

A Component-Based Application Framework for Manufacturing Execution Systems in C# and .NET

Reinhard Füricht¹, Herbert Prähofer², Thomas Hofinger¹, Josef Altmann¹

¹Software Competence Center Hagenberg
A-4232 Hagenberg / Austria

{reinhard.fuericht, thomas.hofinger, josef.altmann}@scch.at

²Institute of Systems Science, Johannes Kepler University
A-4040 Linz / Austria

hp@cast.uni-linz.ac.at

Abstract

This paper describes the design and realization of a component-based application framework to develop Manufacturing Execution Systems (MES). Manufacturing Execution Systems (MES) are a recently defined category of industrial software for the plant floor/manufacturing environment. The overall goal has been to enable the development of MES software systems by composition and extensions of prefabricated building blocks. The framework-based development of MES applications guarantees significant reduction in development time and cost by increased software quality. The framework is grounded on two supplementing approaches – on the one side, an event- and constraint-based modelling approach is used to represent equipment states and production workflow and, on the other side, object-oriented and component-based software technology have been used as the underlying realization concepts. The framework has been realized in Microsoft's new C# programming language and .NET Framework.¹

Keywords: application frameworks, component-based software development, .NET Framework, C# language, manufacturing execution systems, workflow modelling.

1 Introduction and Motivation

Manufacturing Execution Systems (MES) are a recently defined category of industrial software for the plant floor/manufacturing environment (MESA 1994, OMG 1997). MES is considered to be a key technology to improve manufacturing performance capabilities (MESA 1994). While corporate management is naturally concerned with long-term objectives, there is a growing realization that MES can have considerable short-term impact on mission as well as performance (MESA 1994). MES close the information gap between enterprise wide resource planning-systems on the one side and lower level control systems on the other side (Figure 1).

MESs are intended to provide plant-wide insight into the production process, informing about the state of production, production performance, and emergence and

allocation of production costs to products. MES may improve better resource planning and allocation, allows to supervise the process run, and allows to promptly react to abnormal events. *Product tracking*, as the core functionality of a MES system (MESA 1997) has the main objective to accompany and supervise the manufacturing process. Based on requests from the production manager or upper *Enterprise Resource Planing* (ERP) system, the feedback information from low level *Supervisory Control and Data Acquisition* (SCADA) systems, and inputs from the user/operator it has to be in the position not only to know the current state of production and state of all products, but also to recognize abnormal or critical states in the production process.

Current MES software systems (MESA 1996, Huch 2000, Miklovic 1999) are either mainly based on traditional technology or they are often built from scratch. Most of the MES software systems on the market are characterized by inflexible, rigid and monolithic architectures (Boyle 1999). Although most systems come with a user friendly, graphical specification and configuration interface to configure the software for a particular application, they usually fail to address special features which are out of their limited scope. The result is cumbersome, long time adaptation work, which annihilates the advantages of a graphical specification.

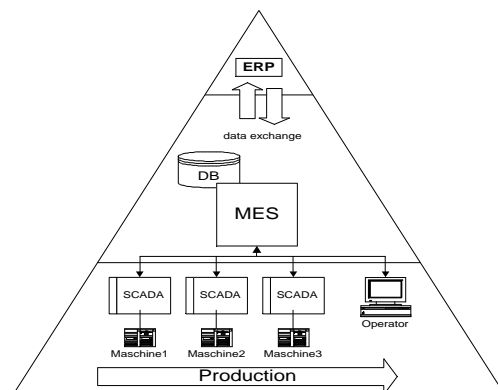


Figure 1: MES between ERP and SCADA

In the work presented, we have adopted a different approach. Instead of building a closed system, we have decided to build an open and extendable component

¹ Copyright © 2002, Australian Computer Society, Inc. This paper appeared at the 40th International Conference on Technology of Object-Oriented Languages and Systems (TOOLS Pacific 2002), Sidney, Australia. Conferences in Research and Practice in Information Technology, Vol. 10. James Noble and John Potter, Eds. Reproduction for academic, not-for profit purposes permitted provided this text is included.

framework (Fayad and Schmidt 1997, Fayad and Johnson 1999). This framework allows building MES software systems by reusing prefabricated and adaptable components. Moreover, it also facilitates specializing and extending the component framework according to special requirements.

The component framework consists of interoperable, adaptable and extendable components. The flexible design of the framework enables to shorten the software development process. Furthermore, with this component framework customer specific requirements can be realized with a minimum effort of that needed with conventional MES tools.

The framework combines two different approaches which supplement each other.

β First, we have adopted an event- and constraint-based modelling approach to represent equipment states and production workflows. This approach differs from classical workflow modelling approaches (Van der Aalst et al. 2000, Lei and Singh 1997, Dumas and Hofstede 2001) as it emphasizes events and constraints in workflows instead of sequences of work steps. With this approach it is possible to represent workflows with independent activities but with all applying restrictions and constraints.

β Second, we used the new C# and .NET object-oriented and component-based software technology (Platt 2001, Archer 2001) as the underlying implementation platform. This allowed us to realize the event- and constraint-based modelling approach in a natural way. Additionally, the component model of C# and .NET has advantageously been used to set up the component framework. The decision to use C# and .NET was a quite obvious one. Initially, we attempted a realization in COM+, which soon turned out to be too inflexible for our purposes. With the emergence of C# and .NET technology we experienced that it suites our requirements in an optimal way.

This paper emphasizes the synergism of those two approaches.

The remainder of the paper is structured as follows. In the next section we discuss the MES application domain and basic design decisions for the framework development. In Section 3 we briefly review the C# and .NET software technology which has been used in framework development. Section 4 then introduces the workflow modelling approach and its realization. Section 5 outlines the layered architecture and section 6 presents a test application. In section 7 we summarize the work and give an outlook to future developments and research directions.

2 Conceptional Background

In this section some background information about the system development is given. First, the characteristics and challenges of a MES system are outlined. From those basic design decisions are derived which have influenced the development of the framework.

2.1 The MES Domain

Recall from the introduction that the main objective of a MES software system (OMG 1997, MESA 1997) is to accompany and supervise the production process in terms of state of orders, products, and production equipment. A MES has to be able to supervise the execution of production and react to abnormal events, e.g. delays in productions steps. It has to be able to give information about the production process and state of products, as well as compute performance characteristics. A MES software system does this mainly based on feedback information from lower level control components and plant operators. In summary, a MES system has to cope with the following main tasks:

- β gather, filter, and process the information flow from the environment, which contains the SCADA, ERP, and operators
- β capture the actual production state and the state of production flow
- β supervise production process and detect any abnormalities
- β give accurate and timely information about any relevant production indices.

In building a MES system one is confronted with the following main challenges:

- β processing the multiple, heterogeneous, asynchronous event-based information flow from the environment
- β coping with the highly dynamic behaviour of the production system and represent the complex production process
- β reacting appropriately to a variety of different events occurring in the production process.

Product tracking (MESA 1997) – the functionality of tracking the state of products and production equipment – is in the centre of any MES system. From product tracking other important MES functionality can be derived, e.g. storing trace histories or computing performance indices. To accomplish product tracking, MES software has to represent:

- β the state of production equipment and state changes of equipment based on feedback information on progress in production
- β the workflows of products, i.e., the work steps required to produce a particular product, together with the alternatives and applying requirements and constraints
- β how the work steps can actually be carried out on equipment, i.e., what it means when a work step is carried out by equipment, in particular, what indicates that a work step is started and finished and what the consequences in terms of states of products are.

2.2 Basic Design Decisions

The characteristics of the MES domain has led to the following main design decisions for building the MESComponents application framework: (1) providing a set of extendable libraries of reusable and adaptable components from which MES applications can be composed, (2) basing on a multi-layered architecture (Buschmann et al. 1996) where the kernel functionality is strictly encapsulated and abstracted from the

environment, and (3) relying on event-based information flow from outside to the kernel, an event-based triggering of internal behaviour, as well as a change propagation mechanism to spread data changes from the inside kernel to the outside interfaces.

2.2.1 Component Libraries

We have strived to build components from which a wide variety of MES applications can be created. The objective is that MES software systems mainly can be composed of prefabricated building blocks, but also that components can be extended and customized easily.

The MES Component framework provides component libraries for user interactions, for interfacing with the lower level control systems through *Ole for process control* (OPC) servers, and in particular, for representing the production process. The kernel contains are components to model the production equipment and the products and their production workflows (see section 4).

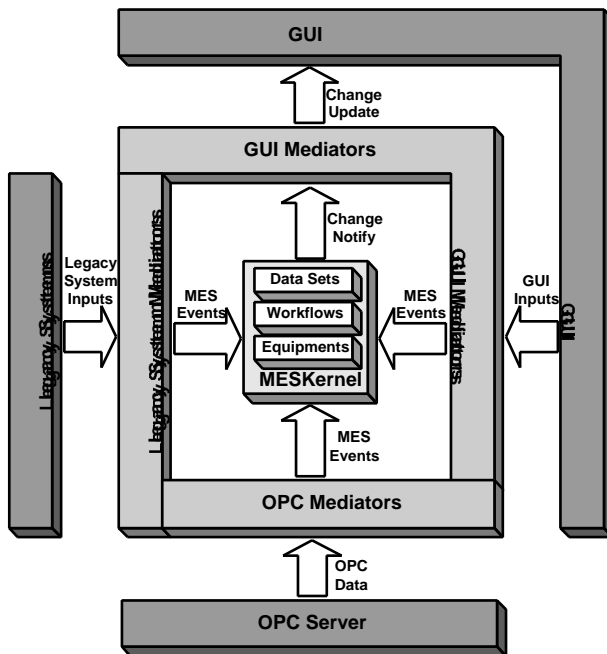


Figure 2: Multi-Layer Architecture

2.2.2 Multi-Layer Architecture

The multi-layer architecture as presented in Figure 2 identifies 3 main layers:

- β **MES kernel:** In the centre the MES kernel incorporates the core functionality of the MES system. All relevant information is combined and processed here. The kernel has to store and maintain all the information relevant for the operation of the system. Components for equipment and workflows represent the production status and production flow. Additionally, special data objects hold user relevant data (orders, processing units, processing data, etc.) in the form of memory-resident relational data tables (see Section 3, Typed DataSets).
- β **External layer:** The external layer implements the external access to the system. Different types of system interfaces are implemented based on a common abstract

specification. For example, interfaces to the control system providing access to the plant data through an OPC server are realized based on the same abstract specifications as inputs from the operator. For typical user tasks, such as viewing and editing various data tables or viewing and controlling equipment, high level, reusable and adaptable components are provided.

- β **Mediator layer:** Between the external layer and the kernel system, a mediator layer handles all input/output information flow. The mediator layer works with the common abstract specification of the external layer and, in that way, abstracts the kernel functionality from the various forms of interactions. Information flow between the internal data models and the external views are automatically kept synchronized. Additionally, this layer is responsible for decoupling the kernel and the external layers in terms of threads, i.e., it handles that the kernel functionality and input/outputs are carried out in different threads to cope with the different execution demands.

2.2.3 Event-Based Execution

The component framework works mainly event-based (unlike message-based systems, see Figure 2). Events originating from actions in the external layer are abstracted in the mediator layer and are handled in the kernel layer. Within the kernel layer, the events are first handled by the equipment components and then forwarded to the respective workflow components. We designed special event protocols for the event flow from the external layer to the mediator layer (UIEvent) and for the event flow from the external layer to the kernel layer as well as for the kernel internal event flow (MESEvent).

A publisher-subscriber mechanism (Gamma et al. 1995) is responsible to propagate data changes of data objects, which occurred during event processing. The mediator layer listens to those state changes and causes attached data views to be updated appropriately and external system parts to be informed.

3 C# and .NET Technology Reviewed

The .NET Framework (Platt 2001) and the C# language (Archer 2001) are Microsoft's new software development platform for building Windows and Web applications. The .NET technology resembles the Java technology, e.g., the C# language is similar to Java (with some extensions) and .NET programs run in the *Common Language Runtime* (CLR) which is conceptually similar to the Java virtual machine. However, currently only Windows operating systems are supported with the advantage that .NET programs usually run more efficiently on Windows than Java programs do.

C# and .NET provide a full set of features ranging from classical class libraries to build Windows applications to features supporting the development of Web services. For the MESComponents framework we mainly used the features which we will review briefly in the following.

Delegates and events

The C# language incorporates unique concepts for type-safe function pointers and events (Archer 2001). A delegate is an object oriented function pointer, i.e. function pointers can be handled like normal objects. Additionally, delegates are declared with a particular parameter interface, which means that a delegate can only refer to methods with the same parameter interface. In this way, working with delegates becomes type-safe.

Events in C# are object members and are declared as delegate types. An event allows to register handler methods. Firing an event means that all handler methods are called in sequence with the sender and an event argument object as parameter (Figure 3). The event/handler concept is similar to Java's event/listener concept with the advantage that the delegate concepts allows direct connection between the event and the called handler methods.

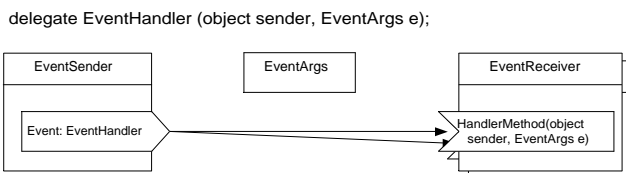


Figure 3: Event/handler example²

In the example in Figure 3, the EventSender object declares an event to be of delegate type EventHandler. The event receiver declares handler methods with the same parameter interface as the delegate type. Several handler methods can be added to the event and are called with the sender object and an EventArgs object as parameters when the event is fired.

The event/handler concept has extensively been used in our framework to realize an event-based flow of execution.

Meta-data and reflection

The .NET Framework provides full access to meta-data at runtime (MSDN 2001). Information about types is stored in the intermediate language (which is analogous to Java's Byte Code) and is available at runtime. The package System.Reflection delivered with the .NET core provides programmatic access to meta-data. It allows accessing any type information of objects, such as description of fields, properties, methods, and events: Moreover, it also allows calling methods indirectly through the meta layer.

We extensively used the meta-data layer and the System.Reflection package to make components adaptable and for component composition. For example,

² For graphical representation of events and handler methods we have adopted the notion known from SDL (Jacobson 1992) for outgoing and incoming events, respectively.

the meta layer allowed us to specify connections between events and event handlers in a declarative way.

Component model

C# and .NET support software component development (Burke 2000). The C# language provides special language concepts to define properties and events of components. The *attribute* concept allows the programmer to define custom information about classes and their members. The information about properties, events, and other members of components as well as the attributes are analyzed by the system and are used for design-time and run-time presentation and adaptation of components.

The System.ComponentModel namespace provides classes that are used to implement the run-time and design-time behaviour of components. The System.ComponentModel.Design namespace contains classes that can be used to design custom support for components at design time and access the services provided.

We have developed the components within the MESComponents framework according to the .NET component model and have used and extended GUI controls from the System.ComponentModel.Design namespace to realize adaptable GUI components.

Data sets and typed data sets

The .NET Framework contains an extension of the ADO technology (Esposito 2001) to provide access to external data sources, like databases or XML files, in a disconnected way. The ADO.NET architecture enables building components that efficiently manage data from multiple data sources. Furthermore it provides the tools to request, update, and reconcile data in multiple tier systems.

The ADO.NET architecture encompasses two distinct groups of components: *content components* and *managed-provider components* (Thai and Lam 2001). The content components include the DataSet, DataTable, DataRelation, and other supporting components which contain the actual data content in the form of a memory resistant data base. The managed-provider components on the other side assist in data retrieval and update from external data stores.

We have used ADO.NET to define and store all user data, with the advantage that connections to external data stores are supported. Additionally, the .NET development environment provides convenient design tools to graphically define data tables and to generate C# code from the graphical specifications (called *Typed DataSets*).

4 The MES-Kernel

This section describes the core-functionality of the kernel layer of the framework. This layer provides general components which implement MES functionality as generic as possible to guarantee openness, flexibility, and adaptability for a wide range of MES application systems.

The kernel system works event-based. For the realization of the kernel functionality one single event type has been defined – the `MESEvent`. The `MESEvent` type is intended to signal any relevant events for the MES system. `MESEvents` are propagated to trigger the flow of execution. Figure 4 illustrates the `MESEvent` protocol. The delegate type `MESEventHandler` is declared with two method parameters – the sender which is of type object and the event argument object of type `MESEventArgs` holding event data. The `MESEventArgs` object holds a name for the event and an event value of base type object, i.e., any value can be distributed with a `MESEvent`.

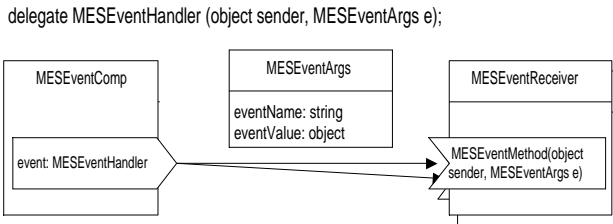


Figure 4: MESEvent protocol

The following main component types are distinguished: *equipment components*, *workflow components*, and *mapping definitions*.

Equipment components

Equipment components model the various types of production equipment. The main objective is to represent the current state of the equipment based on the feedback information from the environment. Reactions to expected and unexpected events originating from the external environment are codified in the components. Moreover the equipment components themselves signal events, which indicate relevant happenings within their scope.

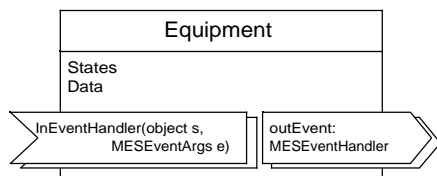


Figure 5: Model of equipment

In Figure 5 a model of a production equipment component is depicted. It contains the following elements:

- β **States:** States are used to represent the current status of equipment, e.g., *running* or *idle*.
- β **Data:** Different data fields may store information on equipment, like products, material and other resources residing in the equipment, the work steps currently running, etc.
- β **Handlers for incoming events:** Handler methods are intended to handle incoming events and perform the appropriate state and data changes. Also, the handler methods will fire outgoing events (see below).
- β **Outgoing events:** Equipment components declare several events which signal relevant changes in the equipment to other parts of the MES system.

Equipment components for larger production units can be composed of smaller equipment components.

Workflows

Workflows are associated with the different types of products and are used to specify how the production of products has to occur. Workflow modelling is first of all based on the concept of a *workflow event*, which is any event relevant for the execution of the workflow. Together with an event the reaction to this event is codified, i.e., changes in status of the workflow itself, but also changes to the product, used resources, and materials.

A set of basic workflow components are available from which specialized workflow components can be derived and from which more complex workflows can be composed. The most basic workflow component is the `WorkStep` (Figure 6), which represents an elementary work to be carried out on a product. A `WorkStep` has a start and a finished event. Analogous to equipment components, workflow components have their state, data, and handlers for incoming events and event declarations to signal own events.

Workflows mainly differ from equipment components in the way complex workflows are built. Complex workflows are composed of simpler workflows by gluing them together through *relation* and *constraint components*. Relations allow representing sequences and alternative courses of work steps. However, instead of a sequence of work steps as in classical workflow modelling (Van der Aalst et al. 2000, Lei and Singh 1997, Dumas and Hofstede 2001) we emphasize the notion of constraints. A constraint can be any condition stated on events and attributes of a workflow. For example a *duration constraint* may state that the occurrence of an event is constrained to lie within a predetermined interval. Another important type of constraint are constraints which guarantee the availability of resources before the work step is started.

Constraint components are attached to events of workflow components. Whenever an event occurs, the constraint checks if all the conditions are met for that event occurrence. Or a constraint may expect an event to occur within a certain time interval and will react when the event becomes past-due. The constraint component will then fire a violation event which allows other parts of the system to react accordingly.

Figure 6 illustrates an elementary work step as an essential building block in a workflow. It has two incoming events, the *start* event and the *stop* event;; they are considered to occur when the work step starts and when it is finished, respectively. It defines a material to work on and a product, which is the result of the work step, and it defines the required resource. Constraints define that the start event can only occur when the material and the resource are available. Furthermore, a time constraint is defined which states that the stop event has to occur in a certain period after the start event, i.e., the work step should not last indefinitely long.

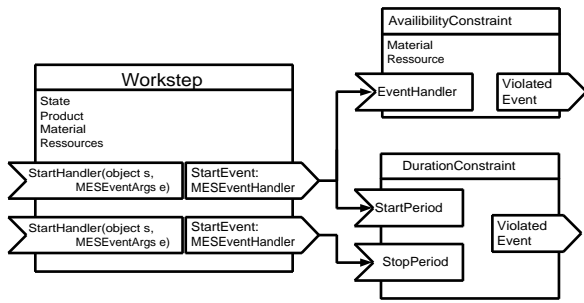


Figure 6: Work step with availability and duration constraint

Mapping definitions

Mapping definitions are the concept, which glues together the equipment and the workflows. Beside the equipment and workflow definitions, one has to define how a workflow is carried out on equipment. This mapping definition is essential as it has to associate the material, products, resources, and events in workflows and equipment with each other. A mapping defines

- β which events from the equipment components have to be mapped at what time to which events in the workflow; this is of special importance since it defines which events signalled in the equipment components are relevant for which workflows and should be handled by which handler methods in the workflows,
- β which resources and materials used in production and defined in the workflows correspond to which resources and materials available at the equipment.

Figure 7 illustrates mappings of a simple work step to a machine. It is defined that

- β products, materials and resources at the machine are processed by the work step,
- β the start event and the stop event signalled by the machine are associated with the incoming start and stop events of the work step

Instantiations of mappings occur during the manufacturing process as reactions to events and are done in the event handler methods of equipment and workflow components. Mappings are highly dynamical and lie within the heart of the dynamics of a MES system. In particular, through the mapping of the events from the equipment to the events in the workflows, events occurring in the manufacturing process get forwarded to the workflows and allow the workflows to progress.

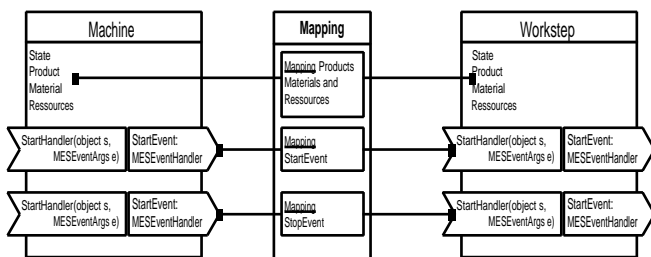


Figure 7: Mappings of workflow to equipment

In the current version of the MESComponents framework, a central mapping component is responsible for event propagation within the kernel system. The

central mapping component allows the instantiation and removal of event mappings, handles forwarding of events to handlers, and allows defining priorities and resolving conflicts between handlers when mappings from one event to several handlers exist concurrently.

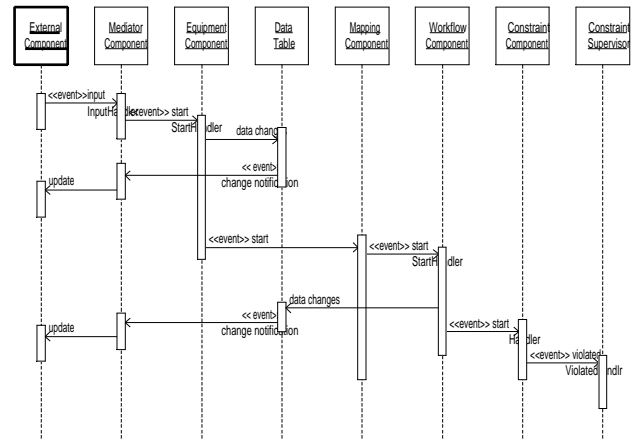


Figure 8: Event flow example

4.1 Event-Based Flow of Execution

In the following we will show the MES dynamic process in some detail. With the various types of components in hand, the MES system works as follows (see Figure 8):

- β An event originates from the external layer and is forwarded as MESEvent to the kernel. There, it is received by an equipment component and the respective handler method is called. In the handler method the state and some data are changed which results in a propagation of the changes through the mediator layer back to the external components. The event then is eventually forwarded as an outgoing event of the equipment component.
- β When a workflow component is currently mapped to this equipment component (through the central mapping component), the event is propagated to the workflow component where the handler method of the workflow is called (e.g. a StartHandler of a work step). As a reaction the workflow component eventually changes its state and the states of product, used materials and resources. Again those changes are propagated back to the external layer. Also, other (e.g. subordinate) workflows may be created, or workflows may be mapped to or removed from equipment. Then eventually one of the components events will be fired.
- β When a constraint is attached to the outgoing event, the constraint is activated and checked. If the constraint is recognized to be violated, the constraint itself fires a violation event. In the current version of the MESComponents framework all the constraint violation events are received by a constraint supervisor module, which is the unique authority responsible for handling constraint violations (in the current implementation of the MES framework, a message is sent to the operator informing him of the constraint violation).

5 The Layered Architecture

This chapter describes the layered architecture of the MESComponents framework. The objectives of the layered architecture are as follows: (1) strictly encapsulating and abstracting the kernel from the environment, (2) supporting various forms of user interactions, (3) providing high level, configurable components for user interaction and interfacing with external systems.

The aforementioned goals have been tackled with the following approaches:

- β Abstract specifications of basic functionality define how to interact with the system in a very general way and provide specifications for the development of different interface components.
- β A set of components are provided which are realized based on the abstract interface specifications and provide basic, high level functionality often needed, e.g., selection of an object out of a list of objects, editing an data table, interfacing to an OPC server, etc.
- β The mediator layer and the set of components for this layer are used to handle the interactions between the external layer and the kernel.

In the following, we first describe the interface specifications and components of the external layer. Then we show how the mediator components handle the interaction between external and kernel layer components.

5.1 The External Layer

To guarantee well-defined interactions between external components and mediator layer, we have defined a set of elementary interface specifications, which define very general forms of functionality found in the external layer. The following interfaces have been defined:

- β `IObjectInput`: is the basic interface for input of an object value; it defines an event (`ObjectInputEvent`), which is intended to deliver an input value.
- β `IObjectOutput`: is the basic interface to output and represent any value on the GUI; defines a method `Output(object outputValue)` to be called with a new value for output.
- β `IObjectSelection`: is the interface allowing the selection of an object from a collection of objects; defines a method to set the collection of selectable objects and an event (`IObjectSelectionEvent`), which is fired when a selection has occurred.
- β `IObjectListRepresentation`: is the interface allowing the representation and manipulation of collections of objects; it has methods to initialise the set of objects and to create, to add, and to remove objects; derives from `IObjectSelection` and hence also owns the selection event.
- β `IObjectEditor`: is the interface enabling the editing of objects; it defines a method to set the object to be edited and an event (`IEditedEvent`) to be fired when the object has been changed.

Based on the aforementioned interface specifications, a set of adaptable components for high-level, recurring

interaction tasks have been realized. Examples of such components are:

- β `OPCEventInput`: is a component to interface with the OPC server; it implements the `IObjectInput` interface; it is configured by specifying the OPC item (measurement from the plant) to listen to and an event condition (e.g., a threshold is met) when to fire the input event; it fires an `IObjectInputEvent` whenever the event condition occurs with the actual OPC item value as input value.
- β `GUIEventInput`: is a component to allow an `IObjectInputEvent` with any value to be put in by the operator from the GUI; it is configured by a standard GUI control, eg. a text field or a button.
- β `GUIStateControl`: it implements the `IObjectOutput` interface and is used to graphically show the state of equipment; it is configured by a mapping of state values to images.
- β `GUIEditableListControl`: it implements the `IObjectListRepresentation` and the `IObjectEditor` interfaces; it is used to represent a list of objects in the form of a grid view and to edit them; it is configured by setting the list of objects, the properties (columns) to show and the allowed fields to edit; it uses the .NET component editors to allow representation and convenient editing of objects of arbitrary type.

5.2 The Mediator Layer

The components of the mediator layer are based on the abstract interface specifications of the external components. They handle the interaction between external layer components and internal data objects. Additionally, the mediator layer decouples the external from the internal layer to run in different execution threads; the advantage is that the internal functionality is not slowed down by input/output. Moreover, the mediator layer is the point where remote interaction with external system parts, running on remote clients, can occur.

The mediator components are configured by setting their respective peer objects in the external and kernel layer and also can be adapted by declaratively defining ways to filter and transform information. Configuration is done through the meta-data layer by defining properties of objects to retrieve information from and events to listen to. By relying on the abstract interface specifications in the external layer, it has been sufficient to provide one mediator component for one interface specification or a unique combination of interface specifications. In the following some of the mediator components are outlined shortly:

- β `MEDObjectInput`: is an elementary, simple mediator component which works with the `IObjectInput` interface; it listens to the `IObjectInputEvent` fired by the `IObjectInput` component and forwards this as a `MESEvent` to be processed in the kernel.
- β `MEDObjectOutput`: it works with the `IObjectOutput` interface and automatically maintains the connection between a property of an internal data object and its external representation; it is configured by

setting the property to listen to and the GUI component to visualize the property.

MEMListRepresentation: it works with the IObjectListRepresentation interface and maintains the connection between a collection (data table) of objects in the kernel and its external representation; it works with the RowChanged events from the data tables; it is configured by setting the collection of objects to represent and the external GUI component; additional filter mechanisms and presentation options can be defined.

MEEditableView: it relies on the IObjectListRepresentation and the IObjectEditor interface; it has the same functions as the MEDListRepresentation component and, additionally, handles the IEditedEvent and updates internal data objects.

Figure 9 shows the connection of a GUIEditableListView to a DataTable object through a MEEditableView component. The mediator component is parameterised with a RowFilter, which defines which rows of the table are actually presented in the view, and a PropertyFilter, which define the columns shown.

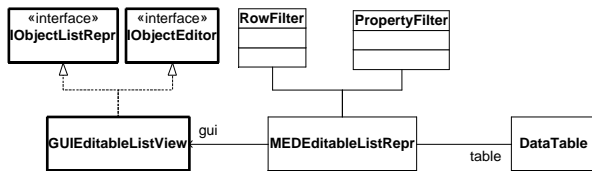


Figure 9 Mediator connection between data table and view component

6 Building a Test Application

The MESComponents application framework has been tested by building a MES software system for a copper milling plant. The test application has been selected from an existing MES system, which has been built with conventional means. The most challenging part has been extracted from the real system. This part proved difficult (or even impossible) to be realized in the existing system. For example, one requirement was that multiple coils can be processed simultaneously in the plant, i.e., that one coil is wound up at the end of the line while the next coil already is wound off and started to be mill cut. The numerous and different exceptions and appropriately reacting to them has been another challenging part.

Figure 10 shows the user interface of the plant operator where he is supported in supervising the production process. The plant winds up coils at the down-swift, mill cuts the coils by a milling cutter, then cuts the coils into pieces by a cutter component, and finally up-winds the cut coil pieces alternatively at two up-swifts.

Building an application based on the framework comprises the following steps: (1) extending the basic components for equipment and workflows from the library to deal with application specific data, (2) building a model of the plant using the equipment components, (3)

implementing the dynamics of the workflows using the workflow components, and (4) configuring the external interface layer and the mediator layer and connecting them to the plant components. In the following paragraphs, we outline how the copper milling plant application has been built in this way.

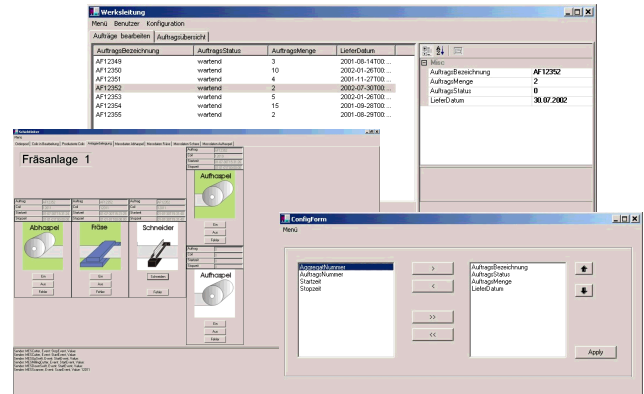


Figure 10: Screenshot from the copper milling plant application

Based on the elementary Machine component, which realizes start, stop, and error event behaviour, a special component CopperMachine has been derived which updates the user specific data on event occurrences. Similarly, a CoilWorkStep has been derived from basic WorkStep to set user specific data at work step events. Furthermore, the mapping of a CoilWorkStep to a CopperMachine has been defined which establishes the correspondence between the coil data of the work step and of the machine. All those extensions where straight-forward.

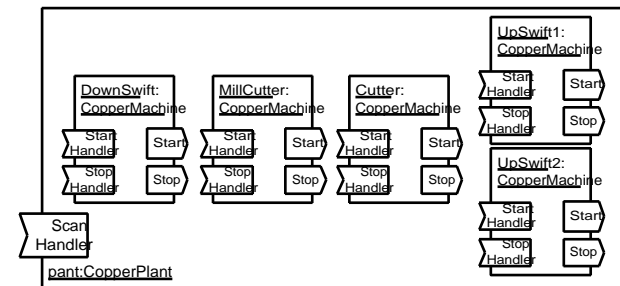


Figure 11 Model of the plant

A model of the whole plant CopperPlant has been implemented by instantiating CopperMachine components as depicted in Figure 11. Additionally, the CopperPlant defines its own handler method ScanHandler for a scan event which is the input event signalling the arrival of a new coil at the plant.

Workflows have been defined for order execution (OrderWorkflow) and for processing one coil (CoilWorkflow). As an order defines a set of coils to process, the OrderWorkflow mainly consists of CoilWorkflow components. Additionally, an OrderWorkflow implements event handlers to listen to the start and finished events of its subordinate

CoilWorkflows and fires a finished event when it recognizes that all its coils have been processed.

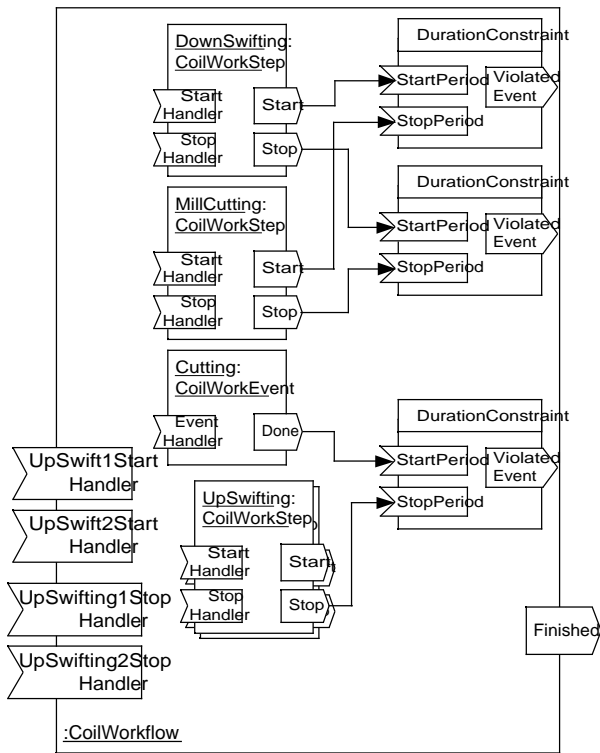


Figure 12: Coil Workflow

The most crucial part of the realization of the application example has been the definition of the CoilWorkflow. (This also required the biggest amount of source code programming, although the biggest effort was to really understand how the workflow actually evolves). The CoilWorkflow has a structure which mainly parallels that of the plant, i.e., for the machines corresponding work steps exist (Figure 12). However, the UpSwifting work steps are created dynamically upon each start event from the UpSwift. Therefore, the CoilWorkflow has event handlers to react to the start events of the UpSwifts which are used to create new UpSwifting work steps each time a new start event from an UpSwift is received. It also has event handlers to listen to the stop events of its UpSwifting work steps which are used to check when all coils are wound up and the whole workflow is finished. Additionally, a set of constraints are attached to the work step events to supervise timing constraints, e.g. that the stop event in the UpSwifting work step has to come some time after the event from the Cutting work step (an important constraint to monitor as the stop event at the UpSwift has to be provided by the operator).

The last step in the realization of the MES system is configuring the external interface layer and the mediator layer and connecting them to the plant components. This was straight forward to realize. For presentation and editing of the user data, the GUI and mediator components have been used and connected to the internal data tables as outlined in Sections 5.1 and 5.2.

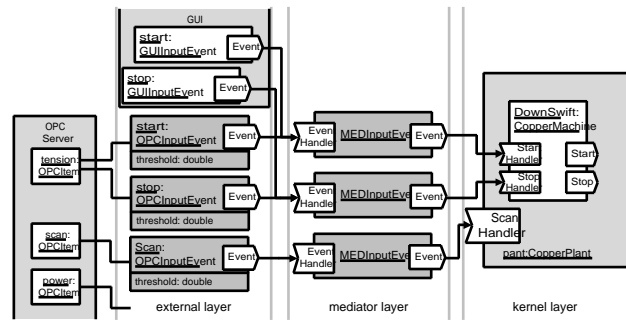


Figure 13 Connecting the plant to the lower level control system

The connection of the plant model to the lower level control system has been implemented as depicted in Figure 13. The interface to the OPC server has been built by using **OPCEventInput** components for the start and stop events for all machines in the plant. Those were parameterised by defining the OPC items to listen to and the condition when to fire an event; e.g. for the DownSwift the item to listen to is the swift tension and condition is that the tension exceeds a certain threshold. Mediator components **MEDObjectInput** are then used to establish the connection from the **OPCEventInput** components to the handlers in the Machine components. Additionally, start and stop events for machines can be triggered by the operator from the GUI (in case when the OPC system does not work properly). **GUIInputEvent** components are used at the operator interface and the events from those components are connected to the mediator in the same way as the events from the OPC.

7 Summary and Outlook

In this paper we have presented a component-based application framework for building MES software systems. The objective of the framework is to facilitate building MES application systems mainly by composition from prefabricated building blocks. The framework is grounded on a layered architecture and provides a set of reusable, extendable, and adaptable components. In the focus is a scheme and a component library to model plant equipment and production workflows. The system is open in respect to extensions and adaptations to meet special requirements. By a restricted but complex application example we were able to show that the framework allows building of advanced MES software systems and that the development effort can be reduced significantly. The open nature of the framework allowed us to implement the complex workflows on the source code level. In this way it was feasible to realize all the special requirements.

The framework has been realized in Microsoft's new .NET technology and the C# language. Many advanced features have successfully been used. For instance, the delegate/event concept has been used to realize the event-based execution model in the framework, the meta-data layer and the **System.Reflection** namespace has been used extensively to realize adaptability and composition of components, and the framework components have realized according to the component model of .NET. Additionally, the ADO.NET framework

for memory-resistant data tables has been employed to model any user specific data. Although, we worked with the Beta 1 version of the .NET Framework, the technology showed to be surprisingly stable and mature and we did not encounter any major problems (except the partial or completely missing documentation).

Working with the MESComponents framework has shown one major difficulty which is known as a general shortcoming of application frameworks: building applications systems out of the framework requires a lot of expertise, experience, and detailed knowledge of the framework (Fayad, Schmidt and Johnson 1999). To improve the framework in that respect, we pursue a combination of the framework approach with a generative approach (Czarnecki and Eisenecker 2000). An application-specific, high-level description language should be provided to allow the specification of application-specific parts, most notable, the workflows and equipment models. This would bring the framework close to commercial MES software systems, which usually provide graphical means for describing the application, but without giving up the openness of the framework approach.

8 References

- ARCHER, T. (2001): *Inside C#*. MS Press.
- BOYLE, W., (1999): Distributed manufacturing execution framework. In *Domain-specific Application Frameworks*. 121-138. FAYAD M. and JOHNSON R. (eds.). John Wiley.
- BURKE, S. (2000): *Creating Designable Components for Microsoft Visual Studio .NET Designers*. msdn.microsoft.com/library/, Microsoft Corporation.
- BUSCHMANN, F., MEUNIER, R., ROHNERT, H., SOMMERLAD, P. and STAL, M. (1996): *Pattern Oriented Software Architecture – A System of Patterns*. John Wiley.
- CZARNECKI, K. and EISENECKER, U. W. (2000): *Generative Programming*. Addison Wesley.
- DUMAS, M. and TER HOFSTEDÉ, A. H. M. (2001): UML Activity Diagrams as a Workflow Specification Language. In *Lecture Notes in Computer Science* 2185:76-82. Springer-Verlag.
- ESPOSITO, D. (2001): *ADO .NET for the ADO Programmer*. MSDN Online Library, Microsoft Corporation, msdn.microsoft.com/library
- FAYAD, M. and SCHMIDT, D. (1997): Object Oriented Application Frameworks. *Communications of the ACM*, 40: 32-38, ACM Press.
- FAYAD, M., SCHMIDT, D. and JOHNSON, R.E. (1999): *Building Application Frameworks*. John Wiley.
- FAYAD, M. and JOHNSON, R. (1999): *Domain-specific Application Frameworks*. John Wiley.
- GAMMA, E., HELM, R., JOHNSON, R. and VLISSIDES, J. (1995): *Design Patterns*. Addison Wesley.
- HUCH, B. (2000): *Analysis and Optimization of processes in manufacturing through manufacturing execution systems*. Diploma thesis (in German). Technical University Graz, Austria.
- JACOBSON, I. (1992), *Object-Oriented Software Engineering*. Addison-Wesley.
- LEI, Y. and SINGH, M. P. (1997): A Comparison of Workflow Metamodels. In *Proceedings of the ER-97 Workshop on Behavioral Modeling and Design Transformations: Issues and Opportunities in Conceptual Modeling*, Los Angeles.
- MESA (1994): *The Benefits of MES: A Report from the Field*. MESA International, www.mesa.org.
- MESA (1996): *MES Software Evaluation / Selection*. MESA International, www.mesa.org.
- MESA (1997): *MES Explained: A High Level Vision*. MESA International, www.mesa.org.
- MIKLOVIC, H. (1999): MES Market: Spectacular but short lived. *Monthly Research Review*, Gartner Group.
- MSDN (2001): *Metadata and Self-Describing Components*. MSDN Online Library, Microsoft Corporation, msdn.microsoft.com/library.
- OMG (1997): *Manufacturing Domain Task Force RFI-3 Manufacturing Execution Systems (MES)*. Document number mfg/97-11-01. Object Management Group, Framingham, MA.
- PLATT, D. S. (2001): *Introducing Microsoft® .NET*, MS Press.
- THAI, T. and LAM, H. Q. (2001): *.NET Framework Essentials*. O'Reilly.
- VAN DER AALST, W. M. P., BARROS, A. P., TER HOFSTEDÉ, A. H. M. and KIEPUSZEWSKI, B. (2000): Advanced workflow patterns. In *Proc. of 7th International Conference CoopIS*, Eilat, Israel. LNCS 1901:18-29, Springer-Verlag.