Model-based Approach to the Development of SCADA applications

Branislav Atlagic, Mihaly Sagi, Dejan Milinkov, Bojan Bogovac, Stanko Culaja
Faculty of Technical Sciences at University of Novi Sad
21000 Novi Sad, Serbia
oscada@googlegroups.com

***Abstract* – This paper presents a methodology for the development of SCADA applications, derived from the SCADA model of a specific industrial plant. It supports all stages in development process, from design of field installation to the specification, coding and verification of complex control application in simulated environment. It is implemented in the extent of a general purpose SCADA/DCS solution, and partially based on international standards ISA-S88 and IEC 61131-3.**

Keywords*: *SCADA, control application, S88 batch control, IEC 61131-3.

I. Introduction

Industrial plants tend to be more complex, both in their size and the way they are operated by computer-based control system in real time. Essentially, a control system consists of underlying field installation (transmitters, control devices, wiring, IO modules/controllers, cabinets etc.) and a SCADA (*Supervisory Control and Data Acquisition*) system that performs supervisory control and reporting [1]. A SCADA system is usually distributed, in line with the nature of a process system.

The very core of each SCADA, observed as a software system, is a real-time datastore implementing the SCADA model of a process segment under its control. All SCADA functionalities (as software components / procedures) closely relate to this domain specific model, usually seen as the *process model*, or SCADA *configuration model*. Most of the functionalities are standard for a SCADA solution, while some must be customized or added as an application specific code (*control application* in the rest of the text). SCADA application development, as a whole, includes HW design (field solution) and SW design (set of control and GUI procedures), and its verification. Designed field solution produces specification of IO signals, protocols and other parameters needed to organize IO communication to the field devices. This specification is actually *IO model* of an industrial plant, and a starting point for building a configuration model and development of control application. The whole SCADA application development process, often a complex and non-trivial task, can be organized through a set of interrelated phases, as proposed by Fig. 1.

To cope with this complexity and to meet user requirements, time and budget constraints, contemporary SCADA solution has to provide set of specialized tools for system design and development, with a focus on design, coding and verification of control application [2]. Each of the tools adds some information to the final SCADA program model, representing a certain industrial plant in real time.

Part of the development process is the provision of application specific simulation tool, derived from process/configuration model and suitable to test and debug control application before the final installation in the factory.

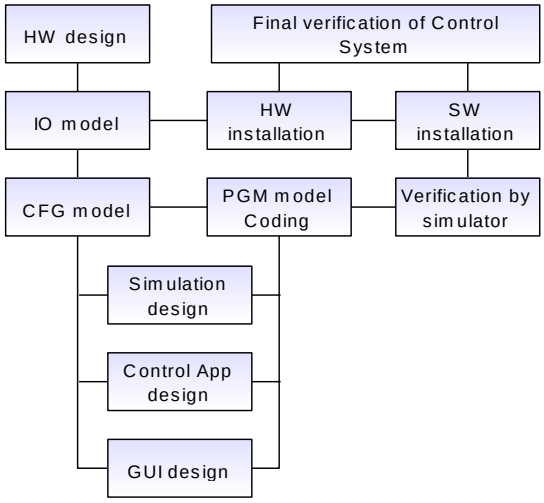


Figure 1. Proposed SCADA application development process

The key question is how to design general purpose SCADA, its model and the chain of development tools to ease implementations of a SCADA solution in real and complex industrial plants.

This is a subject of the paper that will present methodology developed for a general purpose SCADA system aimed to fulfill various requirements of real industrial applications, including the most complex batch control. It is implemented through a set of specialized tools, providing well defined framework for design and development of a control application. Those tools support:

- Design of field solution and generation of SCADA IO model.
- SCADA Configuration, i.e. modeling of an industrial plant.
- Simulation of this plant, derived from the configuration model (process model).
- Specification of a control application in the form of IEC 61131-3 SFC charts (steps, actions, transitions).

- Development framework for semi-automated C coding of procedures defined by SFC charts, running on top of the process model of an actual plant.
- Verification and testing of control application in simulated environment.
- End user training before the final installation.

This paper and a concept presented here are mainly based on long term experience in design and implementation of a SCADA system named GAUS [3]. Most of the model-related issues discussed here were at least partially implemented in this SCADA solution. Compared to other SCADA systems accessible to the authors, GAUS proved its superiority in data processing efficiency and memory consumption. The idea of developing a new solution named oSCADA was to preserve this level of efficiency, while adding new features important for its future applicability in most demanding applications. Most important of this new features are:

- support for huge applications (millions of data points),
- unrestricted distribution of uniform SCADA stations in multiple SCADA layers,
- improved configuration management and dynamic reconfiguration,
- introduction of multiple contexts hosting different configurations (multi-version),
- cross platform support,
- use of advanced C++ options, etc.

Listed features together can hardly be found in any of presently available SCADA solutions.

From all of this, oSCADA can be read in two ways, both like open or object SCADA, due to the efforts to provide really open network/software architecture, or the deep impact of object oriented paradigm to its internal program organization. More about oSCADA architecture and its implementation can be found in [20], to which this paper is a supplement.

Still, the development framework presented here is general enough to be easily implemented in a different environment.

II. Related work

There are two major reasons driving models in the center of IT development process: the complexity of real-world problems that need to be solved, and reuse of previously developed software components. In both cases, models define not only pieces that can be combined, but also a coordinate system and rules required to do this in a proper way. To fulfill its goals, model has to be domain specific, providing clear concepts closely related to a real-world problem, and to the engineers dealing with this problem.

In this paper, focus is placed mainly to the control application, segment of SCADA system software implementing supervisory control procedures specific to a certain industrial plant. The rest of SCADA provides a lot of support invisible to SCADA application designer.

This is why general modeling approaches based on UML and Petri nets are not quite suitable for SCADA application design [4,5,6].

The standard IEC 61131-3 is domain specific, but it is oriented to programming a single PLC (*Programmable Logic Controller*) unit. Currently a lot of effort is invested in its further transition to distributed PLC application, aside from SCADA [7,8]. Therefore, 61131-3 philosophy is different and cannot be fully used for description and programming of SCADA control applications.

Finally, ISA S88 is selected as a standard fundamental for the solution presented in this paper. Many authors had recognized its expressiveness and layered abstract views of an industrial plant [12-15].

III. Basic requirements of industrial control systems

Industrial control systems are usually classified in two major categories, primarily by the nature of the controlled process, and its impact to the architecture of the control system and applied control philosophy.

In its initial meaning, SCADA as a term generally applies to control over geographically dispersed process systems, like pipelines or power grids. These systems implement continuous control philosophy, meaning that the goal of a control procedure is to establish and maintain a stable state of a system for indefinite period of time. In real word, some amount of material (or energy) will continually flow through a system. SCADA systems of this kind are truly distributed (Fig.2) and therefore strongly affected by the communication failures, which limits the scale of automation of this systems. System operations most often include human operator into the chain of action.

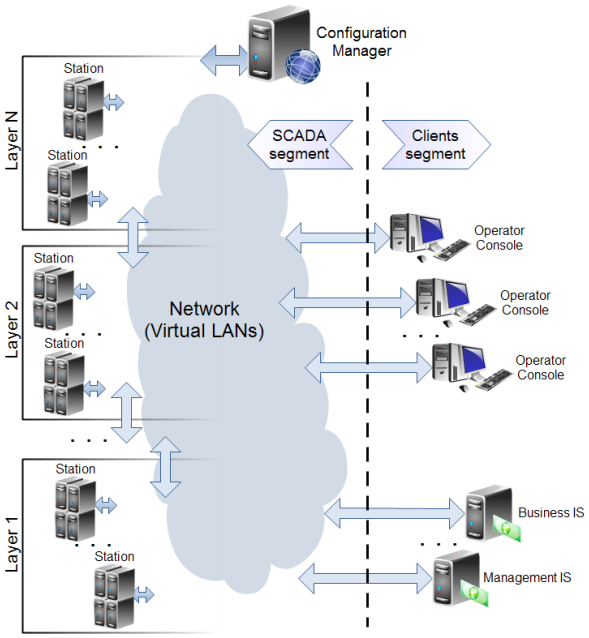


Figure 2. Advanced networked oSCADA architecture

DCS (*Distributed Control System*) predominantly marks systems that control some industrial process at a single location. It is usually an industrial plant (or its segment) with its own control system. Throughout this

text, plant will mean exactly this – a segment of industrial process controlled by a single control system. Also, term SCADA will be used to denote both kind of internally (in software) quite similar control systems. Differences between pure SCADA and DCS are mainly in underlying communication infrastructure, sizing and expected data processing performances.

DCS implementations are usually based on fieldbus networking, providing fast and reliable communication to the field devices acting as process IO modules or process controllers (Fig. 3). Reliability of data exchange makes it viable to deploy full automation of production processes, which is a real difference compared to traditional SCADA.

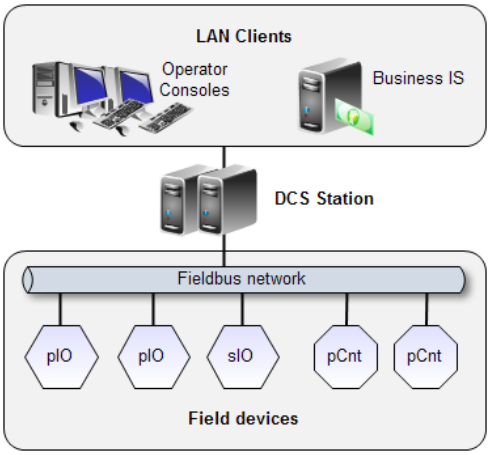


Figure 3. DCS oSCADA architecture

By its nature and implemented control strategy, industrial processes managed by DCS systems are both continuous or batch controlled.

Typical continuous-control plants are refineries, with distributed IO and a lot of synchronized PID controllers preserving flows, pressures or temperatures at a given set-points.

Batch-control processes, in contrast, provide end users ways to submit their request for execution of a set of operations under the conditions set by user. Those conditions are specified by a user supplied *recipe* - a list of operations organized in the phases, along with the parameters of their execution (including type and quantities of materials involved).

According to the standard ISA-S88 [11], a batch process is defined as "a process that leads to the production of finite quantities of material by subjecting quantities of input materials to an ordered set of processing activities over a finite period of time using one or more pieces of equipment".

S88 is usually referenced as the batch-control standard, but it is more than just that. It defines and explains levels of hierarchy and modularity of both industrial plants and their control systems. On that ground S88 is the first model relevant for general purpose SCADA design, as a conceptual high-level model of industrial plant that a control system design should follow.

The standard S88 defines physical model of a plant (Fig. 4), with the following structural elements (layers):

- Each plant has (at least) one processing *cell*, an abstract entity containing all equipment required to execute a limited set of production activities - operations, defined for that cell.

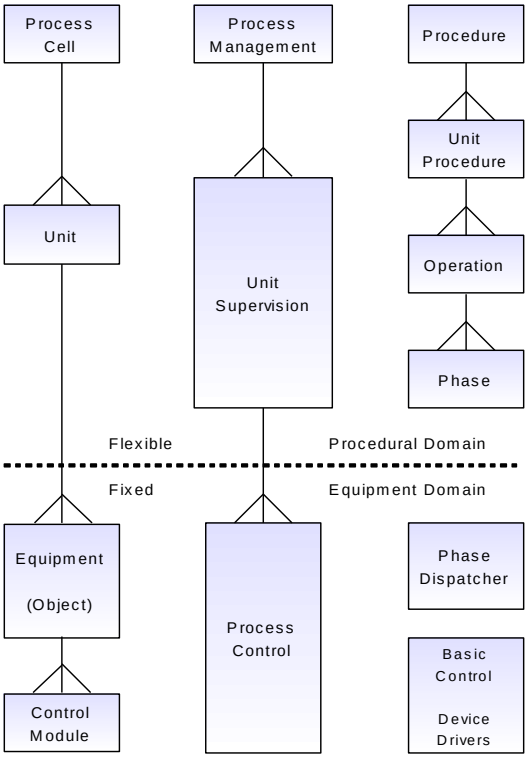


Figure 4. Plant model by standard S88

- *Unit* represents a segment of a process cell, with some separate functionality (set of the operations mutually related by the type), usually around some key equipment at the factory floor. For example, unit relates to a mixer and a set of dozers, pumps and valves requisite to feed that mixer, or to unload it. Consequently, unit is appropriate to be assigned with a control procedure, or *recipe*. Unit still belongs to the flexible, procedural domain of a control system.
- At the fixed or equipment domain, *equipment modules* describe real technical objects performing some activities, defined by the technology of the production process. Materials involved in this activities change their nature, from raw to final products. State of each of equipment is acquired and/or controlled via set of *control modules*, i.e. end field devices installed at the factory floor.

A process is managed through the unit supervision and physical process control. Control procedure is an ordered set of recipes (unit procedures), each of them specifying operations that are to be executed sequentially. S88 standard provides guidelines for writing and handling recipes (unit procedures), but has nothing to do with the internal specification of operations.

Standard S88 gives a valuable contribution to better apprehension of processing plants, end segmented view of its organization. It is simple and obvious to anyone with practical experience in automation of industrial processes [16,17].

In the attempt to standardize diverse environments for PLC programming, IEC has published standard 61131-3 prescribing common elements (data types, variables, functions and program units) and five programming languages with distinct roles [9,10]. *Sequential Function Chart* (SFC) is intended to graph the sequential behavior of a control program. Basically, SFC chart specifies state machine of a program through the steps (states) and transitions (conditions changing active states) (Fig. 5).

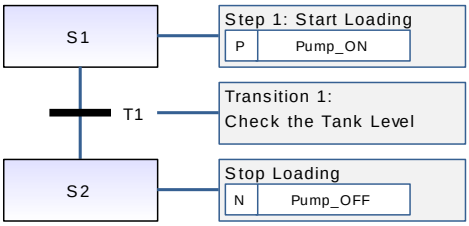


Figure 5. SFC steps and transitions

At least one step has to be active, meaning that several active steps in parallel can execute control *actions* involving physical control over the process. By its nature and graphical expressiveness, SFC is ideal to define internal structure and organization of the operations introduced by S88. SFC allows another step forward in decomposition of a control problem into manageable parts. It is perhaps even more important that SFC, as a domain specific programming language, provides communication between the people of different backgrounds, skills and rolls in a control project. Therefore it was selected to be a part of general purpose SCADA solution presented in the following section.

IV. Requirements of General Purpose SCADA Design

In this paper, "general purpose" implies a SCADA system suitable for both SCADA and DCS applications in wide area of industrial plants. It was already stated that internally software solution looks just the same. There are three top demands for general purpose SCADA system: truly distributed architecture, high processing performances, and support for both continuous and batch control applications.

Distributed SCADA is a hierarchical system built by the stations interconnected via communication network, either TCP/IP or kind of fieldbus. All SCADA stations share a single configuration model, where different stations play different roles inside the control system. Some of them are oriented more to the low-level data acquisition and the physical control, while others are mostly responsible for the high-level control or the decision support. Data replication must provide efficient mechanism for reliable data synchronization among the peer stations, including the propagation of the commands down to the place of the execution.

SCADA performances are usually measured by the number of changes that system can handle in real time, which includes data crunching (conversion and checking of the input data), event generation and storing to disk, and later distribution of changes and events to the other stations. Data distribution and storing are generally limited by the speed of the network or the disk, but they can be postponed for some short period of time. On the contrary, data crunching which involves access to local real-time datastore (often a memory database), cannot be postponed without sacrificing quality of the real time control. Therefore data crunching throughput is the primary performance index for SCADA system. Obviously it has to be high for general purpose SCADA, but from different reasons. In modern SCADA applications, huge size of model with millions of data points (SCADA variables) demands fast handling even though the data acquisition is relatively slow. Large DCS systems are supposed to handle only tens of thousands of signals, but the expected time constraints for data acquisition and control period are extremely high.

Third demand is obvious, general purpose control system should provide adequate development environment for implementation of different control strategies for continuous or batch industrial processes. This environment includes internal software organization and run-time components, but also a set of different software tools for specification, coding and verification of a control application.

oSCADA is a concept and an initial solution developed to meet this requirements. It provides memory and datastore efficiency adequate to meet high-end SCADA application. For example, oSCADA can crunch 2M of changes per second on ordinary desktop computers. In run-time, it uses around 500MB of memory for each 1M of data points. It is also truly distributed and practically free in network topology. There is a single executable for all kind of oSCADA stations, different in their roles. A station customization according to its role is done by a set of installed DLLs providing selected protocols or functionalities [19,20].

V. oSCADA Application Development Environment

The development of a SCADA application starts from a collection of usually unstructured information and requirements given by the end-users. This is initial process specification that needs to be transformed into the detailed hardware and software projects. The early design phases are crucial for the success of the whole development process, because the faults made here are difficult to correct later.

oSCADA concept includes a methodology for specification and modeling of the controlled process system, supported by a set of development tools conducting the application designer through the development process. This section describes the steps to be taken during the model-driven development of an oSCADA application. At the same time, it proposes described methodology as the general one, in believe that it could be useful for other SCADA designers.

A. Hardware design

The goal of hardware design is to select a field solution appropriate for an application, i.e. set of field devices and other components required for implementation of that solution in a plant. It includes definition of all signals, protocols, wirings, IO interface modules and process controllers that are to be installed, in order to collect process IO data and execute control.

A tool named *HwDesigner*, based on a concept of *scenarIO*, supports hardware design (Fig. 6). Basically,

scenarIO is a specification of hardware components involved in connection of a certain type of field device to control system, including cables, safety/protection components, mounting accessories, interface modules, etc. Also, each *scenarIO* is assigned to a Visio/AutoCAD drawing showing the details of the connection. For example, pressure transmitter can be connected via analog 4-20mA signal, or digitally by protocols HART or Profibus. A design engineer is responsible to prepare a *scenarIO* for each of these options, supported by oSCADA. All scenarIOs are stored in *HWdesigner* database, and the application designer can choose those that are optimal for his/her project. Also he/she has to add additional information like the tags, addresses etc.

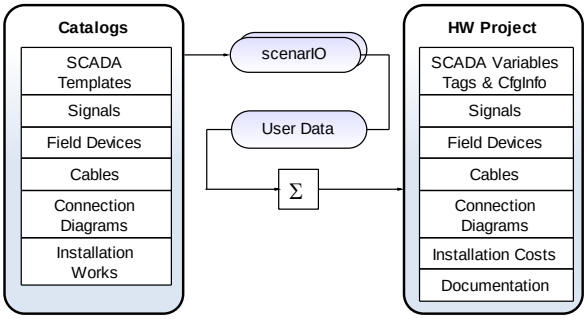


Figure 6. Hardware design process

The output of *HWdesigner* is a project containing lists of process variables, field devices, connection diagrams, and accompanying textual project documentation. For SCADA software designer, the most valuable output of this phase is a list of SCADA variables and their properties important for the next step.

B. Process modeling

Process model implemented in oSCADA follows principles given by standard S88. It is designed to support wide range of potential industrial applications, introducing a high-level software model of the physical process based on catalogs and process variables (Fig. 7).

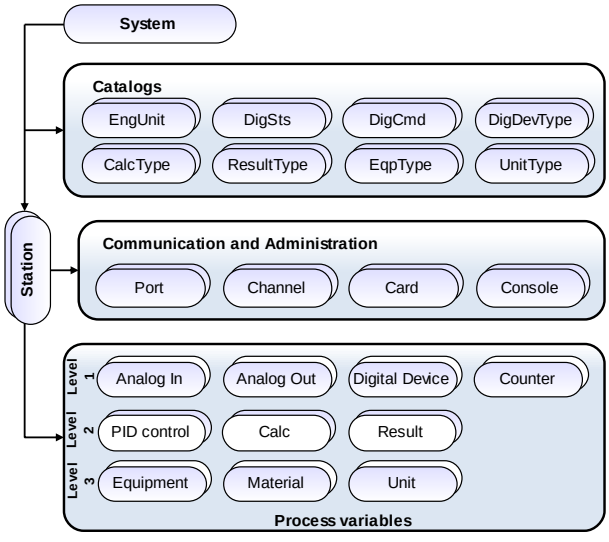


Figure 7. Process model – oSCADA configuration

Catalogs define legal set of various application-specific values and data-types, referenced later by configured process variables. For example, digital devices catalog defines types of used control modules (e.g. on-off valve with two limit switches), including legal set of states and commands associated to this type of modules. States and commands have to be previously defined in their own catalogs. The idea of introducing catalogs is the provision of the structured metadata (data-types) required for the later configuration of process variables (instances) in a controlled way.

Process variables, divided in three levels, represent various components and attributes of a physical process. The current state of the process variable is always expressed by its engineering unit value or through the associated state/command pair if the variable is digital in its nature. These values are derived during the processing of acquired process data and low-level control procedures. First two levels of process variables are common for the most of SCADA systems, and correspond with the physical model of the process cell.

Third level of the process variables directly supports development and implementation of the control application software. Equipment variable, used both for continuous and batch control, organizes a set of low level variables used to supervise and control an existing technical object in the plant. It provides way to write control procedures at the higher level ("object oriented"), through a list of lower level process variables used to transfer the control to the end control modules. Each of equipment has its cataloged type, and can be analog or digital by its nature and representation within the SCADA. Unit variables are the link to user batch control procedures, which are to be executed using defined set of equipment modules and materials. There is a 1:1 relation between a unit and its executive equipment.

The process model is often called configuration model, because it defines SCADA configuration - real-time data structures residing in the very center of a control system. A configuration tool *oConfig* facilitates building a model for a certain application. One of its outputs, particularly relevant for the following code development phases, is a special header file with exported catalog entries. This file provides a link between the application control code and application specific data types imposed by process model.

C. Process simulation

Standard simulation tools present at the SCADA market are usually protocol oriented. They provide slave side of a certain protocol (i.e. Modbus), and respond to SCADA acquisition by sending process data (values) kept internally in its datastore. Usually, there is a support to change this data manually or via some function (random, ramp, sinusoidal etc.). This is a convenient way to verify protocol implementation or test SCADA performances, but very painful for verification of developed control applications code.

Role of plant simulator *oSimulate*, presented here, is to provide simulation environment suitable for thorough testing and debugging of the oSCADA application software. The basic idea is to simulate the behavior of a plant through the simulation of the control modules reaction to the legal commands issued by the control application. Plant simulator is induced from standard

oSCADA acquisition/control station, by stripping off the basic process I/O acquisition, and adding the simulation procedures written in C and founded on oSCADA process model. Those procedures simulate the behavior of the control modules and equipment when subjected to a command.

At the low level, control module is a digital device (like valve, switch or pump), where simulation produces the new state (change of its inputs) in response to a command. For example, a valve turns to OPEN or CLOSE, in line with the received command.

Simulation of equipment is a bit more complicated, but still simple. For example, tank level should rise if the input valve is open and pump started, and vice versa.

In both cases, simulation procedures deal with the earlier defined types of control modules, making it relatively simple to prepare adequate simulation for virtually any industrial plant. Disturbances from the legal reactions, i.e. malfunctions of the field devices, can be generated manually (using standard user interface), or by a random fault generator.

Additionally, in a certain application requires more complex analog simulation, it is provided option to include an arbitrary C-code Simulink model. *oSimulate* will feed it with input values, execute the Simulink periodically and use simulated results to alter plant state consequently.

D. Customization of GUI subsystem

Next step is to define graphical forms and user menus/dialogs requested for appropriate visualization and GUI control of physical process. This is done using a specialized tool for definition of technical diagrams forms, and by programming the additional user dialogs (C# for example). The importance of this phase is in the first visual and UI contact of end-users with the new application. It is the first checkpoint where the mutual understanding between the customer and developer teams (usually different in the technical skills and culture) can be verified.

E. Development of control application for continuous process systems

Continuous control is always present in SCADA applications, as a default control layer. It relates to low level variables and equipment variable from the top level. Careful analysis of control code from several applications had pointed out the functions that should be extracted in a separated, application specific module. Those functions extend and customize standard processing of oSCADA variables up to the relevant user requirements.

At the low level, functions are provided for calculated variables (user-defined calculation procedures) and for digital devices (spontaneous changes and command-execution verification). Equipment variables are periodically checked by a user-assigned function that implements a certain control procedure. All this functions are type-based, written for a class of control modules existing in plant, and registered in one of the catalogs of the process model. This ensures compact and "easy to understand" control code.

F. Batch control procedural model

oSCADA batch control processing is organized according to the standard S88 [11]. The aim of the procedural control is to direct ordered execution of the processing activities required to produce a batch. Top-level batch procedure contains lists of recipes (or unit procedures in S88 terminology) that are to be carried-out sequentially. Assigned to a single unit, recipe specifies input/output materials, equipment, phases and operations involved in its execution. At the end of its execution, a recipe can transfer the report list (involved materials report, quality marks, etc.) to its successor. This allows the ending recipe to collect all relevant data about the whole batch execution, data that can be saved in reports database or further directed to the business information system.

Standard oSCADA library includes batch/recipe management functions allowing users to submit their request in a formalized way (Fig. 8).

Recipes and batch procedures are user-defined entities saved on disk as text files. After the activation, a batch procedure is held in a ready list. Each of its recipes waits while the corresponding unit is occupied (busy). After its activation, a recipe starts phased execution of the specified operations. A phase is incremented when all of its operations are completed.

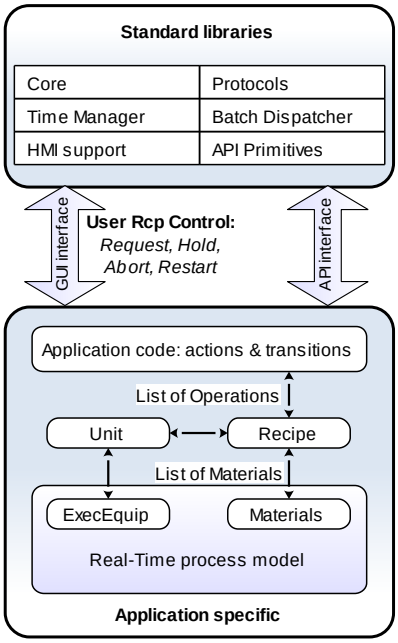


Figure 8. Organization of Batch Control Software

A recipe ends when its last phase is completed. During the recipe execution, system operator has visual and manual control. When it is running, a recipe can be temporarily suspended and later restarted or aborted. Batch procedure ends when all contained recipes are completed. After its completion, batch procedure can be deleted or repeated.

As stated by S88 model, the operation is a software procedure executing a well-defined technological action specific to a given plant. Each operation is characterized by a set of user-defined arguments that specify the way in which it is going to be performed. The user has to supply these arguments when the recipe is created or requested (activated). Several types of arguments are allowed, providing support to pass numeric values and/or process

variables (like materials or equipment) required for the execution of the operation.

Following figure (Fig. 9) shows an excerpt from recipe text file, describing an operation to be executed by oSCADA. This operation, dosing a fluid into a mixer, has two arguments. The first argument is the tag of a *material* variable representing a fluid, and the second is the requested quantity. The third argument could be a dozer (i.e. equipment variable representing a reservoir and a pump) that is to be used, if this fluid can be transferred to the mixer in several different ways. In the given example third argument is not required, because a mixer itself is determined by the unit to which a recipe is assigned, and the its further association with the appropriate equipment variable.

```
[OPERATION]   <FDOSE>
  OperType     FDOSE
  RcpPhase     2
  NoParam      1
  ParamList
      Parameter    230.00   <Quantity>
  NoPVid       1
  PVidList
      PVid         DOP      <Fluid>
[STOP]        <FDOSE>
```

Figure 9. Example of an operation specification

Figure 9 illustrates how this operation would be specified in a recipe. This recipe segment shows that FDOSE operation has to be done during the recipe phase 2, and identifies 230l of a fluid DOP that should be filled into a mixer.

G. Organization and coding of operations

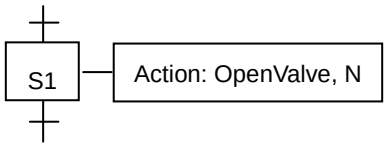
In the approach presented here, operations are organized using standard IEC 61131-3 and implemented in a C/C++ programming environment. The basics of oSCADA solution are:

- A combination of standard programming languages is used. IEC 61131-3 standard language SFC - Structured Flow Chart is used to define structure of the operation in a form of readable and easy-to-understand graphical chart. SFC is irreplaceable in this early design phase, providing simple graphical syntax as a common language for the facilitated communication between the design teams with different technical culture. Use of programming language ST (*Structured Text*), recommended by standard 61131-3 to complete SFC charts, would be quite impractical for oSCADA application development and debugging. It would ask for additional tools like ST interpreter and debugger, which would only add complexity and no benefits compared to standard Visual Studio environment. From this reasons, oSCADA operations are programmed in C programming language, ensuring compatibility with current IT development/ debugging tools.
- A special tool, realized in form of a SFC editor combined with C-code generator/editor, assists in the development process.
- Operations are organized in a project, which contains the list of enclosed operations, and the specification of the number and types of arguments for each operation.
- An operation is defined by its SFC chart and its corresponding C/C++ program module. SFC chart organizes the operation through the steps, transitions and the actions. Action and transition procedures are coded in C, using a set of API primitives to the oSCADA real-time datastore and the control library.
- To preserve reentrancy of operation's program code, the operation should use only a set of assigned dynamic variables.
- Each time an action is assigned to a step, or a new transition is defined, an empty function with predefined arguments is automatically inserted into a C/C++ program module. It is named same as the corresponding action or transition. Programming a body of generated functions is made easier by a specialized editor that helps in browsing through the configured catalog, process variables, operations arguments and dynamic variables.

This approach is an attempt to combine SFC and C/C++ languages in a natural way. The following figures show examples of SFC fragments and their C/C++ code extensions. They should provide instant view into the code that implements control application, and prove its clarity and readability.

Following figure (Fig. 10) illustrates a typical action programmed in oSCADA manner. The action named *OpenValve* sends a command OPEN to a digital device using its tag for a reference. This code uses two oSCADA API primitives:

- `putDigCmd` whose function is probably obvious, and
- `getID` - get ID of a process variable that is referenced by its name.

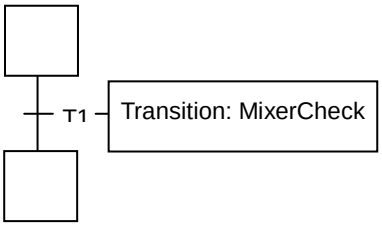



```
static bool OpenValve(STEP *s, UNIT *unit )
{
   putDigCmd( getID("ST:VALVE"),CMD_OPEN );
   return 0;
}
```

Figure 10. Code of a typical SFC action

Figure 11 displays a transition *MixerCheck*, which make sure that the low-pressure condition no longer exists in a mixer. This is done by reviewing the current value of an analog input process variable representing the pressure transmitter installed in a mixer. This variable is accessed as an element of the equipment module that is assigned to the unit executing the recipe. This equipment is designated as the executive equipment (*XE*), while *mixPRESSURE* is the

offset of pressure variable in its element list (exported by the configuration tool).




```
static bool MixerCheck(STEP *s, UNIT *unit)
{
   if( getValue(elmXE(mixPRESSURE)) < 0.9 )
      return 0;
   else
      return 1;
}
```

Figure 11. Code of a typical SFC transition

H. Application software test and verification

Application software test and verification phase, the last one before the on-site installation, is crucial for the successful realization of the whole control project. oSCADA provides plant simulation environment for thorough testing/debugging of the application software. Application code is tested in software environment identical to the run-time, using standard C/C++ debugger with no restrictions, while the plant simulator in an authentic way depicts the behavior of a real process connected to the SCADA station. This tool also can be successfully used for the training purposes, because its user has the same feeling as if he/she is using real HMI console of future SCADA system.

VI. Conclusion

Modern SCADA systems are rapidly changing their traditional role becoming a heart of much larger and more complex integrated control systems.

Traditional SCADA systems are extended with complex decision-support functionalities based on process simulation in real time. This is particularly true for smart grid and power distribution systems, where market already demands large SCADA efficient enough to handle more than 10 million of data points in real time.

Industrial plants typical for DCS control are also increasing their size, along with much higher real-time and safety requirements (redundancy and high availability). Application development methodology, briefly presented here, is primarily oriented towards this class of applications. It is based on standards S88 and 61131-3, which should provide clear concept and common language for all the people gathered around the control project, different in their skills and understanding of the development process.

This paper is an effort to propose a solution that could satisfy complex requirements of distributed batch control applications. Its main goal is to increase flexibility of the basic SCADA system and provide SCADA developer tools supporting each of development steps.

Acknowledgment

This work was partially supported by the Ministry of Education and Science of the Republic of Serbia under the projects TR 32031 and III 44009, year 2011.

References

[1] Bailey D., Wright E., “Practical SCADA for Industry”, Elsevier 2003. ISBN 07506 58053.

[2] Wang L., Chen Tan K., “Modern Industrial Automation Software Design Principles and Real-World Applications”, IEEE publications 2006. ISBN-10 0-471-68373-6.

[3] B.Atlagic, D.Kukolj, V.Kovacevic, M.Popovic, "Application development environment of an integrated SCADA system", EUROCON 2003, Ljubljana 2003

[4] B. Selic, L. Motus, “Using models in real-time software design”, Control System Magazine, IEEE 23 (3) (2003) 31–42.

[5] F. Basile, P. Chiacchio, D. Del Grosso, “A two-stage modeling architecture for distributed control of real-time industrial systems: Application of UML and Petri Net”, Computer Standards & Interfaces, Volume 31 Issue 3, March 2009.

[6] K.Sacha, “Verification and implementation of software for dependable controllers”, International Journal of Critical Computer-Based Systems, Volume 1 Issue 1/2/3, February 2010.

[7] C.Gerber, H.M.Hanisch, S.Ebbinghaus, “From IEC 61131 to IEC 61499 for distributed systems: a case study”, EURASIP Journal on Embedded Systems - Embedded System Design in Intelligent Industrial Automation, Volume 2008.

[8] T.Heverhagen, “Integrating UML-RealTime and IEC 61131-3 with Function Block Adapters”, IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC 2001).

[9] IEC task force 65B/WG7/TF3, International Standard IEC 1131-3: Programming languages for Programmable Logic Controllers, IEC, 1993.

[10] F.Bonfatti, P.Monari, U.Sampieri, “IEC 1131-3 Programming Methodology”, CJ International, 1997.

[11] ANSI/ISA-S88.01-1995, Batch Control, Part 1: Models and Terminology, ISBN: 1-55617-562-0, USA, 1995,

[12] E.Torp, “Using the S-88 Standard for Batch Control”, Automation Expo Conference, Cincinnati 1999.

[13] J.Parshall, L.Lamb, “Applying S88: Batch Control from a User’s Perspective”, ISBN 1-55617-703-8, USA, 2005.

[14] D.Fleming, V.Pillai, “S88 implementation guide: strategic automation for the process industries”, ISBN 0-07-021697-5, USA, 1999.

[15] Z.Verwater-Lukszo, “S88 for Researchers and Scientists”, World Batch Forum June 2004.

[16] F.Lovering, “Using Basic Control to Make Recipes Simple”, World Batch Forum 2006, Zemst, Belgium

[17] C.Case, “Applying ISA S88 to Small, Simple Processes”, Rockwell Automation, World Batch Forum 2006, Zemst, Belgium.

[18] B.Atlagic, D.Kukolj, V.Kovacevic, M.Popovic, "Application development environment of an integrated SCADA system", EUROCON 2003, Ljubljana 2003.

[19] Atlagic B., Milinkov D., Sagi M., Bogovac B., “High-Performance Networked SCADA Architecture for Safety-Critical Systems”, ECBS-EERC 2011, Bratislava.

[20] Atlagic B., Sagi M., Milinkov D, Bogovac B, Culaja S., “A way towards efficiency of SCADA infrastructure”, accepted for ECBS 2012, Novi Sad.