Abstract—The following article is based on research work done in the development of a modular process data system. Based on a sequential main program and interrupt driven hardware interfaces, a software implementation without an operating system was implemented. By means of special software structure called Linked Object List, object oriented design was implemented with the procedural language “C”. Due to this design a reusable and measure system was achieved which enables a high degree of flexibility concerning the hardware configuration and system customization at the user site.

I. INTRODUCTION

The appropriate design of a system is one of the essential topics at the beginning of a new development project. According to the intended purpose of a device the first step is to model the system in order to get a structure for the implementation of the required features. In general the implementation of the system requirements is split in hardware parts and tasks which are done in software. In case of the hardware design the solutions for the challenges are mostly clear and supported by fundamentals of e.g. digital logic laws and several design methods. If we think of the software part a lot of problems have to be solved without such clear fundamentals. Object-oriented design is one of the paradigms which promise a way for designing stable and reliable software. A problem arises in this context if the used microprocessor platform is not supported with a compiler for an object-oriented programming language. In this case only the system modeling could be done in terms of software objects and their relations, the implementation has to be done in a procedural language.

Controlling a process in an industrial environment requires the measurement of the relevant process parameters. As processes are mostly very complex, process lines have to be introduced to structure the system. In order to master the complexity, distributed small process controllers are connected with a host computer at a central point in the plant. They are intended to measure and control the relevant process parameters of the line independently. After an initial configuration such systems can be seen as slaves of the host computer. Standardized busses (e.g. CAN, RS485) are used for the necessary data exchange.

To understand the several tasks of such a small process data and control system the requirements are summarized in the following chapter.

II. REQUIREMENTS OF THE PROCESS DATA SYSTEM

For economical reasons, a small and powerful process computer was developed. A main goal of the system was the universal usage in different fields of applications. Therefore a generic approach with a high capability of user configuration was intended. From the vendor’s point of view an open system was demanded concerning the extensibility of the functionality by means of software. This feature was necessary to enable customer specific changes without the need of reprogramming the main parts of the software. A further requirement was the capability of updating the system in the field.

In general the requirements are specified in the following enumeration.

A. Measurement

By means of dedicated transmitters the measurement of density, temperature and sound of velocity should be performed. Derived quantities like concentrations in several units in combination with different types of transmitters have to be calculated (evaluation unit). Depending on the product line editable sets of polynomial coefficients should be processed and stored.

B. System Inputs

Additionally to the used transmitters, generic inputs for existing process equipment were intended. By means of digital and analogue inputs switches like product line selectors or counter inputs as well as current and voltage levels can be evaluated.

C. System Outputs

For controlling actuators in the product line digital and analogue outputs are demanded. With these outputs limit monitoring relays or current loop control are feasible.

Further outputs, implemented as RS232 and RS485 were required for the communication with higher process hierarchies. A user interface was intended with a graphical display and five keys at the front of the process control system. By means of this interface the configuration process takes place and necessary measurement and status readings can be shown.

Updating or extending the software has to be done by means of a PCMCIA-Slot [3] on the front side.

D. Data/Program Storage

The internal storage has to be able to hold permanently stored standard- and customer specific application programs.

For the necessary parameters (e.g. scaling, limit monitoring, calculation) sufficient non volatile data storage was required.

A picture of the developed process data system is shown in figure 1.
III. DESIGN CONSIDERATIONS

Based on the user requirement specification, at a first step the system requirements were modeled in hardware and software parts.

A. Hardware Structure

In order to achieve an economical solution, efforts were made to reduce the complexity of the system and to lower the number of components. Finally, the best compromise was achieved by using a high integrated microcontroller [4]. By means of the integrated peripheral components of this controller (e.g. external bus interface, asynchronous serial bus, capture/compare unit, interrupt controller) a standardized way to connect the desired components was available.

The internal code and data storage were implemented by means of the external bus interface. The persistence of the data storage was guaranteed with a non volatile RAM, those of the code storage by means of a FLASH memory.

For the implementation of the user interface a graphical LCD was chosen and connected with the external controller bus. A minimal keyboard with five switches was connected to one of the controller’s ports.

To implement the desired functionality of updating the system in the field and of extending the built in software a PCMCIA storage card frame was designed to carry a removable memory card.

The communication line to the host computer was established with the serial line available in the controller. With the added RS485 and RS232 driver it is possible to communicate with one of the both standardized connections.

The development of the digital and analogue outputs and inputs as well as the proprietary transmitter interfaces [2] was done in former development projects and will not be a topic of this article.

The following block diagram [figure 2] is intended to give an overview of the components.

B. Software Structure

In contrast to the hardware design, the software structure is not so simply derived from the specifications. On the one hand standard applications which are available for microcontroller designs and digital circuits like LCD interfaces or PCMCIA implementations are not available, on the other hand basic conditions of the system (e.g. operating system, programming language, development platform) have to be defined.

One paradigm in the field of software development is the usage of object-oriented methods to analyze, model and implement the software requirements. Object-oriented design [1] requires the modeling of the problem by means of a data structure called Class. This data structure does not only incorporate the data (attributes, member variables) but also the functions (methods, member functions) which work with the data inside the class. The goals of such a design are to hide the complexity (abstraction) and to protect the data effectively. If we model our system according to these principles we can structure our software into different classes (objects) and their interfaces (methods). A system build up in this way supports the developer with a lot of advantages concerning reuse, maintenance and stability.

A problem occurs if the used microprocessor platform is not supported by a compiler for an object-oriented language. For many controller platforms there only a C-compiler is available but no C++-compiler.

A typical solution for this problem is a trade-off between the economical/technical decision for a microcontroller and the software development restrictions concerning the object-oriented language. But the missing of an object-oriented language does not imply that we have to develop our software without object-oriented principles. A good practice is to analyze and to model the software requirements with object oriented tools and methods (e.g. UML -diagrams [6]). The implementation is done later in a procedural language with a few restrictions.

For the implementation of object-oriented designs with the procedural language C, a trade-off can be achieved by replacing the C++-class with the C-structure. The methods (member functions) can be replaced as function-pointers in such a structure. Protection mechanisms are possible by putting such a C-structure into a single C-File...
and to control the visibility of the attributes (member variables) by means of an H-File (header file). Therefore access to the members of the C-structure is only possible by means of dedicated member functions (e.g. get(), set()). Another aspect is the instantiating of classes as objects. This feature can be implemented with constant arrays of such structures.

It should be clearly mentioned at this point that this trade-off is not a replacement for the full functionality of object-orientated languages like inheritance, polymorphism or exception handling.

IV. SOFTWARE DESIGN OF THE PROCESS DATA SYSTEM

To analyze and model the requirements of the process data system, UML diagrams [5] were taken to get an overview of the system. As a first result of the requirement analysis the use case diagram [figure 3] was generated.

A central part of the system represents the application program which is optionally extensible with programs on the memory card (update, user applications). Depending on the configuration of the system process variables are measured, calculated and distributed to the host computer or to the graphical display.

A. Process Data Object

The next step in the software development process was the modeling of the system by means of classes (objects). In order to handle the demanded versatility, a data object class [9] was introduced. Every process parameter, variable or constant should be an instance (object) of this class. A few attributes of the object control the behavior of the process value in the device. For every component of the system one attribute of the class is responsible for controlling the data handling. For instance, if the process variable density should be measured and shown on the display a member variable HARDWARE-MASK and DISPLAY-MASK has to be set appropriately.

Depending on this object definition, every module in the software has to be written generically. This implies that the respective program part has to evaluate the attributes of the data object before handling the process value.

Process data objects are very flexible concerning their handling in terms of definition. Hence user specific object definitions are able to extend the constant object definitions of the main system. Therefore a flexible way of customizing the default system is possible. For instance, the system software can be extended by means of an external data memory which stores additional process data objects.

Based on this idea a class diagram was designed to get an overview of the system components.

The configuration process is one of the central tasks in the system. Program handling, data measurement and distribution as well as the handling of the transmitter and process environment options should be handled according to the configuration data. To manage the versatility of transmitters, process lines and process equipments a generic algorithm was necessary.

B. Implementation Issues

Based on the system modeling and the demanded requirements, a few basic conditions have to be defined. One decision concerns the usage of an operating system, another one the selection of the appropriate programming language. In order to meet economical aspects and based on experiences from former projects, a decision was made not to use an operating system. Supported by the fact that the given requirement does not need operating system support like e.g. file system, network stack or...
multitasking, lower memory requirements have been achieved. The real time aspect in this context has already been solved in a former project with the implementation of the measurement tasks based on interrupt driven transmitter routines.

The second question concerning the programming language was solved with the selection of the used microprocessor for economical/technical reasons. As the development platform of this microcontroller [7] supports only the procedural language “C”, the object oriented design has to be implemented as a trade-off in a procedural way. Based on these decisions the main frame of the software was implemented as a sequential main()-procedure supported with interrupt driven routines for interfacing the process equipment (e.g. transmitters). To support distributed development and reuse for further projects, the graphical system for the LCD was implemented as separated procedures and linked by means of a software-interrupt (Graphic-BIOS). As mentioned above the object-oriented design was implemented with the procedural language “C” by using some language trade-offs. One of the central points in this context was the implementation of the process data object class by means of arrays of C-structures.

C. Precedural Object Class Implementation

According to the language “C” the process data object was implemented as C-structures. As there are a lot process data objects in the system, a constant array of such structures was defined. Instead of instantiating objects, a constant array of structures was defined which represents the main definition of the system behavior. Every structure incorporates a name, a type and behavior attributes for the process data. Based on the type of data a method, implemented e.g. as a function pointer, is responsible to access the respective value.

Based on the requirement to extend the software at the customer site special process data objects were introduced with the object type OBJ-LINK, and OBJ-LAST. Due to this definition it was possible to connect several process data object arrays in a way which is called Linked-List. By means of this linked object list it was possible to extend the basic definitions of the system behavior with customer definitions at compile time or later in the field by means of structures at the mobile memory card.

A central procedure was introduced to search for an object in this linked list. Based on the object’s type in the structure this procedure was able to evaluate the OBJ-LINK type and to connect several arrays of structures. Depending on the start point of the search (e.g. system memory, customer memory, PCMCIA card) a certain priority was achieved which is usable to override (customize) basic system definition [figure 5]. For instance with the introduction of new object definitions on the memory card, new process data readings or menu items are available in the process data system. The following code fragments are intended to give an impression of the structure definition and the outlook of the object list.

```c
#define OBJ_MENU (OBJ_ID|12)
#define OBJ_FLOAT (OBJ_ID|6)
#define OBJ_ROBJ (OBJ_ID|11)
#define OBJ_FUNCT (OBJ_ID|1)
// a lot of additional types ...

typedef struct
{
    NAME name;
    union
    {
        unsigned long c_li;
        NVPARA * p;
        FUNC_adr funct;
        CUST_adr cust;
        unsigned long * i;
        float * df;
        char * str;
        NAME * nstr;
    } data;
    char d_name[MAX_DLEN+1];
    char d_unit[MAX_DLEN+1];
    char d_format[MAX_DLEN+1];
    char hierarchy[MAX_BRANCH+1];
    unsigned char hardware_mask;
    unsigned char handle_attrib;
    unsigned char obj_type;
} OBJECT;

const OBJECT Test_Object_Table[] =
{
    // Menu Item
    "Main-Menu", NULL,
    ... ... ... ... ... ... ...
    {0,0,0,0,0,0,0,1}, TM_NONE,
    D_MENU,
    OBJ_MENU },
    // Float Value
    "Temp", (float*)&MEAS_results.pte,
    " Temp.", "[ \xf8"C "]", "%8.2f",
    {0,0,1,5,5,6,ALL,13},
```

Figure 5. Object List Control Principle
As shown in this structure and their definitions, not only attributes concerning the visualization and the hardware are available but also definitions for the format and the hierarchies in menu trees.

The Device Builder component [figure 4] was implemented as a state machine in the main()-procedure of the program. Depending on the state function (Data Visualization, Menu Selection, Parameter Edit or Adjustment, System Configuration) criteria are set which influence the search procedure (e.g. getobject()). So if every component uses the linked object list with the process data definitions, a pure generic system is established which enables a high degree of dynamic concerning the implementation of hardware- or software options.

V. RELATED WORKS

Innumerable publications describe different methods of achieving reusable object-oriented software designs. One of these methods is called Design Patterns [8]. The idea is to use elegant and proven solutions for dedicated problems to model the core of the software at the start of the design.

The method of mastering the complexity and the high dynamical efforts of hardware/software options in the system by means of object list controlled procedures is similar but not identical to a design pattern called Command [8]. This pattern is described as an encapsulation of a request in an object. By means of this pattern clients could issue request to objects without knowing anything about the operation requested.

In this context process data objects can be seen as such requests to other clients (components of the system). In this work the idea was modified and extended with features of linked lists to handle the demanded customization. A further idea was the implementation of such a pattern in a procedural language.

The idea of a linked list Process Data Objects has been already published [9] by the authors this year in a short form and is extended in this work with more details.

VI. CONCLUSIONS

In the development project of a new process data system high requirements concerning the implementation of different hardware options and customization features were fulfilled. For the implementation of the hardware requirements standard microprocessor techniques were used. In contrast to this, the software problems were solved by means of a new principle called object list control. Based on object-oriented methods a software design was generated which deals with a central data structure called process data class. Because of a missing language support for the chosen processor platform the design was implemented with the procedural language “C”. For economical reasons no operating system was used.

The central software structure was implemented as a state machine within a sequential main loop supported with interrupt driven data acquisition. The process data objects class was implemented as a linked list of arrays of C-structures. Every component of the system was designed to use this object list to get information about the handling of the process data. Depending on this structure a high degree of flexibility was achieved and the requirements concerning customization at the vendor’s site or in the field were met.

Experiences from the vendor confirm the compliance to the requirements and customer fulfillment. Product line specific changes are feasible without reprogramming the system. Adaptation to different process equipments are done only by means of configuration.

REFERENCES