Distr.
GENERAL

CES/SEM.47/18
17 January 2002

ENGLISH ONLY

STATISTICAL COMMISSION and
ECONOMIC COMMISSION FOR EUROPE

COMMISSION OF THE
EUROPEAN COMMUNITIES

CONFERENCE OF EUROPEAN STATISTICIANS

EUROSTAT

Joint UNECE/Eurostat Seminar on Integrated Statistical Information Systems and Related Matters (ISIS 2002)
(17-19 April 2002, Geneva, Switzerland)

Topic III: Object-oriented technologies, component architecture

ARCHITECTURAL DIRECTIONS AT STATISTICS NETHERLANDS

Invited paper

Submitted by Statistics Netherlands [1]

I. ORGANIZATION OF THE INSTITUTE

A. From subject orientation to process orientation

1. In recent years, Statistics Netherlands has been restructuring from a subject-oriented organization towards a process-orientated organization. The most important feature of the reorganization and the way we are making statistical figures is the transformation of the traditional stovepipe-based organization into a much more integrated way of making statistics.

2. Generalized process architecture was at the basis of this new organizational model. The three-database model (INPUT / THROUGHPUT / OUPUT) was used as a thinking model for structuring the institute into three statistical divisions, a division of economic statistics (INPUT / THROUGHPUT), a division of social statistics (INPUT / THROUGHPUT), and a division that is responsible for national accounts and all the publications of the institute (OUTPUT). In this setting a start was made with the integration of the stovepipes into single big processes. This is as much of a statistical problem as an IT exercise. In this paper we will focus on the IT aspects.

3. With respect to the three-database model, it has to be emphasized that there are not really three databases. The term database refers to a rather large collection of data, very often in files and directories. In this context, database means that all this data belongs to the same process; a better term should have been the three-process model. It is of course our ambition to have most of our data in real databases or data warehouses. This lies at some point in the future, however.

[1] Prepared by Marton Vucsan (mvcn@cbs.nl).

B. Software development

4. With the change in the organization towards a more general structure, the software development was also directed towards building fewer but bigger applications. After some experimentation it was decided recently that decentralized development within a framework and facilitated by the central methods and technology division is the preferred way. The main reason for this decision is the fact that the statistical divisions have to be able to take full responsibility and thus need to have their own resources. For special projects or just extra capacity there still is a central department with methodologists and developers.

II. SOFTWARE ARCHITECTURE

A. Environment

5. The development environment is an important factor in architectural decisions. After making a choice for the Microsoft environment, it is more or less obvious to follow the strategic directions of this manufacturer. Software development is to be done in the Microsoft environment with a number of restrictions. For normal decentralized development Visual Basic is to be used together with Visual Source Safe. Use of databases is encouraged and so is the use of the OLAP engine. Emphasis is placed on the use of standard tools like Excel, Access, etc. together with VBA. For special projects, like the creation of standard software etc. use of C++ is permitted.

6. All software created should be able to run in the production environment of Statistics Netherlands without major change. This condition is not as strange as it seems. What it means is, that basically it is not allowed for developers (and their project managers) to import nice or handy components from the WEB or other sources. If a project needs a particular piece of commercial or free software, a formal request has to be made to the IT department. If it fits in the infrastructure, the IT department will procure the software and will make sure that maintenance and continuity are well taken care of.

7. Although Microsoft Transaction Server is not implemented at the moment, COM+ is supported throughout the whole network with the exception that is it not allowed to install or run components at file servers or database servers.

B. Our own standard software

8. Aside from the Microsoft software there is a growing body of self-developed standard software (standard to Statistics Netherlands but proprietary in a wider scope) for statistical purposes. This software has a considerable influence on development practices throughout the institute. The most important are listed below:

- StatLine is the dissemination package of Statistics Netherlands. All publications created are to be delivered in StatLine format. To this end there are a number of utilities like StatBuild (component) and XLS2STR. The StatLine product is more an application than a tool; it is a complete set of software to disseminate statistical figures, be it on CD-rom or on the Web. Statline requires a standardised format of the data and as such structures the development of software.

- Blaise is primarily meant to be a tool for doing surveys and partial data cleaning. In its development during the years it has been gradually picking up more functions. In fact Blaise is more and more becoming a statistical toolkit with all sorts of statistical functions. Recently a beginning has been made in supplying part of the Blaise software as a component pack. Important components are the Blaise kernel, the Blaise field selector, the BOI acces-component for accessing data outside Blaise and some components that are able to link graphical images to Blaise data.

- The ARGUS software suite is meant for statistical disclosure control of microdata and aggregated data. The basic engines of this software are available as components albeit OCXs.

- SLICE (for Statistical Location, Imputation, and Correction of Errors) is a component pack designated for automatic plausibility checking of statistical data and for imputation and editing. At the moment it is still using its own class tree, the Slice Record, but in the near future also a version that uses the Cristal interface will be available.

- Cristal is a middleware product tries to unify the interface to statistical objects. The basis of Cristal is a class tree definition containing all the statistical concepts needed. Things like classification, hierarchy, variable etc. are defined in this class tree. Technically speaking Cristal is a predefined area of computer memory with a statistical structure and layout. Typical use of Cristal would be that a developer declares it into his own program (=address space) and uses one of the already existing managers (is data loader) to fill the Cristal structure with data. This data would typically come from a file or a database, but it could also be a datacom source like GESMES. With existing templates and examples, data loaders or managers can be easily written. These managers are written as components that implement a predefined interface, which is part of the Cristal component itself.

C. Architecture

9. Principal guidelines for developing applications at Statistics Netherlands are, that multi-tier architecture is to be applied wherever possible. This logical separation into presentation layer, application or business rule layer and storage layer causes developers to create much more structured programs than before. The concentration of the "working logic" into the application layer, free from presentation related statements, facilitates also the use of components. Of course, for non-interactive applications for high volume processing this will default to plain client-server because of the absence of a presentation layer. However, the same principles of design will apply.

10. To free the components from messy file I/O or database I/O Cristal is used as an interface. Not only will interaction with statistical data be done in a standardized fashion, as more developers are creating so called managers (data load/unload routines) there will also be a large repertoire of I/O components ready for use. For data warehousing and other high volume processing, Cristal is to serve only for use with subsets of the data. To this end, data managers (the load/unload routines) will use a query or other subset command to limit the amount of data. Statistics Netherlands has only begun to combine the various components and Cristal. At the moment it is mostly used as interface in applications. However even there the availability of various components to load/unload data in the Cristal structure pays off. Cristal is still under development and its use is not yet formalized.

11. Developers and designers are encouraged to use components. Aside from a number of pure technical components, like the Stingray grid, the emphasis here lies on the somewhat bigger statistical functions like Bascula, a weighting component or Slice, a set of data editing components, etc. The goal of Statistics Netherlands is to create and import a fair number of components with a high level of functionality. Development will be separated into the design and building of components and combining components into applications. The latter will, to a certain extent, have extra code to glue it all together so it won't be just the combining of components.

III. EXPERIENCES FROM THE FIELD

A. Administration of components

12. Administration has proven to be an elusive subject. There are quite a number of dilemmas's here:

- The central administration leaves something to be desired, but it has been proven to be difficult. A number of complications arise for the people that build the components. Especially for the components that arise out of successful applications it is hard to find parties willing to take full and lasting responsibility (maintenance is the problem here).
- Most of the time you don't want newer versions of components to change the behaviour of old applications at an uncontrolled point in time. With the activation of a new version of a component in an existing application almost always renewed certification and testing is involved. This means that many versions of the same component will coexist. In the Microsoft world this phenomena is known as the DLL hell. The best workaround at the moment is to give new versions a different identification. At the moment an administration policy is being worked out to this effect.
- Central administration and deployment will almost always involve a contract. These contracts will most probably not only cover maintenance but also say something about functionality of a component. Not every project that creates a few handy components will be ready to answer these questions. This is a mechanism that will discourage the central deployment of small and trivial components, which is not undesirable.

13. For the time being, the department that creates the most components anyway, the department of Methods and Technology, is doing some administration of its own to keep track of the components. This leaves many questions unanswered however.

B. Using components

14. We have a long history trying to get developers to share code and we have almost always failed to get them to do so. The most we achieved was collegial copying of source lines. With components, with their runtime interface, it seems to be different. The primary cause seems to be the fact that the components, which are developed these days, are components that cover a specific problem domain. In the old days there were subprograms available for specific functions like SORT, PRINT, SEARCH etc. and the functionality was most of the times related to the problem domain of computer programming. Now most developers would feel challenged to write this kind of code themselves, so code sharing or routine reuse was not much of an issue. With the newer breed of components that cover most of the times a set of complex manipulations with the data, or do weighting etc. there is a need to use these components. For these components it is not trivial for the developers to write the code themselves. Aside from that, the output of most components is certified which will not be the case for home grown code.

15. Another important aspect is the fact that with the use of multi-tier architecture the developers create code much more suited for reuse than while they were building the traditional monolithic applications. Especially the business logic layer is most of the time written as a component. Developers are designing this piece of code purely on a functional basis free of presentation or storage related issues. Reuse of these components in different applications is much easier.

C. The design process of components

16. The design of a component is a difficult process. It will not do, to give a number of developers the task of building components. It is equally not very productive to do an inquiry in the office what components are needed. Experience leads us to the point where we have to conclude that there are two sources of good components:

- Specific applications have parts in their design that are more or less suited for use in other related applications. The identification of the part that is to be used as component is done by the designer and deployment is strictly local. With the architectural requirement to develop multi-tier applications, it has become much easier to derive components from applications.
- Successful existing standard applications have parts in their design that are more or less "universal" and would be very useful as a component. A good example of this is the Blaise application. The identification of the part that is to be transformed into a component is mostly rather informal.
- In the methodological field a theory or new process leads to a well-described piece of software for which the component form is almost natural. In former times these pieces of software existed as subprograms with interfaces for other computer languages as not all calling sequences of the different languages were the same. Nowadays a COM+ component is a very good form for these programs.

17. There is no set of formulated rules or criteria for designing components. The Microsoft framework is used.