Management of large communication networks: A case study

Ivan Vrana¹ and Jan Vrána²
¹Katedra informačního inženýrství, PEF ČZU Praha, vrana@pef.czu.cz
²Komix s.r.o. Praha, vrana@komix.cz

Abstract: In the previous paper [6] authors outlined three basic implementation methods and discussed aspects of how to compare suitability of different approaches for building and operating a system, which supports a telecommunication network configuration management. Based on those evaluation criteria this case study will assess and compare main features and effectiveness of individual alternative implementation methods. It is explained, why the implementation method of the Dynamic relational data storing has proven the most efficient out of the three alternatives. This method was successfully implemented in Eurotel (currently Telefonica O2) network management and this implementation enabled an immediate view on all managed network settings, a dramatic speedup of most activities, reduction of errors, better exploitation of capacity of individual network elements and significant reduction of hardware costs. The method could be generally utilized for solution of a broad category of real problems.

Keywords: Communication network management, computer aided support, implementation methods, Dynamic relational data storing, comparison criteria.

1. Introduction

A problem of management of large telecommunication network, which is composed from many communication nodes of various types and of complex internal structure, is described in paper [6]. Individual nodes of the network are interconnected by links of various types and levels like cables connecting individual devices and equipment or mutually corresponding adjustments of parameters of interconnected nodes.

Management of such a network consists of maintaining and adjusting parameters in communication nodes so that the network had desired total throughput, immunity to local dropouts, capability to provide additional services and other required properties. Management should be able to prepare the network for planned excessive load by redirecting some parts of traffic to the less overloaded nodes, reconfiguring the network in case of drop-out of some nodes, incorporating new nodes or, on the contrary, removing unused or defective nodes from the network, etc.

Before any intervention to the network, operating staff should collect, get acquainted and put into mutual context information about a present state and then propose a new setting of parameters in order to achieve required effects. This is a very time consuming, laborious and non-effective work, which is also frequently a source of mistakes since a volume of information about a current state of the network rapidly increases with its size and complexity. A suitable computer support should allow network managers and administrators to quickly and effectively browse and examine individual parts of the network, their settings, their mutual relationships and to make all interventions easier and more effective. Effective
importing and storing of information about the managed network and the consequent processing of this information is a major and the most important problem of any computer supported management of a telecommunication network. In paper [6] were described three different implementation methods to store and manage data in implementation a system for management of large telecommunication networks:

- General object-oriented approach
- Fixed data structures
- Dynamic relational data storing

and their basic characteristics.

2. Evaluation criteria

In [6] also evaluation criteria were presented for comparing suitability of different implementation methods for building and operating a system for the telecommunication network configuration management and also for comparing main features and effectiveness of individual alternatives. Crucial aspects in assessment of alternative implementation methods are effectiveness of the given alternative and its return of investments. All these aspects can be expressed in terms of the following factors:

- costs of development of initial functionality of the application,
- costs of maintenance and extension of (additional) functionality of the application,
- a runtime efficiency of the developed application.

As also shown in [6], for a more objective assessment each of these factors should be decomposed with respect to costs of individual phases of the project life-cycle, additive or multiplicative character of the project complexity and typical functions implemented by individual methods of data storing.

In the next paragraphs we shall compare the above mentioned implementation methods in a case-study of the Eurotel (currently Telefonica O2) network management.

3. Comparison of implementation methods

3.1 General object-oriented approach (GOA)

In this implementation method a general object model is stored in a relational database with the aim to achieve generality and flexibility. This implementation method is based on transformations of a domain model into a general model of objects and their associations, where physical data model with relational data storing is simplified into two basic entities: Object and Association and into two auxiliary entities: Object type and Association type, see Fig. 1.

We shall gradually evaluate this method with respect to previously mentioned evaluation criteria from [6].
3.1.1 Initial functionality development

Design phase
It is necessary in a design phase to design algorithms of the application logic but the main goal is to develop a logical model of the application domain. An application domain such as a considered example of the GSM network leads to the following model dimensions expressed by numbers of:

- Network elements: $1.6 \times 10^5$ (objects),
- Components of network elements: $5.5 \times 10^3$ (objects),
- Rows of parameter tables: $5.5 \times 10^6$ (objects),
- Individual attributes: $1.1 \times 10^8$ (objects),
- Logical relations between rows of parameter tables: $2.8 \times 10^7$ (associations),
- Historical values of attributes: $3.3 \times 10^8$ (objects),
- Relations between attributes and their historical values: $3.3 \times 10^8$ (associations).

Details can be found in [3, 4]).

![Fig. 1 - Structure of relational database for a general object approach](image)

The resulting model will then consist of $8 \times 10^5$ objects and $9 \times 10^8$ associations. Such dimension of the model needs powerful tools to support its building, but unfortunately, it is not easy to find any suitable tool on the market.

Implementation phase
The GOA method and its data management enable that the application logic was designed and consequently implemented by a general and abstract way. In this philosophy, the density of algorithms is higher and therefore the size of the final program code is smaller. But the less convenient aspect of a higher algorithm density is its bigger internal complexity with higher demands at an abstract thinking and qualification of programmers who implement these algorithms. Development of metadata which control general algorithms, metadata maintenance and importing should be also included into implementation of initial functionality. There are tens of
program branches to be designed and developed. In the present time a gross volume of metadata needed for a mobile operator ranges in about $10^9$ database records.

**Debugging and testing phase**

Tested algorithms and their implementation is abstract (i.e. of a high density) so it is necessary to test less program code but testing is more demanding then in a case of a simple code. Debugging of a logical model which controls the GOA application is an important factor. Logical model is a meta-program for abstract algorithms of application logic and it should be debugged analogically as other programs. In contrast to standard programs, this meta-program does not have functional character and therefore, the common development tools cannot be utilized for its debugging. This is a common debugging feature of all applications controlled by metadata.

### 3.1.2 Maintenance and extension

A main characteristic of the GOA based data storing is an easy maintenance and extendibility of these applications. Most of maintenance and extension works deal with an application logical model (meta-program) and only a small fraction of works deal with the application logic itself.

**Design phase**

Design of changes and extensions of an application data structure restricts itself at the application logical model. That work requires abstract thinking but a support can be built in application logic for most changes so that they can be conducted also by end users. This of course reduces costs. As in the case of design of initial functionality, complexity of changes and extensions depends on a volume of added application logic and also on a fact which portion of existing logic will be concerned. When application logic algorithms are of a high density then even extensive changes can usually be accomplished by only a small change of an application logic, which reduces costs. Change has an *additive character* which manifests itself by proportional adding of a new functionality and with minimum changes to existing one. Higher complexity of changes due to higher code density is a negative consequence because even a small change of application logic can influence a large part of its functionality.

**Implementation phase**

Implementation phase practically does not exist, because it only consists either in generating new scripts to enter a modified logical model into a host database, or implementation is accomplished simultaneously with a design. Analogically to implementation of initial functionality the costs depend on the scope of the fully new functionality induced by a change. This leads to minimum or zero costs.

**Debugging and testing phase**

It has same character as in previous implementation phase. Testing again practically does not exist for changes to application data structure while scope of testing is merely proportional to the scope of extensions of functionality.
3.1.3 Runtime efficiency

Computational and memory requirements of the designed application and induced response time are main factors which decide about usability of the new application. Therefore, these factors should not be underestimated. Let us recognize two parts of the runtime efficiency:

- Memory complexity and
- Computational complexity or overall response time.

Both parts of runtime efficiency directly depend on a scope of application domain. Mainly dependence of runtime efficiency on changes of the size of application domain is particularly important, i.e. how runtime efficiency will change with a possible increase of the application domain size.

A memory complexity of the application follows from a scope of the logical application domain model and also from a manner how this model is physically stored. In our case of telecommunication network, the GOA has typically the following memory complexity:

- Number of objects in a logical model is \(4.5 \times 10^4\). Each object is stored in a host database as a row of an Object table, so that this table will contain \(4.5 \times 10^4\) rows.

- Number of associations in a logical model is \(4.8 \times 10^9\). Each association is stored in a host database as a row of an Association table, so that this table will contain \(4.8 \times 10^9\) rows.

Computational complexity of the application depends on algorithms performing individual operations and also on a data volume.

- Selection of one historical value of an attribute,
- Selections of all attribute values of the given object,
- Selection of all objects which are a neighbor to the given object belong to the most important operations.

A common feature of all these individual operations is repeated joining of tables of the above mentioned high size. This causes high computational complexity and a long response time even with an index access. This is substantial namely for mass operations, e.g. selection of all attribute values which were indirectly aggregated into a given object. It corresponds to selection of all rows of the given table of parameters (e.g. all entries of a routing table for a communication network).

3.1.4 Main advantages

Main advantage of the GOA is its generality and universality. Complexity of application development is proportional mainly to scope and complexity of the application domain. Complexity of all phases rapidly decreases in later life-cycle phases. There is also little complexity associated with a migration of the developed application into other application domain.
3.1.5 Main disadvantages

A principal disadvantage of GOA is its poor runtime efficiency. That is why utilization of applications based on GOA is limited only to very small application domains.

3.2 Fixed data structures (FDS)

In this implementation method a fixed structured relational model is stored in a relational database in order to achieve maximum runtime efficiency of the created application. Entities are stored in a host relational database and functionality of the system is described by a set of algorithms to process individual entities and their relations.

The FDS is very specific with respect to the given application domain. This brings higher runtime efficiency at the cost of lower flexibility. Structure of the host database of application is fully subject to the given application domain. It has character of mutually interlinked application specific tables, as schematically shown in Fig 2.

![Fig. 2 - Fixed data structure of application](image)

Transformation between a logical model, structure of the host database and structure of data management is relatively complex here. In contrast from GOA, it is therefore difficult or impossible to develop application logic separately from development of data management. Interface between both these parts of application has a form of a great number of very concrete functions. Due to this arrangement it is possible to individually optimize separate branches of data management and application logic and to achieve maximum performance and effectiveness. Extensive utilized potential of relational technology (see e.g. [1], [2]) enables to solve selected types of operations effectively.

3.2.1 Initial functionality development

Design phase.

In the FDS case, design of algorithms is very tight to a design of data structures. As algorithms consume proportionally most of the whole application in this case, the main development effort will be paid to it. Fortunately, this type of activities is well
supported by theoretical sources [1, 5] and also by many commercial development tools (CASE tools).

Unlike in the GOA case, in the FDS case the model dimension is proportional rather to diversity of the logical domain than to its total range. It means in practice that repeating structures can be described only once and reused. In the case of selected example the model consists of the following numbers of:

- Types of switches: 1 (entities),
- Parts of switches: 310 (entities),
- Logical relationships m:n that lead to entities: 387 (entities),
- Logical relationships (1:n): 1472 (relations).

The resulting model will then consists of 698 objects (entities) and 1472 associations (relations).

**Implementation phase**

If suitable CASE tools were used in a previous design phase, then implementation of initial functionality based on FDS can be considerably automated and thus it can be simpler and faster. Formalized CASE design creates foundation for an automated implementation. If automated implementation is not possible, then manual implementation is tedious and lengthy due to a large number of implemented algorithms. But the complexity coefficient of this activity is low.

**Debugging and testing phase**

As a consequence of low density of a program code of an application in the FDS, scope of debugging and testing activities is proportional to a number of functionality branches. This number grows proportionally with an overall scope of application domain but also with its heterogeneity.

**3.2.2 Maintenance and extension**

Whilst specificity of the FDS was a benefit during a design, implementation and debugging of initial functionality, this specificity turns into the greatest disadvantage for maintenance and extension of the application.

**Design phase**

A suitable utilization of CASE tools during design of initial functionality with the FDS can reduce a coefficient of time consumption. The same CASE tool can also make implementation easier. But benefit coming from CASE tools decreases during successive loops of the application lifecycle because of inconsistencies between application and its logical model described in the CASE. These inconsistencies often arise e.g. due to manual (non-reported) interventions into a development process.

Scope of necessary extra works directly depends on the size of that part of a logical model and its implementation, which should be compared and matched. This component of complexity quickly grows during individual development cycles.
Implementation phase

Complexity of implementation of changes in an application based on the FDS depends on a character of implemented changes, on character of already completed program code and also on used tools. If the implementation was fully automatically generated by a CASE tool and this implementation was not later manually modified, then implementation of new changes by this tool is merely a formal step. But this situation seldom happens as mentioned in a previous paragraph and the newly generated implementation should be manually completed.

Debugging and testing phase

Complexity of debugging and testing depends on all factors mentioned in this paragraph: i.e. on a scope and character of existing program code, on a size of application domain, on a scope and character of a change and on the way implementation was accomplished. If all previous phases of a life-cycle were performed automatically by a CASE, then testing phase is formal, but this situation practically does not occur.

3.2.3 Runtime efficiency

Runtime efficiency is the strongest feature of the FDS method. This method makes it is possible to achieve maximum runtime efficiency in comparison with other implementation methods due to extensive utilization of “natively” stored data in relatively small and properly normalized structures in a relational database which allow to fully utilize optimized capabilities of database engine for mass operations. High runtime efficiency is also achieved due to the possibility to individually optimize each branch of functionality.

The above described illustrative case-study of the telecommunication network has typically the following quantitative parameters for the FDS method:

- Max 5500 database tables holding data from individual areas of network elements.
- Each DB table consists in average of 30 columns.
- Each DB table contains in average 1000 rows of actual values and 40000 rows of historical values, in total 41000 rows.
- A number of other DB objects (indexes, integrity constraints, etc.) correspond to a number of tables and columns.

FDS does not utilize metadata so that its volume expressed by a number of rows in a DB table is 0. A volume of a “payload” data is \(5500 \cdot 41000 = 2.3 \cdot 10^8\) rows in this case.

All described operations in the case of the FDS can be performed by one optimized SQL query, that is, with maximum efficiency. Individual selection, modification and other SQL queries are integral parts of the application logic so that they can be individually treated and optimized by all means that modern relational DB engines offer.
3.2.4 Main advantages

The main advantage of the FDS method is the possibility to achieve maximum runtime efficiency through efficient utilizing of DB engines capabilities and through individual optimization of all operations. This predetermines FDS to building applications with extremely large volume of data in its domain.

Another advantage is a relatively low initial development costs. A gross labor costs are corresponding to the scale of the application domain but existing development tools and experience of development teams can be effectively utilized which can significantly reduce the total costs.

3.2.5 Main disadvantages

The fundamental disadvantage of the FDS is its rigidity with respect to changes of the application domain structure. Although the initial development costs are relatively low, maintenance costs and costs of further development are very high and they tend to significantly grow. This aspect is unfortunately substantial for a further “life” of developed application.

3.3 Dynamic relational data storing (DRD)

The objective of the DRD is to achieve a very high flexibility with respect to changes of an application domain while simultaneously preserve a high runtime efficiency based on efficient utilization of a relational database engine. That was why, in this implementation method a general object model is stored through dynamically constructed structures in a relational database.

The DRD is founded on the following principle: Similarly as in the GOA, the data structure of the application domain is described by a logical model. This logical model is transformed into a system of entities and relations, i.e. into a form similar to a logical model of a data structure of the application domain used by the FDS method. The transformed logical model of a data structure of the application domain is stored in a host database in a form of metadata in a data space separated from “payload” data. This logical model is a key control element of behavior of the application based on this method and serves as a metadescription or a metaprogram (a control element) for most of business-logic algorithms. According to the metadata, a relational structure of specific tables and their relations is then dynamically generated, which in fact corresponds to the structure used by the FDS. Generated specific relational structures are then accessed through dynamically composed SQL queries according to the given request and to the metaprogram. Execution of such a dynamically built query effectively utilizes capabilities of the relational DB engine particularly for mass operations and is very similar to execution of query in the FDS.

In this way, the DRD successfully combines majority of advantageous aspects of both extreme approaches: the GOA and the FDS methods, since the “payload” data is always stored in a fixed “native” structure, where they can be efficiently processed, but at the same time, it is possible to change structure of stored data and behavior of the application automatically, just by changing the metaprogram. These changes could be performed at any time and without any intervention to the program code.
During modeling of the application domain a general logical model of the application domain is created, where objects, attributes and associations are its basic constructs. The transformation of the general logical model of the application domain to its image converts an originally general object system into a relational system, thus into a system described by concepts of entities and relations. This transformation is therefore manageable only for application domains, which data side could also be described in entity and relation terms, which is possible in majority of practical cases. See numerous literature resources dealing with “classical” information systems for concrete methods of data description of application domain.

The principle of the storing data in DRD is illustrated on Fig. 3. The general logical model of the application domain is stored into tables in a host database corresponding to an E-R schema in a left part of the Fig 3. Thus a metadescription is stored there, see e.g. [5]. According to the metadescription stored in the static tables, further tables and association between them are dynamically generated in the host database. They serve for storing real data of the managed system, i.e. rows of entities. Tables in the host database after generating the dynamic data part are symbolically represented in the right part of Fig. 3. This transformation takes advantage of tabular features of some sets of the logical model objects. These tabular sets are further transformed into dynamically generated tables in the host database, which are then accessed in a relational manner. It results in increased efficiency of processing of stored data. Practically arbitrary application domain or its logical model could be generally described in this way. Thus a firm structure and general transformation algorithms remain identical for arbitrary application domain or for arbitrary modification of the given domain. Only a dynamically generated part varies, which is generated from a logical model (metadescription) of the given application domain and which corresponds to all specific requirements of this domain.

![Fig. 3 - DRD static and dynamic part](image-url)
3.3.1 Initial functionality development

Design phase
The design phase of the DRD method is much like in the GOA method. The main portion of the costs is consumed by developing a metamodel and transforming it to metadata. In the DRD however, constructed hierarchical logical model of the application domain is split horizontally into two parts which are then maintained separately and differently:

- Logical part, i.e. units, relations, entities and their attributes is still considered as metadescription and stored as metadata separately.
- Physical part, i.e. values of individual attributes of individual entities and relations between individual values are considered as “payload data” and stored also separately.

This results in significantly reduced volume of metadata necessary for the DRD method in comparison to the GOA method \((10^5 \text{ vs. } 10^9)\). Fig. 4 shows this comparison of metadata/payload data volumes for individual described implementation methods.

Implementation phase
As for the GOA, algorithms are of high density thus requiring abstract thinking combined with the necessity to develop also a metaprogram. On the other hand, a gross volume of the algorithms is relatively low.

Debugging and testing phase
Tested algorithms and their implementation is abstract (i.e. of a high density) as in the case of the GOA, so it is necessary to test less program code but testing is more demanding then in a case of a simple code. Debugging of a logical model is similar to the GOA. This is a common debugging feature of all applications controlled by metadata.
3.3.2  Maintenance and extension

Analogically as for the GOA, the main characteristic of the data storing based at the DRD, is an easy maintenance and extendibility of these applications. Most of maintenance and extension works deal with an application logical model (metaprogram) and only a small fraction of works deals with application logic itself.

Design phase

Analogically as in the GOA, most of the changes can be implemented by only modifying metaprogram, which (with a suitable application support) can be moreover performed by the end user, thus without any further development costs. Nevertheless, the rest of changes deal with high density algorithms, therefore it requires skilful and experienced developers, which is on the other hand compensated by modifying only a low gross volume of algorithms.

Implementation phase

As mentioned above, implementation phase in most cases practically does not exist because it only consists either of generating new scripts to enter a modified logical model into a host database, or implementation is accomplished simultaneously with a design. Analogically to implementation of initial functionality, the costs depend on the scope of the fully new functionality induced by a change. This leads to minimum or zero costs.

Debugging and testing

Again, testing practically does not exist for changes to application data structure, while scope of testing is merely proportional to the scope of extensions of functionality.

3.3.3  Runtime efficiency

In previous two considered aspects, i.e. in costs of implementation of the initial functionality of an application and in costs of implementation of adjustments and extensions of application functionality, the application based on the DRD is very similar or practically identical with a work complexity of applications based on the GOA. But from the point of view of runtime efficiency, in contrast to the GOA, the DRD is very similar or practically identical with a runtime efficiency of applications based on the FDS.

The runtime efficiency is a very strong feature of applications based on the DRD method. The main reason is a relationally normalized storing of managed data in relatively small tables in the host database in a natural way, so that it is possible to fully utilize relational properties of the host database engine for the most critical operations with data. There exists also a possibility to individually and separately tune the efficiency and performance of each individual branch of the business logic, without influencing performance of other branches. A part of optimization could be accomplished (with a minimum work complexity and with a big benefit) by administrative interventions into the host database, e.g. by creation of indices, by splitting the most frequently queried tables, etc. Another possibility is to arrange algorithms for data access in such a way, that they generate SQL queries in an optimized form. But this optimization is a very difficult approach with a relatively low
benefit. Because the dynamically created data part of the host database is practically same in both the DRD and the FDS methods, also quantification of this structure for our illustrative example is same for both methods. Therefore, we can only recall this quantification:

The above described illustrative example of a telecommunication network has the following quantification for the DRD method:

- Max 5500 database tables holding data from individual area of network elements.
- Each DB table consists in average of 30 columns.
- Each DB table contains in average 1000 rows of actual values and 40000 rows of historical values, in total 41000 rows.
- A number of other DB objects (indexes, integrity constraints, etc.) corresponds to a number of tables and columns.

A volume of metadata expressed by a number of rows in a DB table used by the DRD for the given example is $1.7 \cdot 10^7$. In this case a volume of “payload” data is $5.5 \cdot 10^3 \cdot 4.1 \cdot 10^4 = 2.3 \cdot 10^8$ rows.

A realization of all operations for data manipulation is practically same in the DRD as is in the case of the FDS. The only difference lies in the fact, that the respective addressing of the host database engine by a corresponding SQL query (which realized required function) is preceded by a composition of this SQL query. This composition is yielded by an entered request and a consequent processing of a metadata description of the application domain. A dynamical composition of executive SQL queries is a penalty for a comfort of universality, generality and for an easy expandability of applications based on the DRD method with respect to the FDS method. Thanks to the fact, that metadata are stored separately from managed data and that their overall volume is smaller in the DRD with respect to the GOA, also an access to metadata is much faster. Therefore, even a process of a dynamic composition of an executive SQL query does not represent a serious overhead in comparison with a time, which is consumed by the host database engine for an analysis and execution of often very complex SQL query. The process of analysis of metadata and composition of SQL queries could be moreover efficiently supported e.g. by suitable pre-fetching of some data from a database to the memory.

3.3.4 Main advantages

The DRD method combines principal advantages of both extreme methods. Its generality enables to create applications, which are very general and largely independent on a concrete application domain, which further have very low costs of later maintenance and expansion. This predetermines utilization of the DRD method in application domains, which quickly and often change their structure. Once developed applications are capable to be relatively easily seeded in another application domain.

The relational nature of storing and processing of managed data gives outstanding operational features and a runtime efficiency to applications based on the DRD method. It predetermines this method for utilization in application domains, which are very data and structure extensive.
3.3.5 Main disadvantages

The main disadvantage of the DRD method comes from a large abstractness of algorithms based on this method and also from a necessity to develop and maintain metadata. Globally, these disadvantages result in higher demands on qualification and capabilities of a development team.

4. Conclusions

The Dynamic relational data storing - DRD method has proven to be the most efficient out of three alternatives and it was successfully implemented in building a real-life system for management of a GSM network of the Czech mobile phone operator Eurotel (currently Telefonica O2) (currently Telefonica O2). The new system replaced many manually performed agendas of a network planning department. There are following main direct outcomes of the new system:

- Up-to-date view on all managed network settings on either daily or on-demand basis;
- A dramatic speedup of most activities (from formerly many days of intensive paper-work to typically few seconds), which allows instant changes of network configurations according to changes of traffic or e.g. failures of some elements and also very fast introduction of new customer services;
- A dramatic reduction of errors introduced into the process of implementing changes which directly imply a dramatically improved availability of the network;
- Improvement in exploitation of capacity of individual network elements and thus significant reduction of hardware costs.

The example presented in this case-study of the telecommunication network management serves as an illustration of quantitative properties of individual alternatives of large systems management. This example can be generalized to management of an arbitrary network. A great number of real problems of management of large and complex systems can be transformed to the abstract case of network management. The DRD method, which enabled to effectively solve the task of a network management, could then achieve a broad utilization for solution of many real problems.

Acknowledgement

This work was supported by the Ministry of Education, Youth and Sports of the Czech Republic (Grant No. MSM 604070904 – Information and knowledge support of strategic management and Grant No. 2C06004 – Intelligent instruments for assessment of relevance of content of data and knowledge resources).

References

Management of large communication networks: A case study


