CMX - A GENERIC SOLUTION TO EXPOSE MONITORING METRICS IN C AND C++ APPLICATIONS

Felix Ehm, Yves Fischer, Georgia-Maria Gorgogianni, Steen Jensen, Peter Jurcso,
CERN, Geneva, Switzerland

Abstract

CERN's Accelerator Control System is built upon a large number of C, C++ and Java services that are required for daily operation of the accelerator complex. The knowledge of the internal state of these processes is essential for problem diagnostic as well as for constant monitoring for pre-failure recognition. The CMX library follows similar principles as JMX (Java Management Extensions) and provides similar monitoring capabilities for C and C++ applications. It allows registering and exposing runtime information as simple counters, floating point numbers or character data. This can be subsequently used by external diagnostics tools for checking thresholds, sending alerts or trending. CMX uses shared-memory to ensure non-blocking read/update actions, which is an important requirement in real-time processes. This paper introduces the topic of monitoring C/C++ applications and presents CMX as a building block to achieve this goal.

INTRODUCTION

CERN's accelerator control system [1] is essential for operating the accelerator complex; hence its availability, performance and correct functioning are critical. Consequently, a pro-active approach to problem resolution is desirable, where anomalies are detected and corrected even before operation is affected.

In terms of software, the control system is comprised of some 3500 processes written in Java, C and C++. The latter two are in comparison to Java rather "black boxes" with very limited support for identification and diagnostic of problems. Simple process existence checks, and probing their functionality regularly (e.g. *"they do what they supposed to do"*) as well as a manual core dump after a problem is suspected, are the usual ways.

However, blocked threads for example cannot be automatically detected. Another example is a "delayed problem": software is updated during working hours which introduces faulty code. Although working in the beginning it eventually stops in the night, for example, as an internal message queue has filled up. The consequence: Experts have to be called in and the resolution of the situation takes much longer than during the day. Both problem types negatively impact running costs and overall service availability.

To overcome such situations and to improve pro-activeness, the detailed "health" (state) of each process must be known at any time, which in turn requires two mechanisms:

1) Each process exposes internal numeric values and character data (*metrics*) indicative of its state.
2) A centralized system for monitoring, offering the full range of features like history, trending, status displays and notification in case of values breaching pre-configured thresholds.

DIAMON [2] is such a centralized system at CERN and currently monitors computers and the presence of required processes. For Java services it additionally accesses metrics exposed via the *Java Management Extensions* (JMX) [3] and uses this information to further determine the overall health state. For C/C++ programs in contrary, we found no suitable equivalent to JMX or a similar technology which fits the requirements for our (real-time) processes.

Consequently, we set out to implement a light-weight library, providing a sub-set of JMX's extensive functionality, and accordingly named the library CMX. The current state of CMX is presented in this article.

REQUIREMENTS

The following high-level requirements were identified for CMX:

- Exposure of numeric values and character data outside the process context.
- Low latency operations for reading or updating metrics, actions must deterministically finish independently from their result within 10 milliseconds.
- Operable in a disk-less environment.
- Lightweight dependencies and minimal memory footprint, < 200 Kilobytes.
- User-friendly C and C++ API, which allows dynamic registration of metrics.
- Simple integration with existing monitoring systems.
- Portable to Microsoft Windows, Linux and LynxOS [4] operating systems.

Why Can't We Use Existing Solutions?

The search for existing generic solutions showed that this area is quite uncovered. The only promising technology is the *Simple Network Management Protocol* SNMP [5]. It is an IETF protocol for managing devices on IP networks. Devices that typically support SNMP include routers, switches, servers, workstations, printers, modem racks and more. It is being widely used by many hardware vendors to allow remote monitoring of their network devices and supported by the majority of

ISBN 978-3-95450-139-7

monitoring solutions on the market (NAGIOS, ZABBIX, MRTG, etc.).

For investigating SNMP we used the commonly known *net-snmp* [6] package. Metrics are exposed via a build-in Agent and a *Management Information Base* file (MIB) [7] indicates what metrics can be read.

Apart from the fact that it is not directly designed to be used within (complex) software, the result of the evaluation showed that the management of MIB files is not trivial. It requires thorough understanding of SNMP concepts and the implementation of the agent calls is not straight forward, e.g. metrics cannot be added/remove easily at runtime. During tests, the memory footprint exceeded our limits and update roundtrip times were too slow for real-time processes.

CMX ARCHITECTURE

Enabling Inter-process Communication

The requirements in terms of low-latency and availability on low-performance machines lead to the decision to use *Shared Memory* (SHM) technology for inter-process communication.

POSIX.1-2008 [8] defines the UNIX System V (SysV) SHM Interface (part of *X/Open System Interface*) as well as the user space utilities for managing SysV shared memory. SysV SHM is supported on all target platforms (Linux, LynxOS, Microsoft Windows).

SHM segments can be created and attached to the program's memory space using system calls and are identified by system generated (and unique) numerical *Segment Identifiers* (SIDs).

To access the very same segment across several applications the user can define a static user key. However, the usage of keys should be reduced to the minimum to avoid key collisions between unrelated applications. Therefore, CMX uses only one such user key as explained in the following.

Shared Memory Structures in CMX

Within CMX, an internal look-up data structure provides information on all registered processes. It is stored in a SHM segment and called the *Main Registry*.

At initialization time, CMX will try to access the main registry and in case it was not created before (SHM segment with common key '100' does not exist) automatically instantiates it.

Metrics and other information of a process are stored separately as *Component* data structures. For their SHM segment only system generated SIDs are used instead of keys because, as mentioned before, they may not be unique. The SID is saved in the registry as illustrated in Figure 1. In general, an executable can have more than one Component. At program initialization, CMX creates one by default (with empty `component_name`), to store a standard set of process related information like owner, start-up time and operating system, etc. (see Figure 1).

With each entry in the main registry, the Component's layout version is stored to allow backward compatibility in case it changes. An example here is to be able to read SHM segments from processes that use different Component layouts from older (but not newer) versions of CMX. This is of significant operational advantage, as it is not necessary to upgrade all programs by a newer version of CMX on one computer at the same time and thus, supports the idea of "smooth upgrades" [7, 9].

A cleanup procedure executed with each Component creation makes sure old entries (process is non-existent) are properly deleted.

A Component can store two types of metrics: characters (*strings*) and numbers. The allowed amount of each metric has to be set at the creation time of the Component and cannot be modified afterwards. In this range, however, metrics can be added and removed as demanded. The characteristics of the two types are:

- Numbers: 64-Bit floating point.
- Strings: "dynamic" length, predefined size (default 255) at Component creation applies to all strings within.

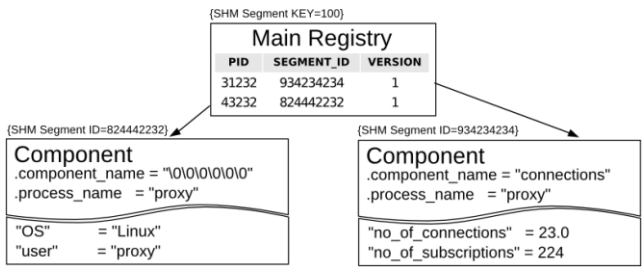


Figure 1: CMX's data structures in System V Shared Memory.

Locking and Data Integrity

While CMX is virtually non-blocking and has a very low execution overhead, it is required to protect internal data structures against race conditions. This is implemented using POSIX semaphores and timed operations (`semtimedop()`). The timeout can be configured by the user via the API and hence, can be optimized to the characteristics of the local environment. CMX uses 100 milliseconds as default semaphore timeout.

There are three mechanisms in CMX to ensure data integrity:

1. Only one process at a time is able to modify the main registry. This is independent from a process updating a metric as Components can be referenced directly (see 2).
2. Each Component is protected against concurrent access (read and write) using locks: only one process can update or read data from a Component at a time (see *Performance Analysis* section for more details on the consequences).
3. CMX ensures that changes to a Component are restricted to the owner process only. This is achieved through the update function which checks if the caller has the same process id as the one stated in the Component field.

ISBN 978-3-95450-139-7

Code Example

Figure 2 shows an example of how a metric is exposed using the CMX API (for better readability the error handling has been removed).

```
// Register process
fb_process_register();
// add a new component
comp_uuid = fb_comp_register("component");
// add a new metric in the component
fb_metric_add(comp_uuid, "counter1");
// set the new metric
fb_metric_set(comp_uuid, "counter1", my_number);
// free cmx resources
fb_process_unregister();
```

Figure 2: Code example for enabling CMX in C.

Operating System Implications

The use of SysV SHM segments and semaphores in CMX is regulated by the operating system limits.

The most interesting setting is SHMSEG, which is the number of maximum allowed shared memory segments per process and defaults to 4096 on Linux. This is sufficient for our environment, but should be known as a limitation if more Components have to be created. It is still possible to raise it by changing the system configuration.

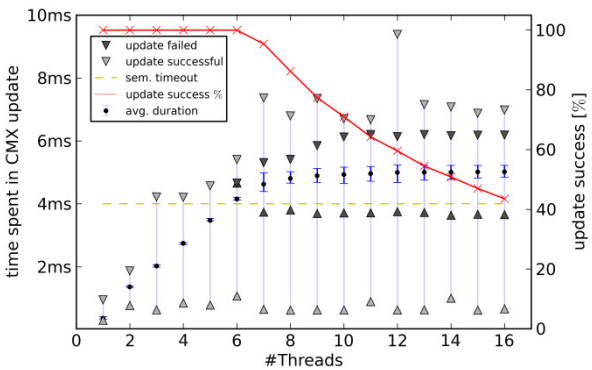


Figure 3: Latency effects on updating metrics at high frequency.

PERFORMANCE ANALYSIS

Figure 3 shows the latency effect of concurrent access on one component for a raising number of threads. For this test*, 100 numerical metrics are registered in one Component and each thread tries to update each of them with an arbitrary value as fast as possible. One update can either end successfully or return with a lock timeout error.

In this scenario the locking timeout is set to the very low value of 4 milliseconds (dashed line in Figure 3). When running, each thread tries to acquire the shared lock of the Component and measures the time after the lock was acquired, to update all 100 numbers and to release the lock. Each failed lock is recorded. The percentage of successful updates per test is shown with the solid line in Figure 3. Up to six threads, the latency raises, but all locks are acquired in-time. With more threads, the update success rate decreases as the line indicates.

A call resulting in locking error takes 4 to 6 milliseconds before returning with an error code. The additional 1-2 milliseconds overhead is considered as a consequence of the operating system thread-scheduler.

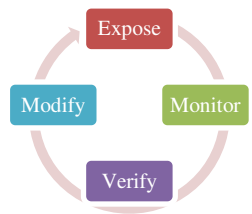
The tests have shown that CMX provides very low latency for updating metrics and shows deterministic and real-time behaviour and therefore satisfies the corresponding requirements.

USAGE AND NEW POSSIBILITIES

Meaningful Information

The fact that CMX enables exposure of process metrics, has little value unless the data provides useful information e.g. indications that problems are building up.

Developers possess detailed knowledge about the processes they support. However, as C/C++ process metrics is an as of yet relatively unexplored domain for them, they will need to learn in an iterative process what constitutes meaningful metrics for the particular process they're responsible for.



Simplified Root Cause Identification

As CMX enables establishing a more detailed state of individual C/C++ processes, it becomes easier to identify the root cause of problems in cases where multiple services are involved – which in turn will help decrease downtime.

Exposure of Runtime and Build Information

CMX is able to provide runtime (see Figure 4) and compile time information of the executable. The latter is possible via its built-in *Manifest* file, which is automatically inserted into the final binary during build time and which includes information like code version, compiler version, platform and more. The very same is also available for all statically linked libraries.

This new feature enables dependency tracking and enriches the diagnostic tools for developers.

```
User              abcdefgh
Processname       cmxctl
PID               32772
Hostname          computer.cern.ch
OS                Linux
OS Kernel         2.6.32-358.14.1.el6.x86_64
Version Level     #1 SMP Wed Jul 17 08:30:19 CEST 2013
Hardware type     x86_64
```

Figure 4: CMX command line tool showing information on the program.

* Test machine used: HP G7 with 2 x Intel X5660 (12 physical cores), running Scientific Linux CERN 6.4 with Linux 2.6.32-358

ISBN 978-3-95450-139-7

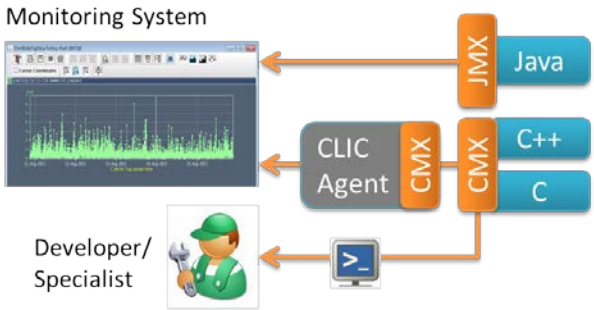


Figure 5: Integration of CMX into the DIAMON monitoring system.

Integration with Monitoring Systems

With programs exposing internal information via CMX library it is now possible to utilize this data further on. The CLIC monitoring agent is part of the DIAMON service and can obtain metric information via its CMX plugin and subsequently provide them on its network interface to external services. Due to its very low memory footprint of less than 2 Megabytes it also runs on hardware with small resources and the limited dependencies allows porting it to various platforms. In fact, it is installed on all real-time computers, high end servers, operational consoles and around 500 virtual machines. In total more than 2000 machines are monitored through the CLIC agent.

From this point on service managers of C/C++services can profit from the full range of features, like value history trending, status displays and notification in case of values breaching pre-configured thresholds.

Planning and Preventive Maintenance

By following the history of metric values over time it is possible to make statements about their future development. Software and hardware can be better tuned or earlier upgraded to changed requirements, e.g. more clients start to demand data from a server.

Discover Service Dependencies

With CMX's support for exposing strings, it is possible to capture patterns of how processes connect to each other by exposing host/process names of clients. This can, for example, being used to inform if the service will be modified or taken down for maintenance.

FUTURE

For our domain we have identified the following plans for the future:

- Collect **more experience** in production environment.
- **Integrate** CMX into the majority of CERN's control system components.
- Development of a alternative **read-out tool** to the CLIC agent, e.g. based on HTTP.
- **Make the project public**: e.g. open source project.
- Elaborate usage for **other domains**, and continue to **improve CMX** in respect to new requirements.

CONCLUSIONS

With the new CMX library, a software developer has a simple and intuitive API which offers a time-saving way to expose internal information on (real-time) C/C++ processes. For the first time, it is possible to inspect these programs – without using debugging tools – during their execution.

CMX is fully integrated into DIAMON, and thus, allows inspecting information remotely in the same way as it is now for Java processes using one central interface.

Pre-failure recognition and detailed diagnostics, which are essential for running complex infrastructures, are now possible and the first experiences within CERN's accelerator controls group show that it enhances the monitoring and diagnostic capabilities of C/C++ programs.

The challenge of providing minimal latency operations with guaranteed data integrity was an important aspect in the design. Hence, its careful implementation as well as thorough testing resulted in a major part of the development. Although the name is derived from JMX, CMX provides only a subset of the functionality in JMX. A built-in agent acting to incoming network requests, for example, was not in scope of the CMX as this implied adding complexity for proper handling and ensuring security aspects.

A main factor in this chain, however, is the quality of the metrics which has to be ensured by the developer of the application.

REFERENCES

[1] CERN's Accelerator Control Group, http://cern.ch/be-dep-co.

[2] W. Buczak, M. Buttner, F. Ehm, P. Jurcso and M. Mitev, "DIAMON - Improved Monitoring of CERN's Accelerator Controls Infrastructure," 2013.

[3] Oracle, "JMX Interface Technology," http://www.oracle.com/

[4] LynxWorks, "LynxOS, Real-Time Operating System," http://www.lynuxworks.com/

[5] IETF, "SNMP Protocol," http://www.ietf.org/rfc/rfc1157.txt

[6] "Net-SNMP," http://net-snmp.sourceforge.net/

[7] IETF, "Management Information Base, RFC-1156," http://www.ietf.org/rfc/rfc1156.txt, 1990.

[8] Posix.1 2008, IEEE Std 1003.1™, 2013.

[9] V.Baggiolini, D.Csikos, P.Tarasenko, Z.Zaharieva, M.Arruat and R.Gorbonosov, *Backward Compatibility As A Key Measure For Smooth Upgrades To The LHC Control System,* CERN, 2011.

ISBN 978-3-95450-139-7