Multi-Processor Defense Applications Implemented Utilizing the Message Distribution Framework (MDF)

Embedded Systems Real-time Application Framework

P. Barile, J. Cook, E. Kosinski
Lockheed Martin MS2
(philip.m.barile, joseph.j.cook, edward.kosinski)@lmco.com

Introduction

The digital signal processing demands of advanced radar systems require ever shrinking latencies and steadily expanding throughput bandwidths. Multi-processor computers with high compute-density and fast communication networks are often the configurations of choice.

Along with the stringent processing demands are the ever shrinking production schedules and steadily expanding requirements. Building embedded applications to exploit multiple processor architectures without reinventing them every time together with changing requirement specifications presents a challenge. In today's COT's-friendly defense environment, delivering system performance and scalability without sacrificing portability has become extremely challenging.

Open standards such as C++, POSIX, MPI and VSIPL address the portability issues. Combined with the COTS multi-processor hardware configurations that are becoming more powerful and abundant along with a product development team that understands object-oriented analysis and design methodology will put you on a successful path to meet customers' current and future needs.

Message Distribution Framework Services

Traditional object-oriented application development addresses solving the requirements of a single problem. An application framework [1] is more appropriate to address the problem domain and solve a class of problems.

The Message Distribution Framework (MDF) is an Object Oriented application framework implemented in C++ and tailored to the stringent requirements of embedded digital signal and data processor applications need for meeting high throughput and low latencies in a multi-processor embedded architecture. The MDF provides an application development and execution framework by entering a partnership with the application developer.

The application developer provides the "main" application routine, a set of application specific Components and a Network Topology Specification (NTS) and links in the MDF libraries. The "main" routine simply invokes a few MDF methods sequentially: `create`, `init`, `run`; and hands over control of the process to the MDF. The MDF will invoke the application Components' `run()` methods with the agreement that the Components will return control to the MDF.

The major MDF services include support for inter-Component application messaging and Component execution control.

A single application executable can be deployed across multiple processors, wherein each instantiation can take on specific functionality as dictated by the specification within the NTS. Additionally, large programs may find it necessary to divide applications into multiple, domain specific executables; each mapping to multiple processors. These separate executables are coalesced by the MDF into a single meta-application through the use of a common NTS file.

Control Distribution Service

Once the MDF assumes execution control of the application, all processor mapped Components wait on application specific control messages to initiate their execution.

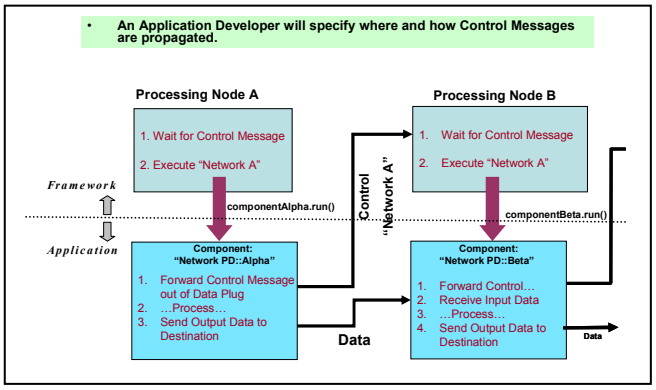

Figure 1: Control And Data Flow

Using MPI as a portable standards-based communication mechanism, MDF allows for run-time selection of processor topologies for application Components. It provides the mechanisms to allow Components to forward application specific control messages to remote processors and down to the Components participating in the specified Network.

This scheme supports two distinct control schemes: message driven and free spinning.

Message driven can be further divided into orthogonal and pipeline control. Orthogonal control allows one processor to be deemed "master" and orchestrate all other processors from a central location. Pipeline control allows the flow of control to pass along the network with the data.

Additionally, the MDF supports free spinning execution by providing an "auto" network feature that allows Components access to processor resources independent of control messages.

Component Interface

The MDF provides a Component interface API that requires the user application to extend it with at least one specific implementation. It is within the application-specific Component that the domain specific algorithms are implemented, such as, digital signal processing (DSP) functions, external device control and communications, and other compute-intensive data reduction and processing.

Run-time Construction using Component and Connection Factories

Using the Factory design pattern [2], the MDF defers construction of application Components to application initialization, thereby allowing processors to instantiate only the objects, and therefore the heap memory resources, that are needed on that processor. Similarly, implementation specific Connections are constructed at application initialization. This feature allows application designers to remap Components and change Connections without the need to re-compile the application.

Connections: Inter-Component Communication

Connections provide the inter-Component communications through an abstraction layer, relieving the Component from implementation details of the specific Connection. The Connection interface API provides blocking and non-blocking send and receive methods, allowing Component designers to choose the method of throttling data flow.

Network Topology Specification

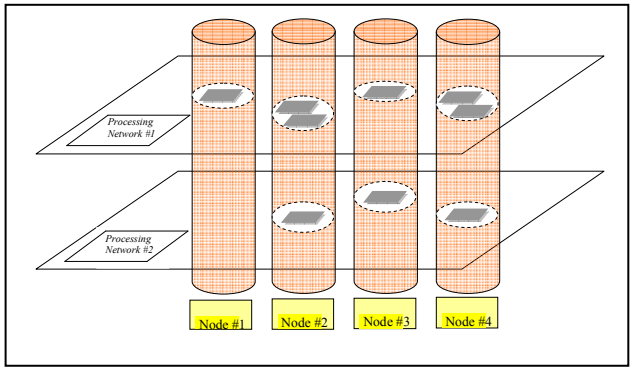

Figure 2: Network Topology

The Network Topology Specification (NTS) is implemented in Extensible Markup Language (XML) syntax and is parsed by the MDF service at program initialization. Its content is defined by the application designer and specifies the application specific Component-to-processor mappings. NTS provides the means to partition the Components into "networks" that span one or more processors that are controlled via the Control messages. Through the Component mapping and Connection specifications, the NTS allows the application designer to specify the data and task parallelization scheme, such as Block, Block Cyclic, Block Overlap, Round-robin, point-to-point and broadcast.

Multiple Target Support

Using industry standards such as C++ and MPI, MDF allows support of multiple platforms such as Solaris, MCOE and Linux. This allows application developers to develop and test in the environment most appropriate considering the available resources and to regression test on the target platform.

Conclusion

The MDF provides the application software designer with an extensible and flexible ready-made architecture containing high levels of abstraction with simple interfaces to minimize radar application design/development time. The MDF development effort can pay itself off through the repeated generation of applications within the same domain.

Other advantages include rapid remapping of processing functions to different processors, and changing their inter-processor communication without code recompilation. This flexibility reduces overall system cost by reducing the total number of processors required to meet system requirements and providing low development overhead for tailoring network topologies in reaction to changing requirements.

Additionally, Object Oriented Design encourages code and function reuse and reduces future maintenance costs.

References

[1] eds. M. Fayad, D. Schmidt, R. Johnson, *Building Application Frameworks: Object-Oriented Foundations of Framework Design,* Wiley, 1999

[2] E. Gamma, R. Helm, R. Johnson, J. Vlissides, *Design Patterns,* Addison-Wesley, 1994