

**METHOD AND APPARATUS FOR  
ENCODING DATA TO BE SELF-  
DESCRIBING BY STORING TAG RECORDS  
DESCRIBING SAID DATA TERMINATED BY  
A SELF-REFERENTIAL RECORD**

This is a continuation of application Ser. No. 08/233,297 filed Apr. 26, 1994, now abandoned.

**BACKGROUND OF THE INVENTION**

**1. Field of the Invention**

The present invention relates to data processing systems. More specifically, the present invention relates to a method and apparatus for encoding data in such a way that the data describes itself, so that the data may be manipulated in a predetermined manner associated with the indicated data type.

**2. Background Information**

As software development becomes increasing prevalent and, used in circumstances which formerly applied discrete electronics to perform desired functions, needs for testing and verification of the functionality of software become more and more important. One prior art method of testing software includes the use of so-called probes and traces. Probes are a means of distinguishing points in code, and perhaps, generating an event record, although other things can be done. A trace is a time-linear (or, at least, causally ordered) collection of records of some set of the distinguished events in the execution of a program or set of programs. A probe, like a test point in circuitry, is a place at which a developer may evaluate the performance of the running program. Some probes may be selectively enabled or disabled depending upon operating circumstances, so that object code can generate probe information without requiring that a programmer modify the source code for the program. Thus, probes within an executable routine may be selectively enabled or disabled in groups at run-time without modification of the underlying source code. Thus, a programmer may determine faults in a program without knowing anything about the internals of the operational code (although this is certainly helpful).

We may categorize probes into two classes: manual probes; and automatic probes. Manual probes are inserted by a programmer by hand into the source code of a program. They are used by the programmer for detailed debugging and performance analysis. A sub-class of manual probes are known as semantic probes with effects that are documented as part of the program or library interface. Typically, these are used to provide external debugging or performance analysis information. Automatic probes are those which are inserted into existing run-time programs by tools without direct programmer manipulation of the source code. Automatic probes may be inserted by a pre-processor or by operating on processed binaries, and may provide such information as procedure exit and entry points. Automatic probes may be selectively enabled and disabled at run-time to specifically analyze certain performance problems.

One of the problems with prior art probes and tracing is that data is expected by the test engineer only in a certain format. Typically, the data is presented in a raw form, wherein the test engineer must determine what the data returned from the probes represents. Thus, the test engineer evaluating an executable program which generates information from probes must have an intimate understanding of the probe information provided at program run-time. Unfortunately, in many instances, such information is not

available, or may be obscured by the original programmer of the application program under test. Thus, it is desirable that a program generating probes provide information about the probes in a standardized manner so that diagnosis of the programs under test may be most easily accomplished without a detailed understanding of the internal functioning of the program under test. Prior art techniques of inter-process communication in environments such as testing, typically require information extraneous to the data to discern the type of data returned from the program.

**SUMMARY OF THE PRESENT INVENTION**

A computer-implemented method and apparatus in a computer system of processing data generated by a first application program in a second application program during runtime. During runtime, the first application program generates a record including a plurality of fields, wherein at least one of the plurality of fields contains data generated by the first application program. Other of the plurality of fields containing descriptive information regarding the data. In implemented embodiments, this may include, fields for representing checkpoints in the program, such as a relative address at which the checkpoint occurred, and/or a time at which the checkpoint was reached. The record also includes a reference (e.g. a pointer, relative or absolute) to a tag record. The tag record describes the plurality of fields contained in the record (e.g. the names and types of the fields). The tag record further recursively references a plurality of tag records each referencing an associated tag record identifying fields in a referred-to tag record. This continues, recursively, until ultimately, a root record is referenced including a self-referential tag identifying the fields in the root record. The second application program then may receive the record (e.g. during computer system runtime, for example, during execution of a test suite), and references the tag record and each of the plurality of tag records, recursively, until reaching the root record in order to identify the data by referencing the plurality of fields in each of the tag records. In this manner, the data contained in the record is thus self-describing. The second application program then manipulates the data according to the identification of the data specified by the record, the tag record and each of the plurality of tag records, such as by filtering data contained in the record which is not required by the test suite, or by reformatting the data into a form more appropriate for examination by a user or a post-processing program.

**BRIEF DESCRIPTION OF THE DRAWINGS**

The present invention is illustrated by way of example and not limitation in the figures of the accompanying in which like references indicate like elements and in which:

FIG. 1 illustrates a system upon which one embodiment of the present invention may be implemented.

FIG. 2 shows a relationship between an application program under test and a test suite program and the communication between those two executable programs.

FIG. 3 illustrates the processing of various trace information and intermediate information as may be performed in a system implementing the embodiments of the present invention.

FIGS. 4-7 show various data structures which may be created and used in implemented embodiments of the present invention.

**DETAILED DESCRIPTION**

A portion of the disclosure of this patent document contains material which is subject to copyright protection