Proceedings of the 2001 Particle Accelerator Conference, Chicago

A FLEXIBLE AND CONFIGURABLE SYSTEM TO TEST ACCELERATOR MAGNETS

J.M.Nogiec, J.DiMarco, H.Glass, J.Sim, K.Trombly-Freytag, G.Velev, D.Walbridge,
Fermi National Accelerator Laboratory, Batavia, IL 60510, USA

Abstract

Fermilab's accelerator magnet R&D programs, including production of superconducting high gradient quadrupoles for the LHC insertion regions, require rigorous yet flexible magnetic measurement systems. Measurement systems must be capable of handling various types of hardware and extensible to all measurement technologies and analysis algorithms. A tailorable software system that satisfies these requirements is discussed. This single system, capable of distributed parallel signal processing, is built on top of a flexible component-based framework that allows for easy reconfiguration and run-time modification. Both core and domain-specific components can be assembled into various magnet test or analysis systems. The system configured to comprise a rotating coil harmonics measurement is presented. Technologies as Java, OODB, XML, JavaBeans, software bus and component-based architectures are used.

1 INTRODUCTION

The challenges encountered by R&D programs typically require very flexible solutions. In the special case of accelerator magnet development, flexible magnetic measurement systems have to be built. Such systems must be capable of employing various measurement techniques and analysis algorithms while using diverse hardware configurations. Such requirements create conditions that are ideal for software reuse practices but also demand great flexibility.

Software reuse can be characterized by the ability to use the same software modules or designs over and over again. Software flexibility, in contrast to reuse, allows for easily changing the overall functionality or performance of the system. When functionality can be further modified after deployment, we are talking about software tailoring.

2 FRAMEWORK

Using component-based development, a technology that employs software modules designed to be used repeatedly in developing applications, can satisfy both the need for flexibility and the need for reusability.

Broad reuse requires finer grain, simple components whereas convenience calls for large, specialized components. Ideally, reusable components should be customizable to easily fit the new specific application scenario.

In order to promote reuse, a component framework has been developed [1]. This framework deals with generic aspects pertaining to any application architecture and has been built from ground up with components in mind. It offers dynamic linking of components upon loading and serves as the mediator in inter-component communications.

The framework offers a software bus communication architecture where data-driven components communicate via events. Five categories of events have been introduced:

- Data events used to pass processed data.
- Control events sent to stimulate required behavior of components.
- Debug events generated to facilitate debugging and analysis of the system.
- Exception events that are further classified as errors, warnings, and significant system events.
- Property events used to inspect and modify component properties.

Java beans components of various levels of specialization and reuse are offered. A set of general purpose (horizontal) components, called core components, is supplemented by less general, application domain specific (vertical) components. These domain specific components group generic magnetic measurement components and other modules, unique to various specific applications and hardware. A total of 48 components have been developed including:

- Core components.
- Data display components.
- Data analysis components.
- Magnetic measurement components.

The measurement systems are configurable with help of the application description language (ADL), a proprietary dialect of XML. ADL has provisions for describing components and their properties, inter-component communication patterns (links between components), and sequences of control events. An excerpt from a configuration file is shown below:

```
<!DOCTYPE configuration SYSTEM "ems.dtd">
<configuration version="0.1" title="Display Test XML">
<!-- Component definitions -->
<component id="Generator"
        class="ems.measurement.dataDisplay.SyntheticData">
        <property name="delay" value="10000"/>
</component>
<component id="DataDisplay"
                    class="ems.core.components.SimpleDataDisplay">
        <property name="title" value="Data Display Component"/>
        <property name="XPosition"  value="0"/>
        <property name="YPosition"  value="500"/>
        <property name="wrapLines" value="true"/>
        <property name="width"  value="400"/>
        <property name="height" value="250"/>
</component>

<!-- Routing information -->
<route type="Data" origin="Generator"
destination="DataDisplay"/>

<!-- Control signals -->
<control signal="init"  destination="!"/>
<control signal="start" destination="!"/>
```

In this excerpt, a data generator component (*Generator*) is defined and linked with a data display component (*Display*). Various properties are set, then both components are initialized and started by sending appropriate control signals to them ("!" means all components).

3 DEVELOPMENT CYCLE

In the presented system, the traditional development cycle has been replaced by a new one, which uses a component-based development technology. In this technology, systems are assembled from a set of components without traditional programming efforts (see Fig. 1). The selected components are adapted to their roles by setting their properties.

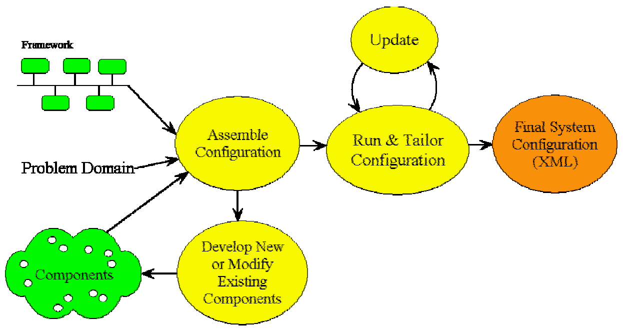

Framework
Problem Domain
Assemble Configuration
Update
Run & Tailor Configuration
Final System Configuration (XML)
Components
Develop New or Modify Existing Components

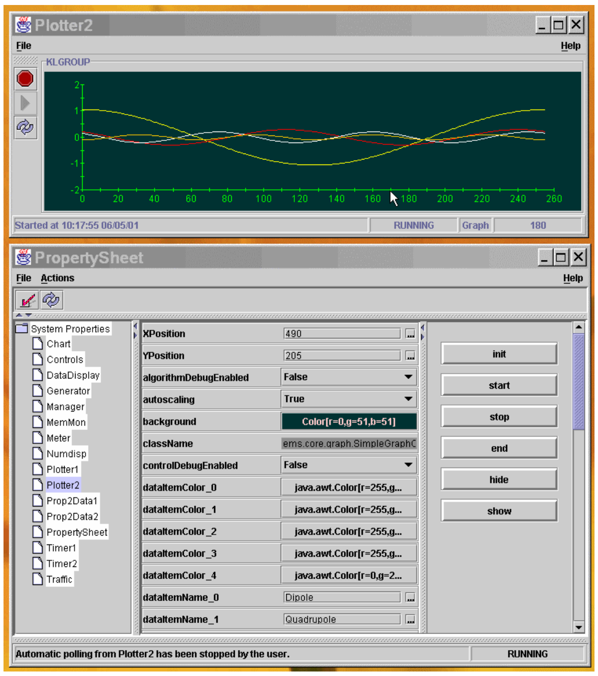
Figure 1: Application development cycle.

Partial definitions of the system, consisting of several components, can be included in the main setup file to reduce the complexity of the setup and offer coarse pseudo-components.

The constructed applications can be further customized (tailored) at run-time to fulfill specific users needs and work situations. This is accomplished with help of the property editor component, as illustrated in Fig. 2. The modified system configuration can be stored as an XML file for future use. In addition, complete configurations are stored together with data before and after each test.

Debugging and exception reporting mechanisms that are integrated in the framework and components support both assembly and testing of applications. Both mechanisms support fine-grained control of the amount and type of information that is reported.

A simulator of the data acquisition hardware has been developed to facilitate configuration, testing, and verification of data processing setup.



Started at 10:17:55 06/05/01 | RUNNING | Graph | 180

XPosition	490
YPosition	205
algorithmDebugEnabled	False
autoscaling	True
background	Color[r=0,g=51,b=51]
className	ems.core.graph.SimpleGraphC
controlDebugEnabled	False
dataItemColor_0	java.awt.Color[r=255,g...
dataItemColor_1	java.awt.Color[r=255,g...
dataItemColor_2	java.awt.Color[r=255,g...
dataItemColor_3	java.awt.Color[r=255,g...
dataItemColor_4	java.awt.Color[r=0,g=2...
dataItemName_0	Dipole
dataItemName_1	Quadrupole

Automatic polling from Plotter2 has been stopped by the user. | RUNNING

Figure 2: Property editor component.

4 SYSTEM CONFIGURATION

A family of measurement programs has been constructed using the above-described process. We will examine a harmonics measurement program (rotating coil system) to test superconducting accelerator magnets. The program (see Fig. 3) consists of the hardware-oriented components (data sources), a chain of data processing components (data processors), data visualization components (data presenters), system status visualization components (i.e. the memory monitor) and data archival components (data sinks). Data sources are the Motor, Z Motion Control and DAQ components. Some of the data processors include the Drift Correction, Fourier Transform, and Magnet Harmonics components. Data presenters are components such as the Plot components and the Numerical Display. System status components include the Memory Monitor and Property Controller. Data sinks consist of database components such as the OODB and the Error Log and Debug Log components as shown in Fig. 3.

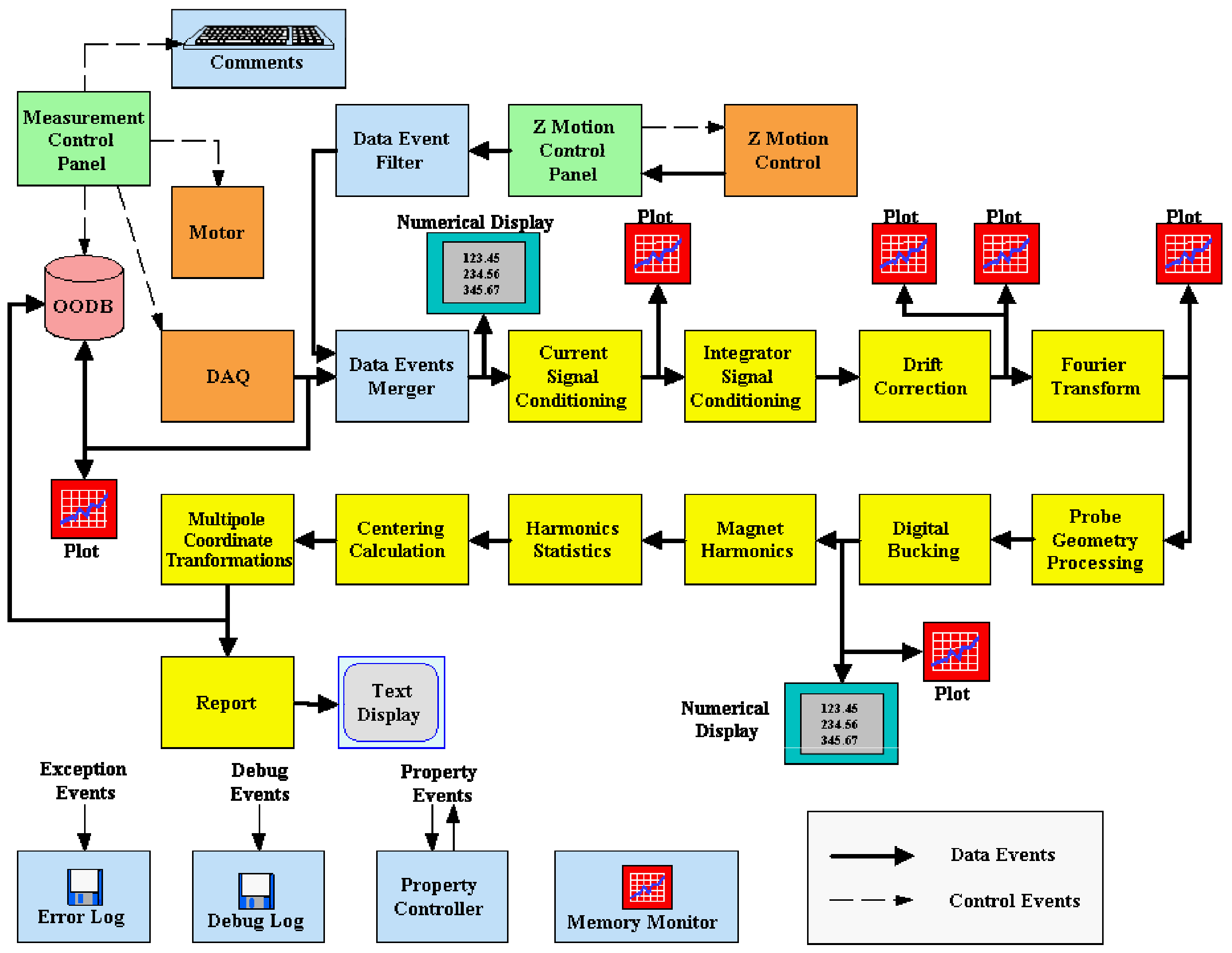


Figure 3: Configuration of the harmonics measurement system.

Overall measurement control and monitoring is provided by the Measurement Control Panel component. Control signals for data sources are passed from proxy controlling components to their implementations on different computers in order to control such hardware components as the positioning system, current, and stepping motors.

Data sources are designed as proxies for their implementations that run under the VxWorks RTOS. Data collected by a multi-channel integrator set and DVM are streamed to the Java DAQ proxy component, together with asynchronous exceptions and debug messages. From the DAQ component, data is passed through the sequence of processing components with some of the intermediate results sent to data presenters (graphs, tabular numeric displays, and text displays). Data events with raw data, selected intermediate results and final results are archived by the permanent storage component. This archiving component is based on the ObjectStore PSE Pro object-oriented permanent repository. In addition, a log of the measurement history is stored in a file.

5 SUMMARY

The component-based technology proved to be uniquely suited for development of test and measurement systems in R&D environments. Using this technology, a flexible and configurable system has been developed to test accelerator magnets in order to help in research studies and development of new magnet designs.

The measurement system is built on top of a component-based Java framework. The system is configurable via XML dialect that describes components and their connections. Tailoring of the system can be done at run-time through a property editor component. Modified configurations can be saved for future use.

The presented solution satisfies both flexibility and reuse requirements typically imposed on R&D systems.

6 REFERENCES

[1] J.M.Nogiec, J.Sim, K.Trombly-Freytag, D.Walbridge, “EMS: A Framework for Data Acquisition and Analysis”, ACAT'2000, Batavia, 2000.