Flexible and Reliable Software Architecture for Industrial User Interfaces

Wolfgang Beer, Bernhard Dorninger, Mario Winterer
Software Competence Center Hagenberg
Softwarepark 21
4232 Hagenberg, AUSTRIA
{wolfgang.beer; bernhard.dorninger; mario.winterer}@scch.at

Abstract

Visualization software plays a major role in controlling and monitoring production machines and facilities. In recent years these software systems have undergone major changes in terms of flexibility, user interaction and large scale enterprise integration capabilities. While traditional machine control systems tend to operate in isolation, modern industrial software systems are integrated in complex production processes that demand for flexibility and openness, also in terms of dynamic change of functionality at runtime. Despite this increased demand for flexibility, the most important requirement for industrial software systems remains reliability and stable operation. Within this work we present an architectural approach for abstracting industrial application models on top of a dynamic and modular runtime framework (OSGi) and a flexible application platform (Eclipse). This approach allows to add functionality at runtime as well as to adapt the rendering of the UI to various technologies.

1. Introduction

With the major success of modern UI concepts in consumer electronics, flexibility, integrity and intuitive user interaction have found their way into industrial user interfaces in recent years. The fast progress of user interface technology in devices like smartphones and tablet PCs, has become a driving force for the development of a new generation of industrial user interfaces. The typical industrial control software is split into a hard realtime subsystem, which is responsible for the direct handling of actuators and sensors, and a non-realtime subsystem for visualization of machine states and for providing a basic user interface to manipulate process and machine parameters (HMI). Production machines (e.g., plastic moulding) usually heavily depend on these configurations, parameters and operational decisions, as well as on error handling, all of which is typically covered by human operators. In former times, such HMIs were barely composed of a set of control buttons and switches combined with some lamps for visual feedback. Today, quite complex HMIs consisting of some sort of tangible user interface and a graphical user interface (GUI) with respective input devices are a frequent sight. In addition, touchscreens became a common replacement for the monitor/keyboard/mouse combination over the last decade. Although industrial HMIs have been using touchscreen interaction for more than 20 years, the introduction of capacitive touchscreen technology in combination with multitouch user interaction and gesture recognition has opened the way for a completely new user experience. Another challenge for the architecture of industrial user interfaces is the increasing demand of flexibility concerning the dynamic loading of functions, or even the integration of third-party software (often referred to as Apps or Plug-Ins). Today, many of these enhancements are integrating with production companies' MES (Manufacturing Execution System) [16], ERP (Enterprise Resource Planning) [12] or CAQC (Computer Aided Quality Control) software systems [18]. Often these external software systems offer state-of-the-art web-based interfaces, so that one can observe great demand for seamless integration of platform independent user interface technologies, such as HTML5. Beside increased demand for flexibility and integration of new UI-technologies, these software systems operate in industrial environments where reliability is still the most important requirement. By using OSGi [15] and Eclipse [3] as a mature and reliable base for our software architecture, we are able to fulfill the requirement for increased flexibility, as well as the requirement for a reliable and stable basis. The remainder of this paper is organized as follows: Section 2 provides a brief overview of efforts in software architectures and technologies for industrial user interfaces and cites relevant related work. In Section 3, we outline our architectural approach. Selected aspects of our implementation are described in more detail in Section 4. Section 5 then discusses open issues and interesting further work. Finally, Section 6 provides a short conclusion.

2. Related Work

Defining software architectures for the implementation of user interfaces to control and parameterize industrial machines and facilities has been a widespread issue over

the last twenty years. As the machine and process control systems grew significantly in complexity and as the control software tends to be distributed within several control units on a typical machine, several research groups work on the problem of defining flexible and distributed software architectures. An early work by Kenneth C. Crater and Craig E. Goldman [9] on the definition of a distributed interface architecture for programmable industrial control systems describes in detail how the user control interface of several control nodes can be distributed by using a HTML interface in combination with a standard web server. A more recent work by Richardson et. al. [14] describes the actual trend of multi-touch user interfaces in industrial applications and discusses the implications of a combination with real-time system requirements. Beside the introduction of multi-touch software frameworks, this work also defines a set of guidelines for the implementation of multi-touch user interfaces that can cope with real-time requirements in industrial control and supervision applications. An early work by Avouris, Nikolaos M. et al. [5] discusses the issue of how to design man-machine interfaces for control and process supervision applications for cooperating agents. Their work focuses on the integration of dialog oriented user interfaces into a general software architecture for agent based control systems. It indicates the importance of user interaction specifically in distributed, task oriented control and supervision applications, which has great relevance for our actual work. In the domain of reliability and stability requirements of industrial user interfaces, ABB Corporate Research performed a scientific study in 2009 on defects in three large industrial control applications [6]. The conclusion of this work was that a high percentage of defects found through the user interface are actually defects in the underlying business logic and middleware software. This statement delivers a strong argument for the introduction of industrial user interfaces that build upon a reliable and flexible software architecture.

3. Architectural Approach

The purpose and contribution of this work is to present an architectural approach for implementing flexible and reliable applications that implement industrial user interfaces for control and supervision of industrial machines and facilities. The proposed architecture fulfills reliability requirements and adds additional flexibility by implementing a specific user interface rendering framework based on the Eclipse 4 SDK.

3.1 OSGi

The OSGi standard is one of the reference architectures for flexible and reliable plug-in, component-oriented middleware. It has been defined for the Java programming language and allows dynamic deployment of software components, which are referred to as bundles. Bundles can be coupled with other bundles through services

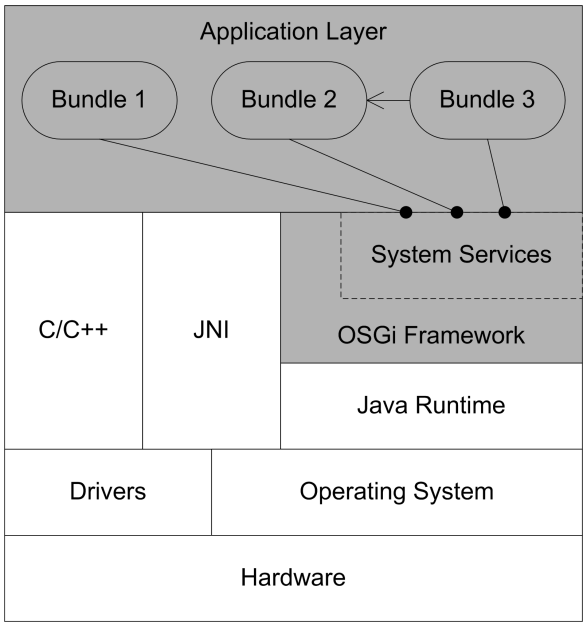

Figure 1. General OSGi system layers

and may be extended by bundle fragments, which are in fact partial components requiring a host bundle for providing their functionality. For software updates or bug fixes, OSGi provides a built-in mechanism to add, remove or update bundles or fragments at runtime. Besides this, OSGi also standardizes the lifecycle of bundles and services including the process of discovery, binding and interacting with these services. Popular and widely used implementations include Eclipse Equinox[2] and Apache Felix[1]. Today, many popular software systems, such as IBMs Eclipse IDE or the JBoss/Wildfly application server, are based on OSGi.

Figure 1 shows the general architecture and system layers of an OSGi application. The gray shaded layers contain the base OSGi framework implementation with its system services that represent the foundation of a custom application in the application layer. OSGi system services provide functionality like bundle and or package management as well as bundle lifecylcle and permission handling. The application layer above hosts the custom software bundles that are combined to an application. Typically, a custom industrial application implements bundles that contain the functional logic, bundles that contribute to the user interface and also bundles that are responsible for control system connectivity, e.g. providing access via OPC[17][10]. However, OSGi initially was intended for headless applications and provides no explicit support for developing user interfaces. Thus, it is helpful to combine the OSGi framework with a flexible means to render an abstract user interface model. This combination resolves the requirement for flexibility, in terms of integration of new user interface technologies, such as HTML5 or JavaFX, as well as the requirement for reliability.

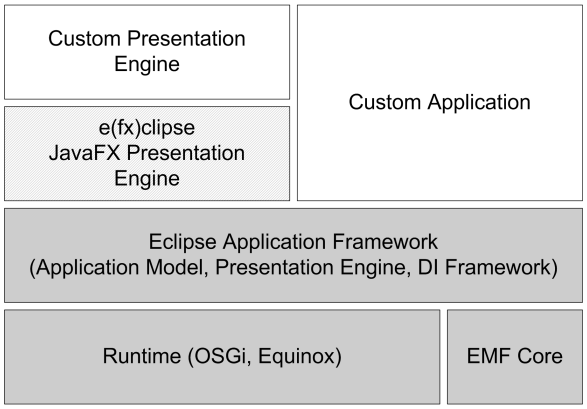

Figure 2. Custom industrial application layer on top of OSGi runtime framework

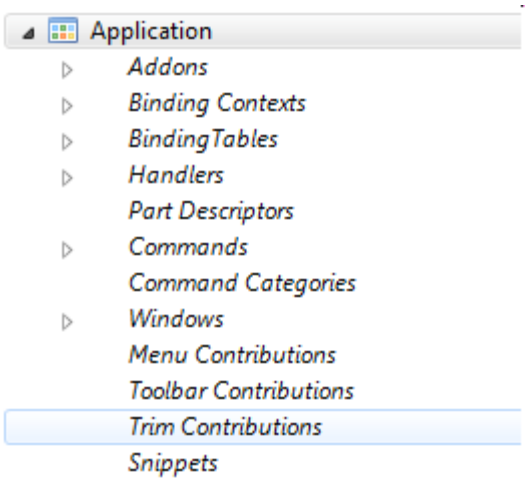

Figure 3. Top level elements of the Eclipse Application Meta Model

3.2 Eclipse

One of the most familiar software systems relying on OSGi is Eclipse [8]. Eclipse is not only a programming environment but also provides a base framework for interactive appplications known as Eclipse Rich Client Platform (Eclipse RCP). The actual version 4 of Eclipse offers a number of enhancements regarding the definition of user interfaces. In terms of a layered architecture model presented in Section 2, Eclipse 4 adds additional abstraction layers for defining the data model of an application in combination with its user interface elements, as it is shown in Figure 2.

The*Custom Presentation Engine* as well as the *Custom Application* layer highlight all application parts, such as the custom applications user interface and the custom applications data model. The Application Framework provides a base for implementing specific presentation engines. Another valuable improvement is the dependency injection support which provides application developers with a comfortable tool to adjust the state of their application at runtime by having necessary data objects and services injected automatically into their implementation classes. The biggest advantage however is gained by the so called Eclipse Application Metamodel [7], which defines a structural frame for custom UI applications. Application designers use it to create their own conctrete application models, which represent the required navigational structure and abstract UI components. In addition, the metamodel may be enhanced if the default metamodel is not deemed sufficient. The custom application model is completely independent of any concrete UI implementation technology. According to this fact, an appropriate implementation of the presentation engine is used to render the user interface of a custom application. Of course, a specific presentation engine may be reused for applications sharing the same UI technology. Example of such renderer implementations exist for Java SWT as well as for JavaFX, but support for virtually any user interface technology is feasible. The abstraction from the user interface implementation technology allows one to introduce a new kind of flexibility by offering various renderers for different platforms.

The default Eclipse Application Meta Model contains a number of model elements shown in Figure 3 with the most important ones being discussed below:

- *Addons* is a list of service objects (Addon) that are instantiated at system startup. An Addon has no visual representation but it can take influence on any part of the system. Every Addon contains an URL that uniquely denotes the Java class that implements the service and its behavior.

- *Commands* are abstract representations of user triggered tasks that can be performed by the system, like *Start Engine* or *Show Help*. At that level, it is neither important how the task is triggered nor where the concrete implementation of the system behaviour resides. Both aspects are modeled using Handlers and BindingTables.

- *Handlers* refer to a concrete piece of code that is invoked when a certain command is executed. It is possible to have multiple handlers per command, but only one handler will be active at the same time according to the current application context. Some components like Window or Part (see below) can define local command handlers and hence override the behavior of other handlers with the same id that are defined on a more global level. For example, a Part might define a custom handler for the Show Help command which overrides the global handler. Whenever the corresponding command is executed while the part is active, the overridden handler will be executed instead of the global one.

- *BindingTables* are used to define a set of user interaction shortcuts and bind them to their corresponding commands. Whenever a keyboard shortcut is triggered, the current active handler of the corresponding command will be invoked.

- *Windows* is the root node of all model objects that make up the user interface controls. There may be more than one window per application. Each *Window* consists of additional structural components, like *Area* or *PartStack* objects. A PartStack contains a number of child controls, but is intended to display only one of them. Of course this is the responsibility of the presentation engine - it would also be feasible to implement a special renderer for a PartStack which is able to display more than one of its children at a time. A *Part* object embodies the atomic portion of the default metamodel. In a machine visualization application, this may be a specific screen, for instance. In terms of UI programming it may be a specific widget container (e.g. a panel) - depending on how the used presentation engine renders a Part object. A Part referes to a Java class that is able to build up this specific Part's UI representation. Of course this implementation class is no more independent of the used UI technology. In other words, the visual content of a Part is not modeled anymore rather than built up in a "traditional way" depending on the UI technology used.

An application model is loaded and interpreted at runtime by the presentation engine which decouples the model from the concrete user interface toolkit. In addition, the engine monitors the application model to detect modifications at runtime and instanly reflect on the displayed user interface. By customizing or even rewriting the presentation engine it is possible to use a different user interface toolkit without the need to adopt the application model itself. Therefore, all components of the application relying on the model only and not on its representation can be retained. As mentioned before, the Part nodes are user interface dependent (due to the directly referenced implementation class), so while the application model itself and also the window definitions can be reused throughout different toolkits, the parts itself have to be rewritten. However, even this constraint might be evaded by employing a solution based on technology agnostic concepts like UIML[13].

4. Implementation

In this section we provide a brief insights in our efforts of applying OSGi and Eclipse with a custom presentation engine for JavaFX to develop an industrial visualization application framework. Figure 4 depicts the simplified architecture of a prototype following our proposed architectural approach.

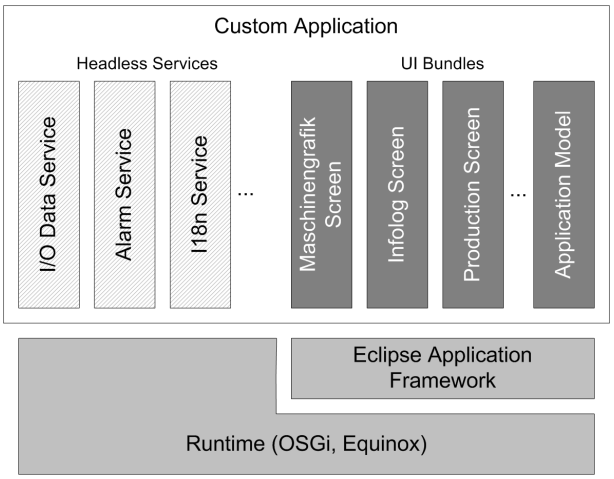

Figure 4. Architectural Overview of an Application Prototype

A visualization application has to provide connectivity to one or more PLCs to read and write sensor and actuator data as well as alarms. This functionality is wrapped in headless OSGi bundles and does not depend on Eclipse or any UI technology. They might be implemented using technologies like OPC(UA) for communications. These services are used by the UI bundles, which not only rely on the OSGi infrastructure, but also on the Eclipse Application Framework. Since "traditional" visualization application usually consist of more or less independent screens, each of these screens could be wrapped in its own bundle - or a set of bundles if decomposition is helpful. In our case there is a central bundle, which only hosts the concrete application model plus some configuration information. Figure 5 shows an excerpt of such an application model for our prototypical application.

Besides a number of Addon objects (e.g. for gesture recognition) and some Command/Handler/Binding definitions, which have been left out in the figure, the model consists of one application window, which hosts an Area node. This area initially is divided into four portions: a fixed header part, two switchable PartStack objects for machine screens and a fixed footer part. The header contians functionality like time and date display as well as a possibility to switch the current machine user. The footer contains some interaction elements for quick navigation. The two PartStack objects represent the screen areas *A* and *B* as seen in Figure 6. While Area B contains a default PartStack implementation, which always displays only one of its children parts *B1..Bn* depending on user selection, we implemented a special renderer for the Area A PartStack renderer. It allows to slide its children parts *A1..An* like desktops on a mobile phone. One such child part is a machine overview depicted in Figure 7, for example.

Being a flexible layer on top of the robust OSGi platform, Eclipse 4 is a perfect base system for a modern user

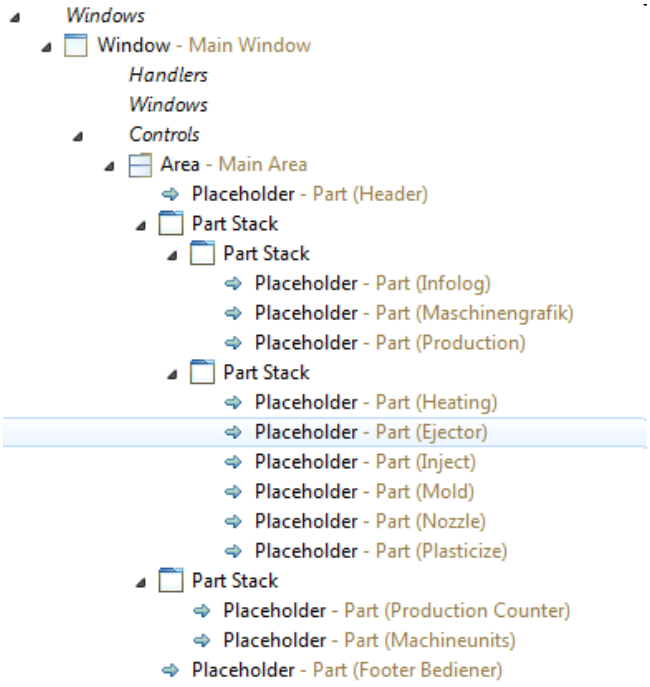

Figure 5. Example for an extendable structural application model

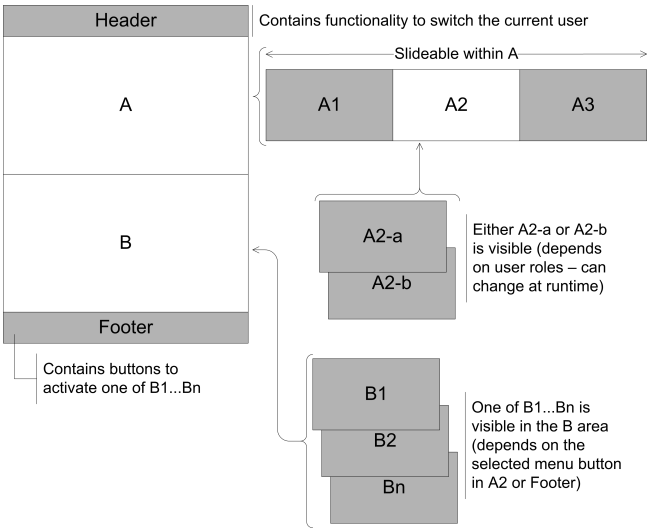

Figure 6. Abstract layout structure for a custom user interface application

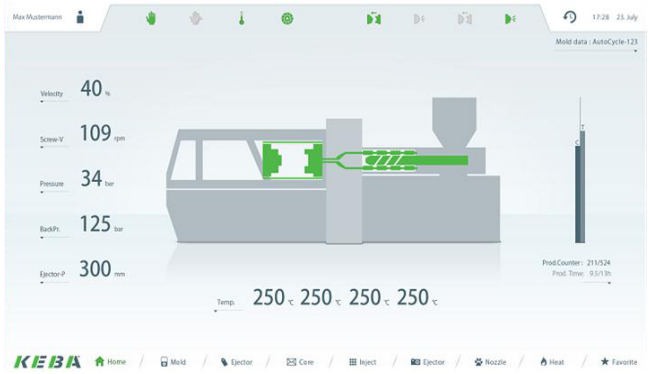

Figure 7. The ”Maschinengrafik” screen Part of our application model - rendered in JavaFX

interface toolkit like JavaFX. Therefore, we use the third party OSGi bundle library e(fx)clipse [4], a JavaFX based eclipse presentation engine. The e(fx)clipse presentation engine is a facade [11] that delegates to renderers, each specialized for representing a certain type of model element. This flexible approach allows us subclassing individual renderers to override the presentation of certain model elements.

An investigation of the default renderers revealed, that they do not fulfill the user interface requirements of our prototype. For example, the default renderer for PartStack shows a tab for each part of the stack, but the requirements demanded that all stacked parts are simply rendered one upon the other without any tabs. In addition, loading and rendering a single part should not cause the user interface to freeze. Instead, long lasting part loading/rendering should be pushed to a background process and a progress indicator should be shown in the meantime. This is a feature not being supported by the default implementation. All respective requirements could be fulfilled by subclassing the corresponding default renderers.

5. Open Issues and Further Work

As we are continuously improving the architecture of industrial software systems in cooperation with our business partners, there are still many issues to deal with. One challenge remaining is to implement a flexible binding of touch gestures with our abstract custom application model. At the moment neither the Eclipse 4 base framework, nor our architectural enhancement allows an application engineer to bind touch and multi-touch gestures with selected model actions. In terms of enabling intuitive user interaction in custom industrial applications this issue seems to be important for further increasing the flexibility and productivity in application design. Another challenge in progress is to provide a user credentials mechanism, which allows to bind certain rights to application model elements and their contained widgets.

6. Conclusion

Within this work we presented an architectural approach for implementing flexible and reliable industrial user interfaces. A main contribution of our work is that we have shown the combination of a reliable runtime platform standard (OSGi) with a flexible application model for dynamic rendering within different user interface technologies based on Eclipse 4 is an appropriate base for industrial HMI applications, especially against the backdrop of varying customization requirements. Eclipse 4 decouples the logical model of a visual application from its presentation by allowing different renderer implementations such as for JavaFX or HTML5. Although Eclipse is Java based, one might consider using even non-Java UI technologies for rendering, such as QT. Our implementation of choice is efxclipse, which uses JavaFX 2 to build and show UI components. The use of JavaFX provides another significant advantage, since its powerful property binding mechanism allows the dynamic and flexible connection of UI widgets to PLC variables. Another benefit springs from Eclipse's base technology OSGi. Its dynamic and modularized nature remarkably eases customization and maintenance of applications by enabling the addition and/or replacement of functionality at runtime. This also includes interactive components plus their PLC bindings. With the further development of the Java language ecosystem it will be even possible to run our Eclipse/JavaFX based applications on systems with limited resources.

References

[1] Apache Felix OSGi Container, http://felix.apache.org/, 2013.

[2] Eclipse Equinox OSGi, http://www.eclipse.org/equinox/, 2013.

[3] Eclipse SDK 4.x: The Next Generation Eclipse Platform, http://www.eclipse.org/eclipse4/, 2013.

[4] e(fx)clipse - JavaFX 2 Tooling and Runtime for Eclipse, 2013.

[5] N. M. Avouris, M. H. V. Liedekerke, G. P. Lekkas, and L. E. Hall. User interface design for cooperating agents in industrial process supervision and control applications. *International journal of man-machine studies*, 38(5):873–890, 1993.

[6] P. A. Brooks, B. P. Robinson, and A. M. Memon. An initial characterization of industrial graphical user interface systems. In *Software Testing Verification and Validation, 2009. ICST'09. International Conference on*, pages 11–20. IEEE, 2009.

[7] F. Budinsky. *Eclipse modeling framework: a developer's guide*. Addison-Wesley Professional, 2004.

[8] M. Clausen, J. Hatje, J. Rathlev, and K. Meyer. Eclipse rcp on the way to the web. *Proceedings of ICALEPCS 2009*, pages 886–888, 2009.

[9] K. C. Crater and C. E. Goldman. Video interface architecture for programmable industrial control systems, Nov. 9 1999. US Patent 5,982,362.

[10] U. Enste and W. Mahnke. OPC unified architecture. *at-Automatisierungstechnik*, 59(7):397–404, 2011.

[11] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. Design patterns: Abstraction and reuse of object-oriented design. *ECOOP93Object-Oriented Programming*, pages 406–431, 1993.

[12] H. Liang, N. Saraf, Q. Hu, and Y. Xue. Assimilation of enterprise systems: the effect of institutional pressures and the mediating role of top management. *Mis Quarterly*, 31(1):59–87, 2007.

[13] U. OASIS, T. Committee, et al. Oasis user interface markup language (uiml) tc.

[14] T. Richardson, L. Burd, and S. Smith. Guidelines for supporting real-time multi-touch applications. *Software: Practice and Experience*, 2013.

[15] The OSGi Alliance. OSGi Service Platform, Core Specification, Release 5, 2009.

[16] P. Valckenaers and H. Van Brussel. Holonic manufacturing execution systems. *CIRP Annals-Manufacturing Technology*, 54(1):427–432, 2005.

[17] L. Zheng and H. Nakagawa. OPC (OLE for process control) specification and its developments. In *SICE 2002. Proceedings of the 41st SICE Annual Conference*, volume 2, pages 917–920. IEEE, 2002.

[18] X. Zheng and D. Chen. Computer aided quality control system for manufacturing process. In *Intelligent Control and Automation, 2004. WCICA 2004. Fifth World Congress on*, volume 3, pages 2819–2823. IEEE, 2004.