about an already known unit or a new one. The look-up tables are hard-coded.
- At last the interface transfers the data to dBaseIV tables, the final destination tables. The application attaches the dBaseIV tables, removes the old data of newly imported units and transfers the data using SQL statements.

The complete data transfer is denoted in figure 1, while figure 2 shows the import part in more detail.

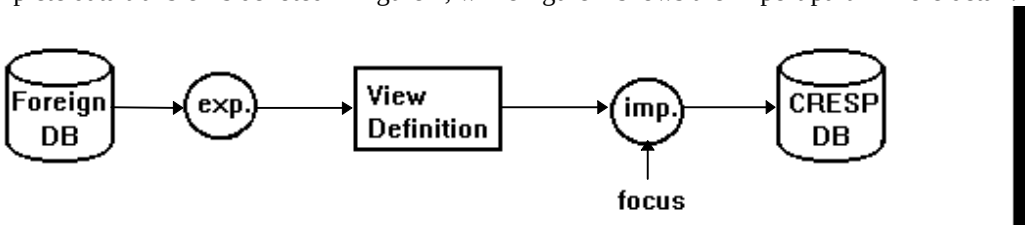

Figure 1 The data transfer from foreign databases to CRESP

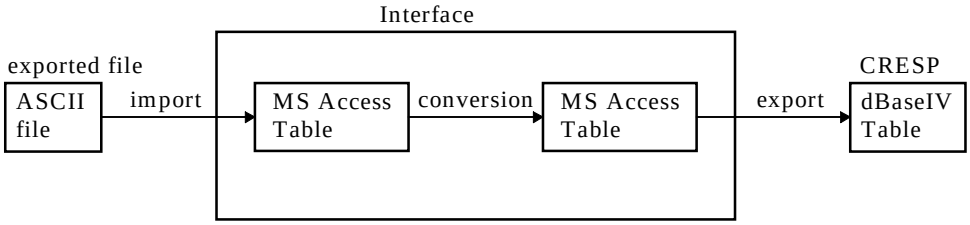

Figure 2 The import part of the data transfer

2.1.2 View definition

THISTLE provides the following exported view of the units and their locations to CRESP:

- A *UNITS* table, which identifies the units. The *UNITS* record is a prerequisite for the following tables.
- A *UNIT-STS* table, which gives the status, strength and current location of the units.
- A *UNIT-EQP* table, which records the unit's equipment holdings.
- A *CMD-REL* table, which records the unit's command subordination relationship.

From now on this view will be referred to as "*GOB1*" and the supporting interface the "*GOB1 interface*". An attribute *update-time* keeps track of the history of the unit. Another view that will identify the movements of units will use this attribute. This view is not yet implemented. The attributes with "No" in the Null-field in the following tables are mandatory for a correct data transfer.

UNITS				
Field	**Null**	**Format**	**Size**	**Description**
update-time		char(12)	Fix	ZULU Date-Time when information was last updated
source		char(8)	Var	command/system sending data
unit-id	No	char(15)	Var	originator-specific unique unit identifier
orbat-type		char(3)	Var	type of orbat (Air, Ground, Naval, Missile...)
name		char(55)	Var	full unit name or nick-name
category		char(10)	Var	main categorisation of unit
en-friend		char(3)	Var	friend or enemy discriminator
country		char(2)	Fix	country code
organisation		char(8)	Var	abbreviation for the organisation the unit is assigned to
arms		char(15)	Var	unit type
command-level		char(5)	Var	level or size of command
symbol		char(6)	Var	code for graphic symbol
comments		char(255)	Var	any short comment

Table 1 GOB1: Units table

UNIT-STS				
Field	**Null**	**Format**	**Size**	**Description**
update-time		char(12)	Fix	ZULU Date-Time when information was last updated
source		char(8)	Var	command/system sending the data
unit-id	No	char(15)	Var	originator-specific unique unit identifier
activity		char(8)	Var	type of activity being performed by the unit
ce		num(2)	Fix	combined pers./eqp. combat effectiveness/readiness code
pers-strength		num(7)	Var	current total number of personnel of any type
location-name		char(30)	Var	name of site/town for current unit location
location-lat		char(7)	Fix	latitude co-ordinate of current unit location
location-lon		char(8)	Fix	longitude co-ordinate of current unit location
effective-time		char(12)	Fix	ZULU Date-Time when information was effective
verific-code		char(4)	Var	verification code for information
comments		char(255)	Var	any short comment on the status

Table 2 GOB1: Units-Sts table

UNIT-EQP				
Field	**Null**	**Format**	**Size**	**Description**
update-time		char(12)	Fix	ZULU Date-Time when information was last updated
source		char(8)	Var	command/system sending the data
unit-id	No	char(15)	Var	originator-specific unique unit identifier
equip-type-categ		char(20)	Var	type-category of equipment reported
equip-type-name		char(40)	Var	type-name (model) of equipment reported
quantity		num(7)	Var	current holding for specified equipment of specified unit
verific-code		char(4)	Var	verification code for information
effective-time		char(12)	Fix	ZULU Date-Time when information was effective

Table 3 GOB1: Unit-Eqp table

CMD-REL				
Field	**Null**	**Format**	**Size**	**Description**
update-time		char(12)	Fix	ZULU Date-Time when information was last updated
source		char(8)	Var	command/system sending the data
sup-unit-id		char(15)	Var	originator-specific unique unit identifier of superior unit
sub-unit-id	No	char(15)	Var	orig.-specific unique unit identifier of subordinate unit
cmd-rel		char(6)	Var	command relationship
verific-code		char(4)	Var	verification code for information
effective-time		char(12)	Fix	ZULU Date-Time when information was effective

Table 4 GOB1: Cmd-Rel table

2.1.3 File structure

The data is exported by ASCII files. There is one file for each table in the view. The files have the following structure:

- Each file contains zero or more records. For an empty table the file contains only a carriage return.
- A record is one line in the file. Records are of variable length and separated by a carriage return.
- A record contains one or more fields. The field separator is "@" (the at sign). The interface specifies the order of the fields. Null fields are fields with zero length. Not-null fields should be of not-zero length and fixed-size fields should be of the specified length from the interface specification.

In Backus-Naur Form this can be constructed as follows:
<file> ::= CR | <record> {CR <record>}
<record> ::= <field> {@ <field>}
<field> ::= {<char>}
<char> ::= A | B | C | ....

2.2 The GOB3 interface

For another interface, called GOB3, only the definition exists, but the interface itself is not yet implemented. This interface will support one view from STAFOR for CRESP. STAFOR, Status of Forces, is an operational database system containing the agreed status of forces allocated to NATO, maintained at SHAPE. The interface definition from STAFOR to CRESP is similar to the *GOB1 interface* definition because they are based on the same view. The big difference is that there are several attributes of THISTLE not used with the STAFOR interface and vice versa. Overall the idea is the same. The databases themselves are very different from each other.

3. Problems and issues

We have described the current state of the project. The improvement or extension of its current functionality is the objective for my master thesis. Now we will discuss some limitations of the existing interface and some possible solutions.

3.1 New views

There are two options to define another view for the destination database. When a database can provide the same data as required for an existing view, an option is to use the same view definition to import data from this other database. Another option is to define a completely new view on any database to insert into the application.

To include the data of an existing view from another database, a new interface has to be written for each foreign database view that will be imported. The same effort has to be performed because the interfaces are hard-coded within the program. Moreover, the interfaces should be adapted each time the view definitions are changing. This could lead to inconsistency.

A similar problem is the creation of a new view on a foreign database to insert into the destination database. In this case it does not matter whether this foreign database already supports views for the destination database or not. Presently this problem must be solved also with writing a new interface.

3.2 Data element mapping

Mostly the mapping of data elements within one interface is one on one. An attribute of the foreign database maps onto an attribute of the destination database with the same domain. Sometimes the domain values of the foreign attribute differ from the domain values of the corresponding destination attribute. Now a specific translation from one value to another is necessary. Another mapping possibility is to map several foreign attributes together onto one destination attribute and the other way around.

3.3 Incomplete data

Data that is mandatory in the destination database might not be provided by the import interface. Using an algorithm or look-up table is not possible. A solution might be user intervention, which has to be avoided as much as possible, or deleting the record with missing data. This implies that non-mandatory data is lost as well, which is preferable over missing mandatory data. Another solution could be introducing a special value, like a null value, to indicate that the value should exist but it does not. This solution cannot be used when (a part of) the key is missing.

3.4 Syntactically erroneous data

The importable data could be in a spreadsheet instead of in a database. A database has constraints on the data. When a date field is required the database will complain when the date is not filled in in the correct format. Errors are easily made in spreadsheets while data is entered manually and not checked automatically. So the data can be imprecise. Spelling errors have to be corrected and may even lead to corruption of data and thus processing.

4. Objective

The plan now is to concentrate on the problems in section 3.1. These problems are slightly different from each other, but that does not matter for the solution. The problems in sections 3.2 and 3.3 are related to the problems in section 3.1, so we try to solve them too. The mapping problem has to be solved to get an interface that imports data from several foreign databases into a destination database, just as all mandatory data has to be available. For the moment we do not look into the problem of section 3.4.

Concerning the first problem writing a new interface for each new database or view that is going to be imported is a huge task. Therefore the objective is to develop a “smart and flexible” software agent to support the data import for CRESP. The result will enable the import of all kinds of views. It will support the import of data from one view of different databases, as well as the import of data from different views of a foreign database. Figure 1 denotes the complete data transfer. Our prime focus will be the import part of the data transfer. The correctness of exported data and the definitions of new views are not within the scope of this project. However, the latter will be discussed briefly.

5. Proposed solutions

Most of the databases used by companies were “stand-alone” databases. One database was used for one application, while another database was used for another application. So there was some freedom in their design because they did not have to communicate with each other. In databases that are in use for several years it is difficult to make changes. Interoperability among different applications developed in this manner is difficult. There are different proposals in literature to solve this problem. Two approaches have been investigated:

- the integration of databases into a global scheme
- and the creation of a federation of databases.

These approaches will be discussed in the next sections.

5.1 Integrating databases into one global scheme

Some authors describe the building of a global scheme over the existing schemes. Examples are Multibase [1] and Mermaid [2]. For the user the database looks like one database with one query language, although there are several different databases with different DBMS’s and query languages. The database schemes of the different databases are translated to relational schemes. These schemes are integrated into one global scheme over which one can define views. The result is a tightly coupled system.

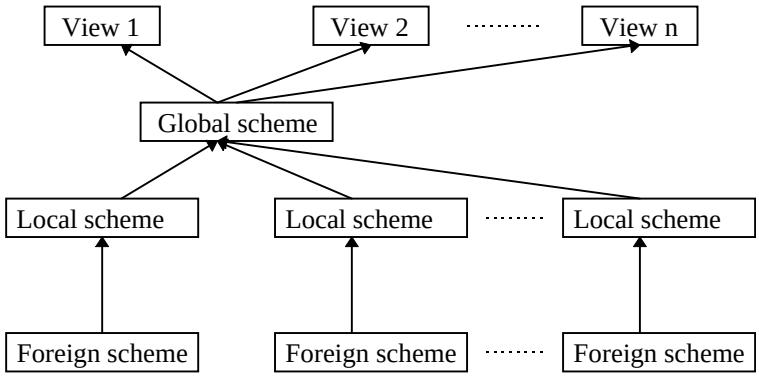

Figure 3 Architecture of integrated database schemes

In this manner the existing databases and their applications do not need to change. All database schemes have to be integrated into one global scheme however and that requires an enormous effort. The process is expensive and difficult. It also tends to be hard to change. This full integration of the databases is not always necessary for an application. In our case it is not useful because CRESP will only use simple views of very complex foreign databases that are varying extremely.

5.2 Get some co-ordination by using a federation of databases

Another solution could be autonomously working systems with import/export modules between them. The databases have to negotiate about using each other's data, but a global database scheme with all the available information is not necessary. This results in a loosely coupled federation of databases. Each database determines what information it wants to export (a kind of view on the database), and puts this in a dictionary. The application uses this dictionary to look up where the required information resides. It distinguishes from composing a global scheme because the dictionary does not contain all available information, only the information the databases want to share with the rest. Examples are the Schooner Interconnecting system [3] and the Remote-Exchange system [4].

The Schooner Interconnecting system sees databases as independently developed components. Each component has a code block and an interface. By using these interfaces the components can communicate with each other. In this case the interfaces form the dictionary. Remote-Exchange uses a sharing advisor to collect the exportable data. This sharing advisor has a semantic dictionary that is a federated knowledge base about sharable information. An application also contacts the sharing advisor when it needs information from a certain database.

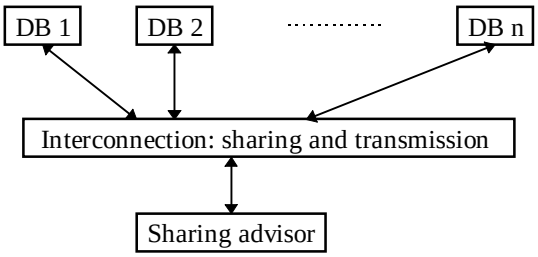

Figure 4 The Remote-Exchange architecture

5.3 The chosen solution

A solution to our problem could be a combination of those two by defining a simple view with all the information CRESP users need. This view is actually a global view over the views the foreign databases export and not a global scheme over all integrated databases. The complexity of the different databases causes an extreme effort for defining a global scheme, while we can express the information needed from the foreign databases in a simple view. Therefore we have chosen for the simple and efficient method. An interface between the foreign databases and the destination database contains the definition of the view. It gets the data of the foreign databases by export modules of those databases. Then it transforms the data into a format the destination database can use and it imports the data using this view.

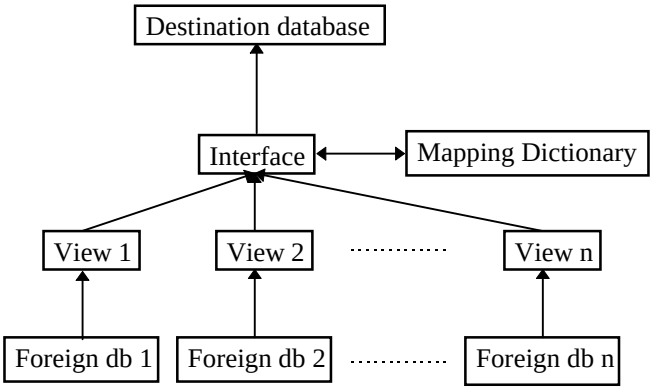

Figure 5 Proposed architecture of CRESP interface

To get a flexible interface the existing interfaces between the foreign databases and the destination database will be specified formally. The result will be a flexible system in which it is easy to add a new database. With a formal description the system is more consistent and complete and it can be a basis for a tool to automate integrating databases. It will be sufficient to describe the export part of a new foreign database in this formal manner to integrate a new view easily into the existing system. The THISTLE and STAFOR databases give a good start for this.

6. A mapping dictionary

To make the proposed interface more flexible the different types of mapping that occur in the *GOB1 interface* are analysed. If it is possible to identify a pattern in the mappings it will be easy to define a mapping dictionary. Such a mapping dictionary provides flexibility as one only has to update this dictionary and possibly add tables for a new view. It is better to use a mapping table per view instead of one mapping table for all views. A global mapping table will include many mappings of attributes that are not present in the view of which the data is going to be imported. The interface has to check these mappings also when transferring the data, although this is superfluous. With a new mapping table for each view the interface can transfer the data faster, because it only has to check the attributes of this particular view. To introduce a new database that exports an already existing view one only has to add this view to the list of available views, but now exported from this database. The implemented mappings are always from one foreign record to one corresponding record of a destination table. The mappings of the attributes of those records can be of six interesting types: 0-1, 1-0, 1-1, 1-N, N-1 and

M-N mapping. Sections 6.1 - 6.4 describe the mappings of *GOB1* for the THISTLE database, while in section 6.5 the remaining mappings are discussed.

6.1 0-1 mapping

With 0-1 mapping a constant or a function without parameters maps onto an attribute of the destination database. No foreign attributes are involved.

6.1.1 Mapping a constant onto an attribute

This is easy to implement by using a **Copy**-function and putting quotation marks around the input value in the mapping table. Now the interface can recognise the value as a constant instead of an attribute name, like *qnty_eval = Copy("HOLD").*

6.1.2 Mapping a function onto an attribute

To map a function onto an attribute one has to write the function that performs the essential actions. For example **PatchCreateTime()** computes the current system time used as value for the attribute *createtime* in the destination database.

6.2 1-1 mapping

Here there are also two possibilities. The attribute domains correspond to each other or a special value conversion is required.

6.2.1 The simple case

When the attribute domains correspond to each other there is no conversion necessary. The interface only needs to know which foreign attribute corresponds to which destination attribute. Fortunately this holds for most of the attributes. In the mapping table the **Copy**-function maps these attributes.

6.2.2 Value conversion

In the other case the attributes also map one on one but the values of the attribute in the foreign database are different from the corresponding values in the destination database. This is for example the case with the *orbat_type/obtype* attribute. When the value of *orbat_type* in the foreign database is "G" this corresponds with the value "GOB" for *obtype* in the destination database. An explicit translation is necessary for each value. The **Translate**-function uses a translation table as a dictionary. It compares for this attribute all FromValues in the table with the found foreign value until a match is found. When there is no match the found foreign value is also the returned value. An update of the translation table is enough to translate another attribute. The translation table has the following format:

TRANSLATION		
FieldName	**FromValue**	**ToValue**
orbat_type	G	GOB
...	...	...

Table 5 Format of the translation table

6.3 N-1 mapping

The **Concatenate**-function maps several attributes of the foreign table to a single attribute of the destination table. For the time being this function uses two attributes as parameters, but it is easy to enlarge this number when concatenation of more attributes is desired. Two foreign attributes are written as one into an attribute of the destination table. For example *locekey* is the concatenation of the foreign attributes *source* and *unit_id.*

6.4 1-N mapping

The foreign attribute has to split into two or more parts with the **Part**-function when it maps onto two or more attributes of the destination database. Each part is a value of an attribute of the destination table. This happens for example with the longitude and latitude attributes. The foreign database stores

this information in one attribute (*location_lon* and *location_lat*, respectively) while the destination database has separate degree, minute, second and direction attributes.

6.5 Other mappings

The mappings that are not present in the *GOB1* data import are 1-0 and M-N mapping. 1-0 mapping denotes the foreign attributes that are not used in the destination database and that are of no further interest for this application. However, they can get interesting when implementing the data transfer the other way around. M-N mapping is not looked for at the moment, but it is possible that in other views this will occur. One could think of data out of two or more foreign tables that has to map onto two or more attributes in the destination database.

6.6 The resulting mapping dictionary for the GOB1 view

To create a table to map every foreign attribute onto his corresponding destination attribute for the *GOB1* view the following functions are required:

- *value1 := Copy(value2)* - value2 is the foreign attribute that is copied into value1, the attribute in the destination database. This is one of the functions that implements the 1-1 mapping. When value2 is a constant instead of an attribute the function is used for 0-1 mapping.
- *value1 := SourceDB()* - returns the database the data is imported from, another option for 0-1 mapping.
- *value1 := PatchCreateTime()* - returns the current system time, also used for 0-1 mapping.
- *value1 := Concatenate(value2, value3)* - value2 and value3 are foreign attributes that form the destination attribute value1 by concatenating value2 and value3, which implements N-1 mapping.
- *value1 := Part(value2, start, length)* - value2 is a foreign attribute that is copied from start to start + length into value1, the destination attribute, where start and start + length indicate positions in a string. This is the implementation of the 1-N mapping.
- *value1 := Translate("attr", value2)* - converts attribute attr with foreign value value2 to an explicit destination value, value1. This is the other implementation of the 1-1 mapping.

One can nest these functions, write extra user defined functions and when necessary one can also use the MS Access functions to map attributes.

When the table name precedes the attribute name, the attribute name exists in two or more tables, but is mapped differently. For example the *cc*-field in the *Units* table is mapped onto the *country* attribute, whereas the *cc*-field in the *My_Eq* table is not mapped at all. When a table name contains a hyphen (-) one has to put the name between square brackets ([ ]).

Because of the declarative style of the mapping dictionary it is very easy to adapt the interface making the import an export of data. Only the table and attribute names have to be exchanged and the functions have to be reversed to obtain an export from CRESP into a foreign database. Currently only the import from foreign databases into CRESP will be used.

This leads to the following table:

MAPPING	
DestAttr	**Function**
activity	Copy(activity)
arms	Translate("Units.arms", arms)
asat	Copy(effective_time)
categ	Copy(category)
cmd_rel	Copy(cmd_rel)
CmdRel.sup_id	Concatenate(SourceDB(), [GOB1_Cmd-Rel].sup_unit_id)
CmdRel.ver_code	Copy([GOB1_Cmd-Rel].verific_code)
com_level	Translate("Units.command_level", command_level)
comb_eff	Copy(ce)
confidence	Copy(verific_code)
createtime	PatchCreateTime()
curflag	Copy(1)
en_friend	Translate("Units.en_friend", en_friend)
eq_type	Copy(Left(equip_type_categ, 20))
GobRep.rep_txt	Copy([GOB1_Unit-Sts].comments)
GobRep.ver_code	Copy([GOB1_Unit-Sts].verific_code)
lat_deg	Part(location_lat, 1, 2)
lat_dir	Part(location_lat, 7, 1)
lat_min	Part(location_lat, 3, 2)
lat_sec	Part(location_lat, 5, 2)
location	Copy(location_name)
locekey	Concatenate(SourceDB(), unit_id)
lon_deg	Part(location_lon, 1, 3)
lon_dir	Part(location_lon, 8, 1)
lon_min	Part(location_lon, 4, 2)
lon_sec	Part(location_lon, 6, 2)
model	Copy(equip_type_name)
My_Eq.rep_txt	Copy([GOB1_Unit-Eqp].comments)
My_Eq.ver_code	Copy([GOB1_Unit-Eqp].verific_code)
name	Copy(name)
no_curr	Copy(quantity)
object_id	Copy(unit_id)
obtype	Translate("Units.orbat_type", orbat_type)
org	Copy(organisation)
pers_str	Copy(pers_strength)
qnty_eval	Copy("HOLD")
react_time	Copy(0)
source	SourceDB()
sub_ id	Concatenate(SourceDB(), sub_unit_id)
symbol	Translate("symbol", symbol)
Units.cc	Copy(GOB1_Units.country)
updatetim	Copy(update_time)

Table 6 The mapping table of GOB1

7. Mapping process

Table 6 is the definition of a mapping dictionary. The interface, written in Access Basic, imports the data from the foreign data files into the destination database using this mapping dictionary. It retrieves the mapping specifications, for each attribute in the destination database, in the mapping table. Then it imports the data from the foreign tables, after the specified conversion has taken place, into the tables of the destination database. Executing code directly from a table is not possible in

Access. In the following some options are described for the implementation of the mappings specified in the dictionary.

7.1 SQL

An option to transfer data is to use SQL. The only restriction is that record by record processing is not possible. The query will transfer the whole table at once. The query would look like this:

INSERT INTO *DestTable*({*M.DestAttr(1), M.DestAttr(2), ..., M.DestAttr(n)*}+)
SELECT {*M.Function(1), M.Function(2), ..., M.Function(n)*}
FROM *ForeignTable*;

Here M is the mapping table and (i) denotes the record number in the mapping table. The following routine describes the Access Basic function TransferData(*ForeignTable, DestTable, MappingTable*) with the last implementation. TransferData(*ForeignTable, DestTable, MappingTable*) has to be executed for all tables defined in the view.

TransferData(*ForeignTable, DestTable, MappingTable*)
Temp1 := empty_string
Temp2 := empty_string
For Each *DestAttr* Of *DestTable*
Select *MappingRecord M* From *MappingTable* Where *M.DestAttr = DestAttr*
Add *M.DestAttr* To *Temp1*
Add *M.Function* With Available Arguments To *Temp2*
End For
Execute "INSERT INTO *TempTable*(*Temp1*) SELECT *Temp2* FROM *ForeignTable*;"
End Routine

7.2 An evaluation function for string expressions

The Access Basic function Eval(stringexpr) is another option. This function evaluates the expression "stringexpr" and returns its value. When the mapping table stores the function and its argument(s) as strings, the interface first looks up the values of the arguments in the current record of the foreign table. Then it puts those together with the function name in a string, so Eval can evaluate it. The result will be the value of the in the mapping table specified attribute of the current record of the destination table. For this implementation the mapping table needs a different format than Table 6. To be able to find the values of the arguments of the foreign table the format has to be the following:

MAPPING				
DestAttr	**Function**	**Arg1**	**Arg2**	**Arg3**
activity	Copy	activity		
arms	Translate	arms		
locekey	Concatenate	source	unit_id	
lat_deg	Part	location_lat	0	2
....	....	....	....	....

Table 7 Format of mapping table when not using SQL to transfer the data

It implies that for a function with four or more arguments, the entire table has to be adapted to this function. This is not a practical solution.

This computation of the values of the attributes of the destination database tables will be done for all records in the foreign table. The following routine describes the data transfer from the foreign table to the corresponding destination table. The description is in a high level:

```
TransferData(ForeignTable, DestTable, MappingTable )
    For Each ForeignRecord Of ForeignTable
        Add New DestRecord To DestTable
        For Each DestAttr Of DestRecord
            Select MappingRecord M From MappingTable Where M.DestAttr = DestAttr
            If M Is Found Then
                Select ForeignAttr1 From ForeignRecord where ForeignAttr1 = M.Arg1
                Select ForeignAttr2 From ForeignRecord where ForeignAttr2 = M.Arg2
                Select ForeignAttr3 From ForeignRecord where ForeignAttr3 = M.Arg3
                tempvalue :=  M.Function With Attributes ForeignAttr1, ForeignAttr2,
                              ForeignAttr3 When Available
                DestAttr := Eval("tempvalue")
            End If
        End For
    End For
End Routine
```

With this implementation one adds a new record to the destination table for each existing foreign record. This means that one needs the temporary destination tables again, because one only wants to keep the latest data of each unit.

One can implement this function in different ways.

- For... Next construction,
- Seek method.

With the for... next construction each record of the mapping table is compared to the attribute in the destination table whose value is searched until a match is found for all attributes of the destination table. One has to do a linear search on the whole table. So on average halve the number of records has to be searched. The seek method uses an index on the key of the table to find the searched record, so just a small part of the records has to be checked. For this option the index has to be available, but that is no problem.

7.3 Get the best implementation

We have analysed three options for implementing the data transfer. To investigate the best one, we compared the execution times of the routine TransferData(*ForeignTable, DestTable, MappingTable*). The results are in the following table:

	20 records	**40 records**	**80 records**	**320 records**
For... Next	± 24 seconds	± 49 seconds	± 95 seconds	<no use>
Seek Method	± 5 seconds	± 7 seconds	± 14 seconds	± 52 seconds
SQL Statement	± 2 seconds	± 2 seconds	± 4 seconds	± 5 seconds

Table 8 A performance comparison of different implementations

The Seek Method and the SQL statement are much faster than the For... Next construction. It is obvious that the SQL statement is the best solution. It achieves the fastest results and the mapping table keeps the same format no matter how many arguments a function needs. All the implementations transfer the data from the foreign tables to the temporary tables. After that the data has to be transferred to the final destination tables, which will take an extra 2 or 3 seconds.

7.4 Improvement of the SQL statement

Currently the SQL implementation only uses insert queries. To make sure that the final destination tables contain the latest information on units the data is stored in temporary tables first. Then the old data on units is removed from the destination tables and all data is transferred with an insert query. Another solution is to use update queries as well as insert queries. The temporary tables are redundant now, while first the existing records will be updated and only new records will be inserted. This solution is also required for the problems discussed later in section 9.3, so from now on we will use the application that uses first the following update query and then the insert query from above:

```
UPDATE DestTable INNER JOIN ForeignTable
ON DestKey = ForKey
SET M.DestAttr(1) = M.Function(1), M.DestAttr(2) = M.Function(2), .....,
      M.DestAttr(n) = M.Function(n);
```

The function TransferData(*ForeignTable, DestTable, MappingTable*) has to be adapted to the following:

```
TransferData(ForeignTable, DestTable, ForKey, DestKey, KeyMapAttr, MappingTable)
    Temp1 := empty_string
    Temp2 := empty_string
    Temp3 := empty_string
    For Each DestAttr Of DestTable
        Select MappingRecord M From MappingTable Where M.DestAttr = DestAttr
        Add M.DestAttr To Temp1
        Add M.Function With Available Arguments To Temp2
    End For
    Execute "UPDATE DestTable INNER JOIN ForeignTable ON DestKey = ForKey
             SET Temp3;"
    Execute "INSERT INTO DestTable(Temp1) SELECT Temp2 FROM ForeignTable;"
End Routine
```

The only disadvantage of this implementation is that update queries take a long time. When all records have to be updated (the insert query is superfluous in this special case) the former test gives the following results:

	20 records	**40 records**	**80 records**	**320 records**
For... Next	± 24 seconds	± 49 seconds	± 95 seconds	± 434 seconds
SQL Statement	± 29 seconds	± 55 seconds	± 96 seconds	± 269 seconds

Table 9 A performance comparison of different implementations

These results are even worse than the For... Next method. However, one only gets these results in the worst case and this implementation has the following advantages:

- This method is the best solution for the problems that will be discussed in section 9.3.
- The format of the mapping table does not change.
- The temporary tables are superfluous, the data is transferred directly into the dBaseIV tables.

Besides, one has to keep in mind that indexes might speed up the update queries, which cannot be tested at the moment while this application is only tested on a local system. So from now on we will only use the application implemented by the two SQL statements.

8. The resulting interface

The interface allows to import data into the destination database using multiple views of multiple databases. The following is sufficient for specifying the entire interface:

- The *interface dictionary* - specifies the structure of the exported foreign text files;
- The *CRESP data dictionary* - specifies the destination tables into which the interface has to import the data;
- The *mapping dictionary* consisting of:
  - The *translation* table - specifies the attribute translations for the *Translate*-function;
  - The *view map* - specifies the connection of tables and views;
  - The *source map* - specifies the connection of foreign databases and views;
  - The *mapping tables* - specify the correspondences of foreign and destination attributes.

The foreign tables have the same structure as the text files exported by the foreign databases. The mapping table and the translation table keep the same structure. Only new records have to be added to these tables. For the definition of available views there are two tables. The first is a *ViewMap* table that identifies which tables belong to which view and what attributes are the keys in those tables. The second is a *SourceMap* table that determines the origin of the data. With the *ViewMap* table the

ForeignTable column identifies the foreign text file and the foreign table, whereas the *DestTable* column identifies the dBaseIV table. Also the keys in the foreign table as well as the destination table have to be remembered and the attributes that identify the relationship between the tables. The two tables have the following format:

ViewMap					
ViewName	**ForeignTable**	**ForKey**	**DestTable**	**DestKey**	**DestAttr**
GOB1	GOB1_Units	Concatenate(SourceDB(), unit_id)	Units	Units.locekey	locekey
GOB1	GOB1_Unit-Sts	Concatenate(SourceDB(), unit_id)	GobRep	GobRep.locekey	locekey
GOB1	GOB1_Unit-Eqp	Concatenate(SourceDB(), unit_id)	My_Eq	My_Eq.locekey	locekey
GOB1	GOB1_Cmd-Rel	Concatenate(SourceDB(), sub_unit_id)	CmdRel	CmdRel.sub_id	sub_unit_id
....	....	....	....	....	....

Table 10 Format of the ViewMap table

SourceMap		
Name	**SourceNode**	**SourceDatabase**
GOB1	ARRC	THISTLE
....	....	....

Table 11 Format of the SourceMap table

8.1 Execution of the import interface

The user interface of the application will ask the user for the following variables and execute the correct actions. The description is in high level in correspondence with the code above for the TransferData-function.

```
Choose SourceNode          /* The node where the database resides                    */
Choose SourceDatabase      /* The database, to import the data from                  */
Compute Available Views    /* Let the user select only from views of the selected    */
                           /* SourceNode and SourceDatabase                          */
Choose View
Choose DestinationDatabase /* The database, to import the data into                  */
Choose Import Option       /* Selection of a full or incremental import, currently only the */
                           /* full import is implemented                             */
Import Data                /* The actual import action of the data                   */
```

In this application the user interface will look like this:

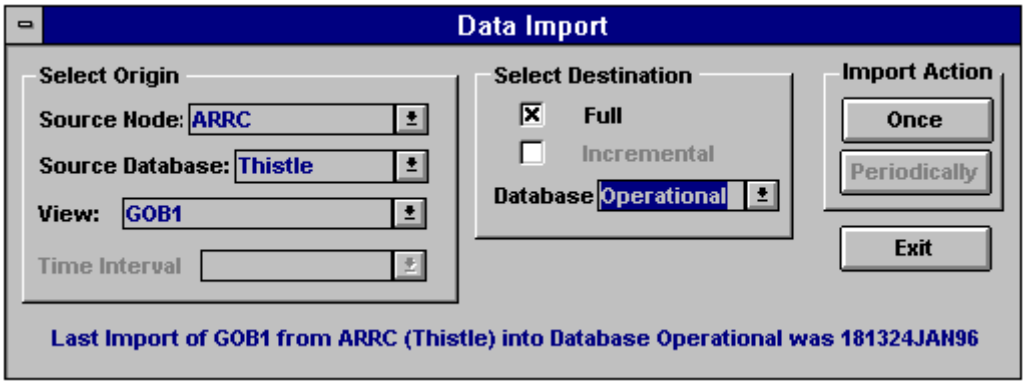

Figure 6 The user interface of the application

The actual data transfer from text files into dBaseIV tables will then be:

```
Routine TransferFromTextFileToDBaseIV()
   For Each Record in View V With V.ViewName = ChosenView
   Get Name Of ForeignTextFile
   Attach V.DestTable              /* Attach the dBaseIV table to the application      */
   ImportText(V.ForeignTable, ForeignTextFile)
                                   /* Transfer data from text file to foreign table    */
   TransferData(V.ForeignTable, V.DestTable, V.ForeignKey, V.DestKey, V.DestAttr,
                MappingTable)      /* Transfer data from foreign table to dest. table  */
  End For
End Routine
```

8.2 Definition of new views

To import data from an already existing view from another database one only has to add the record {ViewName, SourceNode, SourceDatabase} to the *SourceMap* table. However, it must be sure that it is exactly the same view. Otherwise a completely new view has to be added to the application, even if there are minor differences to an existing view.

To create a new view the following actions have to be taken:

- Ensure that the required dBaseIV tables are present in the correct directory. This is the same directory as CRESP program is installed in. The interface itself will make the attachments.
- Create the necessary foreign tables.
- Create a new mapping table.
- Update the translation table with new values.
- Add the names of the foreign and dBaseIV tables to the *ViewMap* table accompanied by the key attributes of those tables and the attribute in the destination table that identifies the mapping between the foreign and the destination table (DestAttr). Also add a new view name.
- Update the *SourceMap* table with the correct source node and database and the new view name.

8.2.1 Naming conventions

Naming conventions are essential keeping the application consistent. The following convention is suggested for adding a new view to the interface.

- For the foreign tables: <ViewName> underscore <Foreign TextFile>, i.e. *GOB1_Units* for view *GOB1* and foreign textfile *Units.ful*.
- For the mapping table: <ViewName> underscore “MappingTable”, i.e. *GOB1_MappingTable*.

MS Access is case insensitive, so it does not matter whether upper or lower case or a mix is used.

9. Case studies

To test whether this interface is as flexible as is assumed the following case studies are examined:

- defining a new view;
- using the same view with a different database;
- key mapping problems.

These will be discussed in the following sections.

9.1 Case study 1: defining a new view

Another view of the THISTLE database could be the *GOB2* view. This is a view that contains the minimum of information needed to display a unit on a map. With this view there is a new concept. *GOB1* maps all attributes from a certain foreign table to one destination table. The *GOB2* view maps one to many, one foreign table has to be spread over more destination tables. The *GOB2* view exists of one *Gob* table that is transferred to the *Units* and *GobRep* tables of the destination database. The *Gob* table has the following format:

GOB				
Field	**Null**	**Format**	**Size**	**Description**
update-time		char(12)	Fix	ZULU Date-Time when information was last updated
source		char(8)	Var	command/system sending the data
unit-id	N	char(15)	Var	originator-specific unique unit identifier
en_friend		char(3)	Var	friend or enemy discriminator
country		char(2)	Fix	country code
organisation		char(8)	Var	abbreviation for the organisation the unit is assigned to
arms		char(15)	Var	unit type
command_level		char(5)	Var	level or size of command
location_name		char(30)	Var	name of site/town of current unit location
location_lat	N	char(7)	Fix	latitude co-ordinate of current unit location, DDMMSSH
location-lon	N	char(8)	Fix	longitude co-ordinate of current unit location, DDDMMSSH
comments		char(255)	Var	free comment field

Table 12 GOB2: Gob table

This view has the following mapping table in which all attributes are preceded by their table names to compel the correct data transfer.

MAPPING	
DestAttr	**Function**
GobRep.lat_deg	Part(GOB2_Gob.location_lat, 1, 2)
GobRep.lat_dir	Part(GOB2_Gob.location_lat, 7, 1)
GobRep.lat_min	Part(GOB2_Gob.location_lat, 3, 2)
GobRep.lat_sec	Part(GOB2_Gob.location_lat, 5, 2)
GobRep.location	Copy(GOB2_Gob.location_name)
GobRep.locekey	Concatenate(SourcDB(), GOB2_Gob.unit_id)
GobRep.lon_deg	Part(GOB2_Gob.location_lon, 1, 3)
GobRep.lon_dir	Part(GOB2_Gob.location_lon, 8, 1)
GobRep.lon_min	Part(GOB2_Gob.location_lon, 4, 2)
GobRep.lon_sec	Part(GOB2_Gob.location_lon, 6, 2)
GobRep.source	SourceDB()
Units.arms	Copy(GOB2_Gob.arms)
Units.cc	Copy(GOB2_Gob.country)
Units.com_level	Copy(GOB2_Gob.command_level)
Units.locekey	Concatenate(SourceDB(), GOB2_Gob.unit_id)
Units.name	Copy(GOB2_Gob.name)
Units.org	Copy(GOB2_Gob.organisation)
Units.source	SourceDB()

Table 13 The mapping table of GOB2

9.2 Case study 2: Using the same view with a different database

To add *GOB1* from the STAFOR database to the interface one normally can use the existing tables for this view. The only thing to be done is adding the record {GOB1, ARRC, STAFOR} to the SourceMap table. However *GOB1* from THISTLE is not exactly the same as *GOB1* from STAFOR. The STAFOR and THISTLE databases are very different from each other and therefore it is not possible to retrieve an identical view from the two databases. So it is better to treat *GOB1* from THISTLE and the similar view from STAFOR as separate cases. Therefore this view from STAFOR will be called *GOB3* and will be treated as a completely new view. The destination tables already present in the application can be used. A new mapping table will be added and of course the foreign tables for this data transfer. Also an extra mapping function is necessary.

- *value1 := ConcatenateIf(value2, value3, value4, value5),* which concatenates STAFOR attribute value2 with value3 or value4 into value1, the destination attribute, depending on value5;

This leads to the following mapping table:

MAPPING	
DestAttr	**Function**
arms	Translate(“arms”, arms)
asat	Copy(effective_time)
ass_status	Copy(com_stat)
categ	Copy(category)
cmd_rel	Copy(cmd_rel)
com_level	Translate(“command_level”, command_level)
comb_eff	Concatenate(readiness_pers, Concatenate(readiness_mat , readiness_trng))
createtime	PatchCreateTime()
CmdRel.sup_id	ConcatenateIf(SourceDB(), [GOB3_Cmd-Rel].opcom_unit_id, [GOB3_Cmd-Rel].opcon_unit_id, [GOB3_Cmd-Rel].cmd_rel)
Units.cc	Copy(GOB3_Units.country)
eff_time	Copy(effective_time)
eq_type	Copy(equip_type_categ)
lat_deg	Part(location_lat, 1, 2)
lat_dir	Part(location_lat, 7, 1)
lat_min	Part(location_lat, 3, 2)
lat_sec	Part(location_lat, 5, 2)
location	Copy(location_name)
locekey	Concatenate(SourceDB(), unit_id)
lon_deg	Part(location_lon, 1, 3)
lon_dir	Part(location_lon, 8, 1)
lon_min	Part(location_lon, 4, 2)
lon_sec	Part(location_lon, 6, 2)
name	Copy(name)
obtype	Translate(“orbat_type”, orbat_type)
org	Copy(“NATO”)
qnty	Copy(quantity)
qnty_eval	Copy(“HOLD”)
source	SourceDB()
sub_id	Concatenate(SourceDB(), sub_unit_id)
updatetim	Copy(update_time)

Table 14 The mapping table of GOB3

9.3 Case Study 3: Key mapping problems

In the previous two case studies it has not been necessary to remember which attributes are keys for which tables, because the mapping between the keys of the foreign tables and the destination tables is identical. This means that the relations are preserved without paying special attention to the mappings of the keys. For example, in the foreign database the *GOB1* view has a 1-N relationship between the *GOB1_Units* table and the *GOB1_Unit-Sts* table, which must stay the same relationship for the *Units* table and the *GobRep* table.

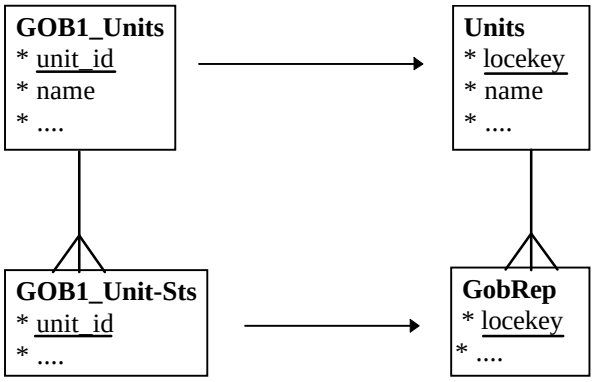

Figure 7 Key mapping between tables

In this case it is very easy. The *unit_id* maps 1-1 onto *locekey*, which is the key in the destination tables. Also when more attributes form the key each separate foreign attribute is mapped 1-1 onto an attribute that is part of the key in the destination table. For example *unit_id* denotes the relation between the units and their status, but the key between the unit status and the equipment is formed by the attribute *unit_id* together with *update_time*, both mapped 1-1 onto the key attributes of the destination table, *locekey* and *updatetim* respectively.

A difficulty arises when in the foreign database a different attribute becomes the key, for example the attribute *name*. Now one has to preserve the same relationships, but cannot just map all attributes. If this is the case one has to check for each *name* in *GOB1_Units* whether this *name* already exists in *Units*. If the *name* is present one has to use the corresponding *locekey* as a key in all destination tables for this particular unit, otherwise generate a new unique *locekey*. For this reason one has to remember the key attributes and the attribute that denotes the relationship between the tables.

For the other tables on the foreign side one has to look up the value of the *locekey* in the *Units* table to make sure that one uses for a new unit the same generated key in all destination tables. Instead of deleting all former data and inserting the new data, one has to update the existing records and append the records of new units. With this construction the data transfer works for the identical key mapping as well as a different key mapping.

10. Achievements and open issues

10.1 Achievements

With my application the integration of views has become much easier. This applies to the integration of existing views from new databases as well as to new views. With the developed application it is easy to import an existing view from a new database. Importing a new view takes a little longer, but is still easy to do. So the problems discussed in section 3.1 have been solved in such a way that insertion of multiple external views into an application by a single program has been achieved.

The main difference between my application and the old application is the introduction of the mapping dictionary; the old application has the mappings hard-coded in the program. Using such a mapping dictionary makes it much easier to adapt the mappings and it furthermore, solves the problem discussed in section 3.2. Also by using update queries as well as insert queries instead of insert queries only, one does not need the temporary destination tables as used in the old application. One can transfer the data immediately from the foreign tables into the target database tables. A drawback is that my application is slower most of the time. The old application takes about 136 seconds to transfer 750 records regardless the state of the dBaseIV tables, while the execution time of my application depends on the number of updatable records. When the interface has to insert new records only (no pre-existence in the destination tables of any record in the data set) the interface takes about 93 seconds to transfer the same data set of 750 records. However, when all records have to be updated, it takes more than six minutes. This is the worst case though and because flexibility is more important than speed, this is not insurmountable. This difference is mainly due to the poor efficiency of the update query. Indexes might improve this performance.

10.2 Open issues

There are still some unsolved items that still require some analysis and research.

10.2.1 Importing erroneous data (section 3.4)

It is possible that the importable data contains errors, for example in text fields or with the use of data from spreadsheets. One can import data from spreadsheets by transferring the data of a spreadsheet in the same way as data from text files. Now one has foreign data in a table that has to be transferred to one or more destination tables. This can be handled in the same way as the import of the data of *GOB2*. However, the restriction is that this approach assumes that the spreadsheet contains correct data, which cannot be taken for granted. Data of spreadsheets is imprecise and therefore it has to be checked on typo's. One can think of using a similar concept as the spelling checker included in MS Word. There is a dictionary that contains words already approved by a user. When the spelling checker finds a word that it does not find in this dictionary it asks the user what to do with it. The spelling checker will give several options. The user can type a correct word, choose one of the suggestions of the spelling checker or ignore the error. One should also be able choosing to add the word to the dictionary, so that the spelling checker recognises the word as a correct one the next time. To complicate the problem further, it is not enough to have a list of all approved words in a dictionary, because one needs to know the approved words for a certain attribute. As the fact that a value is correct for one attribute does not imply that it is correct for another attribute a special solution has to be found.

10.2.2 The mandatory data problem (section 3.3)

At the moment only the *unit_id* is mandatory and a record without a *unit_id* is of no use to the application. The current approach is to neglect such a record, which will be a too simple approach in the long term. User intervention, for example, could be considered as well.

10.2.3 The incremental data transfer

The incremental data transfer is not yet implemented. To do this the used update query has to be changed. Now the update query updates all fields in the table whether the foreign record contains data or not. With the incremental data transfer this would lead to loss of data. So one has to check whether the foreign attribute contains data or just null values before adding the destination attribute and its conversion to the list of updatable attributes in the function *TransferData*.

10.2.4 Unique key generation

Also the unique key generation is not yet implemented. For the time being a random number is chosen as the key for an unknown unit. But this will not necessarily become always a unique number. It cannot be too difficult finding an algorithm to do this and implement the unique key generation very shortly.

10.2.5 Composite keys

Composite keys have not been taken into consideration. Each table has only one attribute (*unit_id* or *locekey*) as key at the moment.

11. List of Tables and figures

12. References

[1] Smith, J. M., Bernstein, P. A., Dayal, U., Goodman, N., Landers, T., Lin, K. W. T. and Wong, E. Multibase -- integrating heterogeneous distributed database systems, Computer Corporation of America, Cambridge, Massachusetts, March 1981.

[2] Templeton, M., Brill, D., Dao, S. K., Lund, E., Ward, P., Chen, A. L. P. and MacGregor, R. Mermaid -- A front-end to distributed heterogeneous databases, Proceedings of the IEEE, vol. 75, no. 5, May 1987.

[3] Homer, P. T. and Schlichtling, R. D. Configuring scientific applications in a heterogeneous distributed system, Proceedings of the 2nd International Workshop on Configurable Distributed Systems, 159-168, Pittsburgh PA, March 1994.

[4] Hammer, J., McLeod, D. and Si, A. An intelligent system for identifying and integrating non-local objects in federated database systems, Proceedings of the 27th Hawaii International Conference on System Sciences, pages 389-407, Computer Society of the IEEE, University of Hawaii, USA, January 1994.

[5] CRESP Technical Specifications

13. Interesting internet addresses

Most of the used articles are found at the internet using Netscape Navigator 1.22 for Windows. The following addresses contain useful information. This can be a collection of interesting links or a list of

publications. At most of the pages with publications one can download the articles immediately in postscript format.

- **http://ccs-www.cs.umass.edu/db.html**, the database page of the University of Massachusetts.
- **http://www-db.stanford.edu/pub/**, the list of publications on databases from the Stanford University, where one can find articles on the Remote-Exchange project at the page of Joachim Hammer.
- **http://bunny.cs.uiuc.edu/publications.html**, the ACM SIGMOD Index of Database Publication Servers, where one can find many addresses with information on database research.
- **http://www.cs.arizona.edu/schooner/index.html**, where one can find all information about research on the Schooner Interconnecting System at the University of Arizona.

14. Used abbreviations

CRESP	-- Crisis Response Prototype
GOB	-- Ground Orbat
NATO	-- North Atlantic Treaty Organisation
Orbat	-- Order of Battle (identification, location, strenght of units)
SHAPE	-- Supreme Headquarters Allied Powers Europe
STAFOR	-- Status of Forces
STC	-- SHAPE Technical Centre

Appendix A Source Code

```
'=============================================================================
' Module: Declarations
' This module contains all declarations of global variables and constants. In this way it is easier to change values
' that are used a lot when necessary. The module also contains some functions necessary to run the application in
' a windows environment.
'=============================================================================
Option Compare Database      'Use database order for string comparisons.
Option Explicit              'Used to force explicit declaration of all variables.

'   User & Kernel Library Functions:
Declare Sub SetWindowText Lib "User" (ByVal hWnd As Integer, ByVal lpString As String)
Declare Function GetActiveWindow Lib "User" () As Integer
Declare Function GetPrivateProfileString Lib "Kernel" (ByVal lpApplicationName As String, ByVal lpKeyName As String,
                                                       ByVal lpDefault As String, ByVal lpReturnedString As String,
                                                       ByVal nSize As Integer, ByVal lpFileName As String) As Integer
'   QPRO200.DLL Library Function
Declare Function WinDir Lib "QPRO200.DLL" () As String

'   Global variables:
Global MyDB As Database                     'Used so much that it is better to use a global declaration
Global MyWorkspace As WorkSpace             'Used so much that it is better to use a global declaration
Global A_Datadir As String

'   Global constants:
Global Const SUPERTITLE = "CRESP PROTOTYPE V1.X"   'Title that appears in header bar at start of application
Global Const DTFORMAT = "yymmddhhnnss"      'Format of date and time for data in the database
Global Const DTGFORMAT = "ddhhnnmmmyy"      'Format of date and time for application
Global Const ENVIRONMENT = "Environment"
Global Const ARRC_DATADIR = "ARRC Operational DataDirectory"
Global Const CRESP_DATADIR = "CRESP DataDirectory"
Global Const FORM1 = "FORM1"                'Name of form which starts the application
Global Const PATH = "C:\CRESP\IMPORT\"      'Directory name that has to be followed by database name to
                                            'identify the place the input data has to reside
Global Const DATADIR = "C:\CRESP\"          'Directory in which the CRESP application has to reside as well
                                            'as this application
Global Const DELIMITER = "AT_SIGN_DELIMITER"

'=============================================================================
' ConnectString provides information about the source of a database used in an attached table.
'=============================================================================
Function ConnectString (DATADIR As String) As String
   ConnectString = "dBASE IV; DATABASE=" + DATADIR + ";"
End Function

'=============================================================================
' ImportAgent identifies the initialisation file for using this application in a windows environment.
'=============================================================================
Function ImportAgent () As String
   ImportAgent = WinDir() + "\IMPAGENT.INI"
End Function
```

```
'=============================================================================
' Module: Mapping Functions
' This module contains all user defined functions to map the foreign attributes onto their matching destination
‘ attributes. All functions have a specific value of an attribute as input and return the corresponding destination
‘ value as output.
'=============================================================================
Option Compare Database        'Use database order for string comparisons

'=============================================================================
' Concatenate pastes two string values into one; nest Concatenate functions for more than two input values.
'=============================================================================
Function Concatenate (Arg1 As String, Arg2 As String, Arg3 As String) As String
   Concatenate = Arg1 & Arg2 & Arg3
End Function

'=============================================================================
' ConcatenateIf concatenates the source with the correct unit_id depending on the input value of Arg4 (OPCOM
' or OPCON).
'=============================================================================
Function ConcatenateIf (Arg1 As String, Arg2 As String, Arg3 As String, Arg4 As String)
   If Arg4 = "OPCOM" Then
      ConcatenateIf = Arg1 & Arg2
   Else 'Arg4 = “OPCON”
      ConcatenateIf = Arg1 & Arg3
   End If
End Function

'=============================================================================
' Copy just returns the input string.
'=============================================================================
Function Copy (Arg1 As String)
   Copy = Arg1
End Function

'=============================================================================
' GenerateKey checks whether the destination key for a certain foreign record already exists or not. This key is
' returned if it exists, otherwise a new key is returned. This function has to be adapted still. Now it returns a
' random number as a new key instead of an unique one.
'=============================================================================
Function GenerateKey (fkey As String, dkey As String, dattr As String, ftbl As String, dtbl As String)
   Dim KeySet As Recordset

   Set KeySet = MyDB.OpenRecordset(SelectKey(ftbl, dtbl, fkey, dkey, dattr), DB_OPEN_SNAPSHOT)
   If Not KeySet.EOF Then
      KeySet.MoveFirst
      GenerateKey = KeySet.Fields(0)
   Else
      GenerateKey = SourceDB() & CStr(Int(100 * Rnd))
   End If
   KeySet.Close
End Function

'=============================================================================
' Part returns the string value of arg1 starting at "start" and ending at "start+length".
'=============================================================================
Function Part (Arg1 As String, start As Long, length As Long)
   If Not IsNull(Arg1) Then
      Part = Mid(Arg1, CLng(start), CLng(length))
   Else
      Part = Arg1
   End If
End Function

'=============================================================================
' PatchCreateTime returns the current system time.
'=============================================================================
Function PatchCreateTime ()
   PatchCreateTime = Format(Now, DTFORMAT)
End Function
```

```
'=========================================================================================
' SourceDB returns the name of the foreign database.
'=========================================================================================
Function SourceDB ()
   SourceDB = UCase(Forms![Form1]![source db])
End Function

'=========================================================================================
' Translate gets a foreign value and determines which value is the corresponding CRESP value of an attribute
' using the Translation table
'=========================================================================================
Function Translate (FromField As String, Arg As String) As String
   Dim tempset As Recordset

   Set tempset = MyDB.OpenRecordset(SelectTransVal(FromField, Arg))
   If Not tempset.EOF Then 'value has to be translated to a value specified in the translation table.
      tempset.MoveFirst
      Translate = tempset![TOVALUE]
   Else 'value does not exist in translation table, so it can be returned without a translation.
      Translate = Arg
   End If
   tempset.Close
End Function


'=========================================================================================
' Module: SQL Library
' This module contains all SQL statements in the program. The statements are kept as global as possible to
' prevent problems with a new view definition as much as possible.
'=========================================================================================
Option Compare Database   'Use database order for string comparisons
Option Explicit


'=========================================================================================
' DeleteEmptyRows removes all rows in table ‘tbl’ where the key ‘fld’ is missing.
'=========================================================================================
Function DeleteEmptyRows (tbl As String, fld As String)
   DeleteEmptyRows = "DELETE FROM [" & tbl & "] WHERE " & fld & " = Null;"
End Function

'=========================================================================================
' DeleteRecords deletes all records in table ‘tbl’.
'=========================================================================================
Function DeleteRecords (tbl As String)
   DeleteRecords = "DELETE FROM [" + tbl + "];"
End Function

'=========================================================================================
' InsertIntoDestTable adds all converted data which has not been imported yet from foreign table ‘tbl2’ to
' destination table ‘tbl1’.
'=========================================================================================
Function InsertIntoDestTable (tbl1 As String, tbl2 As String, ForKey As String, DestKey As String, set1 As String,
                              set2 As String)
   InsertIntoDestTable = "INSERT INTO [" & tbl1 & "](" & set1 & ") " &
                         "SELECT DISTINCTROW " & set2 & “ “ &
                         "FROM [" & tbl1 & "], [" & tbl2 & "]" &
                         “WHERE NOT " & ForKey & " IN (SELECT " & DestKey & “ “ &
                                                      "FROM [" & tbl1 & "]);"
End Function

'=========================================================================================
' SelectKey determines the key value in the destination table corresponding to the value of the foreign key.
'=========================================================================================
Function SelectKey (FromTable As String, ToTable As String, ForKey As String, DestKey As String, DestAttr As String)
   SelectKey = "SELECT " & DestKey & " FROM UNITS WHERE “ & DestAttr & “ = '" & ForKey & "';"
End Function
```

```
'=============================================================================
' SelectLastImport selects last import of View from SourceNode, SourceDatabase and Orbat
'=============================================================================
Function SelectLastImport (View As String, SourceNode As String, SourceDatabase As String, Orbat As String)
   SelectLastImport = "SELECT DTSERIAL
                       FROM IMPORTLOG
                       WHERE TRIM(VIEW) = '" + Trim(View) + "' AND TRIM(SOURCENODE) = '" +
                             Trim(SourceNode) + "' AND TRIM(SOURCEDATABASE) =  '" +
                             Trim(SourceDatabase) + "' AND TRIM(ORBAT) = '" + Trim(Orbat) +
                             "' AND DTSERIAL = (SELECT MAX(DTSERIAL)
                                                FROM IMPORTLOG
                                                WHERE TRIM(SOURCENODE) = '" + Trim(SourceNode) +
                                                      "' AND TRIM(SOURCEDATABASE) =  '" +
                                                      Trim(SourceDatabase) + "' AND TRIM(ORBAT) = '" +
                                                      Trim(Orbat) + "');"
End Function

'=============================================================================
' SelectTransVal determines the destination value of an attribute that is converted with the Translate function.
'=============================================================================
Function SelectTransVal (FromField As String, Arg As String)
   SelectTransVal = "SELECT TOVALUE
                     FROM TRANSLATION
                     WHERE FIELDNAME = '" & FromField & "' AND FROMVALUE = '" & Arg & "';"
End Function

'=============================================================================
' SQLSelectViews returns all views from a given source node and database
'=============================================================================
Function SQLSelectViews (SourceNode As String, SourceDatabase As String)
   SQLSelectViews = "SELECT NAME
                     FROM SOURCEMAP
                     WHERE TRIM(SOURCENODE) = """ + Trim(SourceNode) +
                           """ AND TRIM(SOURCEDATABASE) = """ + Trim(SourceDatabase) + """;"
End Function

'=============================================================================
' UpdateDestTable is used for all units that already existed in the destination tables before the import started.
' The fields of those records will be updated with the new values.
'=============================================================================
Function UpdateDestTable (tbl1 As String, tbl2 As String, ForKey As String, DestKey As String, vallist As String)
   UpdateDestTable = "UPDATE [" & tbl1 & "] INNER JOIN [" & tbl2 & "]
                      ON " & DestKey & " = " & ForKey & "
                      SET " & vallist & ";"
End Function
```

```
'=============================================================================
' Module: Utilities
' This module contains some useful functions used with the data transfer. It contains functions to send messages
' to the user of the application and to add information to a log table when data of a certain view is imported into
' CRESP.
'=============================================================================
Option Compare Database   'Use database order for string comparisons

'=============================================================================
' AddToLog adds current import action to the Importlog table
'=============================================================================
Sub AddToLog ()
   Dim SourceNode As String
   Dim SourceDatabase As String
   Dim View As String
   Dim Orbat As String
   Dim TempSet As Recordset

   SourceNode = Forms.[Form1]![source node]
   SourceDatabase = Forms.[Form1]![source db]
   View = Forms.[Form1]![view]
   Orbat = Forms.[Form1]![target db]

   Set TempSet = MyDB.OpenRecordset("IMPORTLOG")
   If Not TempSet.EOF Then
      TempSet.MoveLast
   End If
   TempSet.AddNew
   TempSet![SOURCENODE] = SourceNode
   TempSet![SOURCEDATABASE] = SourceDatabase
   TempSet![VIEW] = View
   TempSet![ORBAT] = Orbat
   TempSet![DTSERIAL] = Now
   TempSet.Update
   TempSet.Close
End Sub

'=============================================================================
' Attached determines whether the destination table is already attached to the application or not.
'=============================================================================
Function Attached (tablename As String) As Integer
   Dim i As Integer

   For i = 0 To MyDB.TableDefs.count - 1
   'Check all tables to see if the table with name “tablename” is already attached.
      If MyDB.TableDefs(i).name = tablename Then
         Attached = True
         Exit For
      End If
   Next i
End Function
```

```
'=============================================================================
' DataSourceAndOrbat_OK checks whether the input parameters are filled in by the user.
'=============================================================================
Function DataSourceAndOrbat_OK () As Integer
   DataSourceAndOrbat_OK = True
   If IsNull(Forms.[Form1]![source node]) Then
      DataSourceAndOrbat_OK = False
      Exit Function
   End If
   If IsNull(Forms.[Form1]![source db]) Then
      DataSourceAndOrbat_OK = False
      Exit Function
   End If
   If IsNull(Forms.[Form1]![view]) Then
      DataSourceAndOrbat_OK = False
      Exit Function
   End If
   If IsNull(Forms.[Form1]![target db]) Then
      DataSourceAndOrbat_OK = False
      Exit Function
   End If
End Function

'=============================================================================
'   System: CRESP V0.9
'   Module: Utilities
'   Developed by:   AEB/IS
'                   SHAPE Technical Centre
'                   The Hague
'   Date: April '95
'   Notes:
'=============================================================================
Function GetImportAgentParams (Chapter As String, Param As String, Value As String) As Integer
   Dim SpaceHolder As String
   Dim ReturnLen As Integer
   SpaceHolder = String(255, 0)

   GetImportAgentParams = True
   On Error GoTo ErrorHandler
   Open ImportAgent() For Input As #1
   Close #
   ReturnLen = GetPrivateProfileString(Chapter, Value, Space(0), SpaceHolder, 255, ImportAgent())
   Param = Left$(SpaceHolder, ReturnLen)
   Exit Function

ErrorHandler:
   Select Case Err
      Case 53: MsgBox "Error 53: IMPAGENT.INI not found"
      Case Else
   End Select
   GetImportAgentParams = False
   Exit Function
End Function

'=============================================================================
' LastImport returns the string with the message when the last import of this particular view took place
'=============================================================================
Function LastImport (View As String, SourceNode As String, SourceDatabase As String, Orbat As String,
                     DTG As String) As String
   LastImport = "Last Import of " + View + " from " + SourceNode + " (" + SourceDatabase + ") into Database " + Orbat
                + " was " + UCase(DTG)
End Function
```

```
'=============================================================================
' Messagebox puts a message box on the screen on top of the application. One has to remove this message box
' before one can go on with the application.
'=============================================================================
Sub MessageBox (message As String)
   Dim Warning As Integer, MsgDialog As Integer
   Const MB_OK = 0
   Const MB_ICONEXCLAMATION = 48
   Const MB_DEFBUTTON1 = 0

   MsgDialog = MB_OK + MB_ICONEXCLAMATION + MB_DEFBUTTON1
   Warning = MsgBox(message, MsgDialog, "Warning")
End Sub

'=============================================================================
' RefreshLastRetrieved finds date and time of last import of the view that is going to be imported.
'=============================================================================
Sub RefreshLastRetrieved ()
   Dim TempSet As Recordset
   Dim SourceNode As String
   Dim SourceDB As String
   Dim View As String
   Dim DTG As String
   Dim Orbat As String
   Const BLACK = 8388608

   'Check if data source & orbat are selected. If not go & Exit
   If Not DataSourceAndOrbat_OK() Then
      Exit Sub
   End If

   SourceNode = Forms.[Form1]![source node]
   SourceDB = Forms.[Form1]![source db]
   View = Forms.[Form1]![view]
   Orbat = Forms.[Form1]![target db]
   Set TempSet = MyDB.OpenRecordset(SelectLastImport(View, SourceNode, SourceDB, Orbat))
   If Not TempSet.EOF Then
      TempSet.MoveFirst
      DTG = Format(TempSet![DTSERIAL], DTGFORMAT)
      Forms.[Form1]![Text5].Forecolor = BLACK
      Forms.[Form1]![Text5].Caption = LastImport(View, SourceNode, SourceDB, Orbat, DTG)
   Else
      Forms.[Form1]![Text5].Forecolor = BLACK
      Forms.[Form1]![Text5].Caption = "Data retrieval originated by selected Source has not been registered yet"
   End If
   TempSet.Close
End Sub

'=============================================================================
' RefreshToImporting gives a message that import has started. This message will stay on screen during the
' whole import operation and will be removed when the import is finished.
'=============================================================================
Sub RefreshToImporting ()
   Forms![Form1]![Text5].Forecolor = 128
   Forms![Form1]![Text5].Caption = "IMPORTING"
   DoEvents
End Sub

'=============================================================================
' RemoveTableName strips table name of arg to get the sole attribute name when ‘arg’ has the following form:
' tablename.attributename. If ‘arg’ is only an attribute name, arg itself will be returned.
'=============================================================================
Function RemoveTablename (arg As Field) As String
   RemoveTablename = Right$(arg, Len(arg) - InStr(arg, "."))
End Function
```

```
'=============================================================================
' SelectViews computes available views from chosen SourceNode and SourceDatabase. Is called after update of
' SourceDatabase
'=============================================================================
Sub SelectViews ()
   Dim TempSet1 As Recordset
   Dim TempSet2 As Recordset
   Dim SourceNode As String
   Dim SourceDB As String

   'Check if Data Source is selected. If not, exit sub.
   If IsNull(Forms.[Form1]![source node]) Then
      Exit Sub
   Else
      SourceNode = Forms.[Form1]![source node]
   End If
   SourceDB = Forms.[Form1]![source db]

   Set TempSet1 = MyDB.OpenRecordset(SQLSelectViews(SourceNode, SourceDB))
   Set TempSet2 = MyDB.OpenRecordset("CurrentViews")

   DoCmd RunSQL DeleteRecords("CurrentViews")
   If Not TempSet1.EOF Then
      TempSet1.MoveFirst
   End If
   Do Until TempSet1.EOF
      TempSet2.AddNew
      TempSet2.fields(0) = TempSet1.fields(0)
      TempSet1.MoveNext
      TempSet2.Update
   Loop
   TempSet1.Close
   TempSet2.Close
End Sub

'=============================================================================
' Module: Main Module
' This module contains all functions that do the actual data transfer. For the transfer the following routines will
' be called:
' - Initialise at the opening of Form1
' - ImportText at the click on the "import" button, which uses the following routines:
'           - TransferFromTextFilesToDBaseIV, the actual data transfer
'           - some functions to do some administration work
' - TransferFromTextFilesToDBaseIV uses the following routines:
'           - ImportText to transfer the data from text files to tables
'           - AttachdBaseIVTable to make sure the attachment of the destination table exists
'           - TransferData which performs the actual transfer from foreign tables to the dBaseIV tables.
'=============================================================================
Option Compare Database      'Use database order for string comparis
Option Explicit              'Force explicit variable declaration

'=============================================================================
' AttachDBaseIVTable finds out if the attached dBaseIV data files are connected already. If not, attach them as
' specified in the impagent.ini file
'=============================================================================
Sub AttachdBASEIVTable (tablename As String)
  Dim MyTableDef As TableDef

  If Attached(tablename) Then
          MyDB.TableDefs.Delete tablename
   End If

   Set MyTableDef = MyDB.CreateTableDef(tablename)
   MyTableDef.Connect = ConnectString(DataDir)
   MyTableDef.SourceTableName = tablename
   MyDB.TableDefs.Append MyTableDef
End Sub
```

```
'=============================================================================
' ImportOnce imports data only once, actual trigger to start import
'=============================================================================
Sub ImportOnce ()
   'Check if all selection criteria are specified
   If DataSourceAndOrbat_OK() Then
            'Inform User about start of import
            Call RefreshToImporting

            'transfer data from text files via foreign tables to destination tables
            Call TransferFromTextFilesToDBaseIV

            'Register the Activity
            Call AddToLog
            Call RefreshLastRetrieved
            Forms![Form1]![Text5].Caption = ""
            DoEvents
   End If
End Sub

'=============================================================================
' ImportText removes data out of FromTable and inserts data out of ImportFile into it. ImportFile is a text file
' with the full path name specified.
'=============================================================================
Sub ImportText (FromTable As String, ImportFile As String)
   DoCmd RunSQL DeleteRecords(FromTable)
   DoCmd TransferText A_IMPORTDELIM, DELIMITER, FromTable, ImportFile
End Sub

'=============================================================================
' Initialise checks whether the impagent.ini file is in the correct directory and then opens the form.
'=============================================================================
Sub Initialise ()
   'Get Input Parameters from IMPAGENT.INI (Windows Directory)
   If Not GetImportAgentParams(ENVIRONMENT, A_DataDir, ARRC_DATADIR) Then
            DoCmd Close A_FORM, Form1
            Exit Sub
   End If
   If Not GetImportAgentParams(ENVIRONMENT, DataDir, CRESP_DATADIR) Then
            DoCmd Close A_FORM, Form1
            Exit Sub
   End If

   Set MyDB = DBEngine(0)(0)

   'create temporary table to store the views of selected source node and source database in.
   DoCmd RunSQL "CREATE TABLE CurrentViews([Name] TEXT)"
End Sub
```

```
'=============================================================================
' TransferData does the actual data transfer from "FromTable" to "ToTable". It uses the MappingTable to look
' up the mappings, converts the values from the foreign format to the destination format when necessary and puts
' the data in the "ToTable" by using updates for existing units and inserts for new ones.
'=============================================================================
Sub TransferData (FromTable As String, ToTable As String, ForKey As String, DestKey As String, DestAttr As String,
                  MappingTable As String)
   Dim i As Integer
   Dim value1 As String
   Dim value2 As String
   Dim value3 As String
   Dim ToSet As Recordset
   Dim MapSet As Recordset

   Set ToSet = MyDB.OpenRecordset(ToTable)          'open ToTable
   Set MapSet = MyDB.OpenRecordset(MappingTable)    'open MappingTable

   MapSet.Index = "PrimaryKey"
   value1 = ""
   value2 = ""
   value3 = ""

   'For each attribute of ToSet find correct value
   For i = 0 To ToSet.Fields.count - 1
         'Find correct conversion for attribute i in mapping table.
         MapSet.Seek "=", ToSet.Fields(i).name
         If MapSet.NoMatch Then
            'check whether attribute is preceded by its table name in case there are different mappings for this
            'attribute
            MapSet.Seek "=", ToSet.name & "." & ToSet.Fields(i).name
         End If
         If Not MapSet.NoMatch Then
            'list of destination attributes for insert query
            value1 = value1 & ", " & RemoveTablename(MapSet.DestAttr)
            'list of foreign attributes and conversions for the insert query
            value2 = value2 & ", " & MapSet.Function
            'list of attributes and their conversion for the update query
            If Not (MapSet.DestAttr = DestAttr Or MapSet.DestAttr = DestKey) Then
                  'attribute has nothing to do with the key in the destination table
                  value3 = value3 & ", " & MapSet.DestAttr & " = " & MapSet.Function
            End If
         End If
   Next
   If InStr(value1, DestKey) = 0 Then
         value1 = DestKey & value1
         If InStr(DestAttr, DestKey) = 0 Then
            value2 = "GenerateKey(" & ForKey & ", """" & DestKey & """, """ & DestAttr & “””, “””” & FromTable & """", """" &
                                    ToTable & """)" & value2
         Else
            value2 = ForKey & value2
         End If
         If InStr(DestAttr, DestKey) = 0 Then
            value3 = DestAttr & " = " & ForKey & value3
         Else
            value3 = Right(value3, Len(value3) - 2)
         End If
   Else
         value1 = Right(value1, Len(value1) - 2)
         value2 = Right(value2, Len(value2) - 2)
         value3 = Right(value3, Len(value3) - 2)
   End If
   'transfer data from foreign table to cresp table using SQL
   DoCmd RunSQL UpdateDestTable(ToTable, FromTable, ForKey, DestAttr, value3)
   DoCmd RunSQL InsertIntoDestTable(ToTable, FromTable, ForKey, DestAttr, value1, value2)

   ToSet.Close
   MapSet.Close
End Sub
```

```
'=============================================================================
' TransferFromTextFilesToDBaseIV gets data from text files, transforms data according to the specifications
' and transfers data to dBaseIV tables.
'=============================================================================
Sub TransferFromTextFilesToDBaseIV ()
   Dim i As Integer
   Dim ViewSet As Recordset
   Dim FileName, ForDB, DestDB, View, TextName As String

   ForDB = Forms![Form1]![source db]
   View = Forms![Form1]![view]
   DestDB = Forms![Form1]![target db]

   Set ViewSet = MyDB.OpenRecordset("ViewMap", DB_OPEN_SNAPSHOT)

   'transfer data from text files via foreign tables to cresp tables for chosen view.
   ViewSet.MoveFirst
   Do Until ViewSet.EOF
         If View = ViewSet.viewname Then
            'import text files into foreign tables, first derive filename from foreign tablename
            FileName = Right(ViewSet.ForeignTable, Len(ViewSet.ForeignTable) - (Len(View) + 1))
            Select Case Forms![Form1]![Field5].value
                  Case 1
                     TextName = PATH + UCase$(ForDB) + "\" + FileName + ".FUL"
                  Case Else
                     'only necessary when IncrementalToDBase is implemented
                     'TextName = PATH + UCase$(ForDB) + "\" + FileName + ".INC"
                     Call MessageBox("IncrementalToDBase is not implemented yet")
                     Exit Sub
            End Select
            Call ImportText(CStr(ViewSet.ForeignTable), TextName)

            'remove records where key is missing
            DoCmd RunSQL DeleteEmptyRows(CStr(ViewSet.ForeignTable), CStr(ViewSet.ForeignKey))

            'attach dBaseIV tables
            Call AttachdBASEIVTable(CStr(ViewSet.DestTable))

            'transfer data from foreign tables to destination tables
            Call TransferData(CStr(ViewSet.ForeignTable), CStr(ViewSet.DestTable), CStr(ViewSet.ForeignKey),
                              CStr(ViewSet.DestKey), CStr(ViewSet.DestAttr), View & "_MappingTable")
         End If
         ViewSet.MoveNext
   Loop
   ViewSet.Close
End Sub


'=============================================================================
' Form: Form1
' This file contains all event procedures for Form1. Unlike the rest of the code these routines are not stored in a
' module, but are hidden behind the form. One can find them by looking at the property sheets of the elements
' on the form in design view. These routines execute certain actions when for example the form is opened or a
' button is pushed.
'=============================================================================
Option Compare Database         'Use database order for string comparisons

'=============================================================================
' exit_Click closes the database and the form when one clicks on the Exit button.
'=============================================================================
Sub exit_Click ()
   MyDB.Close
   DoCmd Close A_FORM, "Form1"
End Sub
```

```
'=============================================================================
' Form_Close deletes the temporary table CurrentViews in which during the application the views on the chosen
' source node and source database are stored at the closing of the form.
'=============================================================================
Sub Form_Close ()
   DoCmd DeleteObject A_TABLE, "CurrentViews"
End Sub

'=============================================================================
' Form_Open calls some initialising functions for the application.
'=============================================================================
Sub Form_Open (Cancel As Integer)
   Call SetWindowText(GetActiveWindow(), SuperTitle)
   DoCmd SetWarnings False
   Call Initialise
End Sub

'=============================================================================
' once_Click triggers the actual import. It checks whether there are no compilation errors and then calls the
' routine to import the data of the chosen view into the chosen database.
'=============================================================================
Sub once_Click ()
   On Error GoTo Err_once_Click
   ‘no errors found
   Call ImportOnce
Exit_once_Click:
   Exit Sub
Err_once_Click:
   MsgBox Error$
   Resume Exit_once_Click
End Sub

'=============================================================================
' source_db_AfterUpdate computes the views that are defined on the chosen source node and source database, so
' that the user can only choose a view defined on this particular source node and database.
'=============================================================================
Sub source_db_AfterUpdate ()
   Call SelectViews
End Sub

'=============================================================================
' source_db_Change empties the temporary table in which the views are stored from which the user can choose
‘ and does the computation from source_db_AfterUpdate again after the user has changed his mind.
'=============================================================================
Sub source_db_Change ()
   DoCmd RunSQL DeleteRecords("CurrentViews")
   Call SelectViews
End Sub

'=============================================================================
' target_db_AfterUpdate tells the user what was the last time this particular view was imported after the database
' into which one wants to import the data is chosen and all other options are filled in as well.
'=============================================================================
Sub target_db_AfterUpdate ()
   Call RefreshLastRetrieved
End Sub

'=============================================================================
' view_Enter requeries the database to show the user the views he can choose from.
'=============================================================================
Sub view_Enter ()
   DoCmd Requery view.name
End Sub
```

Appendix B The Dictionary

As an example the complete dictionary will be specified for the GOB1 view. This dictionary consists of the following parts:

- The Interface Dictionary - specifies the structure of the exported foreign text files;
- The CRESP Data Dictionary - specifies the destination tables into which the interface has to import the data;
- The Mapping Dictionary - specifies the mappings between the foreign and destination attributes.

B.1 The Interface Dictionary

GOB1 uses foreign text files with the following structure:

UNITS			
Field	**Null**	**Format**	**Size**
update-time		char(12)	Fix
source		char(8)	Var
unit-id	No	char(15)	Var
orbat-type		char(3)	Var
name		char(55)	Var
category		char(10)	Var
en-friend		char(3)	Var
country		char(2)	Fix
organisation		char(8)	Var
arms		char(15)	Var
command-level		char(5)	Var
symbol		char(6)	Var
comments			

Table 15 GOB1: Structure of file Units.ful

UNIT-EQP			
Field	**Null**	**Format**	**Size**
update-time		char(12)	Fix
source		char(8)	Var
unit-id	No	char(15)	Var
equip-type-categ		char(20)	Var
equip-type-name		char(40)	Var
quantity		num(7)	Var
verific-code		char(4)	Var
effective-time		char(12)	Fix

Table 16 GOB1: Structure of file Unit-Eqp.ful

UNIT-STS			
Field	**Null**	**Format**	**Size**
update-time		char(12)	Fix
source		char(8)	Var
unit-id	No	char(15)	Var
activity		char(8)	Var
ce		num(2)	Fix
pers-strength		num(7)	Var
location-name		char(30)	Var
location-lat		char(7)	Fix
location-lon		char(8)	Fix
effective-time		char(12)	Fix
verific-code		char(4)	Var
comments		char(255)	Var

Table 17 GOB1: Structure of file Unit-Sts.ful

CMD-REL			
Field	**Null**	**Format**	**Size**
update-time		char(12)	Fix
source		char(8)	Var
sup-unit-id		char(15)	Var
sub-unit-id	No	char(15)	Var
cmd-rel		char(6)	Var
verific-code		char(4)	Var
effective-time		char(12)	Fix

Table 18 GOB1: Structure of file Cmd-Rel.ful

B.2 The CRESP Data Dictionary

The CRESP Data Dictionary specifies the structure of the destination tables (the dBaseIV tables of CRESP). The required tables for the GOB1 view are:

UNITS	
Field	**Format**
locekey	char(23)
object_id	char(29)
obtype	char(3)
name	char(54)
parent_id	char(54)
sup_id	char(23)
categ	char(10)
react_time	double
en_friend	char(3)
color_code	char(1)
cc	char(2)
org	char(8)
ass_status	char(1)
unit_no	char(10)
arms	char(15)
com_level	char(5)
display	double
ver_code	char(4)
com_status	char(5)
symbol	char(6)
ace_uic	char(9)

Table 20 GOB1: Structure of Units table

CMDREL	
Field	**Format**
createtime	char(12)
sub_id	char(23)
sup_id	char(23)
cmd_rel	char(6)
updatetim	char(10)
updater	char(12)
asat	char(10)
source	char(8)
ver_code	char(4)
rep_txt	char(254)

Table 21 GOB1: Structure of CmdRel table

GOBREP	
Field	**Format**
locekey	char(23)
createtime	char(12)
source	char(8)
updatetim	char(10)
asat	char(10)
updater	char(12)
allegiance	char(2)
commander	char(30)
infosource	char(12)
activity	char(8)
dir	char(2)
pce	double
comb_eff	char(5)
unit_count	double
pers_str	double
report_id	char(10)
location	char(30)
lat_deg	double
lat_min	double
lat_sec	double
lat_dir	char(1)
lon_deg	double
lon_min	double
lon_sec	double
lon_dir	char(1)
utm	char(15)
xcoord	double
ycoord	double
curflag	double
ver_code	char(4)
i_ver_code	char(4)
rep_txt	char(255)

Table 22 GOB1: Structure of GobRep table

MY_EQ	
Field	**Format**
locekey	char(23)
source	char(8)
createtime	char(12)
mat_seqno	char(2)
aa_seqno	char(2)
class	char(3)
confidence	char(4)
eq_type	char(20)
model	char(40)
ser_num	char(20)
sub_ord	char(4)
asat	char(10)
role	char(10)
role2	char(10)
supply_uom	char(3)
allegiance	char(2)
object_id	char(27)
updatetim	char(10)
cc	char(2)
qnty_eval	char(4)
no_curr	double
no_depart	double
no_added	double
no_lost	double
no_auth	double
status	char(10)
comments	char(254)
usage	char(1)

Table 19 GOB1: Structure of My_Eq table

B.3 The Mapping Dictionary

The Mapping Dictionary for the GOB1 view consists of the following tables:

- The Translation table - specifies the attribute translations for all views;
- The ViewMap table - specifies the connection of all tables and all views;
- The SourceMap table - specifies the connection of all foreign databases and all views;
- The GOB1_MappingTable - specifies the correspondences of foreign and destination attributes of the GOB1 view.

So the first three tables are global tables for all integrated views, while the last table is specifically for the GOB1 view. There is also a difference with the former mentioned dictionaries (B.1 and B.2) in the sense that not the structure of the tables but the values in the tables define the Mapping Dictionary. For the GOB1 view the tables contain the following values:

FIELDNAME	FROMVALUE	TOVALUE
UNITS.ARMS	ARMRD	ARMR
UNITS.ARMS	AVN	AAVN
UNITS.ARMS	HOSP	SURG
UNITS.ARMS	MIL/CA	CIMIC
UNITS.ARMS	ORD	OD
UNITS.ARMS	PC	POST
UNITS.ARMS	PSYCH	PSYOP
UNITS.ARMS	SUP	SUPLY
UNITS.COMMAND_LEVEL	AG	GROUP
UNITS.COMMAND_LEVEL	PL	PLT
UNITS.COMMAND_LEVEL	REGT	RGT
UNITS.EN_FRIEND	AS_FR	AF
UNITS.EN_FRIEND	AS_HO	AH
UNITS.EN_FRIEND	AS_IN	AI
UNITS.EN_FRIEND	AS_NE	AN
UNITS.EN_FRIEND	FR	F
UNITS.EN_FRIEND	HOST	H
UNITS.EN_FRIEND	INVOLV	I
UNITS.EN_FRIEND	NEUT	N
UNITS.EN_FRIEND	NOTKN	NK
UNITS.ORBAT_TYPE	G	GOB

Table 23 The Translation table

ViewName	ForeignTable	ForeignKey	DestTable	DestAttr	DestKey
GOB1	GOB1_Units	Concatenate(SourceDB(), GOB1_Units.unit_id)	UNITS	UNITS.locekey	locekey
GOB1	GOB1_Unit-Eqp	Concatenate(SourceDB(), (GOB1_Unit-Eqp).unit_id)	MY_EQ	MY_EQ.locekey	locekey
GOB1	GOB1_Cmd-Rel	Concatenate(SourceDB(), (GOB1_Cmd-Rel).sub_unit_id)	CMDREL	CMDREL.sub_id	sub_id
GOB1	GOB1_Unit-Sts	Concatenate(SourceDB(), (GOB1_Unit-Sts).unit_id)	GOBREP	GOBREP.locekey	locekey
....	....	....	....	....	....

Table 24 The ViewMap table

name	sourcenod	sourcedatabase
GOB1	ARRC	THISTLE
....	....	....

Table 25 The SourceMap table

DestAttr	Function
asat	Copy(effective_time)
categ	Copy(category)
CmdRel.cmd_rel	Copy((GOB1_Cmd-Rel).cmd_rel)
CmdRel.source	SourceDB()
CmdRel.sup_id	Concatenate(SourceDB(), (GOB1_Cmd-
CmdRel.ver_code	Copy((GOB1_Cmd-Rel).verific_code)
com_level	Translate("Units.command_level", command_level)
comb_eff	Copy(ce)
confidence	Copy(verific_code)
createtime	PatchCreateTime()
curflag	Copy(1)
eq_type	Copy(Left(equip_type_categ,20))
GobRep.activity	Copy((GOB1_Unit-Sts).activity)
GobRep.rep_txt	Copy((GOB1_Unit-Sts).comments)
GobRep.source	SourceDB()
GobRep.ver_code	Copy((GOB1_Unit-Sts).verific_code)
lat_deg	Part(location_lat, 1, 2)
lat_dir	Part(location_lat, 7, 1)
lat_min	Part(location_lat, 3, 2)
lat_sec	Part(location_lat, 5, 2)
location	Copy(location_name)
locekey	Concatenate(SourceDB(), unit_id)
lon_deg	Part(location_lon, 1, 3)
lon_dir	Part(location_lon, 8, 1)
lon_min	Part(location_lon, 4, 2)
lon_sec	Part(location_lon, 6, 2)
model	Copy(Left(equip_type_name, 40))
My_Eq.rep_txt	Copy((GOB1_Unit-Eqp).comments)
My_Eq.source	SourceDB()
My_Eq.ver_code	Copy((GOB1_Unit-Eqp).verific_code)
no_curr	Copy(quantity)
object_id	Copy(unit_id)
obtype	Translate("Units.orbat_type", orbat_type)
org	Copy(organization)
pers_str	Copy(pers_strength)
qnty_eval	Copy("HOLD")
react_time	Copy(0)
sub_id	Concatenate(SourceDB(), sub_unit_id)
Units.arms	Translate("Units.arms", GOB1_Units.arms)
Units.cc	Copy(GOB1_Units.country)
Units.en_friend	Translate("Units.en_friend", GOB1_Units.en_friend)
Units.name	Copy(GOB1_Units.name)
Units.symbol	Copy(GOB1_Units.symbol)
updatetim	Copy(update_time)

Table 26 The GOB1_MappingTable