Definition and Application of Second Generation Logging Systems

The development of complex software systems is becoming more and more expensive. The detailed processes within software systems as well as the system behavior are often difficult to trace and to explain. A professional second generation logging solution is able to provide developers and project managers with comprehensive information about the "inside" of their software.

Observing the daily work of a software developer working on a more complex system closely, the tracing and analysis of software processes represents a constantly recurring and burning problem. This seems to be an often not consciously perceived but accepted bottleneck severely hindering the productivity.

Up to now most software developers confine themselves to the output of the data to simple log files or other storage media. But the developer receives hardly any support in regard to the evaluation of this data. This is often done using normal text editors, command line tools such as "grep" or in scripts created for the particular application.
More professional logging solutions promise a lasting productivity and quality improvement and this in *all* activities and phases related to the software, from the development itself up to the operation of the software systems.

1 Application of a logging solution within the software lifecycle

Typical phases of the software lifecycle
• Development and individual tests (unit tests) • Integration and system tests • Initial operation and acceptance • Operation and maintenance

Already during the **development** the software developer is able to perform faster individual tests by visualizing the processes at runtime. This way software developers obtain a better overview than with debuggers and are able to observe processes in real-time. Occurring errors are recorded in detail and thus can be analyzed better. Even if an error occurs later or in a different context, the information which the developer needs for the analysis is saved, including all context information. Recording the communication with the software (programs, subsystems and components) of other developers, teams or parties considerably facilitates also the **integration**, including the verification in regard to sub-suppliers.

During the **initial operation** a professional logging system can be helpful getting past the initialization, configuration, network and device problems. Here in particular you often have also very persistent but only sporadically occurring errors. Also problems caused by modified general conditions can be detected faster. For the **acceptance**, recordings of the processes are important as they can be used to understand and explain the often very complex system behavior and to prove correct function.

During and after the successful putting into operation of the software, a professional logging system enables the **inspection** of the current system status – also for telephone support and remote maintenance. The *recording* of the processes in their real usage environment and under real conditions enables problem analysis at a later time. In some projects also the customer himself requests that the user input will be recorded – if necessary, including processing and results.

2 Logging systems for the recording and analysis of processes

The recording (or “logging” for short) and analysis of processes varies in power and usefulness according to the logging system used.

2.1 First generation logging systems

Logging systems of the first generation confine themselves to the *output* to log files or other storage media and do not provide sophisticated features for the *analysis* of the recorded information. In some cases this may be an appropriate means, e.g. concerning the evaluation of log files of FTP or web servers, where communication with the outside world is recorded using relatively simple and fixed interfaces and formats.

Within software development the *internal* processes of the often very complex systems are much more important, and there are usually many changing components and interfaces involved, so that methods such as the ones mentioned above would be very inefficient here.

However, it is worth mentioning that recently some interesting approaches concerning the standardization and propagation of logging systems of the first generation exist:

- Java 1.4 (Sun) includes a Logging API and also some simple “log handlers”, i.e. output drivers.
- The .NET-Framework (Microsoft) also includes logging support, though in even more rudimentary form.

These examples show that the importance of logging functionality in software development is increasingly recognized. However, the focus is still on only recording the information.

2.2 Second generation logging systems

Logging systems of the second generation allow to influence the type and extent of the runtime recording without the need to modify the source code, ideally even at runtime without need to restart the inspected application. And the user should not be bothered with issues concerning the implementation of advanced logging capabilities with high performance, filtering, decoupled output, resource management (e.g. deleting old information) and forwarding of information.

Still more important, a second generation logging system must include an interactive visualization and analyzing tool. This makes the typical and often-needed analysis tasks much easier *and* many times faster.

Another must is the availability of powerful filtering features. Among others the following are required:

- It should be possible to define filter expressions visually.
- It should be possible to define arbitrary complex logical filter expressions.
- Filters should be able to build on other filters.
- It should be possible to save the filters.
- The filters should be applied easily, e.g. via drag & drop.
- The application of the filters and the display of the filtered data must be quick.

Within pursuit of an analysis task the user should be able to use his/her existing filters and set up new filters based on the current ones. Typically he/she starts with relatively rough filter settings and hides or shows further information step by step, whereby he/she is able to check each of the results quickly.

It should not only be possible to display and evaluate the information location- and time-displaced (offline), but also live (online). Using the filter options mentioned above it is then possible to *monitor* ongoing processes according to different aspects.

As a whole, logging systems of the second generation are based on the realization that not only the recording of information, but *above all* the quick and flexible display and evaluation is important. This puts to mind the known idea of the “information at your fingertips” as once graphically stated by Bill Gates.

Theoretically the user of a first generation logging system is able to upgrade to the second generation by purchasing or developing a compatible analyzing tool for his system. But usually in practice the first option doesn’t exist and the second isn’t realizable and attractive.

If the general decision concerning the use of a logging system is still forthcoming, the following issues should be addressed:

- Does a system of the first generation meet the requirements?
- Do upgrades for this system providing an analyzing tool exist or can they be expected soon?
- Does the system meet the technical requirements?
  For instance: simple interface, support of several log channels, configurable filtering of the output, forwarding, automatic deletion of old data, thread-safety, performance.

3 Selection and introduction of a logging system

A logging system introduced within a starting or ongoing project must usually pay off within this project, i.e. decrease the total expenses and especially the development period of the project. Regarding a quick introduction, in addition to a simple installation a highly intuitive user interface is required. This does not only enable more efficient daily working with the tool but should also allow for a quick and independent acquainting period of the developers with the tool without need for additional external training.

Closely connected with the question of usability are performance issues. First, it must be mentioned that the use of a logging component must not lead to a considerable decrease in performance or unintended system effects or even to instabilities of the entire system. Second, it should be possible, as mentioned above, to promptly access the information recorded via logging, which is why a powerful tool for the analysis of the quickly growing and often immense amount of information should be part of the logging system.

The introduction of a logging system can be carried out step by step. If available it is recommended to use evaluation versions provided by the manufacturers.

A first test case could be the recording of the input, processing and output at the external interfaces of the user system, for example to help with the verification. Another option to test the practicability of a logging system is to apply it in the search for the cause of a currently urgent software error.

iTech Logging 2

Manufacturer: iTech Software GmbH, Berlin (http://www.itech-software.de)

Platform: Windows NT or higher(NT, 2000, XP), Windows 95 or higher (95, 98, ME)

Logging libraries:

- ITLogLib/.NET: Variant as **.NET a**ssembly (DLL) for **Windows .NET languages like Visual Basic .NET an**d C#.
- ITLogLib/COM: Variant as **COM-DLL for all programming- and script-languages, working with COM objects** (e.g. Visual Basic, VBA, Visual J++, VBScript and JScript)
- ITLogLib/Visual C++: Variant as DLL specially used with Microsoft Visual C++
- ITLogLib/Standard C++: Variant as DLL for standard C++ compilers (Borland, GNU, etc.)
- ITLogLib/Java: Variant for (pure) Java on Windows
- ITLogLib/Delphi: Variant for Delphi
- ITLogLib/Win32: Standard Windows DLL (with procedural interface)

Tools:

- configuration tool ITConfigManager
- analyzing tool ITLogBook

Licensing:

- per developer