Service-oriented Communication Concept based on WCF.NET for Industrial Applications

Markus Stopper and Bernd Gastermann

Abstract— **Software Development is subject to a constant process of change. In the meantime Web-services, distributed applications or access to remote services are already the standard. Side by side with these techniques their demands rise significantly. Defined support for security issues, coordination of transactions and reliable communications are expected. Windows Communication Foundation (WCF) - as a part of the present and succeeding .NET Framework - by Microsoft Corporation supports these requirements in line with wide range interoperability. WCF.NET provides the development of distributed and interconnected software applications by a service-oriented programming model.**

This paper introduces a service-oriented communication concept based on WCF specifically designed for industrial applications within a production environment with a central manufacturing information system (MIS) database. It gives an overview about some important design aspects and base service sets of WCF and also shows a factual implementation of the presented service-oriented communication concept in the form of an industrial software application used in plastics industry.

Index Terms—**Service-oriented communication concept, publisher-subscriber pattern, web services hosting, windows communication foundation, ws-discovery.**

I. Introduction

Improvement in efficiency by process mastering is on top of the priority list of manufacturing companies these days. Efficient processes hand in hand with less wastefulness result in more profit. Process mastering enables erratic success especially in productivity, cycle and through-put time as well as quality of production processes. Effective processes without dissipation oriented on a lean value stream, ensure efficient creation of value. Thus the methodical increase of efficiency in manufacturing plays a major role. An important prerequisite for that is to have a clear view of the current production state in real time virtually a production-mirror.

Supervisory control and data acquisition systems (SCADA) and distributed control systems (DCS) respectively take care of this requirement.

Manuscript received December 30, 2009. This work was supported in part by the Automation Research & Development Department of MKW® Austria. Special thanks to the major project promoters Hans and Johannes Danner.

Markus Stopper, *Member IAENG, IEEE & DAAAM International*, was with Department of Production Engineering, Vienna University of Technology, Vienna, Austria. He is now with the Industrial Automation IT Research & Development Department of MKW® Slovakia, Prešov, Slovakia (e-mail: markus.stopper@ieee.org).

Bernd Gastermann, *Member of DAAAM International*, was with University of Applied Sciences FHSTG Burgenland, Information and Communication Solutions, Eisenstadt, Austria. He is now with the Industrial Automation IT Research & Development Department of MKW® Austria, Weibern, Austria (e-mail: bernd.gastermann@mkw.at).

However, in the category of these systems proprietary communication and application solutions can be found frequently. In fact the communication within SCADA or DCS systems is characterized nowadays more and more by the support of miscellaneous TCP-based internet techniques. Workplaces on which visualization and monitoring takes place usually are connected via wired or wireless Ethernet and TCP. The goal of interoperability standardization for communication, however, is far away from being achieved. Previous attempts like OPC are commonly restricted to certain operating systems and protocols, although in the meantime a step forward for operating system/protocol independence is made with OPC UA for example.

For developing ambitious distributed industrial software applications this means, that software development underlies a constant process of change, which makes it seriously difficult to decide for a certain and consciously not proprietary communication technology. Additionally the demands for modern software techniques - like defined support for security issues, coordination of transactions and reliable communications - rise significantly.

Windows Communication Foundation (WCF) - as a part of the present and next .NET Framework - by Microsoft Corporation supports these requirements in line with wide range interoperability. WCF provides the development of distributed and interconnected software applications (with communication interoperability via parameterization) by a service-oriented architecture (SOA) programming model.

This paper introduces a service-oriented communication concept based on WCF.NET specifically designed for industrial applications within a production environment with a central manufacturing information system (MIS) database. It gives an overview of some important design aspects of WCF (next 4 sections), presents the developed SOA-conform communication concept (solution sections) and finally also shows the implementation of the presented solution by describing a corresponding industrial software application.

II. Message Exchange Patterns

WCF offers various ways to exchange messages between client and service. These operation types are often referred to as Message Exchange Patterns (MEPs). In general, the type of operation used to communicate with the service is part of the service, a certain MEP may even place some constraints on the allowed bindings as not every WCF binding actually supports all available MEPs [1].

Throughout the next six paragraphs the various MEPs supported by WCF are presented.

ISBN: 978-988-18210-5-8
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

A. Request-Reply

Request-Reply is WCF's default operation mode. Using this MEP, the client issues a call to the service in the form of a message and blocks until it gets a reply. If the service does not respond within the specified timeframe, the client will get a 'Timeout-Exception'. Using request-reply operations is very simple and resembles programming using the classic client/server model [1].

B. One-Way

In case an operation has no return value and the client does not care about success or failure of the call, WCF offers one-way operations to support this kind of 'fire-and-forget' invocation. Contrary to request-reply calls, one-way calls usually block the client only for the briefest moment required to dispatch the call. As only a request message but no reply message is generated by WCF, values (as well as exceptions thrown on the service side) can't be returned to the client [1].

C. Callback / Duplex

Using duplex communication (or callbacks) WCF allows a service to call back to its clients and invoke a client method. Callback operations are especially useful when it comes to events and notifying the client that some event has happened on the service side. However, in order to enable the service to call back to the client, it has to know the client's endpoint. Therefore, it is necessary for the client to call a service method (see Fig. 1/step 1) first, which saves the callback channel to the client's endpoint for later use (see Fig. 1/step 2). Through that channel it is possible for the service to send messages to the client and invoke certain methods [1].

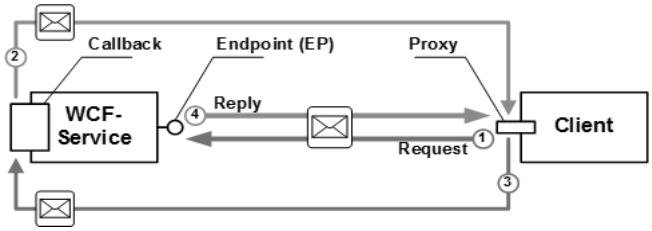

Fig. 1: Base schema of callback /duplex communication [3]

D. Streaming

When client and service exchange messages, they are buffered on the receiving side and delivered only once the entire message has been received. This means, that the client is unblocked only if the request message and the service's reply message have been sent and received in its entirety. This works well for small messages.

However, when it comes to larger messages (e.g. multimedia content) blocking until the entire message has been received may be impractical. Therefore, WCF enables the receiving side to start processing the receiving data while the message is still being received. This type of operation is called streaming transfer mode and improves throughput and responsiveness with large payloads [1].

E. Asynchronous Calls

Using asynchronous calls the client will not block and control returns immediately after the request has been issued. The service then executes the operation in the background. As soon as the operation completed execution the client is provided with the results of the execution. Asynchronous calls improve client responsiveness and availability [1].

F. Queued Calls

Queued messages use Microsoft Message Queue (MSMQ). WCF encapsulates each SOAP message in an MSMQ message and posts it to the designated queue. Thus, the client does not communicate directly with a certain service but with a message queue. As a result, all calls are inherently asynchronous and disconnected. On the service side, queued messages are detected by the queue's listener, which sequentially takes messages from the queue and dispatches it to a service instance [1]

III. Hosting

In order to run and access a WCF service it needs to be hosted in an appropriate environment. Such an environment consists of a process in which the service itself is running. The only requirement for the host is to support .NET application domains. For this reason the following four common hosting methods are to be considered: Managed .NET applications, Windows Services, Internet Information Services (IIS) and Windows Activation Services (WAS). Hosting in any custom application type is possible as long as it supports application domains. Each of these types can be classified to one of two categories: Self-Hosted and Hosted.

A type of host, which the developer has to write on his own is considered a self-hosted environment. As it has to be written completely by the developer himself, these hosts do not contain any manageability features by default. However, this approach offers full control over the hosted service and its life cycle.

Hosted environments – contrary to self-hosted environments – are prefabricated hosts that do not need to be developed and already contain several management features. As these pre-existing hosts handle the service in their own way the developer only has limited influence on the service's life cycle while hosted in this type of host [2].

Each of these hosting environments has certain advantages and disadvantages. Thus, they are not suitable for every application area. In a service-oriented architecture the right host is essential for service robustness. When choosing a hosting environment, it is important to identify the type of service and its availability demands. As the service implementation in WCF is platform agnostic it can easily be ported from one host to another.

Over the next sections each of WCF's standard hosting environments is introduced.

A. Managed Applications

Managed .NET applications are categorized as self-hosted environment as they have to be written by the developer. That also includes graphical applications like Windows Forms and WPF applications but also console applications. This type of host is easy to develop with only a few lines of code, effortless to install and poses almost no demands to the

ISBN: 978-988-18210-5-8
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

system it is running on – except for the .NET Framework itself. Unlike other hosting methods this type is ideal for testing, debugging and demonstrating. It can also be used for user interaction and service control, if necessary. As this is a completely self-made host, it does not contain any features for high availability, easy manageability, recoverability, robustness, versioning and deployment scenarios like IIS does for example. Therefore, it is the developer's responsibility to write the code for starting and stopping the service. This implies that the service needs to be activated manually. The developer thus gains full control over the service's life cycle. Managed applications usually run under limited user accounts and have to be started and stopped explicitly. A service hosted this way is only available when the host is running. However, managed applications support all kinds of transport protocols available for WCF [2].

B. Windows Services

A Windows Service is maintained by the operating system's Service Control Manager (SCM). Although the developer has to write the Windows Service which in turn hosts the WCF service, this Windows Service itself is hosted and maintained by the SCM. This allows the developer to have full control over the life time of the WCF service as it has to be explicitly opened and closed by the code. Windows Services are easy to develop but do not provide any graphical user interface. This type of host is ideal for long-running services as they are continuously monitored by the SCM of Windows. It offers limited support for features like auto-start on system boot and recovery in case of an error. This makes the service available as soon as the computer starts, regardless of whether a user is logged in or not. Furthermore, it allows each service to have its own account and security permission. Windows Services are easy to maintain by administrators as they often do already have knowledge about how to configure them. Nonetheless, installation software is necessary to install a service on the system [2].

C. Internet Information Services Hosting

WCF services can also be hosted using Microsoft's Internet Information Services (IIS) version 5.1 or higher. As IIS is primarily used as web server it only supports HTTP as transport protocol for WCF services. However, IIS 7.0 already includes Windows Activation Services (WAS) which enables support for all WCF protocols. IIS is categorized as a hosted environment and therefore controls each WCF service's instantiation. This can only be influenced by a custom Factory written by the developer. The huge advantage of IIS lies in its many management features like process health monitoring, idle shutdown, process recycling, resource throttling and logging. It offers the same functionality for WCF services as for ASP.NET applications. Since IIS is a complete, stable environment no additional development effort is necessary. Everything is done solely by configuration. In comparison to managed applications and Windows Services, IIS uses automatic, message-based activation. It is important to know that many of its features as well as its activation mechanism can also cause unexpected behaviour compared to other hosting methods [2].

D. Windows Activation Services Hosting

Windows Activation Services (WAS) features all advantages of IIS. Although it is part of IIS 7.0, it can be installed, configured and operated independently. WAS also uses message-based activation for services but contrary to IIS, WAS supports all available transport protocols of WCF, including HTTP, TCP, MSMQ and Pipe. For that purpose it installs listener adapters for each protocol. WAS is only available on Windows Vista and Windows Server 2008 or higher [2].

IV. Publisher-Subscriber Pattern

The publisher-subscriber pattern is a design pattern often used in connection with WCF in order to enable event-based notification of clients through the network. It allows a service to notify its clients that something has happened on the service side. Using the duplex, MEP is necessary for this pattern to work. One-way operation calls are possible but optional.

This design pattern's basic principle is that one or more clients subscribe (see Fig. 2/step 1) at the service in order to receive events. The service then saves each client's callback channel to a list. When an event occurs, the service publishes (see Fig. 2/step 3) it to all subscribed clients in its list. However, in most cases the origin of an event often isn't the service itself but another client or service. In this case, the publishing service just acts as a distributor. The event-publishing client tells the distributor that an event occurred (see Fig. 2/step 2) and the distributor informs all other clients. If a client is no longer interested in receiving events it simply unsubscribes from the service, which then deletes the callback channel from its list [3].

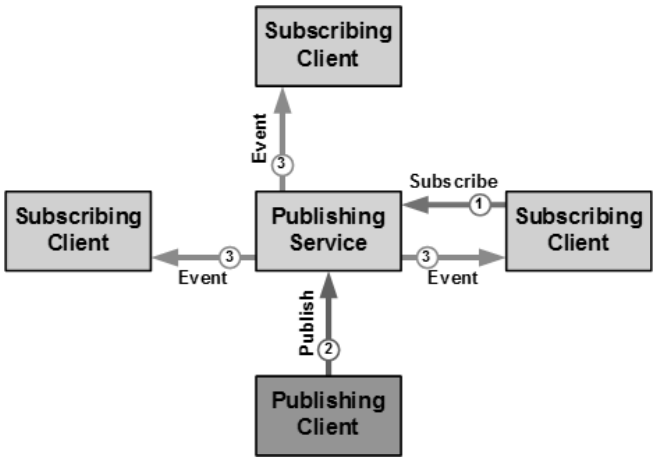

Fig. 2: Base schema of publisher-subscriber pattern [3]

V. WS-Discovery

In order to access a service, it is necessary for the client to know its address. One possibility would be to include all relevant service addresses in the client's code or configuration files. However, as this approach results in a tight coupling between service and client, it is often either not desired or simply not suitable for the client's application area. Therefore, methods exist which allow the client to dynamically locate the desired service and its address.

ISBN: 978-988-18210-5-8
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

One of these methods is WS-Discovery. WS-Discovery is a lightweight WS-* specification to find services in the network. Although WCF implements many WS-* specifications, WS-Discovery is not one of them. Version 3.x of Microsoft's .NET Framework does not provide support for WS-Discovery by default. However, .NET 4.0 will be the first version to officially include this specification. Support for earlier versions is only possible either through custom or third party implementations [4].

WS-Discovery uses SOAP and "User Datagram Protocol" (UDP) Multicasts to find services on the network. WS-Discovery knows four scenarios where each of them has its own message type [4]:

- Hello: As soon as a service is started, it sends a "hello" message via UDP multicast into the network in order to inform other participants about its availability.
- Bye: During shutdown, the service sends a "bye" message using UDP multicast to the network in order to indicate it will no longer be available. Both "Hello" and "Bye" messages are used to reduce network traffic and to avoid service polling by the clients.
- Probe: Clients interested in a specific service send a "probe" message containing information about service type or scope of the service they want to use. All services that match with this information will send a "ProbeMatch" message directly to the client.
- Resolve: In order to locate a service's endpoint and its address the client sends a "resolve" message that contains the service name. All services with this name will answer using a "ResolveMatch" message.

It is important to know that UDP multicasts are by nature restricted to the local subnet. However, in order to circumvent this restriction the use of a discovery proxy is possible [5].

VI. System Architecture

The project driven application-oriented intended purpose for the development of this kind of a service-oriented communication concept based on WCF.NET is, to allow distributed real-time visualisation of production process data written to a central manufacturing information system (MIS) database - with the basic condition to develop the frame model with an underlying reusable distributed system concept utilizable for other industrial applications.

The services of the frame model transfer new inserted data from the relational database over a local area network to the client applications where they are being visualized. This generic system is distributed across three tiers described in the following enumeration:

- The database tier includes both tables and database triggers. These tables contain all the data necessary for visualisation. "Common Language Runtime" (CLR) triggers are coupled with certain relevant tables. The triggers' function is to send newly added data to the services immediately.
- The service tier is characterized by "Windows Communication Foundation" (WCF) services. They are building the link between database and visualization client. Data provided by the database triggers are always forwarded to all subscribed clients as soon as the service receives them.
- On the client tier all transferred data from the services are finally being displayed by the visualization application (client). However, it is also possible to run multiple clients on the same and/or other computers at the same time without affecting each other.

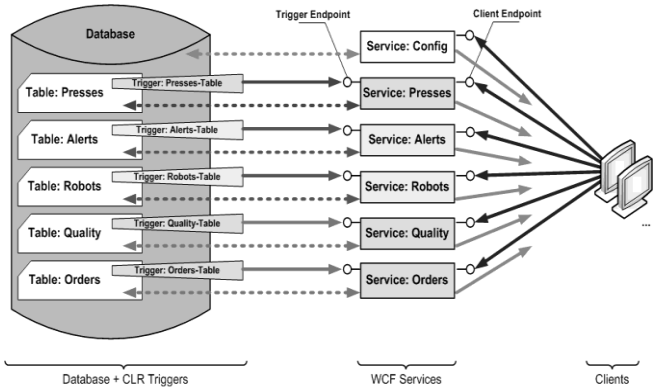

Fig. 3: Schema of the service-oriented communication concept

In this distributed visualization system the database is the ultimate source of the data to be displayed. Whenever a SQL statement (either INSERT or UPDATE) is sent to the SQL server and a new table record is inserted or updated, the trigger associated with the affected table is fired. These triggers are developed using .NET technology and run within SQL server's CLR. These triggers gather all recently added data and send them to one of the services. Each database trigger is associated with its own WCF service and sends data solely to this service.

The service offers two endpoints, one for the trigger and one for the client. Via its trigger endpoint it takes data packets from the trigger and forwards them to all subscribed clients. In order to allow client subscriptions for trigger events from the database, it uses the publisher-subscriber design pattern. To receive these events, a client has to subscribe to the service through its client endpoint which then saves the client's callback channel to a list. In case a trigger is fired and data are received, it sends this packet through each client callback channel in this list. This pattern allows the service to notify many clients at the same time. Additionally, the service implements WS-Discovery to enable clients to discover it in case its address has changed or is yet unknown to the client.

The client receives data packets from all services, stores them in its internal memory and finally visualizes them on the screen. Before the client is terminated it unsubscribes, which causes the services to remove this client callback from their lists so that no more events are delivered.

Through this mechanism, only newly added data can be transferred as database triggers, in general, only fire on INSERT and UPDATE statements. However, to allow visualization of historic data another kind of mechanism has to be used. Therefore, this system uses a simple synchronization mechanism, where the services access custom database views in order to query the necessary data

ISBN: 978-988-18210-5-8
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

from the database. This synchronization is initialized by a specific client call through the service's client endpoint. The service then gathers all data and sends them back to the client using the same callback technique also used for trigger updates. Then, everything is ready to be displayed by the client application.

VII. Client Communication Interface

The client application is designed to communicate with several WCF services at the same time. Each of these services has its own service contract and therefore uses a different message layout with different data types. Furthermore, the client has to provide methods for each service to call in order to allow notification of trigger events.

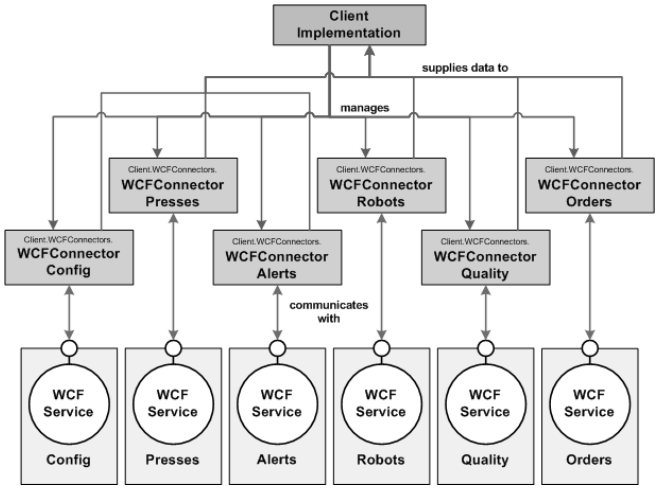

Fig. 4: Schema of client communication interface

Thus, the client uses multiple so called "WCF Connectors" to communicate with all WCF services. Each connector allows only communication with one specific service. This means that six services would also require six connectors on the client side. The intention for this concept was to make the code more structured and readable as well as to avoid ambiguousness between identical service methods like subscription.

Each connector provides its service with a method that the service can call via the callback channel in case an event occurs. This connector method takes the data packet and forwards it to the responsible object of the client's implementation where it is processed, saved and visualized.

VIII. Industrial Application Example

To demonstrate and prove the functionality of the developed service-oriented communication concept based on WCF.NET an industrial application was implemented. The purpose of this application is, to show all important benchmarking data of production to the shift supervisor during the currently running shift. Data of the current shift include real-time data transferred by the triggers mentioned in a previous section. Additionally, the supervisor has the option to display historical data from up to five prior shifts.

The software is able to show data out of the following areas: orders, current production output, moulding presses, robots and quality assurance. These areas are presented per machine, shift and order. For example, the following picture (Fig. 5) shows tabs used to select a machine and an order running on that machine. The active (high-lighted) lights indicate the current status of the machine or order. The machine's status is always visible from everywhere in the application.

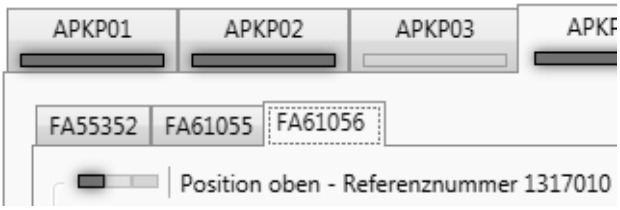

Fig. 5: User interface for selecting machines and orders

On two displays both raw data and charts are shown. Raw data represent information about an order running on a machine of the currently selected shift. For instance, they consist of simple or complex overview values and statistical calculations done in real-time with every single incoming trigger event. Among others, these values are the number of cycles processed by press and robot, combined duration of all alerts as well as standard deviation and midpoint calculations. The following screenshot (Fig. 6) shows the data calculated and summarized by all robot cycles of the selected shift.

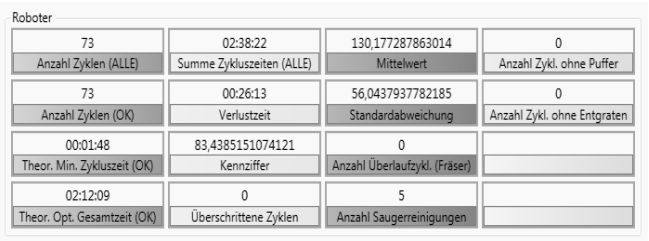

Fig. 6: Calculated and summarized robot data of the selected shift

On the contrary, charts do always show all cycles from the currently selected shift plus its previous shift. Cycles of both press and robot are displayed in the charts.

Additionally, defects registered at quality control are plotted at the quality chart. The basic layout of moulding press and robot charts is simple. On the X-axis date and time of the cycle is indicated. On the Y-axis the duration of that cycle is shown. Quality's chart layout is the same apart from the Y-axis, where the category-ID of the registered defect is applied instead.

Looking at the following picture (Fig. 7), the whole process works as follows:

One of many moulding presses starts producing an item. After the item production process is done, the moulding press control system writes an entry to the press's table of the central manufacturing information system database, which in turn fires the trigger connected to that table.

The trigger collects the data just written to the table by the moulding press control system and sends it to the service. The service forwards that event to every client application currently subscribed, where the cycle is displayed at the appropriate chart and the moulding press's raw data are updated based on the received event.

ISBN: 978-988-18210-5-8
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

In the meantime, that product item is carried to the next place of the production process: the robot station. The robot takes each product and starts deburring, for example.

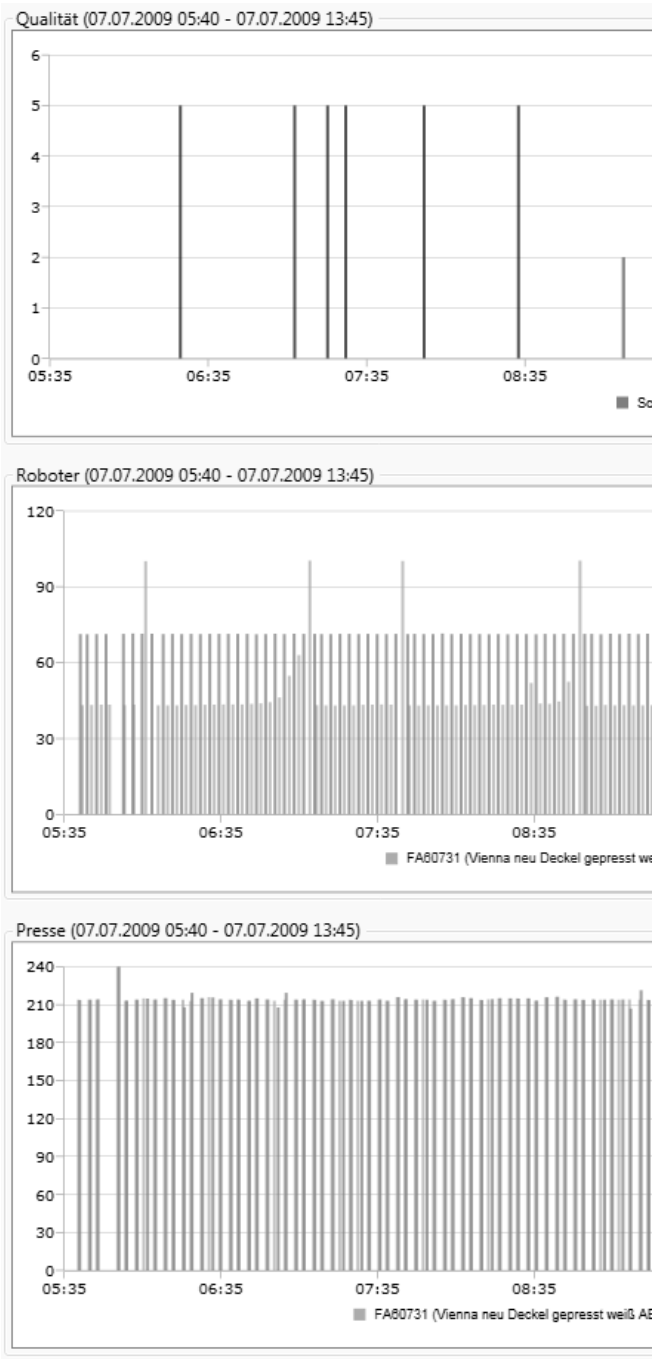

Fig. 7: Client data visualisation via one-to-many service events

After that process is done, the item is transferred to its final destination: the quality assurance testing station. At the same time, the robot control system writes this cycle's data (like cycle length etc.) to the robot's table of the database.

Again, the according trigger is fired and the data are transferred to the client. There, the robot's values and charts are updated.

Finally, the product is checked for defects at the quality assurance testing station. In case a defect is detected this will also be recorded at the database. There are several defect categories like "porous", "filth" or "defect caused by robot". Each of these categories is related to a category-ID.

Instead of the cycle's length the ID-value is shown in the chart when it comes to quality. As with each trigger event, raw values of quality are also updated in real-time.

IX. Conclusion and Further Research

The described visualization and monitoring application respectively the whole software system solution implements the presented service-oriented communication concept based on WCF.NET. It steadily delivers important information about the current manufacturing processes in real-time. The shift supervisor, who uses this software, can filter displayed data by machine, order and shift. This gives him the possibility to compare current production data with those of previous shifts and days just to come to a conclusion how well production went until now and how the machines behave with different settings and optimizations. Thus, it also becomes apparent how changes influence productivity across the complete manufacturing process. All this information also gives an indication of whether the order's deadline can be reached or not. Status lights allow the shift supervisor to check each machine's status (and thus its active orders) at a glance. When comparing the data with other supervisors, everyone is sure to talk about the same parameters.

The industrial application example demonstrates the introduced communication concepts availability for practical use. The presented structural design can easily be adapted to serve similar conceptual formulations.

Due to strict layer architecture of WCF.NET various kinds of transmission modes are supported. If necessary, a comprehensive extension model enables the development of own appropriate protocols.

Although the service-oriented communication concept was designed and the system itself (regarding the projects conceptual formulation) was finished respectively, improvements are still possible. One of them proposed here concerns the current implementation of WS-Discovery. As WS-Discovery is not yet included in .NET version 3.5 a third party library had to be used. However, it is recommended to migrate to .NET 4.0 as soon as it is released as it already implements WS-Discovery by default.

Acknowledgment

The authors are grateful to the Automation Team and the Information Technology Team of MKW® Austria for technical assistance as well as to Johannes and Hans Danner for inspiring ideas and creative discussions.

References

[1] Löwy, J., "*Programming WCF Services*", ISBN: 978-0-596-52130-1, 2nd Edition, Published by O'Reilly Media Inc., Sebastopol, USA, 2008

[2] Peiris, C. & Mulder, D., "*Pro WCF – Practical Microsoft SOA Implementation*", ISBN: 978-1-59059-702-6, Published by Apress Inc., Berkley, USA, 2007

[3] Löwy, J., "*What You Need To Know About One-Way Calls, Callbacks, And Events*", 2006, *Available from:* http://msdn.microsoft.com/en-us/magazine/cc163537.aspx

[4] Microsoft, "*WS-Discovery Specification Compliance*", 2009, *Available from:* http://msdn.microsoft.com/en-us/library/bb736562(VS.85).aspx

[5] WSDD, "*Web Services Dynamic Discovery (WS-Discovery)*", 2005, *Available from:* http://schemas.xmlsoap.org/ws/2005/04/discovery/