Mesh: An Object-Oriented Framework for Modeling of Process Control and Management in Distributed Industrial Environments

Dragan Milicev
University of Belgrade
Faculty of Electrical Engineering, Department of Computer Science
Belgrade, Serbia, Yugoslavia
E-mail: emiliced@etf.bg.ac.yu

Abstract: *Complex industrial systems are characterized by strong co-operation of two enterprise levels: production and management. This includes intensive flow of materials and documents throughout the organization. We present a framework called Mesh for modeling these aspects. In Mesh, a system is modeled by a hierarchical structure of organizational units that are interconnected by ports and channels. Ports and channels serve as media for controlled transport of documents and materials through the system. The framework leaves all the implementation details for process control and document contents to the specialized applications, SCADA and DBMS. This way, the model remains open to the traditional approaches and information systems that may efficiently support unique needs of the enterprise. The framework enables a smooth transition from requirement specifications to the executable model. It also supports all the basic object-oriented principles.*

Keywords: manufacturing execution systems, process control, industrial information systems, workflow management, port-based objects, modeling tools

1 Introduction

Complex industrial systems have always been one of the most demanding users of various software products. Apart from posing some other types of rigorous demands, such as reliability, response time, distributed operation, broad range of users' skills and qualifications, etc., these systems are characterized by co-existence and co-operation of two components. On one hand, they always involve production processes that should be supervised and controlled. On the other hand, the processes are supported and driven by several levels of management. Like in other business systems, this assumes an intensive flow of information throughout the organization. In order to cope with the complexity, a successful software system should provide a strong integration of these two components. Here we propose a framework for modeling such systems.

As an example used throughout this paper, we will present a part of an oil refinery enterprise responsible for oil blending. The ideas for the framework originate from developing of integrated process control and information systems for this enterprise. It has also been the first application and evaluation of the proposed approach.

2 Problem Statement: The Complex Requirements

The problem here is how to specify and develop a software system that can successfully cope with the complexity of a large industrial organization, i.e., that will automate the production process, the management process, and their interaction. We will elaborate the problem by examining the characteristics and requirements of a typical industrial system (Figure 1).

First, the system involves some industrial production processes (Figure 1a) that should be automatically supervised and controlled. This assumes existence of various hardware sensors and actuators connected to the industrial devices. Besides, this assumes complex software devices, algorithms, and modules to support the control. In our example, the process is oil blending. It includes flow of several components (base oils and additives), from specific tanks through pipelines to the mixer (blender), and from the mixer to the destination tanks. The flow is conducted by pumps. There are a lot of measuring devices used to supervise and control the process: flow counters, thermometers, level-measuring devices, heaters, and pump switches.

Second, there is a need for complex data organization and operations (Figure 1b). The management in our example uses numerous standard as well as specific technology and business procedures: recipe management, quality assurance, inventory, accounting, acquisition, etc.

Third, the two levels are tightly coupled and interact in both directions. In our example, product quality analysis is performed several times during the blending process. The blending process itself is not started before the manager has ordered it (which includes a complex preparation procedure). The stocking and selling management processes rely directly on the results of the production process: the quantity of the fluids in tanks depends on the actual flow in the process. The same holds in the opposite direction: the production process can be performed if the quantity of components is sufficient; if not, orders for acquisition are raised.

Fourth, the enterprise is organized hierarchically (Figure 1c). The hierarchy defines both the structural (derived from the process) and role-based (derived from the managerial organization) decomposition of the enterprise. Figure 1c depicts the hierarchical organization in our example, along with the interconnections of the units.

Fifth, there is an intensive flow of *documents* and *materials* throughout the organization. The whole production process is initiated and conducted by documents. The flow of materials is also initiated by documents. The dynamics of the documents and materials flow can be specified in most cases by *scenarios* (Figure 1d). For example, the blending process is initiated by a *production order*. Prior to sending the order to the operator, the manager (technologist) prepares the order: he/she specifies the components, the recipe, the quantity, etc. Upon the reception of the document, the operator acknowledges the order, prepares the devices, and starts the mixer. When a part of the product has been prepared, the process is stopped, and the manager creates an *order for quality analysis* and sends it to the laboratory, along with a product sample. The laboratory performs the analysis and returns the report to the manager. The manager corrects the recipe and restarts the process. (It should be mentioned that every step of this scenario, and the scenario itself, are much more complicated in reality, but have been simplified here for brevity. We wish to emphasize the dynamic nature of the system, and the complexity of the document and material flow.)

Finally, the system is by its nature concurrent and distributed. Many users can initiate activities concurrently, and those activities might need mutual exclusion and synchronization.

3 Overview of Existing Solutions

Traditional SCADA (Supervisory Control and Data Acquisition) systems support well the process-control level of automation. Using SCADA development packages, users can design complex animated panels for process supervision and control [1] as in Figure 1a. The panels consist of graphical objects that represent devices from the real process. Graphical attributes of the objects (e.g., color, filling, blinking, size, position, etc.) can be directed by the signals from the sensors in the process. Besides, users' actions on the graphical objects (e.g., moving, clicking, resizing, etc.) can be forwarded to the actuators in the process. Using scripting languages, users can implement control algorithms. Therefore, SCADA systems are the most appropriate solution to our first requirement. However, they do not provide adequate or any support for most of the other requirements.

On the other side, traditional information systems, based on (most often relational) database management systems (DBMS), can support very complex and often very specific needs for data processing of each organization (Figure 1b). They can be developed by one of the well-known and approved methodologies [2, 3] that offer to user and developer a considerable prospect for success.

However, they weakly (or not at all) meet the other requirements. Besides, even when developed by methodologies that use concepts that directly represent objects and processes from the real world [2, 3], they lead to an inappropriate executable model. Namely, that model is organized simply as a set of operations, performed through user-interface forms, over a shared database, as in Figure 1b. This presentational gap between the specification and the executable model is certainly unattractive to the users. Moreover, even when the methodology is formal, it rarely, if ever, produces smoothly the executable model (code), but the code has to be written by traditional programming approaches [3]. The importance of a smooth and direct transition from the design to the executable model, preferably without presentational differences, has been recognized for long, and methodologies that support such a transition exist in some other domains [4].

Manufacturing execution systems (MES) [5] provide a good framework for process control management and integration with SCADA systems. However, they do not support higher-level business modeling and conceptual integration with traditional information systems. They also do not usually provide a means for hierarchical modeling of the enterprise. On the other side, workflow management systems [6, 7] provide a good conceptual model for modeling business processes. They use concepts such as operations and roles to model the processes, while each operation may be performed either manually (with interaction to the user) or automatically (by enacting an appropriate application outside the workflow management system).

Although our approach belongs to this group, it is has several specific features. First, it allows hierarchical modeling of the organization. Second, it uses concepts that allow incorporation of contemporary object-oriented workflow approaches [8], most notably the interaction and activity behavioral specifications into the structural model. The approach is formal in a way that supports automatic generation of the executable model from structural and behavioral specifications, as it will be described later. Finally, it is particularly dedicated to industrial systems.

As a conclusion, we need an approach that will offer solutions to all our requirements. To do so, it must allow simple incorporation of SCADA systems, because they are best suited for the process-control level. It must also be open to traditional information systems (based on DBMSs) because DBMSs represent a well-studied and mature technology, and provide the best way to support all complex specialties of business that every organization has. Moreover, the approach must provide a good integration and interaction of the two. It should be formal and sufficiently abstract to provide concepts that can be reused in modeling different industrial organizations characterized by the stated requirements. As already stated, it should also support a smooth transition from the design to the executable model, without presentational changes.

4 Mesh: An Integral Approach

We propose a framework that tries to meet all the requirements from our analysis. The framework is called Mesh (Manufacturing Execution Support Hierarchy). Its notation and some basic concepts are inspired by the ROOM method [4] for real-time systems. The framework should be supported by a tool that enables development and execution of Mesh models.

4.1 Concepts and Features

In Mesh, the enterprise is modeled by a hierarchy of *units* (depicted as labeled rectangles in Figure 2a). A unit may represent an operation of the process, or any other organizational (logical or physical) unit. The hierarchy is defined by aggregation: a unit may contain an arbitrary number of other units, and each unit (except for the top one) is a part of another unit.

Hierarchical Mesh model is represented in the same way when specified and executed. In the development phase, the developer specifies the hierarchy. In the execution phase, the Mesh runtime tool allows hierarchical browsing of units (going up and down the hierarchy), may hide or show nested units, and controls access of users to units. In the current implementation, units may not be created dynamically at runtime.

Each unit has its interface. The interface consists of *ports* (depicted as small rectangles on borders of units in Figure 2a). A port may be either input (filled) or output (hollow). Two ports of different units may be connected by a *channel*. Ports and channels serve to transport different kinds of documents and materials. A port may be either for documents or for materials.

A document represents to Mesh a piece of information that can flow through the system. Mesh recognizes a *document type* as a template for *document instances*. Document types are defined in the development phase; document instances are created and changed at runtime. For Mesh, a document may have arbitrary contents. Mesh is not aware of documents' contents. It only takes care of document types, some common attributes of documents (e.g., time and unit of creation), and its flow through the system (e.g., at which unit the document is currently located, which ports it passed and at what time, etc.). The content (implementation) of a document is completely left to the traditional information system (actually, to DBMS).

Materials represent substances that physically exist in the enterprise. As with documents, Mesh recognizes *material types* and *material amounts*. Material types represent substances, parts, semi-products, or products, and are defined at development time. There are (for now) the following relations between material types: a material type may be a *subtype* of another type, a material type may be a *part* of another type, or may be a *substitute* of another type. Material amounts are defined at runtime, and may be transported between units (through ports and channels). A material amount is characterized by its type and quantity (in kg, lit, or other units).

Users can create and then send a document instance through a document output port if the document type of that instance is assigned to that port by the developer. This way, Mesh controls consistency of the system at runtime. When a document instance is sent through the output port, it travels through the channel and appears in the unit that contains the input port connected to that channel. The documents that are currently held by a unit are organized in folders by different criteria (e.g., origin, type, time of creation, etc.). We do not bind the capacity of the unit: any document sent to a unit will be accepted, queued, and processed manually, when the user decides, or automatically, when the destination unit becomes available. In future extensions, we will investigate possibilities for rejecting a document reception, and for providing alternative actions in the sender in that case.

Material amounts are also transported through material ports and channels (with types of materials checked). *Automatic transport* is controlled by the tool: it means that a user initiates the transport and that the runtime tool performs the specified actions. These actions cause (by means described in the next subsection) actual physical transportation of material. *Manual transport* assumes that users (operators) of the source and destination units exchange messages (as kinds of documents) that acknowledge the material move that is by no means controlled by the tool.

Ports may be connected by channels only if they are both either for documents or for materials, they are of complementary direction (input-output, unless one of the ports is a relay port, which means it is connected to a port of a nested unit), and the intersection of their sets of document/material types is not empty. The connection of ports may be done either at development time or at runtime.

Each *event* raised at runtime by Mesh (e.g., a document instance of a certain document type passes through a certain port, a document instance of a certain type arrives at a certain unit, etc.) can cause an *action* defined by the developer. An action represents a part of executable code written in the supporting programming or scripting language. An action can invoke all the operations supported by Mesh (creating, sending, deleting, editing a document, etc.). A concrete execution of an action represents an *activity*. Each activity has its own thread of control and is connected to the event that has caused it.

4.2 Connections to SCADA and DBMS

Mesh tends to provide an abstract hierarchical model of a complex industrial enterprise and to enable and supervise documents and materials flow. All the implementation details for units and documents Mesh leaves to the specialized systems, SCADA and DBMS.

Each unit may have its own SCADA and/or DBMS implementations (Figure 2b). The SCADA implementation is a process-level view to the unit. It is completely provided by the SCADA application, and is represented by an animation panel that controls a specific process. The panel is shown to the user as a pop-up window when the user invokes the operation "View SCADA implementation" for the unit. Similarly, the DBMS implementation is provided by the DBMS application and is represented by a specified form that is displayed when the user invokes the operation "View DBMS implementation." This way, the user has an impression of a hierarchical organization of the enterprise (provided by Mesh), with the access control to the units (also provided by Mesh), and with the combined process and/or business level details that can be viewed for the units (provided by SCADA and DBMS).

Another interaction of Mesh with SCADA and DBMS is through operations of actions. Mesh can send data to or receive data from SCADA and DBMS within an action. One such interaction is used in automatic material transport. When a material amount is sent automatically from a source to a destination port, Mesh executes an operation that sends data to the SCADA application. The SCADA application is then responsible for performing automatic control of physical material transport. SCADA keeps generating a conditional signal that represents the status of the transportation process. Mesh examines the signal and when the transport is complete, Mesh assumes that the material amount has been moved to the destination unit.

Finally, the implementation of documents is again left to DBMS. When the user wants to view or edit the details of a document instance, Mesh invokes a form, controlled by DBMS, that shows the specific document instance. Documents may be arbitrarily complex, may contain arbitrary attributes from possibly several tables of a relational database, or may even contain embedded objects, if supported by DBMS. Mesh itself represents document instances as records in a single table, with the attributes common to all the documents, and with a unique document ID. This ID is used to parameterize the form that is invoked when the document is edited. The form is specified for the document type.

4.3 Object-Oriented Principles

Mesh supports several basic object-oriented principles. First, the principle of types and multiple instances is supported by differentiating *types* and *instances* of documents, materials, as well as of units and ports. A *port type* is defined by its purpose (for documents or for materials), and by the set of document/material types that can be transported through instances of that type. Each port instance has all the properties of its type, but has also its direction (input/output). A *unit type* is defined primarily by the set of port instances that make its interface. Each unit instance of a certain type has the same interface, but can have its own implementation (SCADA or DBMS). Unit and port types and instances are defined at development time and represent the model of the enterprise.

The principle of encapsulation is inherently supported since the implementation of a unit is completely hidden behind its interface: the only way to access a unit from outside is to send materials or documents to its input ports. Besides, a user or an action that initiates an operation in a unit can access other units' ports only through output ports of that unit. Thus, Mesh inherits all the benefits from the port-based object-oriented frameworks that are broadly accepted in other domains [4, 9].

A unit type A can *extend* another unit type B meaning that the A's set of ports is a superset of the B's set of ports. A notion of *unit reference* in Mesh supports polymorphism. A unit reference is a placeholder for all the unit instances whose types extend the unit type of the reference. Thus, a reference is defined by its type (interface). It is placed in the model and connected by its ports to other units or references. At runtime, a concrete unit instance can be connected (dynamically) to the reference (provided that its interface is compatible to the reference's). Then, all the transport that goes through the reference's ports is handled actually by the concrete referenced unit instance.

Finally, we are currently investigating incorporation of design by example and animation methods from Sced [10] into Mesh. The idea is to automatically develop flowcharts for units' behavior from interaction and activity/workflow diagrams (Figure 3). Namely, when an interaction diagram (Figure 3a) is viewed as a sequence of events (receptions of documents and materials) and actions (sending documents and materials, and other operations) for an interacting object (unit in Mesh), a flowchart for object's (unit's) behavior can be derived automatically (Figure 3b), because Mesh is formal in these aspects. We believe that this approach is a very powerful tool for fast and smooth development of executable model from requirement specifications (expressed by scenarios), especially in the context of document flow.

4.4 Concurrency and Distribution

As it is aimed for distributed industrial environments, Mesh must support concurrent multi-user access. Therefore, mutual exclusion must be provided for various operations that change the state of the system. These are, for example, document editing, sending, or channel deletion and creation. The operations can be initiated either by the user, or by the programmed actions. Mutual exclusion in Mesh is provided by the usual locking mechanism. Locking is performed at the unit instance level. Before an operation is to be performed, the corresponding unit instance is tried to be locked. If it is already locked by another operation, this operation has to wait until the unit is unlocked. When the operation completes, it unlocks the unit.

The basis for distribution in Mesh is again a unit instance. One workstation in the enterprise network may be configured to control one unit (and all its nested units) of the logically shared model. The impression of logically shared model in physically distributed environment is easily obtained by using a shared database as a model repository. All static data (units, ports, channels, and document and material types), as well as dynamic data (document and material instances, activities, etc.) are stored in the database. Workstations provide only separate views to the shared model, using standard database access. This way, we can use a traditional, stable, and well-supported client-server technology to address the problem of distribution.

5 Case Study: The Evaluation of Mesh

We have implemented a prototype version of the Mesh runtime tool. The prototype is capable of running the Mesh model in a distributed environment and performing all the basic document operations. We have also implemented interfaces to SCADA and DBMS. We have used Wonderware InTouch [1] as a SCADA package, and Microsoft Access as an application for database user interfacing (displaying forms). As a model repository and information system implementation, we have used Microsoft SQL Server. The communication to SCADA is performed using the Windows DDE protocol, and the database is accessed by the ODBC standard. This way, we have implemented a tool that can be easily adapted to other SCADA and DBMS servers on Microsoft Windows platforms.

Using this prototype, we have modeled the described oil refinery organization by fully implementing document flow and SCADA interaction for the process of oil blending, while putting off the detailed implementation of other units, such as stocking, oil production, inventory, etc. We have connected the Mesh model with the SCADA model of the process, and with the corresponding information system that deals with recipes and other details. We have also implemented several core scenarios of production, as described previously.

We may report some first experiences and impressions from using the Mesh framework for modeling. First and probably the most evident benefit was that the future users of the systems, i.e., technology experts (that have rather modest knowledge of computer and information technology), have surprisingly quickly accepted the concepts of Mesh and its terminology. The effect was a very good and early understanding between the users and developers in specifying the requirements. This has proved that the Mesh framework and its tools for requirements specification (hierarchy, interfaces, and scenarios) are very simple and well fitted, yet sufficiently expressive for the domain. Second, the ability to quickly produce an executable model was proved by developing the whole described system in about 10 man-months. Furthermore, a smooth transition from specifications to the executable model provided a system that had very few inconsistencies with the real users' needs.

There are also some negative experiences from the development. The most important is the following. Although we used only the standard interfacing to other applications (such as ODBC and DDE), we had some problems to integrate them. It seems that one of the most valuable issues of Mesh may turn into its biggest drawback due to incompatibilities or inconsistencies of standard interfaces. We hope that this effect will diminish when contemporary software technology, strongly oriented to applications co-operation, becomes stable.

As we have installed the prototype recently, we do not have much experience from using it. Some first and very preliminary impressions are that we are much more able to quickly respond to the users' changing requirements. This is partially due to the simplicity and expressiveness of Mesh, and partially due to its openness. It seems also that the users do not have many problems in learning and using Mesh as an enterprise supervisory model.

6 Conclusion

Mesh is a hierarchical object-oriented framework that may be used for modeling of a broad range of industrial organizations. Unlike some other design methodologies that use their concepts only in the process of requirements specification and design, while producing the executable model using traditional system or database-access languages, Mesh keeps the same concepts in the executable model. Due to its openness to other systems, SCADA and DBMS, which provide necessary specialties of the implementation, Mesh is a *complementary*, rather than an *alternative* approach to traditional enterprise modeling. It is aimed to enhance the capabilities of other systems, not to substitute them.

Some future extensions will, we hope, further improve applicability of Mesh. We will look for other common abstractions or use cases in the domain, in order to support them in Mesh, obtaining even better reusability. For example, we will study the notion of document lifetime and the ways to specify it and to obtain an executable model directly from its specification.

7 Acknowledgments

The author appreciates co-operation with the National Oil Industry of Serbia—Oil Refinery Novi Sad, where the described prototype has been implemented. The author is grateful to the colleagues who helped in the implementation and application of Mesh: Zlatko Orcic, Vesna Mincev, Sanja Djuran, Maja Sentin, and Olgica Nanovskij. Special thanks are to Vlado Krunic, who was the leading manager in the described process of automation.

8 References

[1] 'Wonderware InTouch Getting Started,' Wonderware Corporation, Irvine, CA, 1997

[2] YOURDON, E.: 'Modern Structured Analysis' (Prentice Hall Int'l, 1989)

[3] LIU, S., OFFUTT, A.J., HO-STUART, C., SUN, Y., OHBA, M.: 'SOFL: A Formal Engineering Methodology for Industrial Applications,' *IEEE Transactions on Software Engineering*, Vol. 24, No. 1, January 1998, pp. 24-45

[4] SELIC, B., GULLEKSON, G., WARD, P.: 'Real-Time Object-Oriented Modeling' (John Wiley & Sons, New York, 1994)

[5] 'Wonderware InTrack User's Guide,' Wonderware Corporation, Irvine, CA, 1997

[6] 'The Workflow Reference Model,' Workflow Management Coalition, http://www.wfmc.org, 1994

[7] GEORGAKOPOULOS, D., HORNICK, M., SHETH, A.: 'An Overview of Workflow Management: From Process Modeling to Workflow Automation Infrastructure,' *Distributed and Parallel Databases*, 3, Kluwer Ac. Publishers, 1995, pp. 119-153

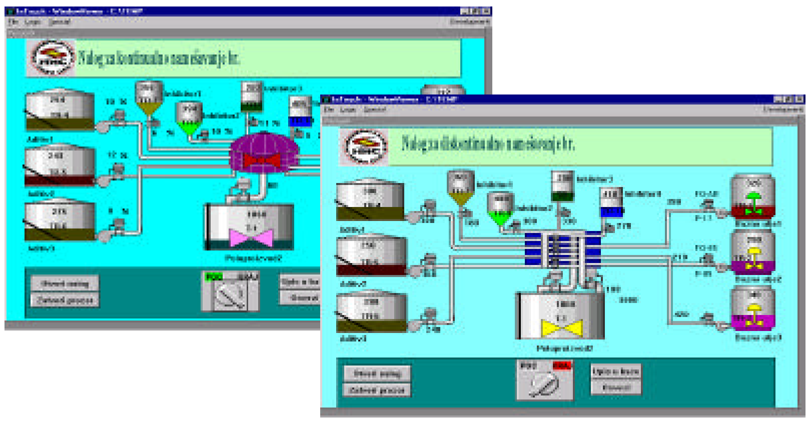
[8] BOOCH, G., RUMBAUGH, J., JACOBSON, I.: 'The Unified Modeling Language User Guide,' (Addison-Wesley, 1999)

[9] STEWART, D.B., VOLPE, R.A., KHOSLA, P.K.: 'Design of Dynamically Reconfigurable Real-Time Software Using Port-Based Objects,' *IEEE Transactions on Software Engineering*, Vol. 23, No. 12, December 1997, pp. 759-776

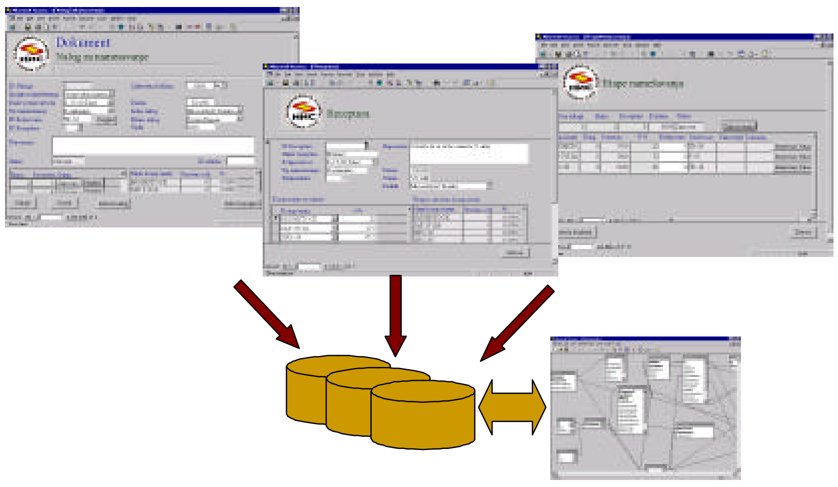
[10] KOSKIMIES, K., SYSTA, T., TUOMI, J., MANNISTO, T.: 'Automated Support for Modeling OO Software,' *IEEE Software*, Vol. 15, No. 1, January/February 1998, pp. 87-94

9 Figures and Captions

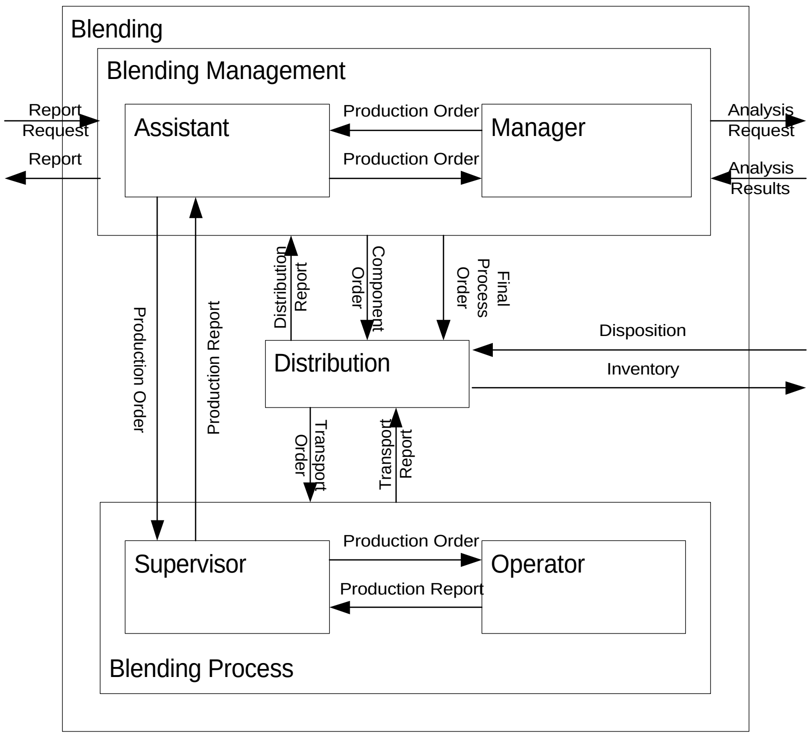
Figure 1: Characteristics and requirements of a complex industrial system



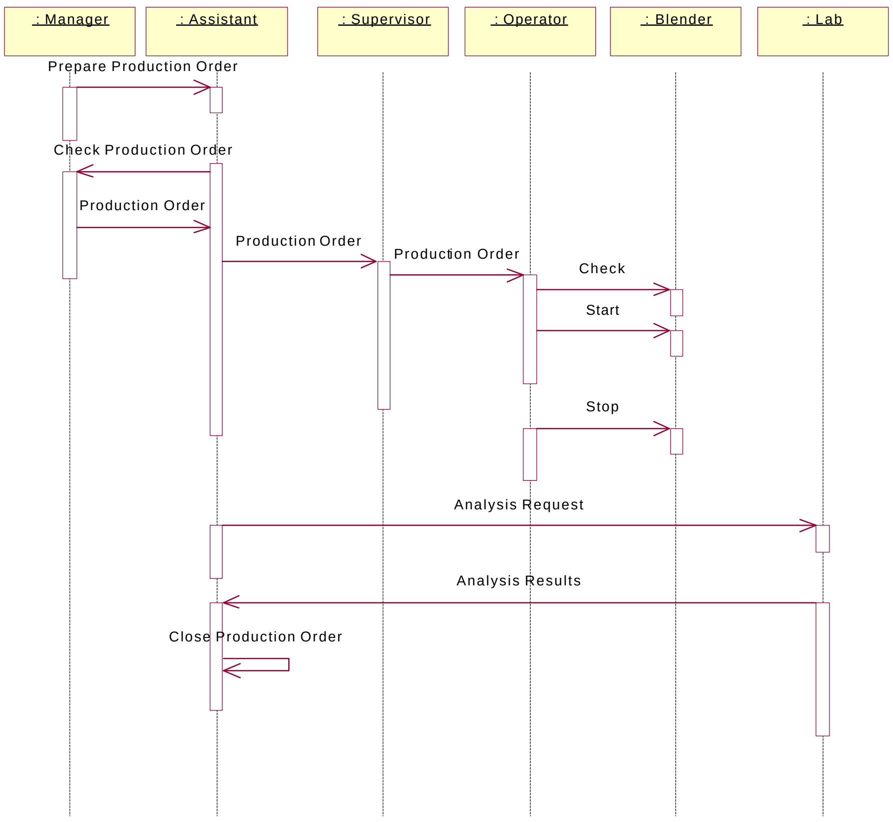
a) Industrial production processes



b) Complex operations upon data

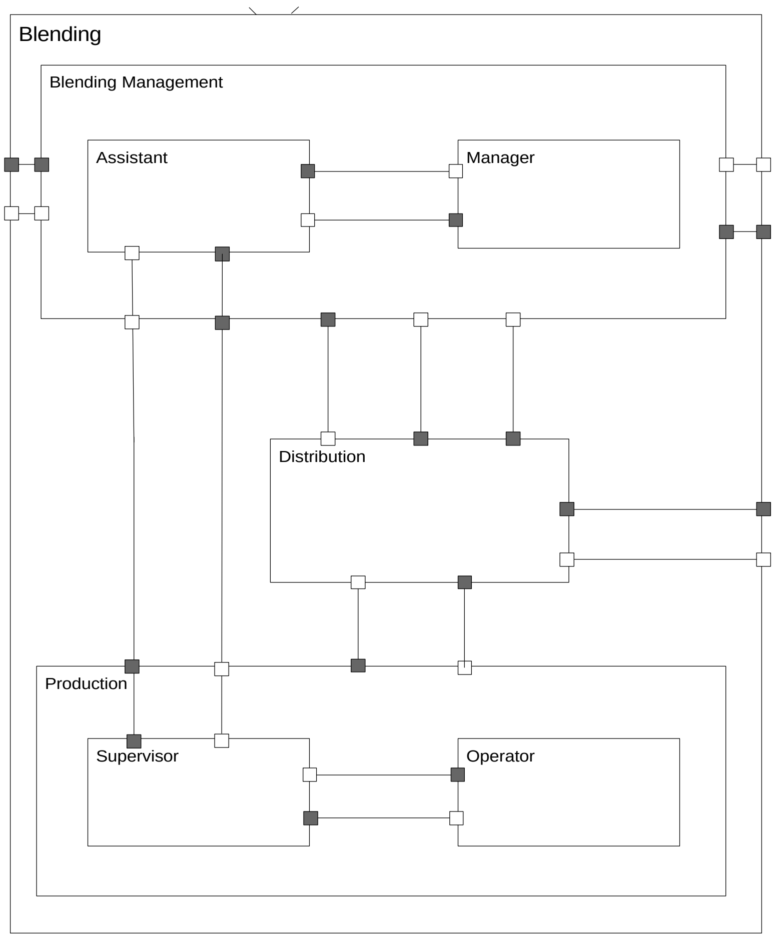


c) Hierarchical structure of the organization

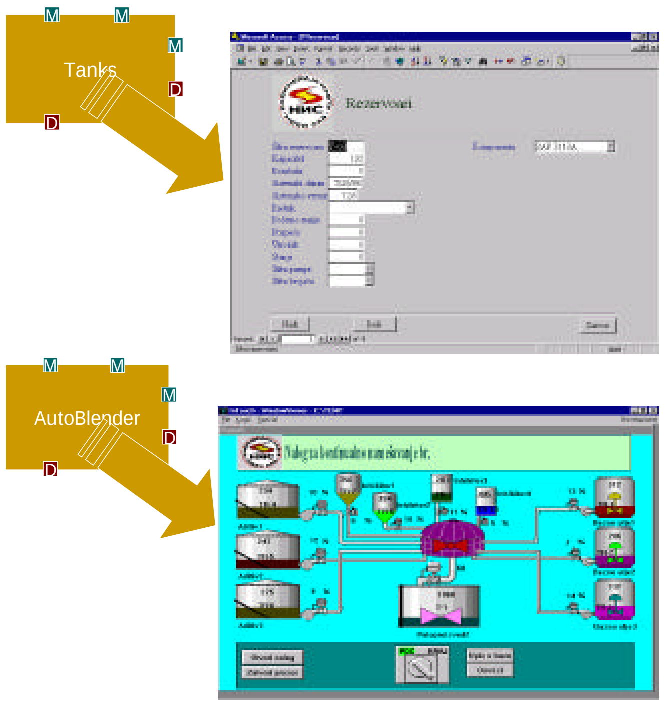


d) Dynamics of the system: flow of documents and materials

Figure 2: Mesh modeling

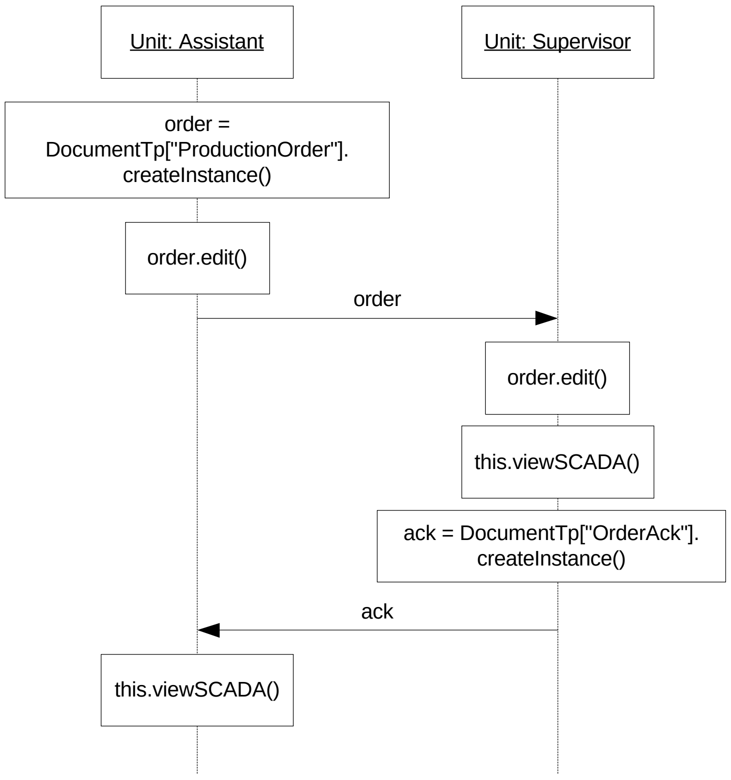


a) Hierarchical structure of units, ports, and channels

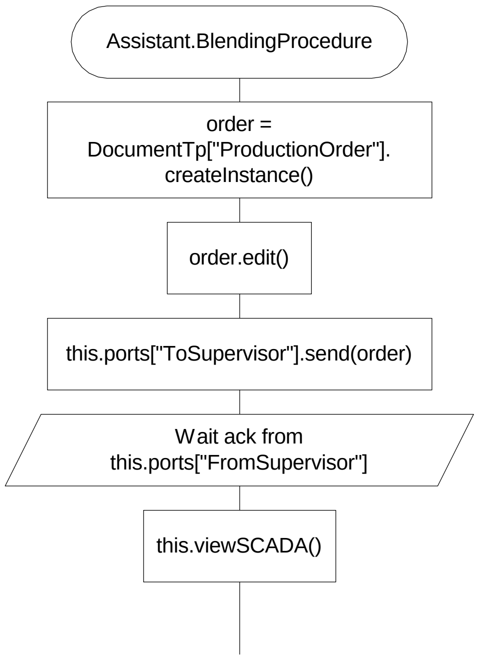


b) Unit implementations

Figure 3: Design by example: derivation of flowcharts from interaction diagrams


Unit: Assistant
Unit: Supervisor
order = DocumentTp["ProductionOrder"]. createInstance()
order.edit()
order
order.edit()
this.viewSCADA()
ack = DocumentTp["OrderAck"]. createInstance()
ack
this.viewSCADA()

a) Scenario specification


Assistant.BlendingProcedure
order = DocumentTp["ProductionOrder"]. createInstance()
order.edit()
this.ports["ToSupervisor"].send(order)
Wait ack from this.ports["FromSupervisor"]
this.viewSCADA()

b) Flowchart for Unit Assistant behavior