FERMILAB-PUB-03-472-TD

Configuring Systems from Components: The EMS Approach

J.M.Nogiec, E.Desavouret, S.Kotelnikov, K.Trombly-Freytag, and D.Walbridge

Fermi National Accelerator Laboratory[*]*, Batavia, Illinois 60510*

Abstract. EMS is an exercise in component technology. It offers rapid development of specialized data acquisition, visualization and analysis systems via assembly from vertical and horizontal components. The EMS architecture allows for agile development of systems and promotes reuse of software. The framework supports a visual builder that shows connections between components and lists component properties. The system offers both off-line setup of properties and run-time modifications. Multi-bus architecture allows for independent routing of data, controls, debugs, and exceptions. The architecture, configuration process, and control of applications through scripting are presented.

INTRODUCTION

Traditional software development processes highly depend on a priori knowledge of all the requirements. The requirements, which are typically defined in the initial phase of the project, are used to develop software. This process frequently leads to monolithic solutions, where the cost of change to the system grows exponentially with time. In the R&D situations, where the ability to respond to ever changing requirements is a sine qua non condition for success, such a process is unacceptable.

The shortcomings of the traditional processes are addressed by the component-based agile development of applications. This approach is centered on the premise that anticipating and understanding every system requirement ahead of time is very difficult, if not impossible. The process is adaptive and is able to quickly reflect changes in user's requirements in produced software.

In the opinion of the authors, two fundamental conditions must be fulfilled in order to successfully employ agile development: a) an architecture that promotes easy changes and extensions to the system must be adopted, and b) a rapid development environment that allows for quick development iterations must exist. The rest of this article shows how both of these fundamental necessities are addressed by the EMS, a component-based system developed by the Systems Development and Support group at Fermilab.

*Operated by the Universities Research Association under contract with the U.S. Department of Energy

EMS OVERVIEW

The EMS is an extensible, Java-based framework, which allows for adding new components and specialization for various application domains. It replaces a traditional code-test-release application development cycle with assembly of systems from a set of components. It offers a rapid development environment in which universal components are supplemented with modules reused from other applications and newly developed components to form new applications.

The core system consists of an architectural framework supporting communication and system assembly, and a set of core components, which are components common to many sub-domains. Examples of such components include graphing components, numerical and textual display components, traffic and memory monitors, file and database I/O components, system debugging components, etc. These basic components are supplemented with domain specific components.

Within the system, components communicate through messages (events) exchanged over a software bus. The bus conveys five independently routed categories of events: data, controls, debug, property, and exceptions. The communication patterns are enforced by the router component and can be defined externally from the communicating components. Both content-based and address-based communication is

provided with the source routing and routing tables methods. Depending on their role in communication, components can be data producers, consumers, both, or neither.

Components, their properties, routing specification, and initial controls are specified in XML, which form the configuration of an application. They are implemented as Java Beans, which are connected to the bus via adapter objects. In order to send and receive messages over the bus, each component has to implement appropriate communication interfaces. Separate interfaces allow for exchanging of control, data, debug, and exception events (Figure 1).

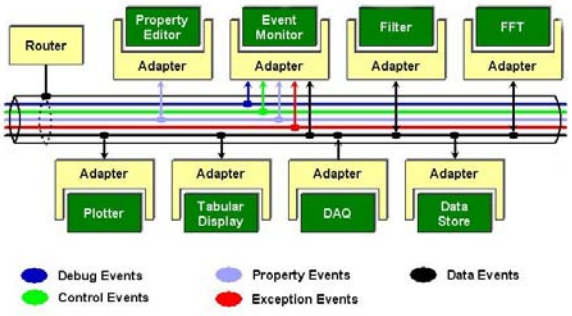

FIGURE 1. Inter-component communication in EMS.

APPLICATION DEVELOPMENT

EMS applications are built from components. The process of configuring the EMS application consists of three phases:

- Selecting a set of components to use
- Bringing components to their desired state.
- Linking components together

This process can be iterative and it can produce a series of executable applications, each of them functionally closer to the required result. Integration of debugging and exception handling solutions in the framework aids in rapid development by speeding up testing and debugging.

Choosing Components

EMS-based applications are assembled from existing EMS components using XML configurations. Users must determine what components are best suited for the needs of their application. Two tools have been developed with focus on the component documentation: 1) the EMS Help Viewer tool that displays the component documentation, and 2) EMS Help Composer, used to assist in producing the user-focused documentation.

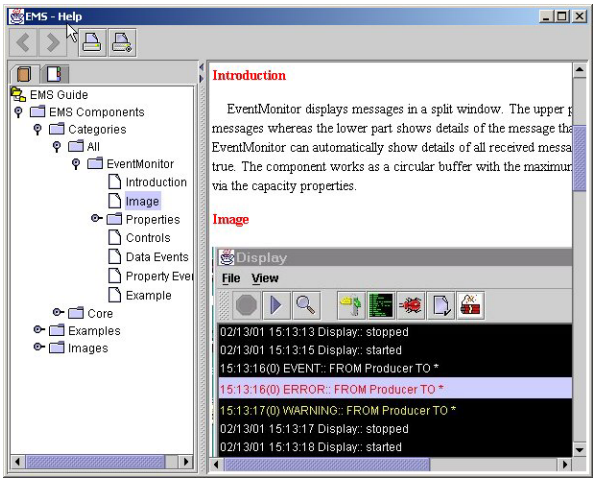

FIGURE 2. Component information.

The EMS Help Viewer (Figure 2) allows selection of the individual component in a variety of ways: alphabetically, by category, by configuration example (XML), or by image. The standard documentation of the selected component includes: name, category, purpose and introduction, image, properties, control signals, data accepted and produced, and an example XML configuration.

Setting Initial Component State

EMS applications consist of communicating components that are typically data driven. This means that a component automatically starts processing data when the data is received. EMS components can change their states in response to control events or external stimuli. Component data processing happens only if the component is in an appropriate state, such as RUNNING.

All EMS components have properties that are used to control the behavior, and allow customization of the component. Properties are modified by the framework in response to property events sent to the component. When debugging a configuration, it is possible to tailor the application by modifying property values dynamically at run-time. This is done using the property editor. Although this may be dangerous during an actual production run, it is useful when developing a new configuration. Dynamically changing a property's value shows how it affects the

behavior of a system without creating new configuration files and re-starting the application for each change. Using this feature can help to quickly refine a system.

Linking Components Together

As has already been stated, components exchange events via a multi-bus, which supports independent "wiring" of components for the different types of events, i.e. property, data, exception, control, and debug events. The system supports unicast, multicast, and broadcast communication patterns. Apart from static wiring specified in the XML configuration, components can specify destination addresses dynamically at run-time.

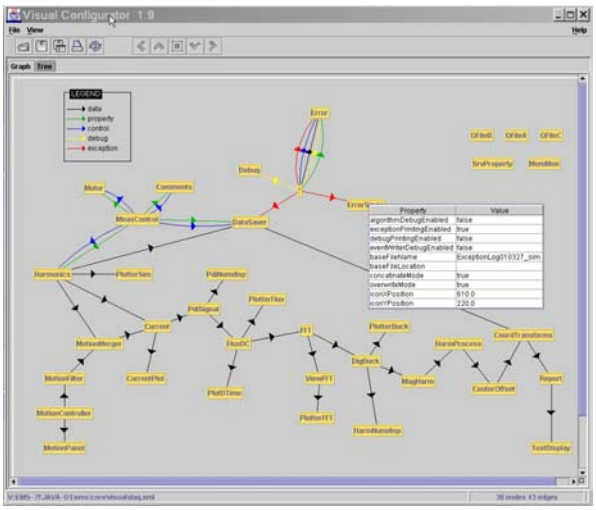

FIGURE 3. Wiring of components.

Figure 3 shows the wiring of an application as displayed by the EMS configuration viewer. Different colors are used to distinguish different event types. By clicking on the component, one can examine its properties.

Scripting

Scripting is the answer to the demand for quickly developing and automating new test or data processing procedures. With the provided scripting facility, users are be able to customize the system to suit their needs, ease their routine tasks, and automate tests. Repetitive tasks and complex procedures can be simplified by providing scripts to handle them. The scripting language augments the Java language and supports the accelerated development of new test procedures. As a result, domain experts are frequently able to add functionality to the EMS programs without the need to program new components in Java.

The Script Interpreter component is responsible for interpreting Python scripts. This component serves as an intermediary between the script and other components. It sends control, property, and exception events on behalf of the script and receives replies. The invoked interpreter runs in a separate thread to allow for concurrent execution of the EMS framework and the script. The user can conveniently run scripts using the Script Control panel, which is a GUI component to monitor and control execution of scripts (Figure 4).

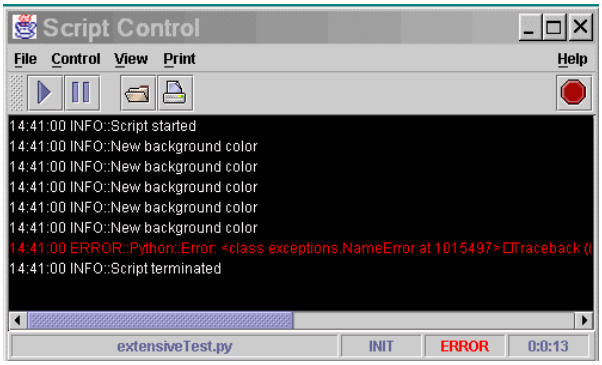

FIGURE 4. Script control component.

CONCLUSIONS

EMS is a black-box framework, which is a realization of an architecture that is extensible and flexible, and therefore fits perfectly the requirements of agile development. It allows for easy changes in functionality, by replacing or adding components, and limits code refactoring to isolated (encapsulated) components. Consequently, one can incrementally build a system through a series of executable releases. At the same time EMS provides for a high level of reusability, by allowing for reuse of the framework and both domain-specific and general-purpose components. The behavior of the application can be modified at run-time via property changes, and extended through scripting.