OBJECT LIST CONTROLLED PROCESS DATA SYSTEM

Anton Scheibelmasser
CAMPUS 02, University of Applied Sciences, Automation Technology, Körblergasse 126, 8021 Graz, Austria

Bernd Eichberger
Graz University of Technology, Department of Electronics, Inffeldgasse 12, 8010 Graz, Austria

Keywords: Object oriented design, process control system, linked object list.

Abstract: 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 modelling could be done in terms of software objects and their relations, the implementation has to be done in a procedural language. 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 flexible system was achieved which enables a high degree of flexibility concerning the hardware configuration and system customization at the user site.

1 INTRODUCTION

Controlling a process in an industrial environment requires the measurement of the relevant process quantities. The use of process lines helps to master the often very complex system structure. In order to master this complexity, distributed small process controllers operate through a network with a host computer located at a central point of the plant. The software design for such remote controllers requires a high degree of flexibility to handle the various hardware options. This can be achieved by using intelligent software algorithms. Depending on the specific hardware interfaces and the process line, an adaptation in the field by means of parameterisation or configuration can be performed easily.

2 DESIGN CONSIDERATIONS

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

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 PC-Card 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 (Stroustrup, 1991) requires the modeling of the problem by means of a data structure called Class. The goals of

Scheibelmasser A. and Eichberger B. (2007).
OBJECT LIST CONTROLLED PROCESS DATA SYSTEM.
In *Proceedings of the Fourth International Conference on Informatics in Control, Automation and Robotics*, pages 351-354

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 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). 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 arrays of such structures.

3 SOFTWARE DESIGN OF THE PROCESS DATA SYSTEM

To analyze and model the requirements of the process data system, UML diagrams (Fowler, 1999) 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 extendable with programs on a 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.

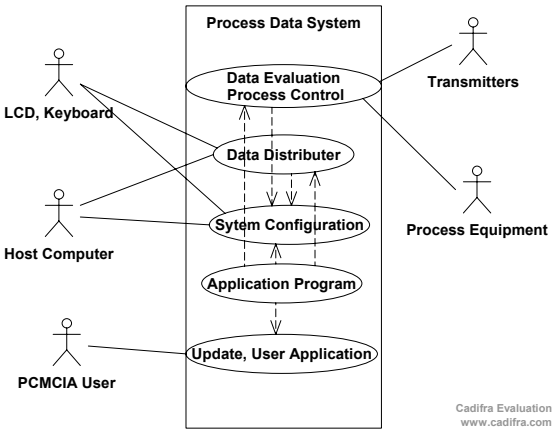

Figure 1: Use Case Diagram of the Process Data System.

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.

3.1 Process Data Object

In a next step of the software development process the modelling of the system by means of classes (objects) has to be done. In order to handle the demanded versatility, a Data Object Class was introduced. Every process parameter, variable or constant should be an instance (object) of this class. A few attributes of the object control the behaviour 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.

A Device Builder component, implemented by means of a state machine, prepares the basic state values for the Process Objects. The state function of the Device Builder interacts with several components in a sequential way. Starting with the invocation of the measurement tasks, the application program, the visualisation and the host communication, interfaces of the respective components are used. The Process Object component plays a central role in the system. Every component has to evaluate the attributes of the handled process data objects to get information about the intended use, the type and the specific handling of the values.

3.2 Implementation Issues

Based on the system modelling 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 (Keil, 2007) 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). One of the central points in this context was the implementation of the Process Data Object class by means of arrays of C-structures.

3.3 Procedural 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 definitions of the system's behaviour. Every structure incorporates a name, a type and behaviour 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 inked object list it was possible to extend the basic definitions of the system behaviour with customer definitions at compile time or later in the field by means of structures located 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, PC-Card card) a certain priority was achieved which is usable to override (customize) basic system definition.

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.

```
typedef struct
   {
      NAME name;
      union
      {
         unsigned long c_li;
         NVPARA * p;
         FUNC_adr funct;
         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[] =
    {
      {
         "Main-Menu", NULL,
         "","","",
         {0,0,0,0,0,0,0,1},
         TM_NONE,
         D_MENU,
         OBJ_MENU },
      {

    "Temp",(float*)&MEAS_results.pte,
    " Temp."," [ \xf8""C ]","%8.2f",
    {0,0,1,5,5,6,ALL,13},
    TM_NONE,
    D_MENU | A_ASSIGN,
    OBJ_FLOAT }
    // a lot of additional objects …
  }
```

As shown in this structure and their definitions, not only attributes concerning the visualisation and the hardware are available but also definitions for the format and the hierarchies in menu trees. The Device Builder component was implemented as a state machine in the main() -procedure of the program. Depending on the state function (Data Visualisation, 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.

4 RELATED WORKS

Innumerable publications describe different methods of achieving reusable object-oriented software designs. One of these methods is called Design Patterns (Gamma, 1994). 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 (Gamma, 1994). 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.

5 CONCLUSIONS

In the development of a new process data system high requirements concerning the implementation of different hardware options and customization features were fulfilled. The introduction of an "Object List Control" structure has significantly enhanced the stability, robustness, maintenance and extensibility of the software structure. Depending on this structure a high degree of flexibility was achieved and the requirements concerning customization at the vendor's site and in the field were met. The object list structure has proved its capability to solve the problem of customization by binding different device configurations (object lists) at compile or at runtime. Based on this list control new device features can be added by means of mobile data cards. From our point of view this method is advisable if there are a lot of configurable options defined in the system or in case of weak or partial requirement specifications. In both cases a high degree of flexibility in software design is needed. The open "Linked Object List" is a generic approach which allows upgrading the system behaviour without recompiling the whole software.

REFERENCES

Stroustrup B., 1991. *What is "Object-Oriented Programming?".* AT&T Bell Laboratories Murray Hill, New Jersey 07974.

Fowler M.,*UML Distilled, Second Edition*, 1999, Addison-Wesley

CADIFRA, *UML Editor, 2005,* www.cadifra.com

KEIL GmbH, 2007, *C166 Development Tools,* www.keil.com

Gamma E., Helm R., Johnson R., Vlissides J., *Design Patterns,* 1994, Addison-Wesley