STANDARD COMPONENTS FOR COMPUTATIONAL MATHEMATICS SOFTWARE

A. Yu. Boboshko

The paper considers the use of standard software components for automated construction of computation-intensive problem-oriented interactive systems. The properties of standard software components are listed and their interactive development is discussed.

Software support systems now include various tools and features that narrow the gap between their information and computing environments and the operating environments (shell systems) of modern OS [1]. Important advances have been made in these OS toward the creation of a single standard user interface, which is one of the main components of System Application Architecture establishing uniform methods and procedures for working with different computers.

The standard user interface includes multiple-window graphics facilities, multipurpose process interaction procedures, and various distributed utilities for managing data and computations. All these are usually developed using standard software components (SSC). In acquiring the features of shell systems, software support systems remain oriented to the solution of specific classes of problems, as defined by the application domain model. Software support systems are required to provide highly automated tools for the development of the user–computer interaction process, creation of the system part of the software product, on-line generation of programs, and data manipulation. These requirements are particularly relevant for computation-intensive software systems and for intelligent packages designed for data processing in physical experiments, mathematical modeling, and computer experiments [2].

The computational process in computation-intensive software systems is characterized by the following features:
- long computations, with correspondingly prominent reliance on checkpoints;
- an on-line information base for storing intermediate results (data conversion during input/output, storage on external media);
- early identification and elimination of nonpromising alternatives;
- batch and interactive processing modes used in various combinations.

Data export/import facilities are characterized by the following features:
- large volume of data and many different data types;
- economical data storage techniques (distributed RAM, external storage, virtual memory, etc.);
- modification of data, special methods of locating the required fragments, support of data integrity;
- error diagnostics and testing of modified data.

Interactive computation-intensive software systems are expected to meet additional requirements:
- user-friendly techniques for entering data for the problem to be solved ("menus", directives, templates, electronic spreadsheets and forms, functional-level languages with step-by-step specialization of problem conditions, etc.);
- visualization of the computational process;
- dynamic viewing of the results;
- combination of different task-execution techniques: compiling, interpreting, stepwise compiling, macro generation;
- dynamic specification of auxiliary program fragments and their linkage to the software system in run time;
- parameter export/import for linkage of functional modules, including new auxiliary programs.

The large-block programming system DISUPP [3] provides the support tools for the development of software with these properties. This system has been used in the development of interactive software with various process visualization features and

a broad spectrum of control tools for computations that allow multiple alternatives and modification of the computational process. DISUPPP generates so-called routing systems, in which the application domain model is defined by a routing graph — a pseudograph \( G(M, R) \), where \( M \) is the set of nodes with a unique root and a unique terminal node, and \( R \) is the set of arcs, possibly loaded with route-selection predicates reflecting information and control links between the nodes. The route through the graph is planned dynamically. Nodes of the following types are allowed:

1) computation steps associated with functional modules implementing individual complete actions during problem solving;
2) assignment node assigning parameter values, e.g., for implementation of default rules;
3) VVOD\((i)\) nodes, where the user enters selected parameter values for the problem;
4) PRINT\((i)\) vertices, where selected results are output to the user;
5) auxiliary nodes FIKT\((i)\) and STOP for simplifying the representation of the set of routes of a single application domain model. Thus, STOP is always the terminal node, and FIKT functions as a switch for combining several routes into one or for identifying branching sections of a single route;
6) pointer nodes to graphs of previously constructed routing systems, imposing a multilevel structure on the application domain model.

Here \( i \) is a supplementary identifier of the specific system node.

In accordance with these features, we can identify the following stages in the operation of routing systems for problem-oriented interactive software, say in computational mathematics:

- selection of the computation method;
- creation of the information environment of the problem, i.e., input of previously prepared data, diagnostics, modification, passing the data to functional software modules;
- creation of the computation environment of the problem, i.e., linkage of modules in various languages, allocation of external storage to large volumes of input, output, and intermediate data;
- execution of computations and dynamic checking of computation results;
- viewing the computation results;
- analysis of the results and decision on how to continue the computations.

These stages are repeated several times during the problem-solving process, and their sequence is not always fixed. Reusable standard program components have been developed for the implementation of these functions, and they are used in DISUPPP for the introduction of new system nodes in the application domain model of routing systems.

Standard software components (SSC) have been developed for interactive input of vectors and matrices (integer, real with single, double, and quadruple precision), with error diagnostics, dimension checking of the input array, data modification, and storage in system libraries. It is often necessary to supplement the functional system modules with auxiliary programs (e.g., subprograms for computing equation coefficients or matrix and vector elements). This is accomplished by SSC that link auxiliary FORTRAN IV and FORTRAN77 programs according to previously developed templates or from system libraries. The auxiliary programs can be modified and linked to the computation environment of the problem (dynamic compiling and editing).

The operating stability of the software system is improved by introducing SSC capable of dynamically linking modules in different languages with fixed export/import. These modules are linked as subproblems at the routing system nodes. This linking method makes it possible to call a specific module which is needed for computations, while keeping the interface fixed. Combined with SSC linking auxiliary programs, this method provides broader capabilities than the mechanism of tunable modules in ADA.

Work with external memory is handled by SSC that create files of required size and organization in external storage. Dynamic viewing of the results produced by a functional module is made possible by special SSC (so-called "peek nodes") that monitor the creation of print protocols and test files containing computation results in external storage. The execution of a functional module can be terminated when necessary.

The basic components listed above make it possible to construct routing systems in which the composition of the functional modules is fixed in advance, but the specific copies of these modules are assembled in run time from macros as additional computations become necessary for refining the composition of objects and links in the routing system. For example, the elements of a matrix can be entered from the keyboard by the user or input from a file in external storage; they can be modified in run time or computed by an auxiliary program constructed for this purpose. The print protocols created by functional modules can be viewed after the execution of each computation step by connecting a SSC that displays the computation results on the screen.

The SSC developed for the description of computation-intensive routing systems have led to the addition of the following system nodes to the DISUPPP software tools: