

DEVELOPMENT SYSTEM WITH METHODS PROVIDING VISUAL FORM INHERITANCE

COPYRIGHT NOTICE

A portion of the disclosure of this patent document contains material which is subject to copyright protection. The copyright owner has no objection to the facsimile reproduction by anyone of the patent document or the patent disclosure as it appears in the Patent and Trademark Office patent file or records, but otherwise reserves all copyright rights whatsoever.

BACKGROUND OF THE INVENTION

The present invention relates generally to system and methods for creating software programs. More particularly, the present invention relates to a visual development system and methods for improved form-based development of software programs.

Before a digital computer may accomplish a desired task, it must receive an appropriate set of instructions. Executed by the computer's microprocessor, these instructions, collectively referred to as a "computer program," direct the operation of the computer. Expectedly, the computer must understand the instructions which it receives before it may undertake the specified activity.

Owing to their digital nature, computers essentially only understand "machine code," i.e., the low-level, minute instructions for performing specific tasks—the sequence of ones and zeros that are interpreted as specific instructions by the computer's microprocessor. Since machine language or machine code is the only language computers actually understand, all other programming languages represent ways of structuring human language so that humans can get computers to perform specific tasks.

While it is possible for humans to compose meaningful programs in machine code, practically all software development today employs one or more of the available programming languages. The most widely used programming languages are the "high-level" languages, such as C or Pascal. These languages allow data structures and algorithms to be expressed in a style of writing which is easily read and understood by fellow programmers.

A program called a "compiler" translates these instructions into the requisite machine language. In the context of this translation, the program written in the high-level language is called the "source code" or source program. The ultimate output of the compiler is an intermediate module or "object module," which includes instructions for execution by a target processor. In the context of Borland's Turbo Pascal and Object Pascal, the intermediate module is a Pascal "unit" (e.g., .TPU file). Although an object module includes code for instructing the operation of a computer, the object module itself is not usually in a form which may be directly executed by a computer. Instead, it must undergo a "linking" operation before the final executable program is created.

Linking may be thought of as the general process of combining or linking together one or more compiled object modules or units to create an executable program. This task usually falls to a program called a "linker." In typical operation, a linker receives, either from the user or from an integrated compiler, a list of modules desired to be included in the link operation. The linker scans the object modules from the object and library files specified. After resolving interconnecting references as needed, the linker constructs

an executable image by organizing the object code from the modules of the program in a format understood by the operating system program loader. The end result of linking is executable code (typically an .EXE file) which, after testing and quality assurance, is passed to the user with appropriate installation and usage instructions.

"Visual" development environments, such as Borland's Delphi™, Microsoft® Visual Basic, and Powersoft's PowerBuilder™, are rapidly becoming preferred development tools for quickly creating production applications. Such environments are characterized by an integrated development environment (IDE) providing a form painter, a property getter/setter manager ("inspector"), a project manager, a tool palette (with objects which the user can drag and drop on forms), an editor, a compiler, and a linker. In general operation, the user "paints" objects on one or more forms, using the form painter. Attributes and properties of the objects on the forms can be modified using the property manager or inspector. In conjunction with this operation, the user attaches or associates program code with particular objects on screen (e.g., button object); the editor is used to edit program code which has been attached to particular objects.

At various points during this development process, the user "compiles" the project into a program which is executable on a target platform. For Microsoft Visual Basic and Powersoft PowerBuilder, programs are "pseudo-compiled" into p-code ("pseudo" codes) modules. Each p-code module comprises byte codes which, for execution of the program, are interpreted at runtime by a runtime interpreter. Runtime interpreters themselves are usually large programs (e.g., VBRUNxx.DLL for Visual Basic) which must be distributed with the programs in order for them to run. In the instance of Delphi, on the other hand, programs are compiled and linked into true machine code, thus yielding standalone executable programs; no runtime interpreter is needed.

To facilitate software development, it is highly desirable to reuse software components or modules—ones which have been tested and debugged. In form-based, visual development environments in particular, there exists a high degree of functionality which is duplicated from one project to another. Often, however, the core functionality must be modified. Even if substantial modifications are not dictated by system design, one nevertheless still must make substantial modifications in order to adapt the functionality to a new project. Today, programmers typically cut and paste from one project to another. The approach is problematic, however. If one desires to make a core change to the underlying functionality, one is required to go to each individual project to which the code has been copied and manually enter those modifications. There is no mechanism to propagate such a change among projects.

These problems are compounded by the use of forms in projects. Each form in a project typically includes a form initial state, which exists in addition to the code. A difficulty arises in how to propagate a change from the base form to projects having dependent forms. Another problem which arises with the current approach to copying forms among projects is that of versioning. Here, as each form (and its code) is propagated from one project to another, it often undergoes some modification. Since a single base form is not maintained from which dependent forms propagate, a proliferation of the form occurs which leads to increased difficulty in managing the development process.

There has been some effort to address these problems with the use of "form inheritance." Current form inheritance