Program Generation Approach to Semi-Natural Simulators Design and Implementation

Emanuil Kirilov Markov, Vesselin Evgueniev Gueorguiev, Ivan Evgeniev Ivanov
Technical University Sofia, TUS
Sofia, Bulgaria
e-mail: {emarkov, veg, iei}@tu-sofia.bg

Abstract—**This paper presents ideas for design, implementation and results of creation of computerised semi-natural simulators of the different heterogeneous objects. The presented approach extends well known solutions for object simulators creation using the program generation technique. Simultaneous co-design of the control system and the simulator is explained as well. Both stand-alone and distributed objects and control systems are included in the presented work. Examples of different implementation of the presented approach are presented.**

Keywords – semi-natural simulation; program generation; distributed control system; hardware-in-the-loop.

I. Introduction

Computer-based simulators of many different complex objects are a reality today. They have become a reality with the expansion of cheap computers since the 80’s of the 20th century. The observation of many different implementations and approaches to simulators creations are available [1][2][3].

Large-scale hazardous objects like planes and nuclear power plants have been simulated for decades. Simulation of various real systems has multiple advantages compared to experimenting and use of the actual system. Some of the advantages are: the possibility to train the personnel to operate various types of machines; to use such simulations in preparations and optimizations of control algorithms; to repeat and analyse specific situations.

The two main classes of simulators – fully numerical simulators and physical simulators mark the boundaries. The first one implements a kind of mathematical model (any type). MATLAB® is a very good example of this approach [8]. The second one is physical (material) emulation of the object. There are a lot of simulators between these two endmost types. Now, we can find Hardware-in-the-loop (HIL) [9], Software-in-the-loop (SIL) [8], Agent-based simulators [10] and other approaches to simulators building inside the boundaries mentioned above. HIL and SIL are a bit opposite because HIL means that one implements a simulation of the object when the controller is ready-for-use. The SIL means that one has a computer-based model of the object and puts in the same environment the code of the controller and runs the two together only numerically. The Agent-based simulation is a new adoption of the component-oriented programming design approach and can be discussed in this paper as a way how to present distributed/de-centralized systems and to implement their simulations.

All these simulators implement some model of the object and communicate somehow with the controller or experimental environment.

A brief examination of the papers and on-line materials today shows many different simulators and simulation environments [4][5][6].

One of the definitions of control systems architecture concerns their geographical position. The variants are ‘concentrated/centralized’ and ‘distributed’ control systems. When we talk about distributed control systems their architecture reflects the structure of the object – distributed on the level of parameters (huge objects) or distributed as points of control. In both cases the control system is influenced by the communication network and all delays introduced by it, loss of packets, etc. [3]. Using the presented below approach both concentrated and distributed systems will be covered.

The paradigm for program generation of many kinds of software is not new. Its implementation in the area of control systems varies in aspects and mathematical background but has a stable place in today’s methodologies. Here we will discuss the version of program generation when the target system is build using pre-programmed library components which are only instantiated and linked in the real implementation. The full code-generation like the one used in MATLAB Embedded Studio [8] or ADA [11][12] based real-time systems is out of the scope of this paper.

The approach presented in this paper is focused on using one and the same tool (program generator) to design both the HIL-type “semi-natural” object simulator and the control system.

The name “semi-natural” means that the simulator is not only a computer, running the object model and connected to the controller via its physical interface, but additionally that the simulator can include parts of the real object’s hardware. In this case we have something that is mostly a simulator but has some elements of an emulator.

The present paper is structured as follows: Section II resents the proposed connection between control systems, objects and their simulators; Section III presents a short description of the used program generator and the formal model implemented by it; Section IV presents the implementation and analyses of several objects and their simulators; Section V is the conclusion.

II. BACKGROUND OF THE PRESENTED APPROACH

To implement a simulator using mathematical models of the object is an old and widely used idea. To implement an object emulator is a very old idea, too. To implement a simulator using both a mathematical model and elements of some physical hardware is a more recent one. To implement a simulator using a physical interface, which is the same as the original object interface is a much newer idea. It has been possible to do that only for the last 20 years when both computer hardware and peripheral devices became cheap enough and versatile. To implement a simulator for a distributed object is an old idea but it has been possible to implement it only in the last few years after the advance in communications and especially in guaranteed real-time communications.

A complex control system has a structure similar to that shown in Figure 1.

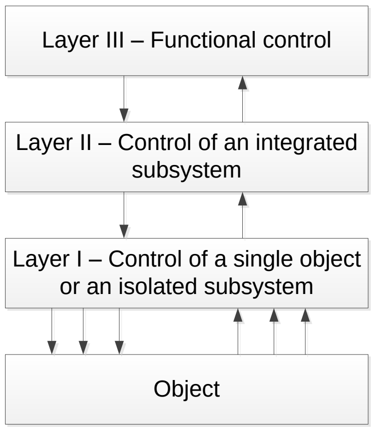
Layer III – Functional control
Layer II – Control of an integrated subsystem
Layer I – Control of a single object or an isolated subsystem
Object

Figure 1. General structure of a multi-layered control system.

Here, we assume the object to be a single one. Later, we will discuss how that view can be expanded for distributed objects. When we implement a computerized simulator with a real peripheral device, we can present it on the diagram shown in Figure 2.

When we combine both a control system and a simulator, the resulting diagram looks like the one shown in Figure 3.

This approach allows different ways to simulate the object to be used:

- full simulation using a physical interface of the same type as the real object (and possibly including parts of hardware from the object);
- partial simulation using a physical interface, emulating the real interface;
- partial simulation using signal exchange based on a type of networking.

In [1], several different variants for simulation are described but here we discuss only those that include parts of real physical hardware.

In all cases, if the object and/or control system has HMI it is presented "as is" in the couple "control system-simulator".

Discussing simulation of a distributed object, we have to include simulation of its distributiveness in the object properties. Depending on the type of the distributiveness we can simulate transport delays, internal status (as temperature) propagation delays and inequalities, distribution of process run (like conveyor and machinery near it) and so on. The complexity of that simulation can be higher than the complexity of the control system. This complexity increases in the case shown in Figure 3, when the control system and the simulator are connected via an additional "line" for activity synchronisation.

III. THE PRGEN – PROGRAM GENERATOR FOR DISTRIBUTED CONTROL SYSTEMS AND OBJECT SIMULATORS

The program generator used for establishing a simulation model is designed for creating distributed real-time control systems. It has been implemented in many different versions over the years [14][15][16]. It is based on an extended Moore machine implementing specific actions in each node of the state machine. Specific elements of its design reflect the possibility to generate both stand-alone and distributed systems. The generated distributed systems can operate as a "virtual mono-machine" or as a component-based (or agent-based) system. A graph representation of the control algorithms is chosen. The system can be described by its activities. Each activity is a separate thread. The activity thread consists of two different graphs:

- A State Transition Graph (STG) – a graph model for modelling the finite automaton, describing the general behaviour of the activity thread. The graph is represented like statechart described in [17].
- A Signal Flow Graph (SFG) – a graph model representing the signal transformation flow (the dataflow [18]). It is built by Function blocks, very similar to Simulink®. It is mainly used to model the continuous part of the system, to handle the I/O and communication drivers and to calculate complex predicates used in transitions of the State Transition Graph.

A. State Transition Graphs

Each system node has at least one activity thread. This thread is implemented by its STG defining the logical behaviour (although there are some implementations containing only one state with an infinite loop to itself). The STG has one entry point (initial node) with no other function than pointing where exactly the execution of the STG should start when the system is started. For each state of the STG one or more SFGs can be attached. For each state one or more transitions should be defined. A transition to the same state is acceptable. Decision making for transition to be performed is based on an associated to each state Binary Decision Diagram (BDD). Values for the BDD's predicates are taken from the node SFGs.

For each STG, an execution period is defined, defining how often the graph activates and executes its current state. The execution period can be modified if necessary during runtime (e.g., when the system is in idle/power-down state scanning inputs and refreshing outputs at a high rate are not necessary).

There are two types of transitions defined: synchronous and asynchronous. When a synchronous transition is executed the graph execution stops and the task goes in sleep mode until the execution period expires. When an asynchronous transition is activated, the task goes directly to execution of the state following the transition, without waiting for the period to expire.

B. *Signal Flow Graphs*

The SFG models the data flow of the system. Typically the SFG entry points are Function blocks representing I/O drivers, communication drivers or in specific cases data calculated by other SFGs. Then the data is passed to other Function blocks, which make transformations, check constraints and conditions and produce output for I/O drivers, communication, user visualization, database logging, etc. Each Function block in the SFG is executed only once per SFG execution. This is ensured by the connections between them. There are two types of connections: activating and non-activating. The SFG execution starts with Function blocks which do not have activating inputs. After their execution, the remaining blocks are checked for activation. The blocks that have all their activating inputs set, start execution and continue until the last block of the SFG finishes its job.

Each Function Block (FB) can have up to three types of inputs and two types of outputs.

The inputs of FB can be: Link Inputs (activating or non-activating); Parameter Inputs; Internal State Inputs. Each input can be linked to only one data source. An input which takes data from two different sources (e.g. from the outputs of two separate preceding FBs) is not available.

FBs can have two types of outputs. Static outputs – can be a data source of unlimited number of link inputs. Point to point outputs – cannot be a data source of inputs. Instead, they can be linked to a static input and change their value.

All inputs and outputs can hold matrix values, allowing complex system models to be generated.

What makes this implementation different from many others is that a communication subsystem is implemented as a number of communication modules using both logical and physical data exchange protocols. Using this approach the general model of the implemented system – control or simulator, is virtual mono-machine. The communication bus is implemented hidden, but observable [20][22].

Extended description of the model of the presented program generator can be found in [14][19].

IV. Simulator projects

Hereafter, we will present three different objects where the control system and the simulator were implemented using the presented approach. All objects have dominantly analogue behaviour, but they are very different in size and general complexity. A comparison between the implemented control system and simulators will be provided as well.

These objects are a fuel tank farm, a business building and a machine for making sausages (food industry). The first two objects are big distributed objects and the third one is a stand-alone machine. The following points will be discussed for each object: 1) why we need to build a simulator; 2) real object structure; 3) how the simulator is built, and 4) problems when the program generator was used to implement the simulator and how they were solved.

A. *Tank farm control system and simulator*

The tank farm control system includes the loading and unloading of tanks, calculation of the available fuel based on several different measured parameters, control of the pipe system. The control system is built by using an industrial OPC-based system connected to the field devices via Profibus connections (both DP and PA).

The need to invest time and money in a simulator of the tank farm has several dimensions: 1) it is used to design and tune different elements and the integrated control system; 2) it is used to train operative personnel; 3) it is used to analyse situations like leaks, inconsistency between different meters, etc.

Each tank has several integrated sensors for external and internal temperature, density, height of the fuel in the tank. Internal temperature is measured using distributed (multipoint) thermometer measuring temperature in several points on different depth in parallel. The density meter and level (height) of the fuel is traversed in the fuel starting from level 0 and finishing at the bottom of the tank. The pipe system includes mass-meter computerized devices and valve control. Valves can control fuel flow but they do not have internal position stabilization and have to be controlled by the main system. A general view of a tank, its sensors/actuators and pipes, is shown in Figure 4.

Having more than 10 different tanks and a large number of pipes makes the control and measurement task rather complicated. All tank sensors and actuators were connected to the upper level control system via Profibus PA in compliance with explosion hazardous areas safety requirements. Massmeters and valves as specific hardware were simulated using other mixed SW-HW simulators of the devices. With this simulating environment in mind, a construction for object simulation was built. The structure of the multilayered tank simulator is shown in Figure 5.

The tank simulator is built using the program generation approach. It is generated by the program generator using building blocks from the block library (pre-programmed and pre-compiled elements like “PID-controller”, “first-order filter”, “ADC driver”, etc.). As has been said, the tank simulator is a multi-layered system. Each element is simulated separately. Every local simulator is implemented using Single Board Commuter (SBC) with an ARM core and the necessary peripheral devices. Each SBC is driven by RTOS and the real-time part of the program generator – the RT interpreter and communication library.

Every low-level element is connected to the upper level simulator layer which coordinates them, implements an upper-level tank model and logic and supports communication to the upper levels of the system simulator. Thus, every complex tank simulator operates as a component of components. Using this approach and combining this simulator for every tank with simulators for connecting pipes and switching valves, we designed and implemented the tank farm simulator. The implemented simulator operated in various types of modes for normal and abnormal operation, to work in real-time and simulated time-flow with the control system. It covered the following areas of use:

- Control functions test and tuning (control system tests, tuning and development);
- Operators education/training;
- Abnormal situations analyses.

The main problems of the implemented simulator were two: 1) the Profibus network; 2) the number of elements to be simulated.

The Profibus communication has one positive side – it has a well-known traffic scheme and communication load and introduced delays can be calculated. It has one drawback – it is very hard to simulate in a real environment because developing of a Profibus slave is a hard, slow, specific and expensive task. To avoid this problem we implemented Master-Slave network over RS 422 media with an upper level protocol similar to the Profibus. All delays were included in the network modules. Removing Profibus from the system for simulation purposes we switched all sensors and actuators to other (supported by them) networks.

The solution of the problem with the number of the simulated elements was solved using an approach from the computer science. We had several tank types, valve types, pipe types and communication links. A template configuration for every different type of device was created. An instance of every specific device was parameterized and loaded into a SBC. All SBCs were connected following the connection scheme. Thus, implementing each template only once and configuring it for every specific instance all simulator elements were built. Based on the component-based system structure, they were connected one-by-one to the controller. This allowed the design group to verify and validate the solution starting from low and going to high system complexity.

B. Business building simulator

The next object that will be presented is a business building. It has many different stores, restaurants and other objects.

The decision to build a simulator for building control purposes was taken for several reasons: 1) experiments with the real equipment are possible and real data were collected but they are expensive and after the beginning of the official object exploitation some of them became impossible (because they include measuring devices of utility companies – electricity, water); 2) Experiments with different scenarios of exploitation are impossible on the real object; 3) training of the exploitation personnel has to be done periodically; 4) the building owner needs an experimental field test for malfunction and abnormal situation analyses.

Building control and data acquisition included electricity load control, all separate electricity, water and heat metering devices for every object, implementation of a number of scenarios for lighting, heating and other equipment control, control for abnormal situations and functioning, full time system log, etc. The building has several hundred sensors – smart or simple, several hundred actuators, switchers and intelligent output devices. This complex object has multi-layered requirements for exploitation. This requires a multi-layered control system. Simulation and testing of the full control system is impossible using standard approaches for computer simulation.

The designed simulator consists of many low-level simulators for every single object. Controlled sub-objects can be separated in several classes of similarity. A single template for every object class was designed. After that they were instantiated and parameterized. The final structure is similar to that shown in Figure 3. The final simulator system has to have about 3000 different function blocks for 7 computational and 3 logical low-level configurations and 2 upper levels implementing simulation of integrated functions. It is very hard to implement such a system. That is why the simulator size was reduces to 1/10 of the real size but including all computational and logical structures. Additionally, modules were implemented simulating human streams, some behavioural scenarios, and abnormal situations in the electricity and water supply (malfunctions injectors). All simulation modules were implemented using SBC ARM-based computers with the appropriate physical periphery and communication. From a generalized point of view the controlled object is a heterogeneous distributed discrete-event system including a number of analogue inputs and outputs and several sub-objects of analogue type of functionality. Many of the controlled elements communicate to the controller via MBUS. The structure of the MBUS connections is shown in Figure 6.

The control system includes a number of PLCs connected to their upper level (a SCADA system running on an industrial PC computer).

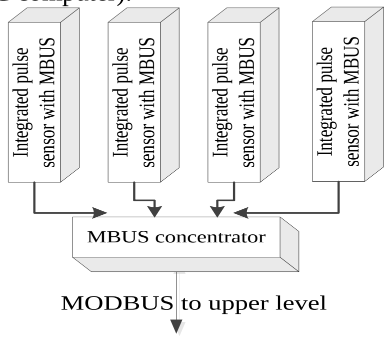
Integrated pulse sensor with MBUS
Integrated pulse sensor with MBUS
Integrated pulse sensor with MBUS
Integrated pulse sensor with MBUS
MBUS concentrator
MODBUS to upper level

Figure 6. MBUS to MODBUS connection

Every control loop is simulated by implementing an object model running on the SBC under RT OS and the real-time interpreter of the generated by the program generator configuration. One of the most important features of the simulator is the ability to be re-loaded with different system internal statuses (contexts) and to re-execute situations which have happened and have been logged. Additionally, the simulator can run in parallel with the real system in fast time and to predict the object behaviour. The implemented integral HMI was used to train building operators to understand and control its functioning and all included subsystems. Additionally, full database for process logging is included.

As in the previous example, one of the problems was the fact that simulation of a MBUS slave is hard. Fortunately, we did not need to simulate it to test functional implementation. There are a lot of MODBUS simulators for the upper level implementations. The communication library of the program generator was extended with modules implementing upper levels of MODBUS protocol. They were connected to the simulator of the low-level pulse sensors. The transmission from the simulation pulse module to MODBUS module was done using substitution of MBUS by I^2C, but following the scheme in Figure 6. All upper layers were kept as they were.

C. *Food industry machine simulator*

The third object that will be presented is a relatively small system from the food industry. It is a machine for making sausages. The size of the machine is about 2×2×3 m. It has very non-linear behaviour depending on many uncontrollable external disturbances – size and quantity of the prepared sausages or meat, quality of the bran for the smoke generator, variations of the current voltage and others. The mathematical model of the machine is statistical. It is a typical non-linear object with internally distributed parameters. Experiments with this machine are expensive and more over they are in-repeatable in general. Testing of the controller and its software is hard. It was decided to design and implement an object simulator to prepare an experimental environment with predictable behaviour and make testing and tuning of all control equipment repeatable. The simulator was based (as before) on a SBC computer with physical periphery as the real object. All analogue and discrete signals were implemented using process I/O devices with high resolution and precision. The mathematical model of the object was implemented using library modules from the program generator library. Additionally, a communication channel to the upper level of the simulator was implemented, which was used to set different parameters of the model (to switch between different sausages and meats and to simulate differences between different pieces of them in the machine). Using this component approach and program generation a simulator was built which covered the real machine behaviour up to 98%. The controller was designed to meet all project requirements. The main problem in this simulation was to build-in enough possibilities to induce disturbances in the object but, because this is mostly mathematical and logical and not a hardware problem. The controller was implemented using standard modules from program generator's library. The only problem was to connect triggers activating these modules and their re-parameterization to the upper level of the implemented simulator. Using external data lines similar to the ones shown in Figure 3 this was implemented.

V. ANALYSES OF THE PRESENTED APPROACH FOR SIMULATORS IMPLEMENTATION USING PROGRAM GENERATION

In the paper, three different objects and the approach to designing their simulators were demonstrated. One is a small but very complex food machine, the second is a huge distributed dominantly discrete object (a business building) and the third is a heterogeneous distributed object including transport delays, a lot of non-linearites and additionally several problems in communication simulation. All designed simulators were of the type 'semi-natural' or 'partial emulators'. They include both specific and general-purpose hardware and operate in real-time software. The presented objects are very different. These simulators are based on one and the same approach – a component design using an automated tool, a program generator. This generator produces a system configuration that is executed by the real-time system. All hardware for the simulators is similar – SBC with ARM processors and physical periphery of the type similar to the real object hardware interface. Depending on the object size the real simulator is implemented using one or several SBCs. They are connected to the implemented control system via analogue, discrete, pulse and communication interfaces.

Analyses of the complexity of the implemented simulators compared with the control system show that they are on similar levels. In all three situations the design phase of the simulator facilitating the good understanding, modelling and design of the control system. The possibility to use simulators running with time speed different form the real time enabled both fast checks of situations and real-time prediction of eventual dangerous object behaviour to be conducted. The numerical nature of the simulators and control systems enables situation analyses using system logs and other status information for events that have happened. Education and training of operators of those systems is based on the same principles. The main difference is the size of the simulator. Hardware implementation depends on the number of inputs, outputs and communication lines to be simulated. The possibility of the program generator to build templates and to instantiate them by sets of real parameters speeds up many times the generation of systems with big number of similar elements (as every component-based system).

Comparing the time necessary for the simulators building we will say that the most hard for design from the modelling point of view was the food machine simulator. The most time consuming was the building simulator because it had the biggest number of I/Os and real elements to be simulated even using template instantiation.

VI. CONCLUSION

The paper presented an approach to building object simulators using program generators and one and the same toolset for the control system and simulator implementation. The presented semi-natural simulators enabled the control system and the simulator to be designed simultaneously. This approach reduces the investments and risks in the design and implementation phases of the control system design. The possibility to use that implemented simulator not only for control system tests but for personnel training and for events and abnormal situations analyses makes them a helpful, relatively inexpensive tool with great flexibility and versatility. The difference from other HIL is the ability to include easily parts of the real hardware together with the simulated one.

ACKNOWLEDGMENT

Parts of the presented work are funded by the Bulgarian NSF under DRNF02/3 project.

REFERENCES

[1] J. A. Carrasco and S. Dormido, Analysis of the use of industrial control systems in simulators: State of the art and basic guidelines, ISA Transactions, Volume 45, no. 1, January 2006, pp. 295-312

[2] A. Negahban and J. S. Smith, Simulation for manufacturing system design and operation: Literature review and analysis, Journal of Manufacturing Systems, Volume 33, Issue 2, April 2014, pp. 241–261

[3] W. Li, X. Zhang, and H. Li, Co-simulation platforms for co-design of networked control systems: An overview, Control Engineering Practice vol.23, 2014, pp. 44–56

[4] The Rapid Automotive Performance Simulator (RAPTOR), http://www.swri.org/4org/d03/vehsys/advveh/raptor/default.htm [last accessed: 08.08.2014].

[5] M. Pasquier, M. Duoba, and A. Rousseau, Validating Simulation Tools for Vehicle System Studies Using Advanced Control and Testing Procedure, http: //www.autonomie.net/docs/6 - papers/validation/validating_simulation_tools.pdf [last accessed: 08.08.2014].

[6] IAEA, Use of control room simulators for training of nuclear power plant personnel, Vienna, 2004, IAEA-TECDOC-1411, ISBN 92–0–110604–1.

[7] M. Johnstone, D. Creighton, and S. Nahavandi, Enabling Industrial Scale Simulation / Emulation Models, In Proceedings of the 2007 Winter Simulation Conference, 2007, pp. 1028-1034.

[8] MathWorks, Generate and verify embedded code for prototyping or production, http://www.mathworks.com/embedded-code-generation/, [last accessed: 08.08.2014].

[9] J. A. Ledin, Hardware-in-the-Loop Simulation, Embedded Systems Programming, Feb. 1999, pp. 42-60.

[10] C. Macal and M. J. North. Agent-based modeling and simulation. In Proceedings of the 2009 Winter Simulation Conference, ed. M. D. Rossetti, R. R. Hill, B. Johansson, A. Dunkin, and R. G. Ingalls, Piscataway, New Jersey: Institute of Electrical and Electronic Engineers, Inc., 2009, pp. 86-98.

[11] D. A. Watt and B. A. Wichmann, W. Findlay, "Ada: Language and Methodology." Prentice-Hall, 1987

[12] A. Burns and A. Wellings, Real-Time Systems and Programming Languages (Fourth Edition) Ada 2005, Real-Time Java and C/Real-Time POSIX, April 2009, Addison Wesley Longmain, ISBN: 978-0-321-41745-9

[13] N. Baldzhiev, V. Bodurski, V. Gueorguiev, and I. E. Ivanov, Implementation of Objects Simulators and Validators using Program Generation Approach, DESE 2011, Dubai, UAE, December 2011

[14] C. K. Angelov and I. E. Ivanov, “Formal Specification of Distributed Computer Control Systems (DCCS). Specification of DCCS Subsystems and Subsystem Interactions”. Proc. of the International Conference “Automation & Informatics’2001”, May 30 - June 2, 2001, Sofia, Bulgaria, vol. 1, pp. 41-48.

[15] I. E. Ivanov and K. Filipova, “Integrated scheduling of heterogeneous CAN and Ethernet-based hard Real-Time network”, Proc. of IEEE spring seminar 27th ISSE, Annual School Lectures, Bulgaria, 2004,vol. 24, pp.481-485

[16] I. E. Ivanov and V. Georgiev, “Formal models for system design”, Proc. of IEEE spring seminar 27th ISSE, Annual School Lectures, Bulgaria, 2004,vol. 24, pp. 564-568

[17] D. Harel, “Statecharts: A visual formalism for complex systems” Science of Computer Programming 8 (1987), pp. 231-274

[18] K. M. Kavi and B. Buckles, “A Formal Definition of Data Flow Graph Models” IEEE Transactions on computers. vol. C-35, no. 11, November, 1986

[19] I. E. Ivanov, “Control Programs Generation Based on Component Specifications”, PhD thesis, 2005, Sofia, (in Bulgarian)

[20] C. K. Angelov, I. E. Ivanov, and A. A. Bozhilov. Transparent Real-Time Communication in Distributed Computer Control Systems. Proc. of the International Conference “Automation & Informatics’2000”, Oct. 2000, Sofia, Bulgaria, vol.1, pp. 1-4

[21] A. Dimov, I. E. Ivanov, and K. Milenkov, "Component-based Approach for Distributed hard Real-time Systems", Information Technologies and Control, 2, 2005

[22] A. Dimov and I. E. Ivanov, Towards development of adaptive embedded software systems, Proceedings of TU Sofia, vol. 62, book.1, 2012, pp. 133-140

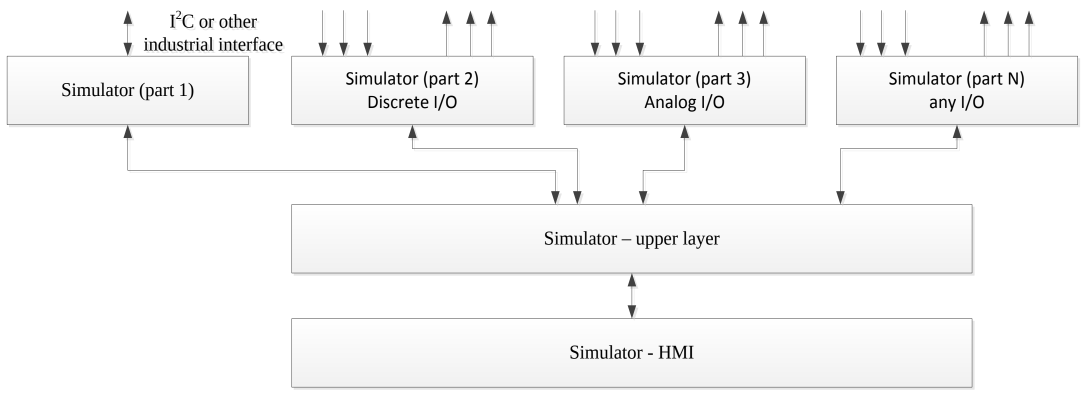
I2C or other industrial interface
Simulator (part 1)
Simulator (part 2) Discrete I/O
Simulator (part 3) Analog I/O
Simulator (part N) any I/O
Simulator – upper layer
Simulator - HMI

Figure 2. General structure of a multi-layered semi-natural simulator.

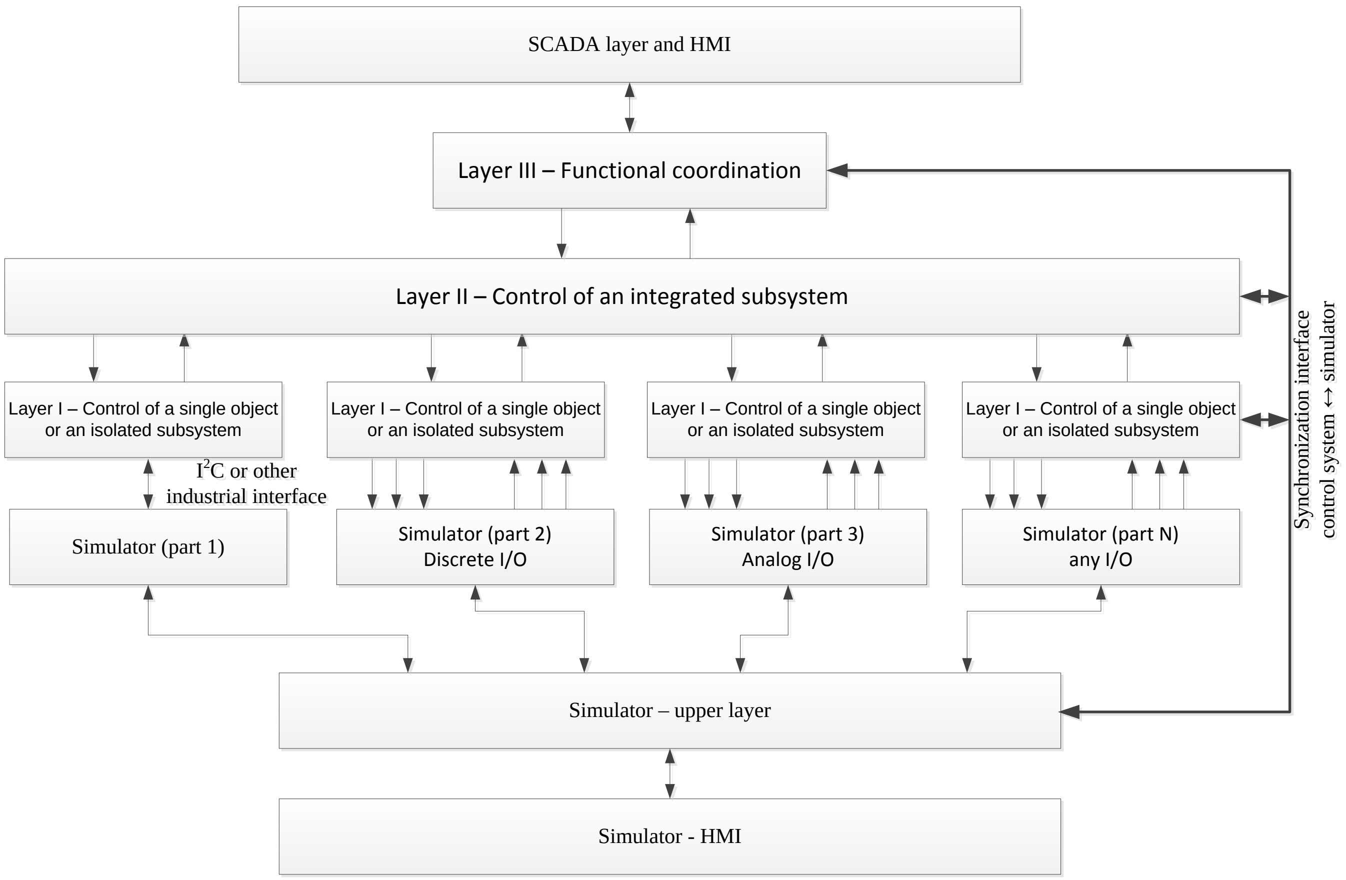
SCADA layer and HMI
Layer III – Functional coordination
Layer II – Control of an integrated subsystem
Layer I – Control of a single object or an isolated subsystem
Layer I – Control of a single object or an isolated subsystem
Layer I – Control of a single object or an isolated subsystem
Layer I – Control of a single object or an isolated subsystem
I²C or other industrial interface
Simulator (part 1)
Simulator (part 2) Discrete I/O
Simulator (part 3) Analog I/O
Simulator (part N) any I/O
Simulator – upper layer
Simulator - HMI
Synchronization interface control system ↔ simulator

Figure 3. Combined structure “control system ↔ simulator”

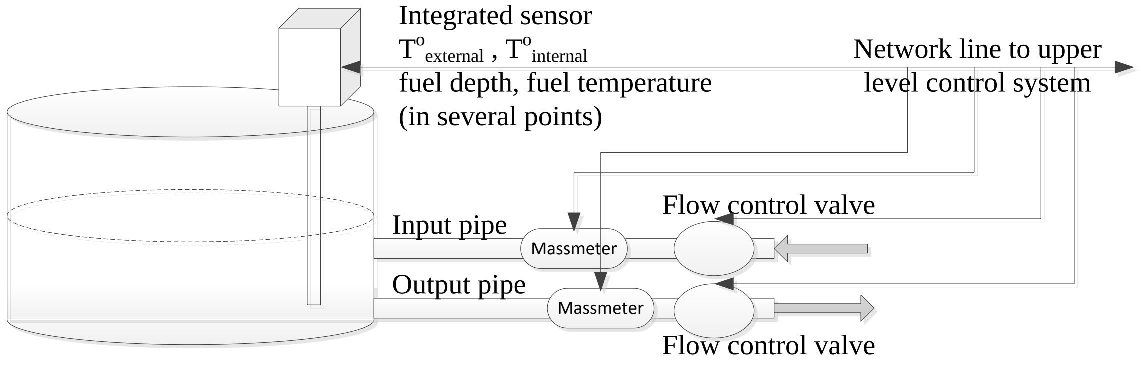

Figure 4. Fuel tank – general structure and sensors/actuators

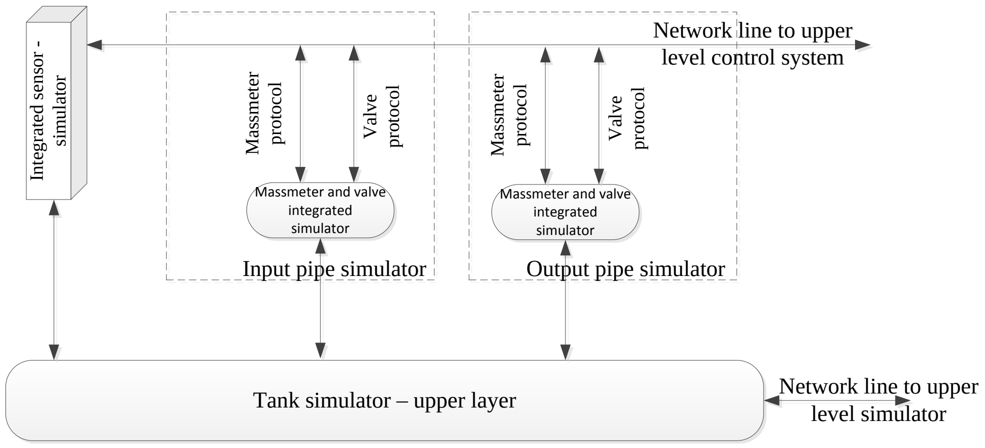
Integrated sensor - simulator
Massmeter protocol
Valve protocol
Massmeter protocol
Valve protocol
Network line to upper level control system
Massmeter and valve integrated simulator
Massmeter and valve integrated simulator
Input pipe simulator
Output pipe simulator
Tank simulator – upper layer
Network line to upper level simulator

Figure 5. Fuel tank – simulator structure