**INTRODUCTION**

The DRAMA API [1] maintains the AAO's primary test environment for distributed instrumentation software and the shared guide used by various other environments. It is a test bed, with each main task running independently to test many different types of different types of tests in a DRAMA system. Tests are run on different hosts in a heterogeneous environment. DRAMA was implemented in 1994, when it was designed to be highly portable at a time when POSIX was available on all machines of interest. It has been run on many flavors of UNIX, VMS, VAX and MS Windows, and provided the ability to write soft real-time applications and with good performance on, for example, SPARC VAX CPSUs. The flexibility offered systems as simple as the AAO's QF system [2] to implement – see figure 1, noting that one of the most appropriate hardware for such systems is a distributed system.

Most work is a DRAMA task is done in response to "Obey" messages – in effect, event-driven messages, implementing "Actions." The design approach implements co-operative multi-threading, where each application can be running on the same host but most daemons require a different language (IDRAK) thread in order to allow all actions to run and for the action itself to be "Kicked" – a message to change behavior in some way (typically not to behave) to cancel the action cleanly.

The approach has worked well and a strongly object-oriented task design approach was implemented for tasks written in C++.

Attemps were made running about 1994 to implement C++ interfaces for DRAMA, but the results were inferior and various difficulties were present. One of the early issues was the poor portability of C++ features, some features such as templates and exceptions were not reliably implemented and were not portable. Another was that we were still learning the best approaches to use.

While DRAMA tasks using threads of various types have been implemented over the years, DRAMA itself has not supported using threads, with few in co-mo multi-executable applications. In such cases, C++ API tasks would be used around DRAMA when using threads, but is in current use of this capability for component testing kept somewhat limited. This is partly due to the absence of supported languages and need for arbitrary software in many cases. No one has been looking at doing the right way to thread support for DRAMA over the years, but the work has not exploited.

C++ [3] was a major example to the C-Language threads are near support using the right approach. The use of C++ templates and exceptions were not reliably implementable and were not portable. Another was the way that we were still learning the best approach to implement.

Attempts were made running in 1994 to implement C++ interfaces for DRAMA, but the results were inferior and various difficulties were present. One of the early issues was the poor portability of C++ features, some features such as templates and exceptions were not reliably implemented and were not portable. Another was that we were still learning the best approaches to use.

The DRAMA way to return control to an action must be implemented, with no use of code, when the DRAMA thread is killed. Whilst it can be a problem to understand the life of an action, it can be done it is hidden by DRAMA2. The API allows reliable tasks to be written quickly. It allows sequenced code to be written for sequenced jobs, but with the efficient non-blocking DRAMA messaging facilities available. Much of the potentially useless, complete of existing sequenced distributed applications is hidden from programmers in DRAMA2, the most common cases.

### Basic Hello World example

In example 1 shows the "HELLO" action is running in the main thread while it is running on "Kickback". In the traditional DRAMA system, when the "Hello" thread is called, the DRAMA message thread, the main thread颐 is called and starts running actions in threads.

The second part of the figure is an alternate sub-Class of "MessageHandler". This class is a handler of messages (what is called an Action Handler). The class must provide the method "MessageHandler::Handle()", which is called when a message is received while a thread is running. The thread is not terminated, and the action is marked as completed. Immediately, another thread can be run on the same thread, even if it is hidden by DRAMA. A message is taken as the target of the thread, but is not processed any further.

Example 1 shows on the right, how the "Hello" action is used with the "MessageHandler". The "Hello" action is a simple action, but can send a simple string message to any thread.

Example 2 shows a simple implementation of a Hello action. Action functions can store state in a structure, which can be used to share state between DRAMA, but the implementation in this case is very clear:

```c++
#include <iostream>
#include <cstdlib>

// Hello World example

int main() {
    std::cout << "Hello World!" << std::endl;
    return 0;
}
```

### Basic Hello World example

**Threaded "Hello World" example**

In example 1 shows the "HELLOWORLD" action is running in the main thread while it is running on "Kickback". In the traditional DRAMA system, when the "HelloWorld" thread is called, the DRAMA message thread, the main thread颐 is called and starts running actions in threads.

The second part of the figure is an alternate sub-Class of "MessageHandler". This class is a handler of messages (what is called an Action Handler). The class must provide the method "MessageHandler::Handle()", which is called when a message is received while a thread is running. The thread is not terminated, and the action is marked as completed. Immediately, another thread can be run on the same thread, even if it is hidden by DRAMA. A message is taken as the target of the thread, but is not processed any further.

Example 1 shows on the right, how the "HelloWorld" action is used with the "MessageHandler". The "HelloWorld" action is a simple action, but can send a simple string message to any thread.

Example 2 shows a simple implementation of a HelloWorld action. Action functions can store state in a structure, which can be used to share state between DRAMA, but the implementation in this case is very clear:

```c++
#include <iostream>
#include <cstdlib>

// Hello World example

int main() {
    std::cout << "Hello World!" << std::endl;
    return 0;
}
```

### Basic Hello World example

**Threaded "Hello World" example**

In example 1 shows the "HELLOWORLD" action is running in the main thread while it is running on "Kickback". In the traditional DRAMA system, when the "HelloWorld" thread is called, the DRAMA message thread, the main thread颐 is called and starts running actions in threads.

The second part of the figure is an alternate sub-Class of "MessageHandler". This class is a handler of messages (what is called an Action Handler). The class must provide the method "MessageHandler::Handle()", which is called when a message is received while a thread is running. The thread is not terminated, and the action is marked as completed. Immediately, another thread can be run on the same thread, even if it is hidden by DRAMA. A message is taken as the target of the thread, but is not processed any further.

Example 1 shows on the right, how the "HelloWorld" action is used with the "MessageHandler". The "HelloWorld" action is a simple action, but can send a simple string message to any thread.

Example 2 shows a simple implementation of a HelloWorld action. Action functions can store state in a structure, which can be used to share state between DRAMA, but the implementation in this case is very clear:

```c++
#include <iostream>
#include <cstdlib>

// Hello World example

int main() {
    std::cout << "Hello World!" << std::endl;
    return 0;
}
```

### Basic Hello World example

**Threaded "Hello World" example**

In example 1 shows the "HELLOWORLD" action is running in the main thread while it is running on "Kickback". In the traditional DRAMA system, when the "HelloWorld" thread is called, the DRAMA message thread, the main thread颐 is called and starts running actions in threads.

The second part of the figure is an alternate sub-Class of "MessageHandler". This class is a handler of messages (what is called an Action Handler). The class must provide the method "MessageHandler::Handle()", which is called when a message is received while a thread is running. The thread is not terminated, and the action is marked as completed. Immediately, another thread can be run on the same thread, even if it is hidden by DRAMA. A message is taken as the target of the thread, but is not processed any further.

Example 1 shows on the right, how the "HelloWorld" action is used with the "MessageHandler". The "HelloWorld" action is a simple action, but can send a simple string message to any thread.

Example 2 shows a simple implementation of a HelloWorld action. Action functions can store state in a structure, which can be used to share state between DRAMA, but the implementation in this case is very clear:

```c++
#include <iostream>
#include <cstdlib>

// Hello World example

int main() {
    std::cout << "Hello World!" << std::endl;
    return 0;
}
```