Tu Tran Le

GENERIS Work Order Management

Work Order Follow Ups Improvements

Helsinki Metropolia University of Applied Sciences

Bachelor of Engineering

Information Technology

Thesis

September 12, 2012

Author(s) Title Number of Pages Date	Tu Tran Le GENERIS Work Order Management 40 pages 12 September 2012
Degree	Bachelor of Engineering
Degree Programme	Information Technology
Specialisation option	Information System
Instructor	Sakari Seppälä, Team Manager, EDM Solutions Jaana Holvikivi, Principal Lecturer

This project was carried out for Process Vision Oy and the main goal was to find solutions for improving the current Work Order Management (a part of GENERIS application). As GENERIS is a standard information system for different energy markets, the current Work Order platform needs a more effective management interface. The process of following up with existing work orders needed to be improved, so that users could effectively manage all work order data within the system. The new user interface should be productive, user-friendly and it must follow a common design pattern.

Improvement ideas were collected based on existing requirements and analogue use cases. It was necessary to implement a prototype for Work Order Follow Ups module using the existing GENERIS Meter Asset Management platform. Since GENERIS is an energy information system that is built on the Windows operating system, the new implementation was done using the Microsoft Foundation Class framework and the C++ programming language. The prototype needed to be evaluated and tested against specific use cases using a test database.

The result of the project was the implementation of new user interfaces which could increase users' productivity in basic use cases. This thesis also analysed the existing workflow definition object in GENERIS platform, which resembled the work order object. There were certain limits which eliminated the possibility to develop new work order object based on the workflow definition object. However, a suggestion for a new data model was also described in this thesis.

Keywords	GENERIS, MAM, user interface, design pattern, data model, energy information system, C++ MFC, Oracle

Contents

Abbreviations and Acronyms

CIS	Customer Information System
EDMS	Energy Data Management System
GENERIS	General ENERgy Information System
GOF	GENERIS Objects and Fields
IT	Information Technology
MAM	Meter Asset Management
MDM	Meter Data Management
MFC	Microsoft Foundation Class
OOP	Object-Oriented Programming
RDBMS	Relational Database Management System
SQL	Structured Query Language
UI	User Interface
XML	eXtensible Markup Language
XSL	eXtensible Stylesheet Language

1 Introduction

Process Vision Oy is an IT company specialized in information systems and applications for the energy business. The company provides solutions for measurement data warehouses, which is a software product called GENERIS [1,1]. Process Vision Oy has focused on developing versatile solutions for the deregulated energy market targeted for distribution companies, energy retailers, balance coordinators and system operators. These solutions consist of wide measurement data warehouses, systems for balance settlement and balance management, data transmission functions and systems for contract and portfolio management [2,1].

In this thesis, an internal development project for Process Vision Oy was described. The main objective was to find improvement solutions for GENERIS Work Order Management, which is an integrated part of GENERIS Energy Data Management System. As GENERIS is a standard system solution for the energy market, it runs various business processes for different market parties. Each customer configuration has its own work order definitions with different sets of data fields. Due to the large number of work orders and their data fields, managing existing work orders effectively has become a challenging task. Thus, it was necessary to improve the current Work Order Follow Ups module, so that users could easily manage all existing work order data inside the system. The improvements needed to be user-oriented and satisfy all technical requirements of the GENERIS coding practice. The new user interface should also follow the current design pattern of GENERIS Browser and be easy to learn by all general users.

Moreover, the data model needed to be improved in such a way that it could support user-defined fields and data exchange between different systems. The new data model should be compatible with previously installed databases and the current function system. Another alternative is to re-use the existing data model of the Workflow definition object in the GENERIS platform. The possibilities of the new data model are analysed and discussed in this thesis.

The main tasks of the project involved implementing a more effective user interface for managing existing work orders and analysing different possibilities to improve the existing data model. The target of the study was a robust, version-controllable Work Order Management with an improved Work Order Follow Ups module based on the existing

Meter Asset Management (MAM) platform. All of the improvements needed to be made based on the analysis of the current system implementation and all analogue use cases in the process.

2 Theoretical Background

2.1 Energy Data Management System

The Energy Data Management System (EDMS) is a system that provides data warehouse solutions for energy markets. The system offers fundamental functionalities for managing and manipulating energy-related information using data-storage devices. An important role of EDMS is to integrate data from physical measurements into a time series format. This format is used to record consecutive measurement readings during a specific time interval. The information source may relate to different aspects of the energy business. The data can be about metering devices, their physical properties and reading measurements. The system can be used for different sectors of the energy industry such as gas, electricity and district heating. It also provides a communication channel between various market parties as well as customers. [3,1-2.]

An important requirement is that the system supports a daily routine in which a large amount of data is processed hourly. EDMS modules are designed to perform the tasks of network operators, balance coordinators and electricity retailers. Metering point-related information such as the network owner, retailer, validity time is also managed by the system. One important feature of a standard EDMS system is the support for advanced data processing operations such as validating, updating, calculating, reporting, invoicing and data transferring between different systems. Moreover, it should be possible to integrate existing data into other customer information systems (CIS). [3,3.]

2.1.1 Meter Asset Management (MAM) Platform

The Meter Asset Management (MAM) platform is an important part of GENERIS Energy Data Management System. It provides versatile tools for managing various aspects of meter-related data as well as data collection tasks for device assets. The GENERIS MAM platform provides advanced solutions for a wide range of energy companies, from a small local electricity company to a multinational network owner. Some of the most significant features of MAM are about asset management and meter reading task management [4,6].

The integrated modules of the platform consist of different utilities for managing energy devices and their data contents, which can satisfy the most significant requirements of an energy data warehouse. The main functions of the MAM platform involve managing metering device details, device data, creating reports, installation and maintenance tasks. As for the district heating and district cooling utility, the process of meter calibration and testing can also be arranged using the system. Some of the basic processes also involve customer-related services (such as registering a new customer and the change of metering configuration or retailer). Moreover, MAM integrates with scheduled remote data collection from an external data source such as a meter concentrator using the provided Application Programming Interface. [4.]

2.1.2 Work Order

Work order is an abstract unit of work requested by customers or a company's staff. Work orders usually take place at the customer side, which is equivalent to a service order in which the location, date and nature of the work are recorded [5,6]. In the energy market, there are many work order types which are related to different aspects of the meter asset management. Typically, the targets of work order involve individual or a site of multiple metering points and meter devices. By creating a new work order, energy companies can conduct various tasks on their meter assets, including meter change, meter readings gathering and estimation, meter configuration, maintenance, connection and disconnection. Work orders data is stored in a database and can be exported to different formats (paper, PDA or structured text file…). [5,8-10.]

GENERIS Work Order is the object type that is used to represent work order inside GENERIS MAM platform. It is used to create new on-site tasks relating to metering assets. Generally, the site where a work order is carried out is a metering point of a network operator. This involves installation, uninstallation, changing and maintenance of meter devices. The manual process of gathering the readings from meter devices can also be requested using a work order item. In brief, GENERIS Work Order object can be considered an interface for handling work order electronically. It can be exported to paper/electronic format as well as take input data from users and save it into the system. [5,12.]

2.2 Usability

In software engineering, the term usability refers to the ease of using or learning how to use an application from the user's point of view. Basically, software is also an object that requires human interaction. For instance, the user interacts with a Windows application by using its graphical user interface. Thus it is important to design the user interface using a generalized usability philosophy, which makes the software easier to learn. Good usability itself is the best practice to take a complex computer system into daily use. It concerns effectiveness, efficiency and satisfaction when the user accesses certain functions of the system in order to achieve specific goals. [6,4.]

The effectiveness of the system is defined by the extent to which the user's intended purposes can be fulfilled with accuracy and good performance. Meanwhile the term efficiency reflects the rate of speed at which users can complete specific tasks. If the users can achieve their targets faster and with less effort, the system is considered to have better efficiency. Efficiency also needs to be consistent across different parts of the system. Finally, user satisfaction is measured based on a user's attitude and perceived acceptability towards the system. It is the key factor that is used to evaluate the usability of a system. [6,7.]

Software with good usability is considered to be user-friendly and provide users with higher quality experience. In order to achieve good software usability, developers do not develop the software solely based on the oriented technology but with the intended users in mind. In other words, the interface is designed based on the user workflow, so that they can get the task done in the easiest manner. This method makes it possible to identify the needed functionalities as well as prevent any design flaw that might have occurred when the product is taken into production. [6,9.]

2.3 User Interface Pattern

In principle, a pattern is a common systematical feature that improves the habituation of the subject. It can be the user interface of a software application or the outlook of a product. The ultimate target of a pattern is to help users to understand the object more easily and have better experience when using it. Thus it makes the process more effective and usable. [7,3.]

The User interface (UI) pattern can be considered a concrete principle to build the user interface design from the ground up. It is a fundamental element and needs to be consistent across different environments on which the applications operate. However, the pattern only sets the standard for user interface. It plays the role of a basic guideline for designers to make the application interface self-consistent. Depending on the application context, the user interface design may differ from the design pattern, for example, to satisfy a special user's need or due to a technical requirement. [7,17.]

As a software project grows both in size and complexity, it starts to integrate more and more functionalities and user interfaces in the same application. For instance, a Windows application is typically a composite of different user interface toolkits. The user interface is built from a handful of simple controls (such as text fields, labels, images and buttons). Users might encounter different graphical interfaces in the same software application. This is the reason why the user interface pattern needs to be taken into consideration. In general, it is the structural and behavioural user interface that improves the habitual familiarity of end-users. A consistent design pattern helps users understand and adapt to the interface more easily within the same application domain. [7,31.]

A pattern can be built based on researching practical user experience. At first, the researcher needs to recognize the target user groups. Since each user group has a unique way of perceiving their surroundings, there is no single pattern which can satisfy the needs of all user groups. A design pattern may be compatible with certain users but not everyone. The ultimate goal of designing interface patterns is to identify the most common behaviour patterns. The researchers need to find out what is generally true and accepted by most users. In order to achieve this target, it is necessary to learn about different users and then remove the odds from the common pattern. Specifically, it is necessary to learn about the following topics from users' points of view:

- Why would users use the design? (their goals)
- What do users do with the design? (their tasks)
- How do users describe the design? (their language)
- How familiar are users with the design? (their skills) [7,17.]

It is important to establish reality grounding by focusing the learning process only on potential users. The learning method can be based on direct observation of on-site users or by conducting case studies and conducting surveys. All of the above methods are regarded as data collecting methods. Another approach would be to design a pattern based on an imaginary subject which would capture the most important aspects of all users in the target user groups. [7,7.]

2.4 Object-Oriented Programming

Object-Oriented Programming (OOP) is a concept used in the process of software development. The basic principle is to organize the application into classes and make the code reusable. In the older procedural programming method, the source code consists of a series of instructions that take place one after another. The developers use logical expression to determine the condition in which a procedure takes place and when the program ends. The same code can be considered to be a unique solution for certain problems only. Thus it is difficult to reuse the same code in different projects. Also in order to solve a large problem, it is necessary to break it into several smaller pieces and work on each of them separately as a single problem. This process needs to be repeated until each problem can be solved directly without further analysis. [8,6.]

However, this is not the case for the OOP approach. The basic concept used in OOP is the object, which contains both data variables and procedures. The analysis of a problem is basically a process of designing all needed objects. The result is a collection of objects with independent data state and methods. All objects can be allowed to interact with each other. This concept brings programming closer to problem solving in real life, which basically consists of many interactive objects. [8,11.]

The principles of OOP involve the use of classes, objects, inheritance and polymorphism. The class acts as an abstract definition for building up object instances. The class may contain data members (or variables) and functions. The processing of an application is based on invoking different sets of functions inside classes and objects. Moreover, each class can be re-used and extended through the use of inheritance and polymorphism. These two concepts provide reusability and extensibility for an application. Inheritance is the process of creating a new class based on an existing class. The sub-class can inherit all data members and methods from the base class (except for those that are specifically not to be inherited). Since different problems may have similar objects, this approach makes it possible for developers to reuse the code and reduce the effort in implementing the solution. It is also easier to maintain the application code since modifying an object can have the same effect on different solutions for different problems. [8,11-13.]

The term polymorphism refers to a unique feature in OOP where a class can have multiple method definitions with the same name and different signatures (return type or parameter list). In other words, the same method will behave differently depending on the context. This concept brings OOP a step further toward solving programming requirements just like in the real-world counterparts. [8,13.]

2.4.1 Modular Software Design

Large software systems are more complex to develop and maintain than smaller systems. They tend to be difficult to modify or extend even though the needed change is simple and applies to only a small part of the application. As the software system grows both in size and complexity, it takes much more time for developers to identify a specific block of code and its effect on other portions of the whole system. A single change may require modifying also other parts of the system. In the worst case, the loop repeats endlessly. The design starts to degrade because the requirement changes in such a way that the initial design was not able to handle it. This results in a system that is impossible to maintain. The requirements are the most volatile in the software life cycle. Thus, it is necessary to decide on an agile approach for the initial system design. [9,2-7.]

The concept of modular software design involves designing a large application system consisting of multiple smaller software modules. Each module has its own function and should not have too many dependencies on other modules. By organizing an application into smaller independent modules, a change that comes to a module will not have any effect on other modules, thus unexpected break would not be likely to occur inside the whole system. This design makes software developing become a process of extending application instead of modifying the same existing code base. [9,8-11.]

In modular software design, higher-level modules should not depend on lower-level modules. In other words, abstractions should not depend on detailed implementations and detailed implementations should always be derived from abstractions. All software modules can communicate using a generic interface definition. This approach makes the software system more sophisticated, easier to maintain and more extensible. It keeps the design of the system simple, clean and effective, no matter how much extensions have been added to the software system. Thus reusability, extensibility and maintainability are important factors which can be achieved by using this design approach. [9,10.]

2.4.2 Microsoft Foundation Class

The Microsoft Foundation Class (MFC) is an extension of C++ which includes a set of library that provides wrappers for the Microsoft Windows Application Programming Interface (API) in C++ classes. The purpose of this library is to simplify the task of designing and developing the Windows application. The tasks of creating and managing application windows can be done through MFC classes and thus, eliminate the need for calling Windows API directly. MFC was equipped with a set of macros and dynamic classes that can be used to handle Windows messaging mechanism, exception and serialization. [10,7-8.]

2.5 Relational Database Management System

Database represents the persistent data inside storage devices such as local hard drives. The organization of the data inside the device is managed by a database management system. Basically this management system is an application that handles the file abstraction layer. It provides other applications and services with an interface for defining and manipulating data. The format of stored data can simply be text or binary data such as image, sound and application data. [11,7.]

The Relational Database Management System (RDBMS) is a concept of representing data in the database system. Data is organized as a set of formally described two-dimensional tables. The table contains columns which represent different fields for a data record (data row). The role of a relational table is to represent an object entity. Each table can have its own attributes, data fields, constraints and connections to other tables. The data amongst different tables may have a relationship with each other. There is a wide range of special objects inside the database which enforce certain data constraints to be valid. Some of them involve the concepts of key, check and constraint. Each RDBMS has its own language for querying data. The Structured Query Language (SQL) is the most common language that is used by many database systems available today. [11,8-13.]

The process of designing a relational database structure starts with modelling the data as entities or objects. It is also called as data modelling process. Each data entity and its relationship can be expressed as a database table, whereas the list of columns is generated based on the attributes (or data fields). The role of modelling is to represent the nature of data graphically and formally. It gives the developer a better view of the real nature of information and the needs for processing data in a specific area. There are several types of modelling data, some of which are relational model, entity relationship model and object-oriented model. [11,20.]

As the hardware has become much faster and the need for effective data management has increased significantly, the relational database has become a popular concept in software development nowadays. Currently, there are a large number of RDBMS alternatives available on the market. Some of the most common names are Microsoft SQL Server, MySQL, IBM DB2, PostgreSQL and Oracle. Specifically, the GENERIS system uses Oracle Database for its data storage and manipulation operations.

2.6 Data Model

The Data model is a method for describing and storing information inside data storage resources according to a specific set of system requirements. In other words, the data model plays the role of data representation for information requirements inside a system.

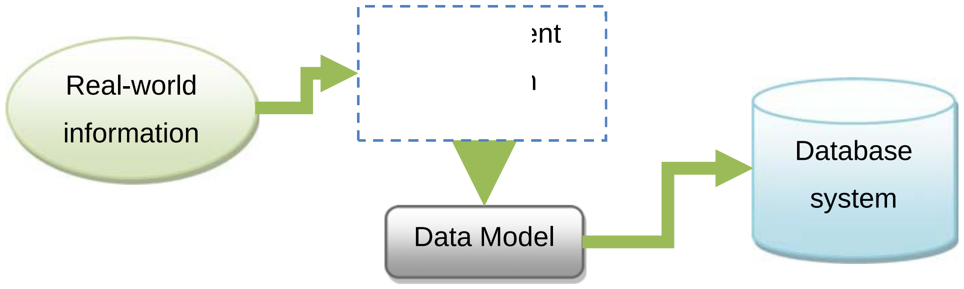


Figure 1. Role of data model in information storage

Figure 1 illustrates the role of data model in the process of storing information into database system. The data model acts as a blueprint for recording the contents of data elements based on the scope of various business processes [12,5]. In other words, the data model can also be considered a representation of the data structure.

3 GENERIS Work Order

3.1 GENERIS Platform

GENERIS stands for General ENERgy Information System, which is an information system platform that provides solutions for a wide range of parties and business processes in the energy sector. The main elements of GENERIS consist of binary files and an instance of the relational database (provided by Oracle RDBMS). Typically, users will interact with the system using a main application named GENERIS Browser and all user data will be stored inside the database. GENERIS is the platform for developing different modules targeted at different aspects of a multi-utility energy information system. Each module has its own set of Windows binary image and database installs. Depending on the licenses installed for the system, the corresponding modules will be installed together with GENERIS.

The current development of GENERIS has provided a possibility for managing balance settlement, billing, contract, portfolio, meter data, meter asset and data validation. Each GENERIS module handles certain parts of the data management system and owns specific sets of GENERIS objects and fields. There are two main modules in GENERIS which are responsible for Meter-related data management:

- Meter Data Management (MDM): The system has been developed for a multi-utility environment to centrally manage all commodities and other types of measurements including power quality and weather conditions. It is in production use for the management of main and sub-metering information of electricity, gas, district heating/cooling, water and solar power. The MDM system handles a large volume of data simultaneously through a highly-optimized processes and effective calculations, which results in a stable and high-performance system. Moreover, the system also provides flexible and advanced search tools which satisfy the need of the smart metering business. [13,2.]

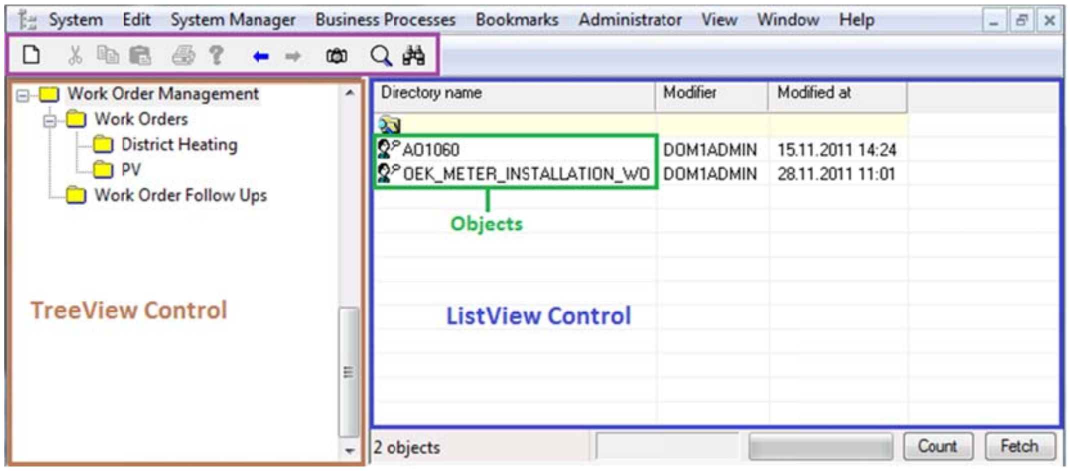
- Meter Asset Management (MAM): The module consists of various tools for managing metering devices. It covers the whole life-cycle of each metering device starting from purchase, storage, installation, maintenance and retirement. Meter reading data can also be collected using traditional methods such as recording data on papers [4.12]. Moreover, the MAM platform offers follow-up and reporting tools for metering-related tasks.

Though each module is designed for specific processes in the energy sector, they can be connected together using the GENERIS Objects and Fields (GOF) system. This system plays the role of an abstract data access layer inside the GENERIS platform. It provides a dynamic way of describing data without specifying the database access details inside the source code. This design offers a flexible structure for organizing data in the form of objects and fields, which simplify the process of searching, accessing and manipulating data in the database.

The GENERIS platform also integrates the access control system through built-in security policies. Each module may have its own security policy, which defines the permissions for GENERIS users. The permissions are enforced for general reading, writing of GENERIS objects as well as workflow definitions in specific processes.

3.2 GENERIS Browser

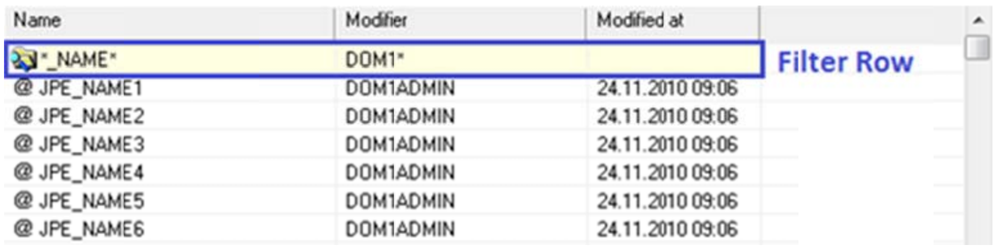
The GENERIS Browser is the main entry-point application for all basic GENERIS end-users. The browser was developed for the Windows operating system and has the layout of Multiple Document Interface (MDI), where users can open multiple child windows under the main application window. The user interface design is quite straight-forward and provides users with direct access to different sections. It contains a main navigation menu, a toolbar, a main browser area, which consists of a tree view control and a list view control. The main screen layout of GENERIS Browser is illustrated in figure 2.



Directory name	Modifier	Modified at
A01060	DOM1ADMIN	15.11.2011 14:24
OEK_METER_INSTALLATION_WO	DOM1ADMIN	28.11.2011 11:01

Figure 2. Main screen of GENERIS Browser

As illustrated in figure 2, the Tree View control on the left allows users to navigate to different parts of the system. The top-level folder hierarchy of the tree is constructed based on the installed application modules inside the system. Each module may have more than one folder node, and below each node users are allowed to create a sub-node for their own purposes. In most cases, whenever users select a folder node from the Tree View, the list of objects will be updated to the list view control on the right. The list view shows the object detail (which is by default, the object name, last modifier and last modified date). The list of detail column can be customized according to each user's preference. Users can view the details of the object by simply double-clicking on an object row. Moreover, users can also apply a filter on the list view control. The filter can be in text, number or date-time format, depending on the column data to which the filter applies.

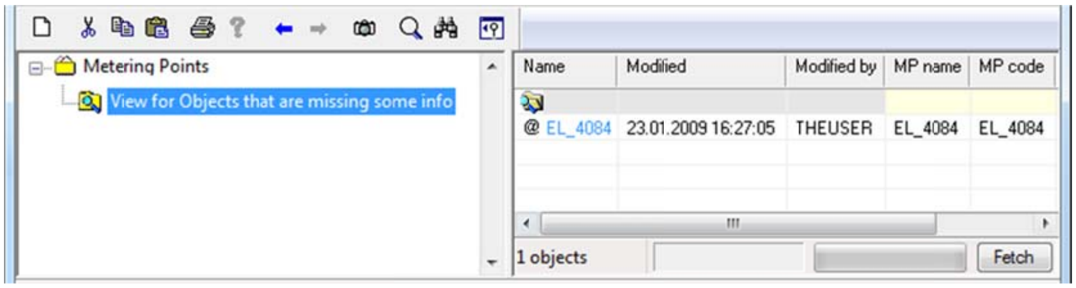


Name	Modifier	Modified at
_NAME	DOM1*	
JPE_NAME1	DOM1ADMIN	24.11.2010 09:06
JPE_NAME2	DOM1ADMIN	24.11.2010 09:06
JPE_NAME3	DOM1ADMIN	24.11.2010 09:06
JPE_NAME4	DOM1ADMIN	24.11.2010 09:06
JPE_NAME5	DOM1ADMIN	24.11.2010 09:06
JPE_NAME6	DOM1ADMIN	24.11.2010 09:06

Figure 3. Filtering of objects inside list view control

Figure 3 illustrates a simple use of the filter based on the object name and its modified date. The text-based filter also accepts wildcard characters such as the asterisk and question mark character. When users want to filter based on the modified date, a popup window will appear and allow users to select the desired date and time.

In principle, each tree node (folder) represents a specific object type (which is technically a C++ class), thus the list of objects will only contain items that belong to the same type. The list of object types is constructed based on the binaries of licensed application modules during the GENERIS installation process. Moreover, it is also possible to create an object view instead of a subfolder inside the tree view. A GENERIS view works using the same principle as a database view. It acts as a filtered list based on user-specific search criteria.

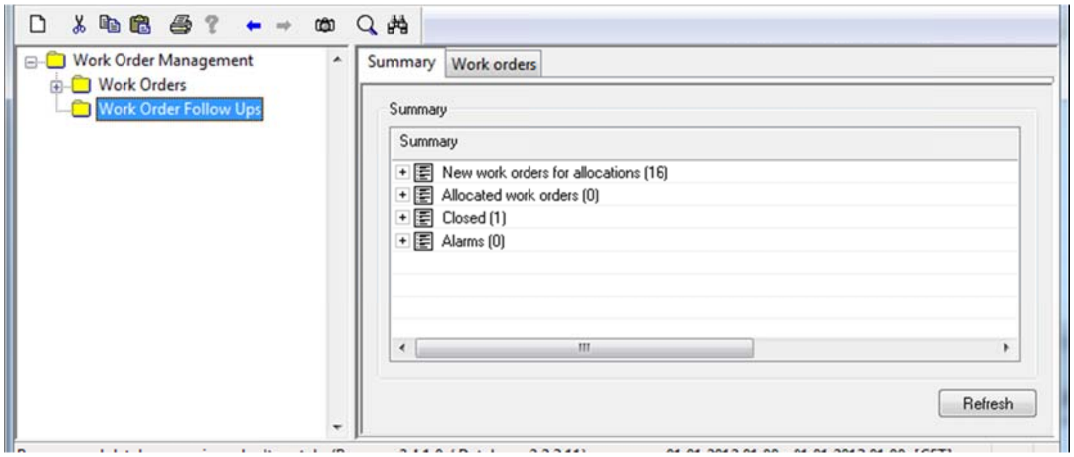


Name	Modified	Modified by	MP name	MP code
@ EL_4084	23.01.2009 16:27:05	THEUSER	EL_4084	EL_4084

Figure 4. An object view in the GENERIS Browser

Figure 4 illustrates the appearance of an object view in the tree view hierarchy. When users select the object view, the list of objects matching the search criteria specified in the view will be fetched into the list view control.

In some cases, when users select a folder from the Tree View, instead of showing a list of objects, the list view will be replaced by a generic panel section. The content of the panel depends on the implementation of the specific module. This design provides flexibility for the user interface design and allows developers to reject the general design pattern in order to satisfy the system requirements. Figure 5 illustrates the content of the GENERIS Work Order Follow Ups module, which is represented by a panel control instead of a list of GENERIS objects.



New work orders for allocations (16)
Allocated work orders (0)
Closed (1)
Alarms (0)

Figure 5. Content of the Work Order Follow Ups module

According to figure 5, when users select the Work Order Follow Ups node, the GENERIS Browser will present a custom-implemented interface instead of a list of the objects. This module allows users to easily manage the list of existing work orders inside the system based on some key criteria. It also provides users with a possibility to quickly view the content of a work order without the need to open a new window. However this module can still be improved to enhance user experience, which will be covered in section 4 of this thesis.

3.3 GENERIS Work Order Object

Work orders are created to request operations on meter devices. The type of a work order depends on the operation that needs to be carried out. The current GENERIS MAM platform supports all necessary workflow types that are used in the energy industry. Some of the most common work order types are maintenance, manual area meter reading, meter change, meter random test, meter configuration, installation, connection and disconnection.

Work order objects can be created under a sub-folder named Work Orders. Basically, a work order object also has connections to other object types such as metering points and metering devices (since work order is often created for a meter device at a metering point).

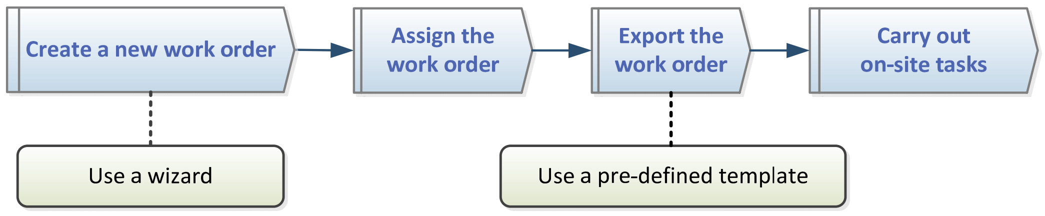


Figure 6. Life cycle of a work order object

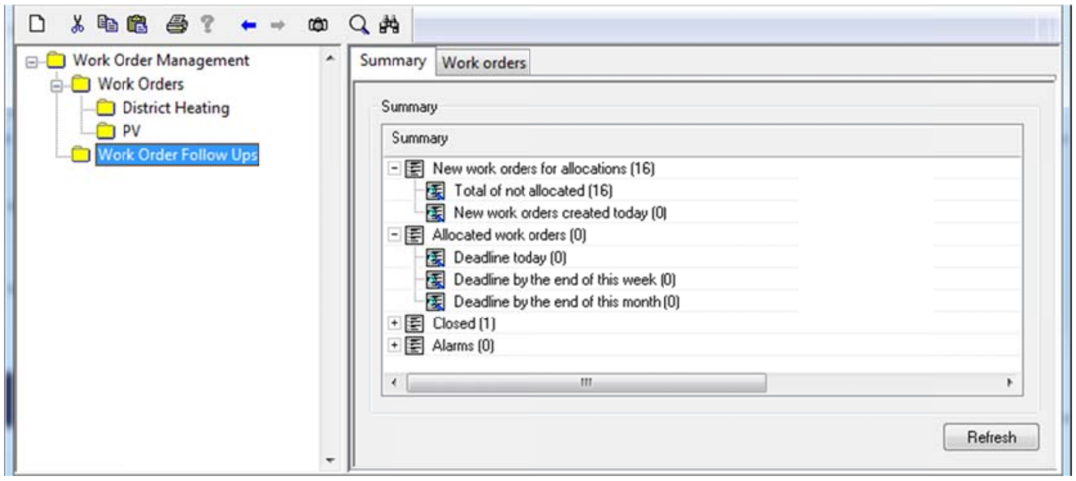
Figure 6 shows a simple lifecycle of a work order object. At first, users can create a new work order object using a wizard. It is necessary to specify all details related to the work order, such as work order type, work order code and description. A work order type may have different sub-types (in the case that there is more than one way of carrying out the selected work order, and the sub-types can also be used for different external systems). For example, in the case of creating a new work order for changing a meter device, there are sub-types which specify the format of the work order (either using paper or an electronic PDA device). Each work order type will require a different set of work order data fields, though all work order types will have a certain number of data fields in common. Depending on the work order type, users might need to specify related object definitions, which are usually metering points or meter devices. If a work order is related to multiple meter devices or metering points, the work order is considered to have more than one task associated with it. It is also possible to attach files to a work order if needed.

The work order object is stored in the database management system. However in order to issue an on-site work order, it is necessary to export the work order object into a specific output format (paper or an electronic document to be used on PDA). This is why each work order needs to have a data template. The data template might be an XSL file (eXtensible Stylesheet Language) or an Excel template. This configuration is very flexible since all customers can have their own data template and design for all exported work orders.

After work orders have been stored in the database, users can view, edit, export and monitor the state of the work orders. Each work order object might contain more than one work tasks, and each task has its own assigner, deadline, comments and multiple states. A work order task status is represented by its state property. Users can also attach optional files to a work task (for external meta-data). Whenever there is a need for collecting inputs for work orders (either by using automatic data import or manual data input), the data is validated and saved into the system using the master data management system (GENERIS EDMS). For example, when there is a need for meter change at a certain metering point, the end readings will be validated and saved for the old meter. Then the old meter will be moved to storage and the new meter will be installed to a metering point from the storage with start readings. In the case of validation errors, such as invalid data values, data inconsistency, users will be notified and required to take manual actions.

3.4 GENERIS Work Order Follow Ups

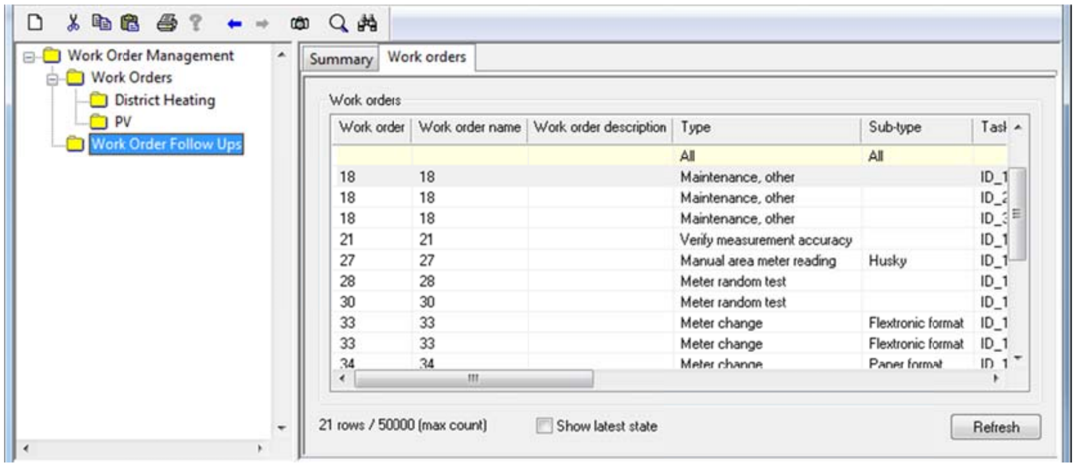
As the amount of data grows both in size and quantity, following up with existing work order becomes a challenging task. Thus it is necessary to use GENERIS Work Order Follow Ups to keep track of existing work orders. GENERIS Work Order Follow Ups can be found as a subfolder under Work Order Management in GENERIS Browser.



New work orders for allocations (16)
Total of not allocated (16)
New work orders created today (0)
Allocated work orders (0)
Deadline today (0)
Deadline by the end of this week (0)
Deadline by the end of this month (0)
Closed (1)
Alarms (0)

Figure 7. GENERIS Work Order Follow Ups Summary

Figure 7 shows the location of GENERIS Work Order Follow Ups module in a sample GENERIS installation. In this case, the Summary tab gives users an overview of current work order situation inside their GENERIS Installation. The filtering criteria are based on the most useful use cases from the user’s point of view. It focuses on all work orders that need special attention (work orders which are late or not assigned to any employee). When users navigate to Work order tab, they will get the details of all work orders that are currently created in the database, as illustrated in figure 8.



Work orders

Work order	Work order name	Work order description	Type	Sub-type	Tasl
			All	All	
18	18		Maintenance, other		ID_1
18	18		Maintenance, other		ID_2
18	18		Maintenance, other		ID_3
21	21		Verify measurement accuracy		ID_1
27	27		Manual area meter reading	Husky	ID_1
28	28		Meter random test		ID_1
30	30		Meter random test		ID_1
33	33		Meter change	Flextronic format	ID_1
33	33		Meter change	Flextronic format	ID_1
34	34		Meter change	Paper format	ID_1

21 rows / 50000 (max count) ☐ Show latest state

Figure 8. The tab view of GENERIS Work Order Follow Ups

Figure 8 illustrates the behaviour of GENERIS Browser when users navigate to the Work order tab in Work Order Follow Ups. The list view control will then be populated with a list of all work order tasks together with all of their states up to present, users can specify to view only the latest state. This view is helpful in case users want to follow up with the recent activities of the work orders.

3.5 Existing Workflow Definitions

The work order is an object type which has similar properties to an existing object type that is named as workflow definition. This object type belongs to the GENERIS System application. Instead of having specific data fields, a workflow has a more generic declaration which is in the form of parameters. A workflow definition object represents a work process which consists of multiple phases and can contain parameters. Each parameter can be simply a text or even a GENERIS object type (which is identified by its own unique identification code).

Users can create multiple instances based on the same workflow definition. Each workflow definition can define multiple application services, where the actual tasks are being done. The Workflow Management platform also offers a workflow group object type, which acts as a follow-up panel for specific workflow definitions. In other words, users can manage the status of all instances created from specific workflow definitions by using a workflow group object.

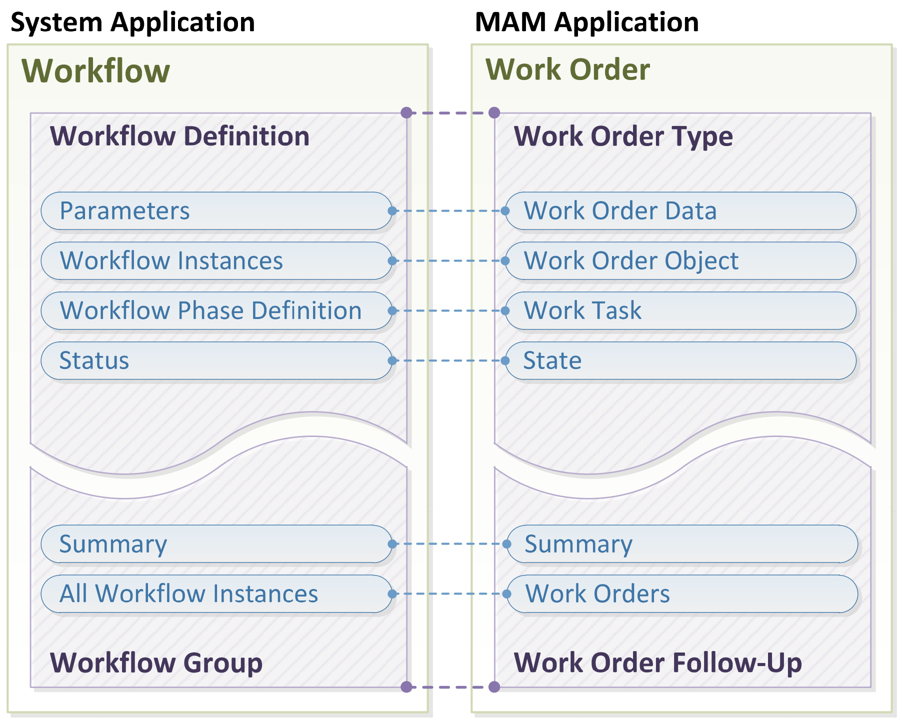
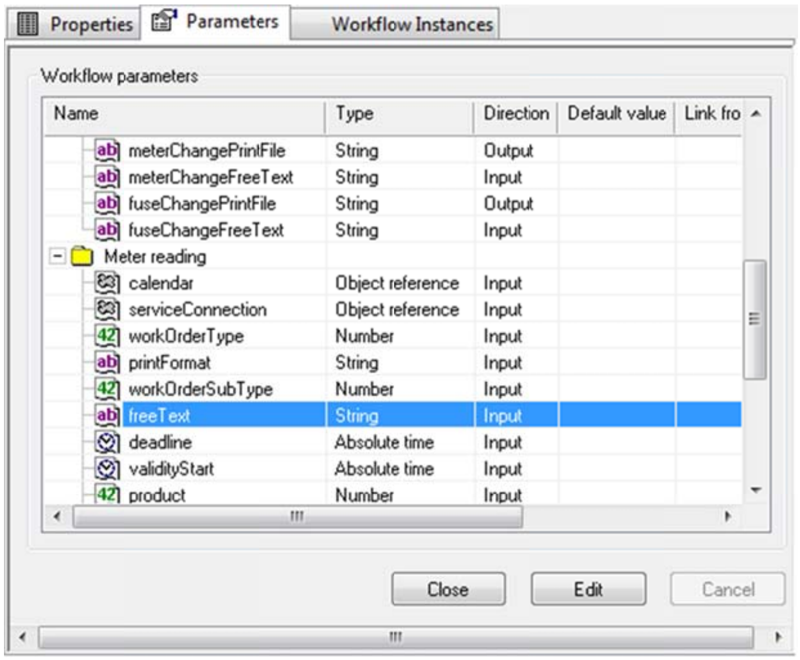


Figure 9. Similarity between GENERIS Workflow Management and Work Order Management

Figure 9 illustrates the similarity between Workflow Management and Work Order Management. In fact, the workflow definition can be considered as the abstraction of work order type. Thus a workflow definition can be instantiated as a work order object. Provided that workflow definition can accept parameters of different data types, it is possible to create work order data as workflow parameters. In this case, the metering device, metering point or owner party can be considered an object parameter of the workflow definition. Moreover, the workflow group object also offers the same functionality as the existing Work Order Follow Ups. It opens a new possibility to develop the current Work Order Management based on the existing Workflow Management in GENERIS. This can significantly reduce the complexity of the system and decrease the number of code lines by reusing the existing codebase.

However, there are certain limitations with Workflow Management that prevented this possibility from coming into practice. First of all, workflows are more suitable for larger processes such as contract managements. A workflow sub-process will be more suitable for the whole process of a work order. Moreover, the current workflow management does not have the interface for exporting/importing data into the existing workflow instances. Since work order is not a static object and it requires data exchange in a bidirectional manner, this problem naturally becomes the biggest obstacle in the process. Users need to export work order data to a custom format and then in some cases, import new data into the existing work order object. In other words, workflow instance can only be used as a work order object unless its interface has support for external dataflow. Moreover, a Work Order platform will be installed on different customer systems with a wide range of custom data fields. The current Workflow Management Configuration is not flexible enough to handle that requirement either.



Name	Type	Direction	Default value	Link fro
meterChangePrintFile	String	Output		
meterChangeFreeText	String	Input		
fuseChangePrintFile	String	Output		
fuseChangeFreeText	String	Input		
Meter reading				
calendar	Object reference	Input		
serviceConnection	Object reference	Input		
workOrderType	Number	Input		
printFormat	String	Input		
workOrderSubType	Number	Input		
freeText	String	Input		
deadline	Absolute time	Input		
validityStart	Absolute time	Input		
product	Number	Input		

Figure 10. Configuration for workflow definition parameters

Figure 10 illustrates a sample configuration for workflow definition parameters. In the current implementation, the list of parameters is represented by a two-dimensional list view control, which shows a parameter in a single row. This user interface design is not effective in the case that the parameter list view contains many rows (there is no sorting or filtering available in this list view control). It is also necessary to have the possibility to specify the custom state for a work task and security policy for each workflow definition parameter.

3.6 Roll-Back Feature

GENERIS Work Order is a robust platform which can handle all basic use cases in the energy market. It also supports the validation task for meter readings from the metering point. However, the current system does not allow users to undo the changes that were made to work order data. The process of reverting data into its original state due to human errors requires several manual steps, in the case of input errors, misspelling or data inaccuracy.

It is necessary for the Work Order platform to provide support for the roll-back feature. It can be used to restore system data to its previous state in the case of data errors. Moreover, it needs to enforce data integrity requirements during the whole process. In some cases, a roll-back can cause the existing data to invalidate itself. Thus, the roll-back operation should ensure that the data is valid before and after the process is completed. For instance, when the target work order is already too old, the rolled-back data state may overlap the new data in the system.

3.7 Binary XML Data Model

As of the current system, user fields are specified inside the source code and needs database scripts for making updates. This process is time-consuming and thus ineffective in the product cycle. Moreover, the management of license and version updates will become complex as the number of versions increase and the size of the database starts to grow.

One possible approach is to design a new data model which is based on the binary XML (eXtensible Markup Language). In general, the binary XML format creates smaller XML documents by omitting the common full-text syntax structure and representing data using a certain binary data format. The result is a document which is faster to parse but impossible to read by ordinary text editors. By making use of binary data, the size of an XML document is greatly reduced at a price of non-readable text for end-users [15]. Thus, it is suitable for performance purposes. It is well supported in many database systems and can also be used for importing/exporting purposes. Another advantage of the binary XML is that it can also replace the traditional means of updating the database using SQL scripts. By migrating into XML, different user fields from customer setups can also be put under version control. Moreover, an XML document can be easily validated against document schema and it is very flexible due to the possibility of XSL transformation [16,13]. Thus, the task of importing/exporting data/configurations becomes much more simplified and effective.

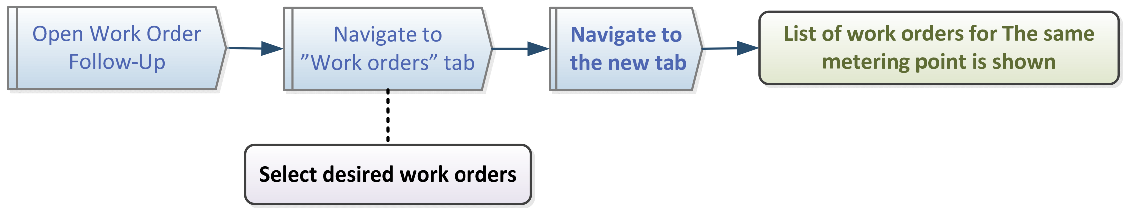
However, it is also important that the migration process needs to be backward compatible. This is to make sure that new implementations will not corrupt existing user data or requires a complicated migration process. Thus a new data interface should be implemented in such a way that it can support not only the new data model but also the old database system. The existing hard-coded user fields need to be revised according to the new data model using database updates.

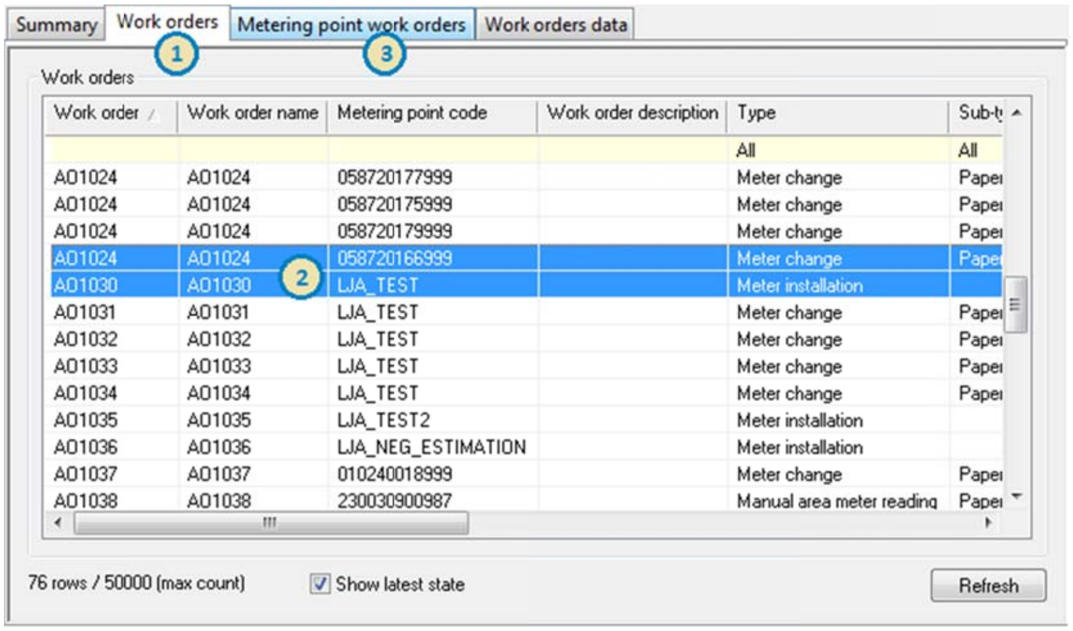
4 Work Order Follow Ups Improvements

4.1 New View for Related Work Orders

Currently, each Work Order object is presented in Work Order Follow Ups as an independent unit. However this is not the real case in practice, since different work order types may have certain connections with each other. Provided that there are multiple work orders for the same metering point (or meter device), users might need to view all work orders grouped by a metering point or meter device. For instance, when users view event-based reading work orders for moving/supplier change, they might want to view other work orders that belong to the same metering points, such as work orders related to a meter change operation. This can help users to arrange the on-site tasks in a more effective manner, for example, if there are two work orders about meter reading and meter change which relate to the same metering point, they should be carried out at the same time.

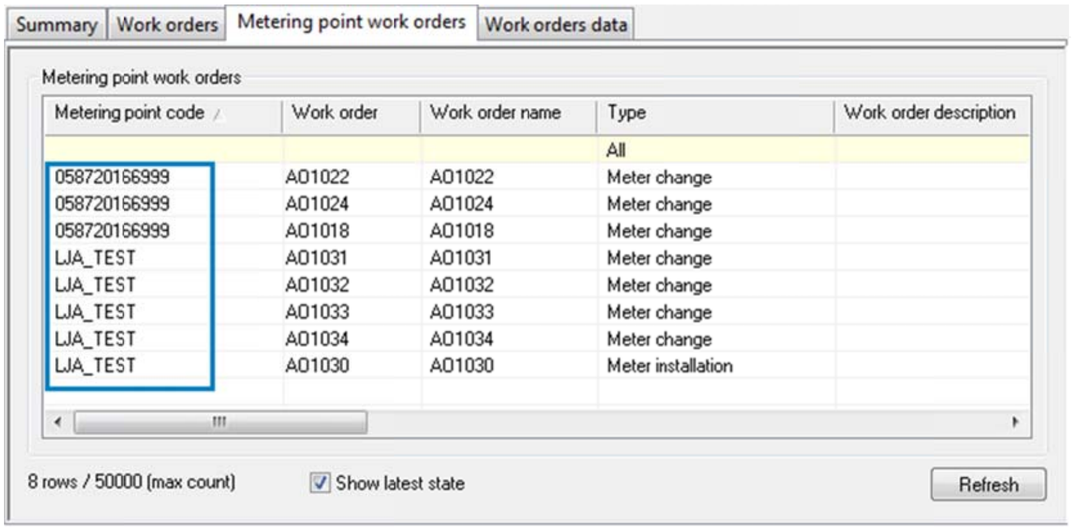
One possible solution is to allow users to select some specific work orders from the Work order tab. Then they can navigate to a new tab and the list of all related metering points will be shown on the screen.


Open Work Order Follow-Up
Navigate to "Work orders" tab
Navigate to the new tab
List of work orders for The same metering point is shown
Select desired work orders



Work order	Work order name	Metering point code	Work order description	Type	Sub-t
				All	All
A01024	A01024	058720177999		Meter change	Paper
A01024	A01024	058720175999		Meter change	Paper
A01024	A01024	058720179999		Meter change	Paper
A01024	A01024	058720166999		Meter change	Paper
A01030	A01030	LJA_TEST		Meter installation	
A01031	A01031	LJA_TEST		Meter change	Paper
A01032	A01032	LJA_TEST		Meter change	Paper
A01033	A01033	LJA_TEST		Meter change	Paper
A01034	A01034	LJA_TEST		Meter change	Paper
A01035	A01035	LJA_TEST2		Meter installation	
A01036	A01036	LJA_NEG_ESTIMATION		Meter installation	
A01037	A01037	010240018999		Meter change	Paper
A01038	A01038	230030900987		Manual area meter reading	Paper

76 rows / 50000 (max count) ✓ Show latest state



Metering point code	Work order	Work order name	Type	Work order description
			All	
058720166999	A01022	A01022	Meter change	
058720166999	A01024	A01024	Meter change	
058720166999	A01018	A01018	Meter change	
LJA_TEST	A01031	A01031	Meter change	
LJA_TEST	A01032	A01032	Meter change	
LJA_TEST	A01033	A01033	Meter change	
LJA_TEST	A01034	A01034	Meter change	
LJA_TEST	A01030	A01030	Meter installation	

8 rows / 50000 (max count) ✓ Show latest state

Figure 11. The new view for work orders belonging to the same metering point

Figure 11 illustrates the new workflow for users when they want to view related work orders. This feature requires the possibility to select multiple work orders at the same time from the Work orders tab. In this case, from the Work orders tab, users select data rows for two work orders with code AO1024 and AO1030 (these two work orders are related to two different metering points). Then by navigating to the new tab named Metering point work orders, the list of all work orders related to the same metering point will be shown on the screen. This workflow can also be modified to become even more flexible, as illustrated in figure 12.

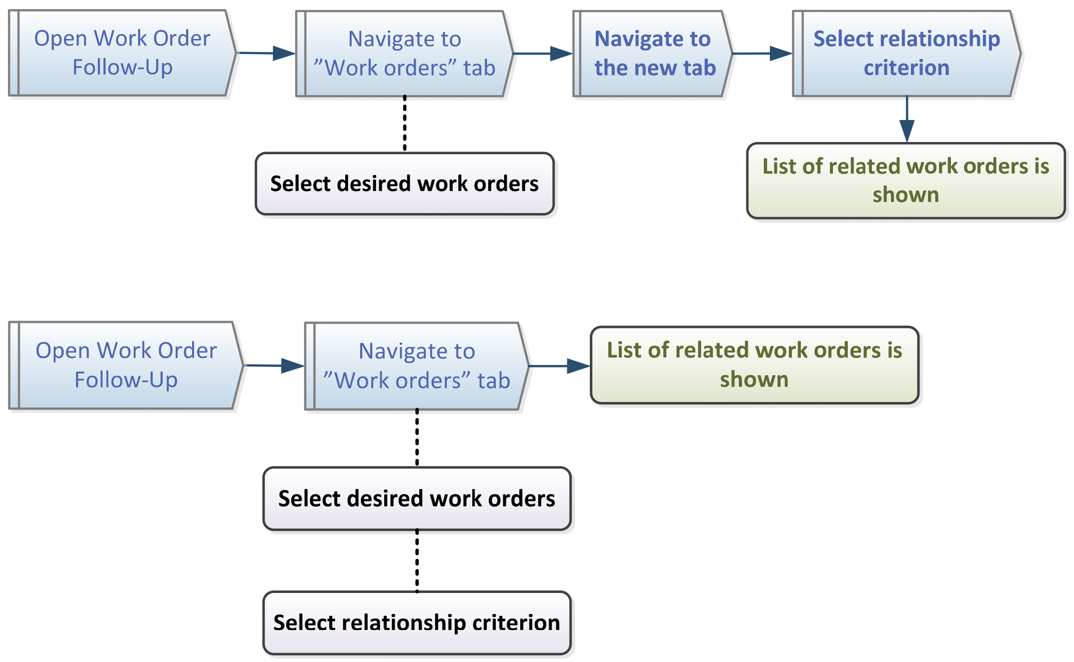
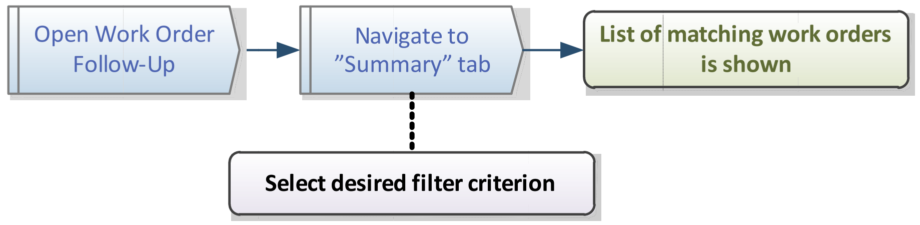

Open Work Order Follow-Up
Navigate to "Work orders" tab
Navigate to the new tab
Select relationship criterion
Select desired work orders
List of related work orders is shown
Open Work Order Follow-Up
Navigate to "Work orders" tab
List of related work orders is shown
Select desired work orders
Select relationship criterion

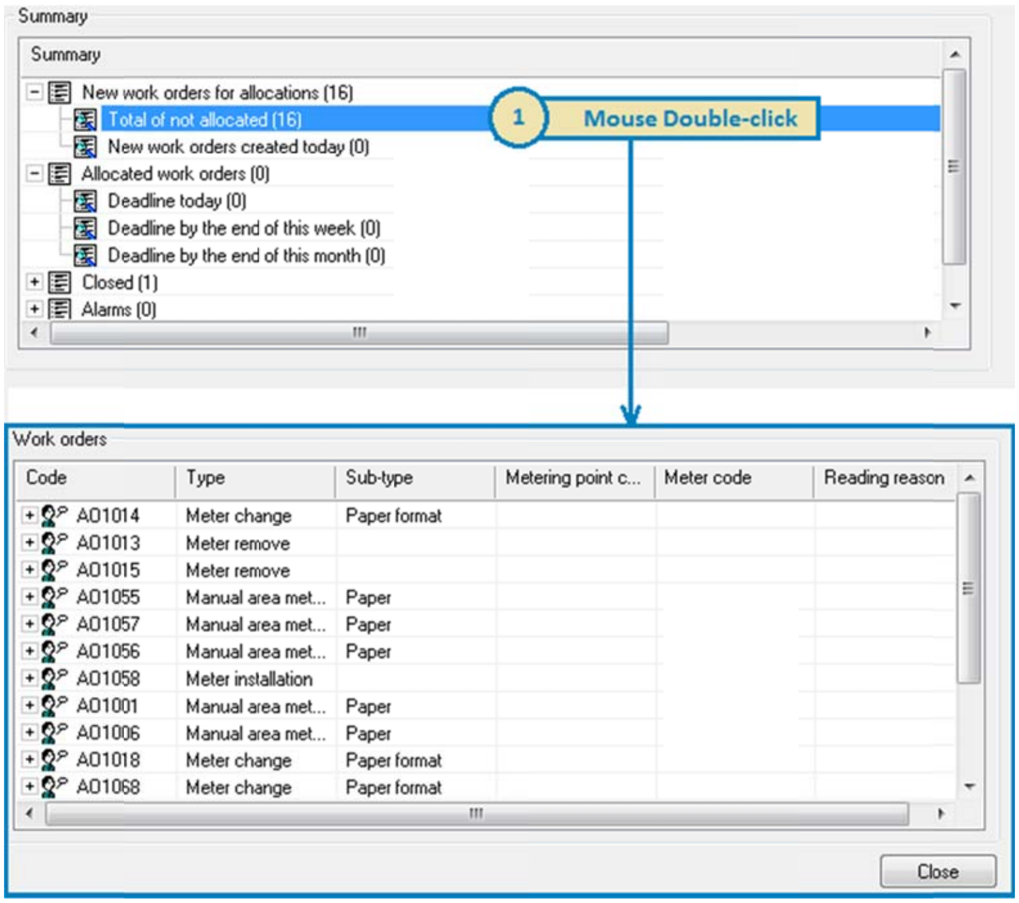
Figure 12. User workflows for viewing related work orders

Figure 12 illustrates another approach for the same purpose with even greater flexibility. Instead of showing only related work orders which belong to the same metering point, users can also view related work orders based on the selected work order data. For example, users can view all other work orders for the same meter device or metering point. The user interface should have a new combo box which contains a list of available data fields for the work order. By selecting the corresponding field name from the combo box, GENERIS Browser will load the list of work orders which are related to the selected work orders by that field. The list of the related fields can be populated from the GOF system instead of specifying list items directly in the C++ code. Whenever users select another item in the combo box, the content of the list view will be updated accordingly. The combo box can be placed directly inside the existing tab and all related work orders will be grouped together by using unique background colours for each data row. It is also possible to place the combo box inside the new tab, so that the new list view would only contain the work orders that the users are interested in.

4.2 Filtering Support for Summary Tab

Currently, the view of Summary tab in the Work Order Follow Ups module lists all pre-defined search conditions. By double-clicking on the desired filter name, the list of work orders which pass the filter condition will be shown to users.


Open Work Order Follow-Up
Navigate to "Summary" tab
List of matching work orders is shown
Select desired filter criterion



- New work orders for allocations (16)
  - Total of not allocated (16)
  - New work orders created today (0)
- Allocated work orders (0)
  - Deadline today (0)
  - Deadline by the end of this week (0)
  - Deadline by the end of this month (0)
- Closed (1)
- Alarms (0)

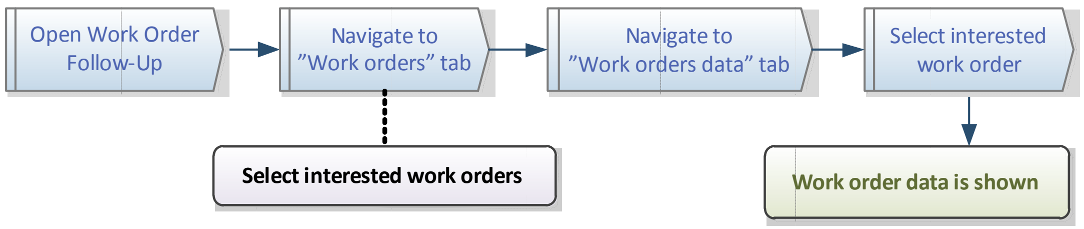
Code	Type	Sub-type	Metering point c...	Meter code	Reading reason
A01014	Meter change	Paper format			
A01013	Meter remove				
A01015	Meter remove				
A01055	Manual area met...	Paper			
A01057	Manual area met...	Paper			
A01056	Manual area met...	Paper			
A01058	Meter installation				
A01001	Manual area met...	Paper			
A01006	Manual area met...	Paper			
A01018	Meter change	Paper format			
A01068	Meter change	Paper format			

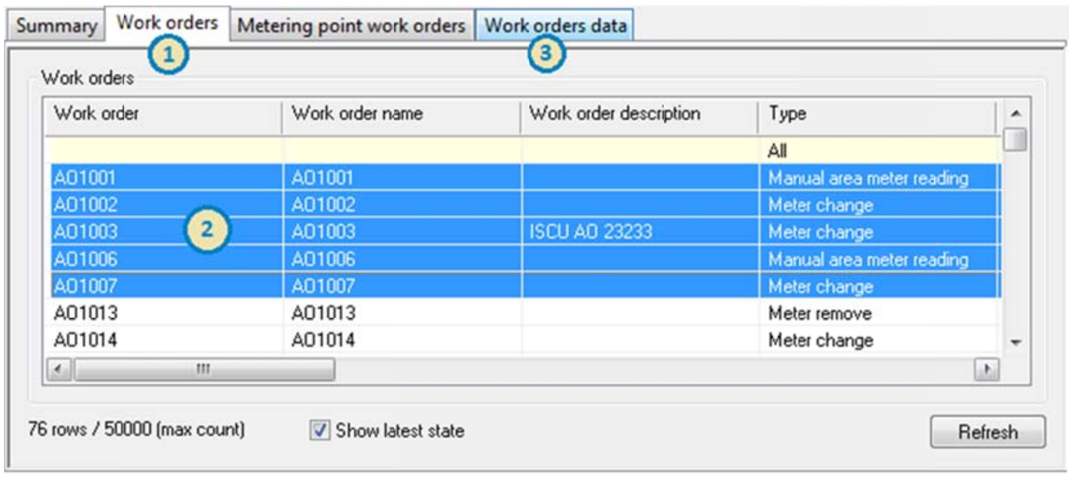
Figure 13. Current implementation of Summary tab in Work Order Follow Ups

Figure 13 shows the current graphical user interface of the Summary tab inside the Work Order Follow Ups module. This design comes with certain limitations as it does not provide users with a flexible navigation mechanism. The resulting work order list does not provide support for filtering or sorting. In the case that users want to apply another filter condition, they need to click on the Close button and then re-select the condition from the previous screen, as the screen view state is not persisted when users navigate away. This workflow can be improved by replacing the current list view control with another list view with the ability to sort/filter the data rows, which is the same as in the Work orders tab. Moreover, the list of conditions should revert to its previous state, so that users will not need to expand the tree structure all over again.

4.3 Work Order Data Tab

The current implementation of Work Order Follow Ups does not allow users to view detailed work order data directly. Users need to double-click on the interested work order from the list view. It is not possible to browse multiple work order data in the same view. The existing workflow can be improved according to figure 14.

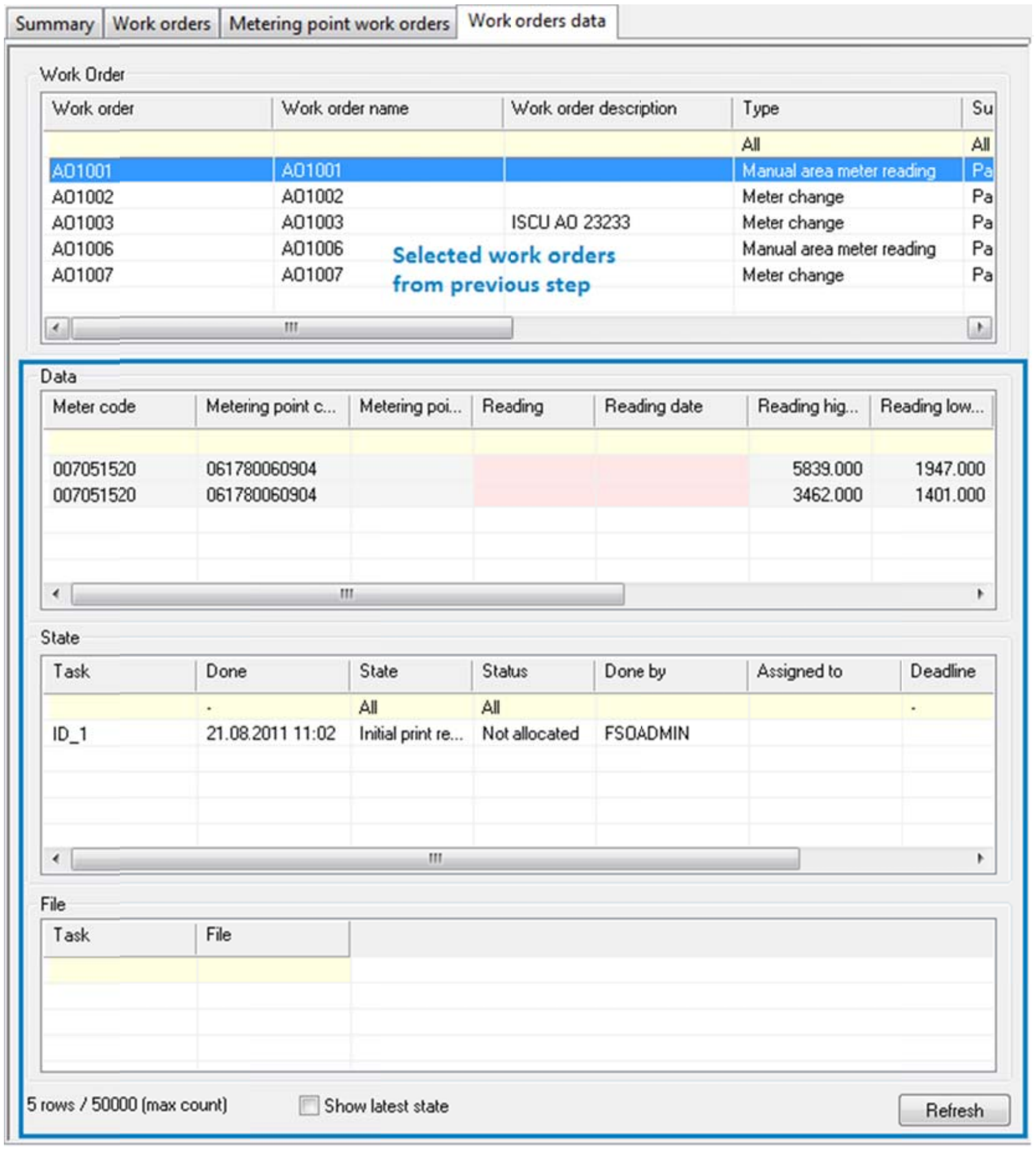

Open Work Order Follow-Up
Navigate to ”Work orders” tab
Navigate to ”Work orders data” tab
Select interested work order
Select interested work orders
Work order data is shown



Work order	Work order name	Work order description	Type
			All
A01001	A01001		Manual area meter reading
A01002	A01002		Meter change
A01003	A01003	ISCU A0 23233	Meter change
A01006	A01006		Manual area meter reading
A01007	A01007		Meter change
A01013	A01013		Meter remove
A01014	A01014		Meter change

76 rows / 50000 (max count)

☑ Show latest state


Summary
Work orders
Metering point work orders
Work orders data
Selected work orders from previous step
Show latest state
Refresh

Work Order

Work order	Work order name	Work order description	Type	Su
			All	All
A01001	A01001		Manual area meter reading	Pa
A01002	A01002		Meter change	Pa
A01003	A01003	ISCU AO 23233	Meter change	Pa
A01006	A01006		Manual area meter reading	Pa
A01007	A01007		Meter change	Pa

Data

Meter code	Metering point c...	Metering poi...	Reading	Reading date	Reading hig...	Reading low...
007051520	061780060904				5839.000	1947.000
007051520	061780060904				3462.000	1401.000

State

Task	Done	State	Status	Done by	Assigned to	Deadline
	-	All	All			-
ID_1	21.08.2011 11:02	Initial print re...	Not allocated	FSOADMIN		

File

Task	File

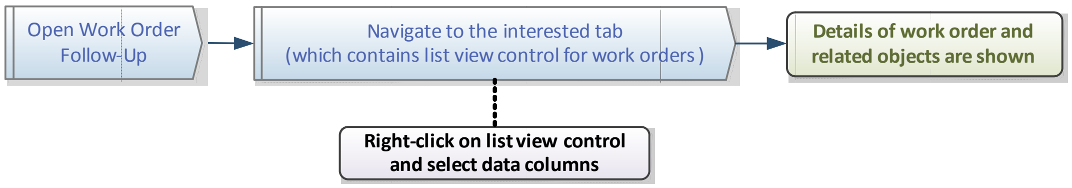
5 rows / 50000 (max count)

Figure 14. New workflow for viewing multiple work order data

Figure 14 illustrates a new workflow which allows users to view the data of multiple work orders. At first, users select work orders from the list view control in the Work orders tab. Then by simply navigating to a new tab, named as Work order data, users can view the data of selected work orders. The top-most list view control lists all selected work orders. In the case that users did not select any work order from the previous step (or they simply navigate directly to the new tab), this list will be populated with all available work orders inside the system (there is a limit on the number of work orders to be fetched). When users select a work order from this list, all data belonging to that work order will be shown in the same screen (in the Data section). This design also allows extended operations on multiple work orders. For example, it is possible to add new sections into the view which allows a wide range of batch jobs to be executed (including exporting and modifying multiple work orders).

4.4 Additional Information for Work Order Data

The current list view for Work order listing that is used by Work Order Follow Ups does not allow users to decide on which data columns are visible. By default the list of columns will be populated based on the common Work order data fields. However, this list of columns might not prove to be useful for all work order types and in all customer setups. The list view control can be enhanced by adding the possibility to specify the list of columns (data fields). The data source may come from all objects that are related to the current work order. This configuration can be implemented based on the existing GENERIS Objects and Fields system. This also allows user-defined fields to be included.


Open Work Order Follow-Up
Navigate to the interested tab (which contains list view control for work orders)
Details of work order and related objects are shown
Right-click on list view control and select data columns

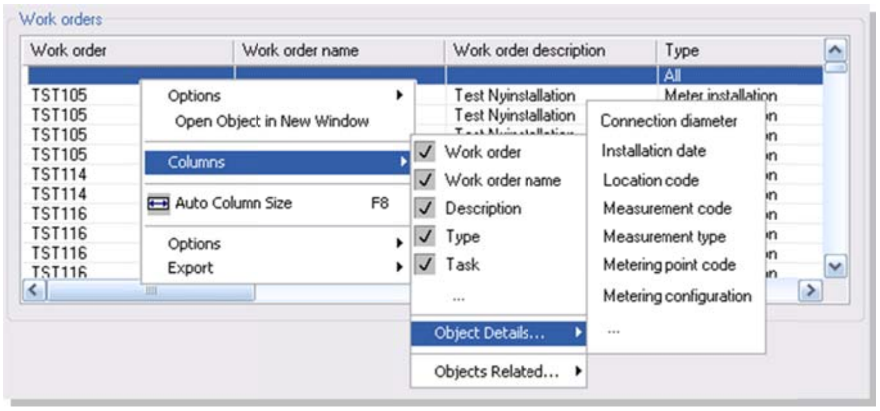


Figure 15. Custom data fields from related objects

Figure 15 illustrates a sample use case for the new function. By right-clicking on the list view control, users have the possibility to specify the list of visible columns. For example, instead of viewing only Work order details, users can also view the information regarding the related metering points or metering devices.

5 C++ Implementations

5.1 C++ MFC Windows Application Structure

One fundamental design of a MFC Windows Application is based on Document / View model. This concept divides an application into two different classes: a document and a view class. As simple as it sounds, the document class defines the application data (or the document itself), whereas the view class is used solely for the presentation of the application. Specifically, the document class contains the data structures, algorithms and specifies the application processing mechanisms. The view class displays the graphical user interface to users. It is responsible for painting the main form as well as handling all message mappings in Windows. In other words, it receives users' interactions at the front end and then takes appropriate actions.

The class for document and view object should be derived from CDocument and CView class respectively. The document and view interact with each other by using pointers. The document object stores a pointer variable which points to its associated view object and vice versa. Each view object has a member field which is a pointer that points to the document object. Whenever there is a change in the document data, the document will then notify all of its views to repaint their client area by calling a method named UpdateAllViews() [14,90].

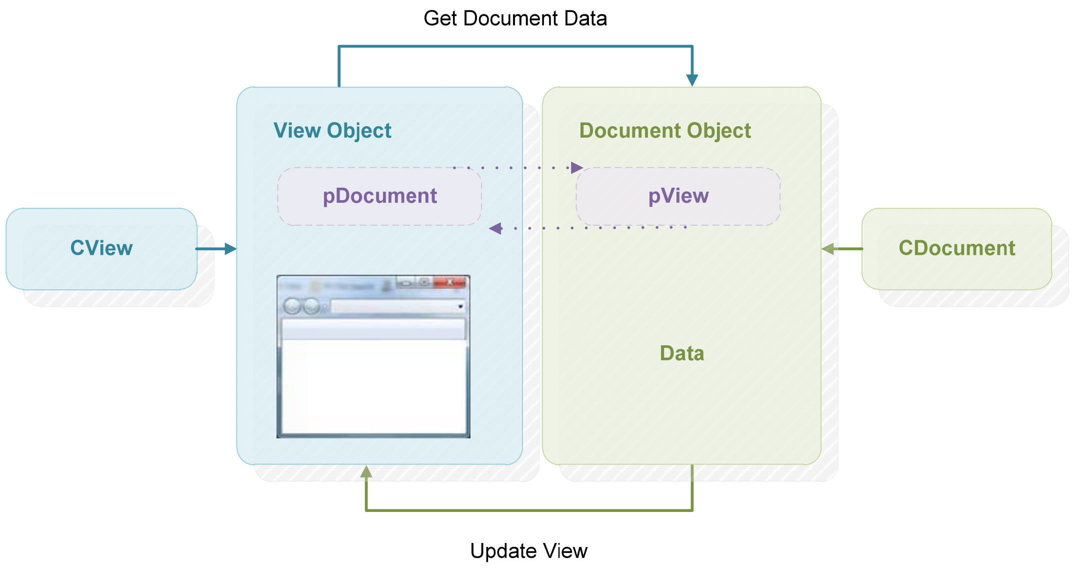


Figure 16. Document/View model

Figure 16 illustrates the relationship between the document and view object in the Document/View model. The view object can be a subclass inheriting from CView and document object can inherit from CDocument. They both have a pointer to the other object as a data member, which makes it possible to exchange data between the view and the document. The update will occur in the case that a user-generated or custom event occurs.

All interactions between users and a Windows application, such as a mouse click and window movement, are built based on the message system. Whenever an action occurs, a message will be created and sent to the appropriate class for processing. Each message has its own handle and is connected to a specific method. Windows application keeps track of the corresponding method for processing each message type using the message map. In order to map a message type to an existing method, it is necessary to make a macro call which takes the message handler and the function reference:

```
ON_MESSAGE (MESSAGE_HANDLER, FUNCTION_REFERENCE)
```

There are different handlers for different types of messages. The developers just need to map all those messages that they are interested in. The remaining unmapped messages will be handled by the framework itself.

5.2 User Interface Implementation

5.2.1 MFC List View Control

The implementation of a list view control involves creating a panel display which consists of a two-dimensional table with or without borders. The list view control may have different display styles. Each item can be simply a named icon or a detailed data grid. In the detailed view mode, the items are represented as a collection of data rows. Each data row has a set of data fields which is represented as columns. The intersection of a list view row and its column is called a table cell. The content of each table cell is a string of text. However it is also possible to add interactive contents to the list view, such as images, checkboxes and colored rows. Data and columns inside a list view control can also be formatted depending on custom implementations.

In an MFC application, the list view control is encapsulated using the CListView class. This class seamlessly integrates the list control with the fundamental Document/View architecture. The list view control can display its contents in different ways:

- Icon view: Each item is represented as an icon with a text label underneath. This is the only view where users can drag and drop the items to any location inside the list view area.

- List view: Each item is represented inside a data row with only one column. The row may contain an icon and text.

- Report view: This is similar to the List view style; however it also supports additional columns to the right. Each item is a composition of multiple sub-items, which are created by the application. Each column is implemented by an integrated header control using a class named CHeaderCtrl. [17.]

The CListView class also uses messaging to handle users' interactions. It provides a set of functions for manipulating the list view contents, such as retrieving and editing list view items. There is no built-in support for enhanced functionalities such as sorting and filtering. However, by implementing custom methods for different message types, it is possible to create an advanced version of the list view control by extending the CListView class.

5.2.2 MFC Tab View Control

MFC provides support for integrating a tabbed view into an application using the Document/View model. In order to implement a tab page inside an MFC application, developers simply need to derive a class from the CTabView class and then add a new view as a new tab. The new view needs to be derived from the CView class and the tab controls will display the view as a new tab. [18.]

Figure 17 illustrates a sample class diagram for a TabView control inside an MFC application. An arrow line demonstrates an inheritance relationship. The beginning of the line is the base class and the arrow points at inheriting class.

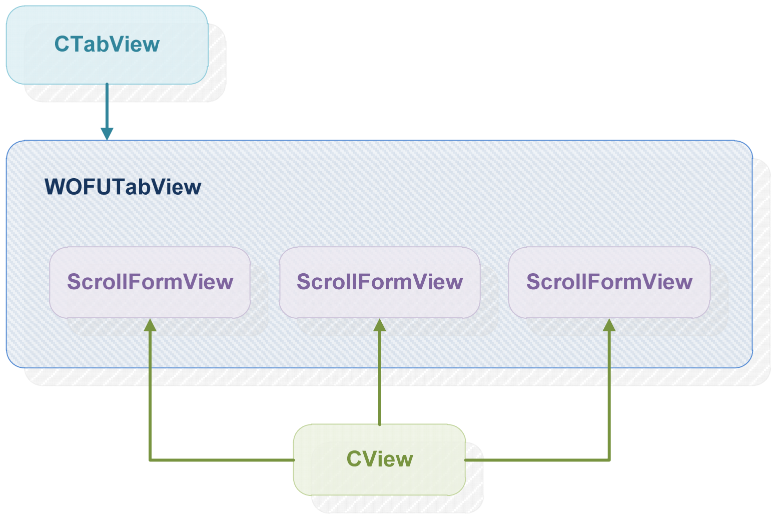


Figure 17. TabView control class hierarchy

According to figure 17, WOFUTabView is a subclass inheriting from CTabView. This class represents the tabbed view of Work Order Follow Ups. It uses three objects which are created from the ScrollFormView class. In this case, ScrollFormView is a subclass inheriting from the CView class. The purpose of the ScrollFormView is to provide a view with scrolling functionalities.

In order to add a new view, it is necessary to use a method which is inherited from the CTabView class which is named as AddView. The signature of the function is illustrated in listing 1.

```
int AddView(
   CRuntimeClass* pViewClass,
   const CString& strViewLabel,
   int iIndex=-1,
   CCreateContext* pContext=NULL
);
```

Listing 1. Signature of AddView function.

According to listing 1, it is necessary to specify only the first two parameters since the remaining parameters already have default values. The first parameter is the pointer to the runtime class of the tab view, which is derived from the CView class. The second one is a string which represents the title of the tab. By default, the new tab will be inserted into the end of the tab groups. In the case of inserting a tab into a different position, the third parameter is used solely for that reason. It takes an integer number which represents the zero-based position of the new tab view.

5.2.3 GENERIS Document

The new functionalities mentioned section 4 should be implemented using the Document/View model. The view is created based on the existing MFC classes and extended controls in the GENERIS Core module. The document object is basically based on the CDocument class. Each document will be created based on the existing document from GENERIS Browser. It has multiple inheritance layers, which include the data access layer and data handling logic. The document also contains data state which is used whenever users switch between different tabs. All controls inside a view share the same document as the tab view. For example, when users select some work orders from the list view control, the list of selected items will be stored inside the document. It is necessary to map that event to a function which handles the data storing operation, as illustrated in listing 2.

```
// Create new Document class deriving from CDocument
class WOFUDocument : public CDocument
{
public:
    WOFUDocument () {}
    virtual HINSTANCE ResourceHandle();
    void SetSelectedWO(bool _state);
    WorkOderObject& GetSelectedWOAt(int _Position);
    CArray<WorkOderObject> & GetAllSelectedWO();

private:
    CArray<WorkOderObject> m_SelectedWOArray;
};


// Create custom list view control class deriving from CListView
class WOFUListViewControl : public CListView
{
public:
    WOFUListViewControl ();
    virtual ~ WOFUListViewControl (){}

private:
    void OnItemChanged(int _iRow);

protected:
    CDocument * pDocument;

};
```

```
// Implement member function of custom list view class
void WOFUListViewControl::OnItemChanged (int _iRow)
{
    if (_iRow < 1 || !GetItemState(_iRow, LVNI_SELECTED))
         return;   // Skip non-datarow

    WOFUDocument* pDoc = dynamic_cast<WOFUDocument*>(pDocument);

    if (pDoc)       // If using WOFollowUpDoc type
    {
         // Data storage operations
    }
}

BEGIN_MESSAGE_MAP(viewObjectName,CView)
          ON_NOTIFY(LVN_ITEMCHANGED,IDC_LV,OnItemChanged)
END_MESSAGE_MAP()
```

Listing 2. Message mapping for list view control event.

Listing 2 demonstrates a sample definition of sub-classes which inherit from the CDocument and CView class. This is needed for implementing the Document/View model and message mapping operation. The code block makes a call to the message handling macro which has the declaration as: BEGIN_MESSAGE_MAP(viewObjectName, CView). This macro is used to start the message mapping process for all user interactions that occur in viewObjectName, which is an object created from CView class. The next macro call defines the event type, target list view control and the method to be executed whenever the event occurs. pDocument is a pointer inside the list view control which points to the document object used in the model. In this example, the document class is a derived class from the CDocument class, and thus it is necessary to cast the document pointer to the derived type. The casting operation is quite necessary in software applications which make use of abstraction and dynamic linking. In this case, the document can also be used whenever a database operation needs to be done on existing data. It contains a pointer to the database handler that is created when users first start GENERIS Browser. Since the document object will be created at the platform level, thus the module itself does not need to reinitialize the object.

6 Discussion

The main goal of the project was to improve the existing work order data model and the GENERIS Work Order Follow Ups user interface. The testing prototypes satisfy all design requirements. They are capable of responding to user input and display all necessary data fields at runtime. Taking the targeted users into account, the user interface has been designed so that it can greatly enhance the productivity of an energy information system. During the development of the system, different sets of solutions for the same problem were taken into consideration; however the selected solutions proved to be the most effective and could deliver the best user experience. The new design minimized the number of mouse clicks and users could get the tasks done using the least required transition steps. The outcome of the project has been evaluated to fulfil the requirements and function properly under the designed environment.

However, GENERIS Work Order Follow Ups can be furthered developed by adding full support for GENERIS Objects and Fields. The current implementation still relies on SQL queries to the existing database view, which limits the capability of the application to query data from other sources such as related objects and their data fields. This feature can be implemented based on the existing GENERIS platform using its built-in functions for querying GENERIS Objects.

The process of applying the design pattern could also be improved further by conducting the study based on user interviews and customer surveys. This method can collect real-world statistics of customer satisfaction. This would require a thorough planning process and careful preparation of the questionnaire materials. It would be more effective to improve the design where it is needed most based on collected feedback data.

The solution of using binary XML for the data model would simplify the version control process and increase the application flexibility. However, structural changes to the existing implementation and new data interfaces need to be created in order to accept the new format of bidirectional data stream. The user interface also needs to be redesigned, so that it can reflect the data workflow as well as new functionalities in an effective and systematic manner.

7 Conclusion

Design patterns help to improve general user experience. The new solutions for GENERIS Work Order Follow Ups will make the system more usable and effective. By minimizing the number of steps that users need to go through to get a task done, the new implementation has removed redundant steps from the process. This means that the user workflow will become more logical and effective. However, the most important benefit is that users can now have access to even more functionalities from a large multi-utility energy information system. In this thesis, the existing user interface's design patterns were studied and based on that, new improvement ideas were developed and evaluated.

The ultimate goals of the project were to achieve better software quality, reduce system complexity, increase efficiency and simplify the installation process. The user interface is a vital part of a software application, since all user interactions are done through the user interface. A good user interface will decrease the training cost, user error rates, support enquiries and at the same time increase productivity. Moreover, by using a highly customizable and effective data model, the MAM work order can become a more dynamic solution for deploying customer-oriented systems. This design eliminates the gap between common configuration and customer-specific setups. It also integrates seamlessly with the current GENERIS platform, which reduces the effort for installation, maintenance and version updates, thus reducing cost. Moreover, the binary XML is a widely-used format which is supported by different relational database systems.

The result of the project was an improved graphical user interface which can help users handle all analogue use cases effectively in their custom setups. The user interface plays an important role and contributes greatly to the effectiveness of system configurations. All background knowledge about usability, data structure, database modelling and programming languages played a vital role in achieving this objective.

References

1 Kauppalehti Uutiset. Process Vision: Seinäjoki Energia, the first new electronic services eGeneris Solution [serial online]. Helsinki, Finland; 24 November 2010. URL: http://www.kauppalehti.fi/5/i/yritykset/lehdisto/hellink/ tiedote.jsp?oid=20101101/12906056340950. Accessed 20 September 2011.

2 Baube F. GENERIS Meter Asset Management User Guide. Helsinki: Process Vision Oy; 2011.

3 Kallio P. GENERIS Energy Solutions. Helsinki: Process Vision Oy; 2002.

4 Einamo J. GENERIS MAM Meter Asset Management: Process Vision Oy; 2008.

5 Surhone L, Tennoe M. Work Order. Saarbrücken: VDM AG & Co. Kg; 2010.

6 Tullis T, Albert B. Measuring the User Experience: Collecting, Analyzing, and Presenting Usability Metrics. Burlington MA: Morgan Kaufmann; 2008.

7 Tidwell J. Designing Interfaces. Patterns for Effective Interaction Design. Sebastopol CA: O'Reilly Media Inc.; 2011.

8 Farrell J. Object-Oriented Programming Using C++. Boston MA: Course Technology; 2009.

9 Martin R, Martin M. Agile Software Development: Principles, patterns and practices in C#. New Jersey USA: Pearson Education; 2007.

10 Swanke J. Visual C++ MFC Programming by Example. Berkeley CA: CMP Books; 1999.

11 Ritchie C. Relational Database Principles. London UK: Thomson Learning; 2002.

12 Ponniah P. Data modeling fundamentals: a practical guide for IT professionals. USA: John Wiley & Sons; 2007.

13 Jokinen A. GENERIS Meter Data Management. Helsinki: Process Vision Oy; 2012.

14 Björnander S. Microsoft Visual C++ Windows Applications by Example. Birmingham UK: Packt Publishing; 2008.

15 Surhone L, Tennoe M, Henssonow S. Binary XML. Saarbrücken: VDM AG & Co. Kg; 2010.

16 Melton J, Buxton S. Querying XML: XQuery, XPath, and SQL/XML in context. San Francisco CA: Elsevier Inc.; 2006.

17 CListCtrl Class. Microsoft Corporation [online].
URL: http://msdn.microsoft.com/en-us/library/hfshke78(v=vs.90).aspx.
Accessed 19 December 2011.

18 CTabView Class. Microsoft Corporation [online].
URL: http://msdn.microsoft.com/en-us/library/bb983705(v=vs.90).aspx.
Accessed 19 December 2011.