PNNL-18716

Prepared for the U.S. Department of Energy
Under Contract DE-AC05-76RL01830

MeDICi: An Open Platform for Sensor Integration

AS Wynne
I Gorton
JM Chase
EG Stephan

September 2009

DISCLAIMER

PACIFIC NORTHWEST NATIONAL LABORATORY
operated by
BATTELLE
for the
UNITED STATES DEPARTMENT OF ENERGY
under Contract DE-AC05-76RL01830

Printed in the United States of America

Available to DOE and DOE contractors from the
Office of Scientific and Technical Information,
P.O. Box 62, Oak Ridge, TN 37831-0062;
ph: (865) 576-8401
fax: (865) 576-5728
email: reports@adonis.osti.gov

Available to the public from the National Technical Information Service,
U.S. Department of Commerce, 5285 Port Royal Rd., Springfield, VA 22161
ph: (800) 553-6847
fax: (703) 605-6900
email: orders@ntis.fedworld.gov
online ordering: http://www.ntis.gov/ordering.htm

This document was printed on recycled paper.
(9/2003)

PNNL-18716

MeDICi: An Open Platform for Sensor Integration

AS Wynne
I Gorton
JM Chase
EG Stephan

September 2009

Prepared for the U.S. Department of Energy
under Contract DE-AC05-76RL01830

Pacific Northwest National Laboratory
Richland, Washington 99352

1. Introduction

MeDICi (Middleware for Data Intensive Computing) is a platform for developing high performance, distributed streaming analytic and scientific applications. Developed at Pacific Northwest National Laboratory (PNNL), MeDICi has been released under an open source license and is based on enterprise-proven middleware technologies including a widely used Enterprise Service Bus (ESB), the standard Business Process Execution Language (BPEL), and open source message brokers. Wherever possible, we have built on existing open source, standards-based systems and integrated them into a coherent whole by creating simplified graphical programming tools such as a Workflow Designer and an easy to use and well-documented integration API. This software development approach allows us to: avoid re-creating complex service integration and orchestration systems, reap the benefits of continual improvements to the technology base, and focus on creating tools and APIs which allow for the creation of re-usable component-based software applications and workflows. These aspects have facilitated rapid adoption of the platform within PNNL for demonstration and operational applications. In fact, MeDICi has been used for a wide range of integration projects including two sensor integration applications described later on in this paper.

The remainder of this white paper is organized as follows: Section 2 provides a high-level description of the MeDICi architecture. In Section 3, the open aspects of the API and tool development are highlighted. Section 4 explains system readiness by presenting relevant demonstrations and deployments. Finally, documentation and licensing details are provided in Section 5.

2. MeDICi Architecture

2.1 Architecture Overview

MeDICi is a middleware platform for building complex, high performance analytical applications. These applications are structured as workflows (sometimes called application pipelines) of software components, in which each component performs some independent processing of incoming data and then passes results to the next step in the workflow.

MeDICi is an open software architecture (depicted in Figure 1) that consists of three loosely-coupled components:

- MeDICi Integration Framework (MIF) – an event-based messaging platform for distributed component integration. MIF is the core of a MeDICi application and

comes with an API for the construction of re-usable components and analytic pipelines.

- MeDICi Workflow – a graphical workflow design environment that is used to orchestrate existing MIF components. Workflow is based on BPEL (Business Process Execution Language) technologies, providing a standards-based recoverable workflow execution engine.
- MeDICi Provenance – an RDF-based data store for capturing and querying workflow metadata that can be used for reconstruction and forensic investigation of application results.

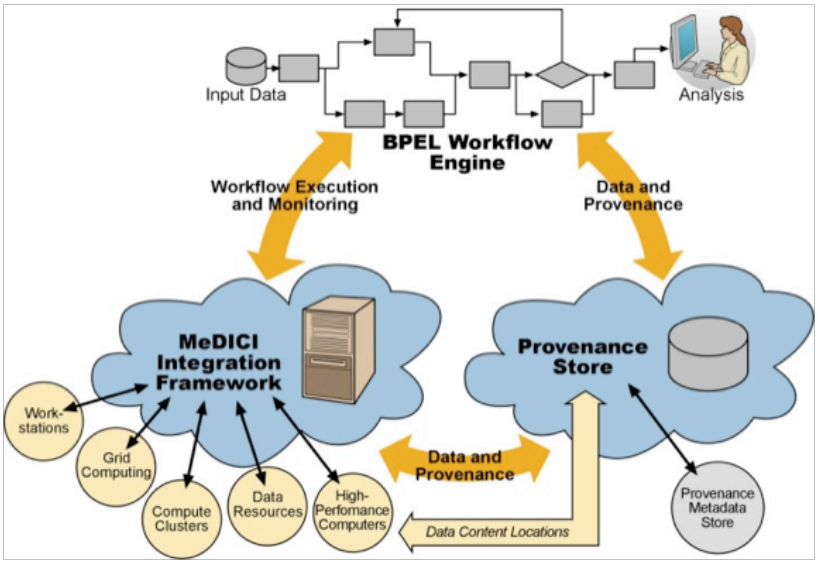

Figure 1. MeDICi Architecture

In a MeDICi Workflow application, the application designer creates a workflow graphically using our Dynamic Workflow (DWF) language. Each task in the workflow calls an associated Web service, which is typically supported by a MIF component deployed in the MIF container but may alternatively be a standard service located anywhere on the Internet. MIF components wrap computational codes that require complex integration, and support a protocol that is designed to minimize the data transfer overheads between elements of the workflow. Optionally, MIF components can record metadata (known as provenance) about the data they receive/produce and the processing carried out. The metadata is passed transparently from a component to the MIF container, and this sends a message to a message queue called *ProvenanceListener*. MeDICi Provenance takes these messages from the queue, and stores them in an RDF store for subsequent analysis by users.

2.2 MeDICi Integration Framework

MIF components are constructed using an easy-to-use Java API that allows a programmer to rapidly create network-ready multi-threaded processing elements. Components consist of a collection of one or more *MifModules*, which perform some data processing function such as transformation, application-specific analysis, or file copy from one host to

another. Communication between modules is abstracted as *endpoints* and may take place over any one of several standard communication protocols such as Java Messaging Service (JMS), HTTP, SOAP, TCP, or by simple Java method calls. Endpoint transports and paths may be changed without regard to the processing logic. This allows MIF programmers to focus on algorithm development without having to worry about implementing complex networking or threading code.

Although the MIF is implemented in Java and provides a Java API, codes that are encapsulated as MIF components can be written in any programming language (e.g. we have experience with C++, FORTRAN, Java, R, Haskell, C#, Matlab) and be distributed across a cluster or a network of processing nodes. A simple caricature of a MIF application architecture is depicted in Figure 2.

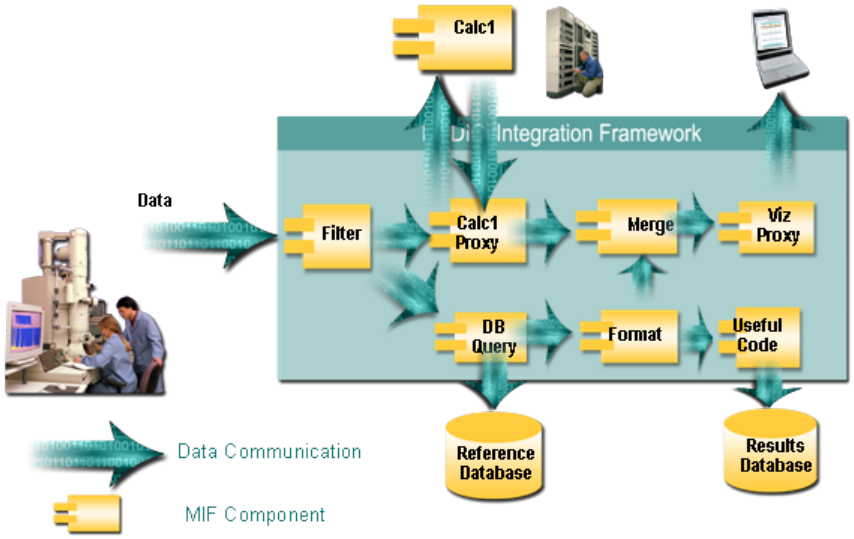

Figure 2. A MIF Application

Local components execute inside the MIF container, executing Java-based application components that are defined in the workflow application. Remote components are wrapped in the same programmatic interface, but utilize a component proxy in MIF that facilities executing the component code outside the MIF container. Remote components are used to implement distributed workflows and to integrate with non-Java codes and high-performance computing platforms. Further, MIF components can be hierarchically structured, allowing more complex functionality to be composed internally within a component.

As mentioned, we have designed the MIF API to be regular and simple to learn. Still, to further ease the burden of MIF application development, we provide two other ways to configure a MIF pipeline: by graphically creating the pipeline using the MIF Component Builder (see Figure 3) or by creating an XML configuration file by hand. The component builder application allows application developers to graphically wire together the components in a MIF application and configure each component with a number of properties, including the underlying application code to execute. The tool then generates

all of the MIF API code needed for an executable MIF application. Components can also be stored in an XML representation in a component library for subsequent use in different MIF applications.

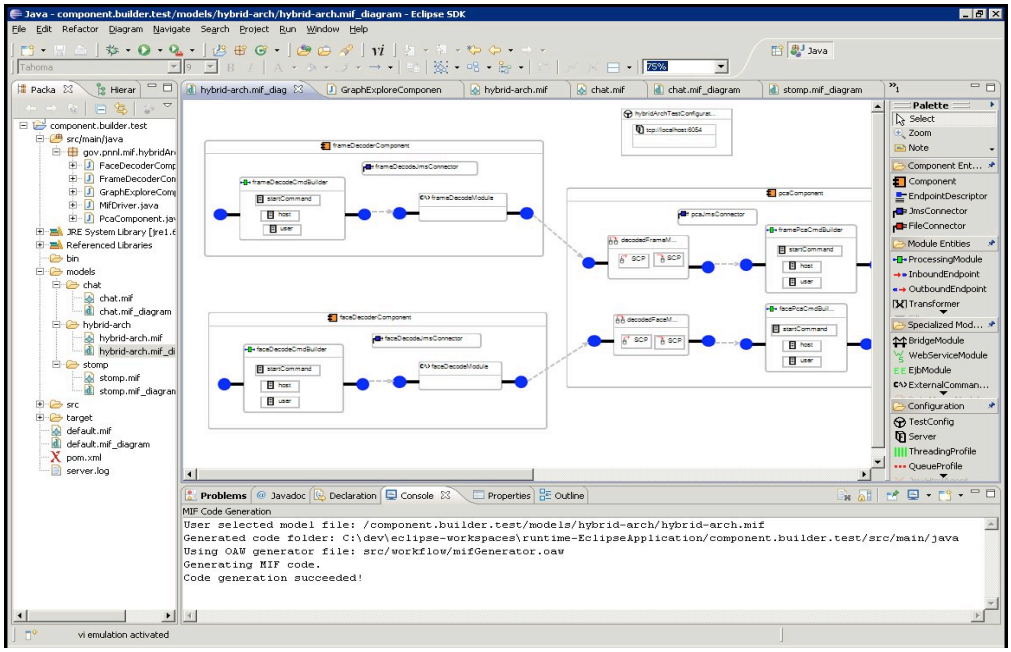

Figure 3. MIF Component Builder

The MIF container environment is provided by Mule, an open source messaging platform (http://mulesource.org/). MIF extends the Mule API to make application construction simpler and to create an encapsulation mechanism for component creation. Our current implementation uses ActiveMQ (http://activemq.apache.org) for messaging and CXF (http://cxf.apache.org) for Web services, but the MIF API is agnostic to the underlying messaging platform. Integrating with third party messaging systems is as simple as placing the correct client jars on the MIF Java classpath – code implemented using the MIF API does not need to be changed. This allows deployments to configure MIF applications using specific technologies that meet their quality of service requirements. In addition, if the provenance and JMS options are not used in a MIF application, then the deployed application does not need these platforms to be started, thus releasing resources within the MIF container.

2.3 MeDICi Workflow

MeDICi Workflow is built upon standard BPEL (Business Process Execution Language) workflow engine technology. The user describes the steps in a workflow graphically using our MeDICi DWF design tool, which defines a visual workflow design language to abstract away much of the complexity of BPEL, and automatically generates a corresponding BPEL representation of the DWF workflow.

In the DWF design tool, designers select icons from a palette that contains a list of all MIF components available for the application. Each icon is associated with an XML description of the properties of the MIF components that must be configured in order to use the component in DWF. These properties also specify the Web service endpoints that the BPEL engine must call to asynchronously invoke each MIF component. MeDICi Workflow applications can also call external Web services that are not MIF components. Designers then connect the icons together into a workflow, add any necessary configuration properties, incorporate control constructs such as loops, branches and parallel executions, and finally generate and deploy the BPEL to execute the application.

A moderately complex DWF workflow is shown in Figure 4. This workflow is used by PNNL scientists to measure atmospheric radiation as part of an ongoing research project. The top-level structure of the workflow is the Task. Within a Task, the designer defines the workflow. The designer drags previously constructed MIF components from the palette into a Task and defines input and output properties for each component. Properties represent the input values that must be passed to the MIF component. These can be constants that the designer sets, or variables that are derived from the outputs of a previous step in the workflow. Global variables can also be defined in DWF and simple expressions can be created by the designer to manipulate global variable values between workflow steps. The workflow designer then connects the components together in a sequence.

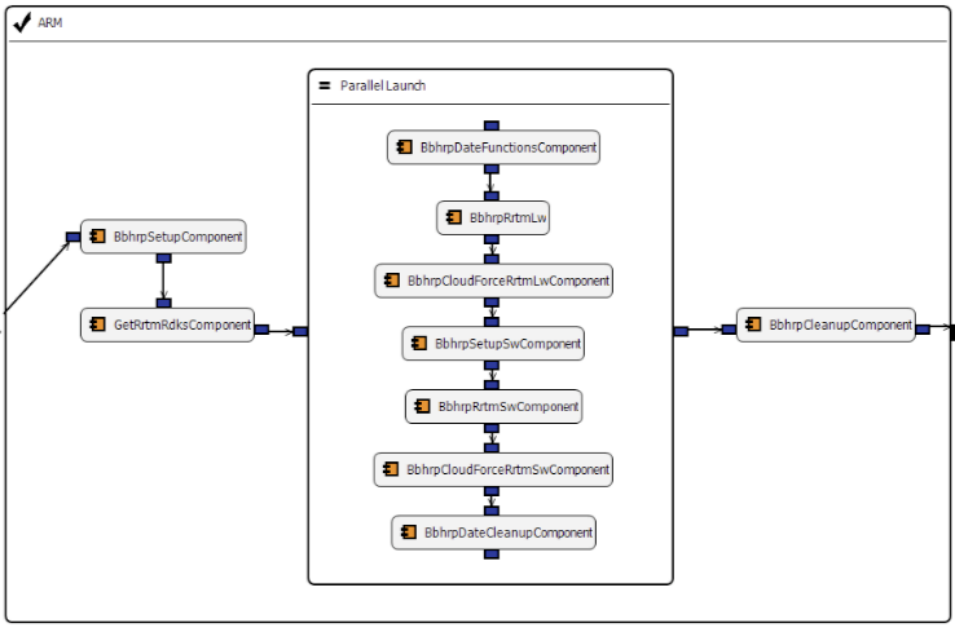

Figure 4. MeDICi Workflow Designer

In addition to sequences of components, DWF supports iteration, selection and parallel control constructs. For example, in Figure 4 a ParallelLaunch construct specifies that the contained sequence of components should be replicated and executed concurrently. A

Data Array Key property of the ParallelLaunch specifies the name of a variable that contains an array of data items to be processed by the concurrent components. Each item in the array is passed to its corresponding component instance within the ParallelLaunch, giving a simple mechanism for passing data to concurrent component instances. An equivalent mechanism is available for collecting the outputs from each concurrent component for subsequent merging and processing.

2.4 MeDICi Provenance

MeDICi features a provenance implementation that tracks the event history of workflows and remote processing. MeDICi provenance includes an asynchronous message-based provenance listener that can capture provenance from MeDICi and other applications using JMS messaging. Provenance is accessed from within a MifModule by using an API called the "Describe Anything" API (DaAPI). The DaAPI provenance listener written in Java is documented and records provenance relying upon the standard Open Provenance Model (OPM) developed by an international community of scientists (http://openprovenance.org). The OPM not only features a model specification, but also provides Java APIs to record the model in XML or RDF/XML formats.

For provenance storage, the architecture supports the commonly used open source RDF store Sesame (http://www.openrdf.org/). Once the provenance is stored PNNL-created tools can be used to filter standard SPARQL (http://www.w3.org/TR/rdf-sparql-query/) query results. Third party open source products such as the Longwell browser can also be used to visually explore the provenance information. Depending on application needs, provenance can be collected in a number of ways, including: capturing it as a stream of data, or by parsing text file artifacts or even binaries. To do this we developed a Java based tool called Defuddle (http://defuddle.pnl.gov/) that extracts data based on a schema specification of the file.

3. Software Environments

In this section we describe the software development environment that a MeDICi application developer would be presented with and give a flavor of the MIF API. For a detailed explanation of the API, please refer to Section 5 for pointers to our online documentation. This section also briefly describes the software development approach used internally to create MeDICi.

3.1 API Development Environment

Since MIF is a Java API, it is suitable to use any Java Integration Development Environment (IDE) running on any operating system for MIF application development. MIF is distributed with a single Java Archive (jar) containing all MIF class dependencies, which is provided for convenience to allow a developer to include it on his/her classpath. Some developers may also choose to simply use a text editor and command line java

compiler to build their applications. However, the graphical tools (Workflow Designer and Component Builder) are implemented as plug-ins in the Eclipse (http://www.eclipse.org) application framework. So, to use these tools it is necessary to install Eclipse, and then install the MeDICi plug-ins from within Eclipse.

3.2 MIF Application Programming Interface

As mentioned in Section 2.2, individual MIF processing elements are referred to as MifModules. As MIF is an event-based system, the output of one MifModule is sent over an endpoint and then becomes the input of another MifModule. MifModules working together to form a reusable task are combined to create MifComponents. So, to create a MeDICi application, one or more MifComponents are combined by either using the Workflow Designer to graphically create an application, or by programmatically using the MIF Java API. In the latter case, the MifComponents are simply added to the pipeline and the MIF server is started using API methods. A full description of the MIF API is beyond the scope of this paper. Please see Section 5.1 for pointers to online documentation that provides full API documents and sample API tutorials.

The MIF API and MeDICi tools described above were created using a software engineering methodology known as Model Driven Software Development (MDSD). MDSD focuses on the creation and manipulation of formalized abstractions known as models. These models can be created and represented graphically as, for example, a UML diagram. Then the models can be used to mechanically generate source code or be transformed into other models. MeDICi employs the Eclipse Modeling Framework (EMF) to perform MDSD tasks. For example, the core MIF API was created by graphically building a model. This allows much of the API code to be automatically generated by the EMF tool chain. Any time the API model changes, the source code can be regenerated and kept automatically in sync with the model. This technique has the interesting side effect of producing constantly up-to-date UML documentation. Even more significant, EMF and related technologies are used to automatically generate an estimated 90% of the code that makes up the graphical tools described above. MeDICi developers have found MDSD to be a valuable asset in having a code base that is easily maintainable and can be understood by engineers and users who are not engaged in active development of the software.

4. System Readiness

MeDICi has been developed as an internal research initiative over the past four years. During this time it has been used in numerous high performance demonstrations over a number of scientific and analytic domains including proteomics, electricity grid simulations, face recognition from streaming video, streaming text analytics, and atmospheric monitoring. At this time, MeDICi is beginning to be used outside of PNNL. For example, the Institute for Systems Biology (www.systemsbiology.org) is adopting it as an asynchronous messaging layer as a part of its popular Gaggle toolkit

(http://gaggle.systemsbiology.net), which is used by biologists to investigate infectious diseases, among other biology related tasks.

In addition, two projects applying MeDICi technologies are particularly germane to DHS interests in the area of COP/C2. These projects are described below.

4.1 Network Sensor Fusion and Analytics

PNNL has created a Distributed Intrusion Detection System (DIDS), which was deployed in production to monitor network activity at the 2006 SuperComputing conference. PNNL maintains a standing demonstration of this system and due to the success of this project, it is in the process of being adopted for use by PNNL's internal network security monitoring team. The DIDS was created as a research challenge to solve a pressing problem for PNNL. PNNL maintains a set of network sensors deployed in several locations around the country to collect network session data. At the time the project was started, sensor data was summarized and indexed offline and was made available 24 hours after collection. It was then analyzed on demand as part of forensic studies of past suspicious activity. While forensic studies are important, a capability is needed which can detect threats as they occur. To this end, the PNNL DIDS is augmenting the capability with analytic components that have the ability to detect anomalous behavior in near real time. To do so, a system is needed which can integrate any number of sensors over a wide geographic range. Since network based threats are increasing in number and complexity, they system must be scalable and have a pluggable architecture which allows components to be deployed and reconfigured easily. PNNL has found that MeDICi is a suitable, high performance integration platform on which to build this system.

4.2 Situational Awareness for Emergency Management

PNNL has used MeDICi to create a Situational Awareness system for Emergency Management (EM). This project is being developed under DHS funding on behalf of the Pierce County Department of Emergency Management (PCDEM) in Pierce Co., WA. This system integrates emergency call data with GIS data and displays it on a map-based graphical client (see Figure Figure 5) that allows EM staff to geographically locate call data in relation to resource information stored in an ESRI GIS database.

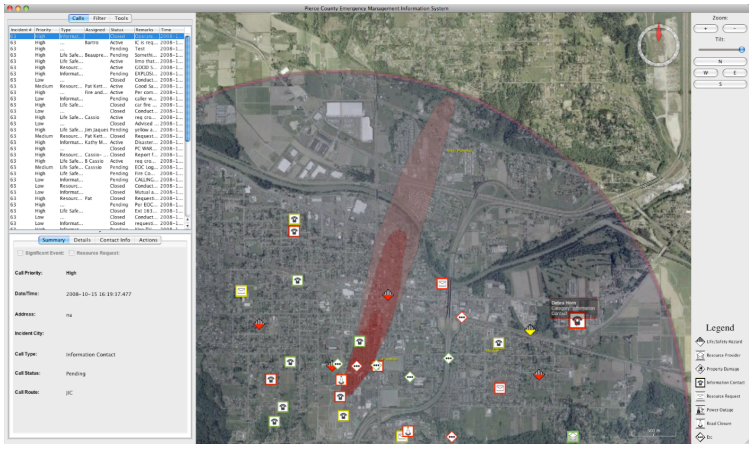

Figure 5. MeDICi GIS Integration

5. Documentation and Licensing

5.1 Documentation

All of MeDICi documentation is contained at the website: http://medici.pnl.gov. There, the MeDICi component model is described, sample code and tutorials are provided, as is API documentation in the form of javadocs. Also a MeDICi publication bibliography is available on the site. The MIF is available for download from the site and is distributed with all binaries needed to create a MIF application as well as all MIF source code. Finally, the MIF API model is available as a UML diagram, which may be helpful for software engineers in understanding the structure and intent of the API.

5.2 Licensing and Third Party Considerations

MeDICi source code and binaries are freely available to all U.S. government agencies under Government Purpose Rights without cost as this software was developed on behalf of the Department of Energy (DOE). The software is also freely available to the public under a BSD-style license known as the PNNL Standard OSS License (http://medici.pnl.gov/standard_pnnl_oss_liscense). As such, there is no requirement to freely distribute changes made to any modified code. Further, all the software that MeDICi links with use open source licenses that do not require that source code changes be re-distributed. There is no commercial licensing model for the product. There is no requirement to purchase additional commercial licenses for third party systems as the software is designed to be used with free software products. However, it is easy to integrate MeDICi with third party commercial products such as ArcGIS when necessary.

MeDICi is completely open to incorporation of capabilities provided by third-party vendors, particularly those released as web services. MeDICi can run on any hardware capable of running a Java Runtime Environment (JRE) of version 1.6 or greater. For systems on which a JRE is not available – for example on specialized high performance computing platforms -- it is possible for MeDICi to launch and manage processes created to run on such systems.

6. Summary

In solicitation number HSBP1209RCOPCC, DHS requests information on integration systems that are built on standard technologies, have open licensing terms, open architectures, and have well-documented, easy to use APIs. These features are desired because DHS envisions a system that is extensible and maintainable into the future. This whitepaper has demonstrated that MeDICi possesses all of these features. Further, MeDICi is a strong sensor integration platform and is being used to engineer a GIS enabled Situational Awareness system for Pierce Co., WA emergency managers. This puts MeDICi in a strong position to meet the DHS' COP/C2 needs.