Program Generation Approach for HIL Simulators Design

Vesselin E. Gueorguiev, Ivan E. Ivanov, Emanuil K. Markov, Desislava Georgieva

***Abstract* — The problem of design, implementation and tuning of HIL simulators is rising together with systems complexity. Many elements of the real system cannot be simulated easily and precisely. In case of hybrid objects the complexity arises. Very hard is implementation of mixed simulators – having both computer elements and hardware elements for extended object simulation. Here is presented an approach of using program generation technique for creation of object simulators. Using one and the same approach for implementation of the simulator and the control system is also discussed. Objects and control systems having different structure are presented in this paper.**

***Index Terms* — program generation; HIL simulation; object-control system co-design.**

I. Introduction

Design and implementation of hazardous objects or objects generation real danger for people is a complex task. To make this process easier and robust from many years object, control system and object simulator are taking part simultaneously. With the era of cheap computers for embedding, computer-based simulators became an usual thing. Many approaches for designing and implementing simulators are available [1][2][3].

Very complex systems like nuclear reactors, planes of any scale and similar are simulated form tens of years. The increasing complexity of medical apparata makes them an object for simulation as well. Together with human body simulators, this is one of the most rapidly growing area of research. To use simulator of a real system has many advantages. Comparable with experiments with real system some of these advantages are: avoiding some dangerous situations; tuning and optimization of control algorithms, alarms and abnormal situations handling; near to reality personnel training; history generation for specific situation analyzes.

Model-in-the-Loop (MIL) [4], Hardware-in-the-Loop (HIL) [4], Software-in-the-Loop (SIL) [5], Agent-based simulators [6] are some of the various types of modelling techniques today. Simulators are varying from fully numerical to mostly physical as implementation. All simulators are built on the idea of implementation of some model of the object. This model (the simulator) has to exchange data with the experimental environment or the controller.

Submittet for review 06.2021.

Vesselin Evgueniev Gueorguiev, FCST, Technical University of Sofia, 1000 Sofia, Bulgaria (e-mail: veg@tu-sofia.bg).

Ivan Evgeniev Ivanov, FA, Technical University of Sofia, 1000 Sofia, Bulgaria (e-mail: iei@tu-sofia.bg).

Emanuil Kirilov Markov, emarkov@tu-sofia.bg

Desislava Georgieva, NBU, New Bulgarian University, Sofia, Bulgaria (e-mail: author@ie-bas.org dvelcheva@nbu.bg).

A brief examination of the papers and on-line materials today shows many different simulators and simulation environments [7][8][9].

The control systems architecture can be defined from different points of view. One of these points is how the object is situated – if it is 'concentrated/centralized' and 'distributed' we have correspondingly centralized or distributed control system. Discussing distributed control systems we see how their architecture answers the object's structure. The object can be distributed on the level of parameters (huge objects). It can be distributed as points of control – many input points positioned (enough) far from each other. In all cases, the control system includes communication subsystem that generates delays, loses data because of different reasons, etc. [3] With the approach, presented below, both concentrated and distributed object simulators, communication network influence, operational reorganization and other problems will be addressed.

Here will be presented the usage of program generation technique for control systems and simulators implementations. These implementations in the area of control systems cover wide range of aspects, mathematical and logical elements, networking and other. The program generation approach is one of the most used in the area of both control systems implementation and simulators implementation.

Program generation itself has many variants – from full code-generators (examples are ADA-based real-time systems and MATLAB Embedded Studio) to configurators, based on pre-compiled libraries and table-generation of the system structure.

In many situations different sets of tools are used for control system implementation and for simulator implementation. Here is presented an integrated approach for design and implementation of both the control system and the hybrid-type object simulator using one program generator.

The present paper is structured as follows:

- Section II resents how objects, object simulators and control systems are elements of one and the same approach;
- Section III presents a short description of the used program generator and the formal model implemented by it;
- Section IV presents several objects, their analyses and how their simulators are implemented;
- Section V includes the conclusion.

https://doi.org/10.47978/TUS.2021.71.02.013

II. Background Of The Presented Approach

We mentioned before, one the main types of simulators is the s-called "Software-in-the-loop" (SIL). This is an implementation of some mathematical and logical model that can be connected to the controller – an other software implementtation.

This idea is old, widely used and implemented in many research environments (like MATLAB/SIMULLINK and other). Next step to make the simulator or the controller more realistic is to include some physical interface. The idea to use the same interface as the original object interface is relatively new.

It became possible to use this approach only in the last 20-25 years after the jump in performance, stability, quality and accessibility of the computer hardware. In that period peripheral devices became enough versatile and cheap, and industrial communication became enough fast and stable.

The structure of a complex control system for complex object is of layered type. The object is the lowest layer. On it is Layer I – the control systems for local sub-object or element. Layer II is the integrating controller – for specific subsystem or object. The Layer III is the functional control – integral functionality/coordination. This structure is very common to the industrial systems for complex automatizaton from mid-80s.

In the next discussions we will assume first a single (monolithic) object. Next will be discussed how the program generation approach for simulators implementation expands to distributed objects and controllers.

The implementation of computer-based simulator with real (physical) periphery we can present as it is shown in Fig 1.

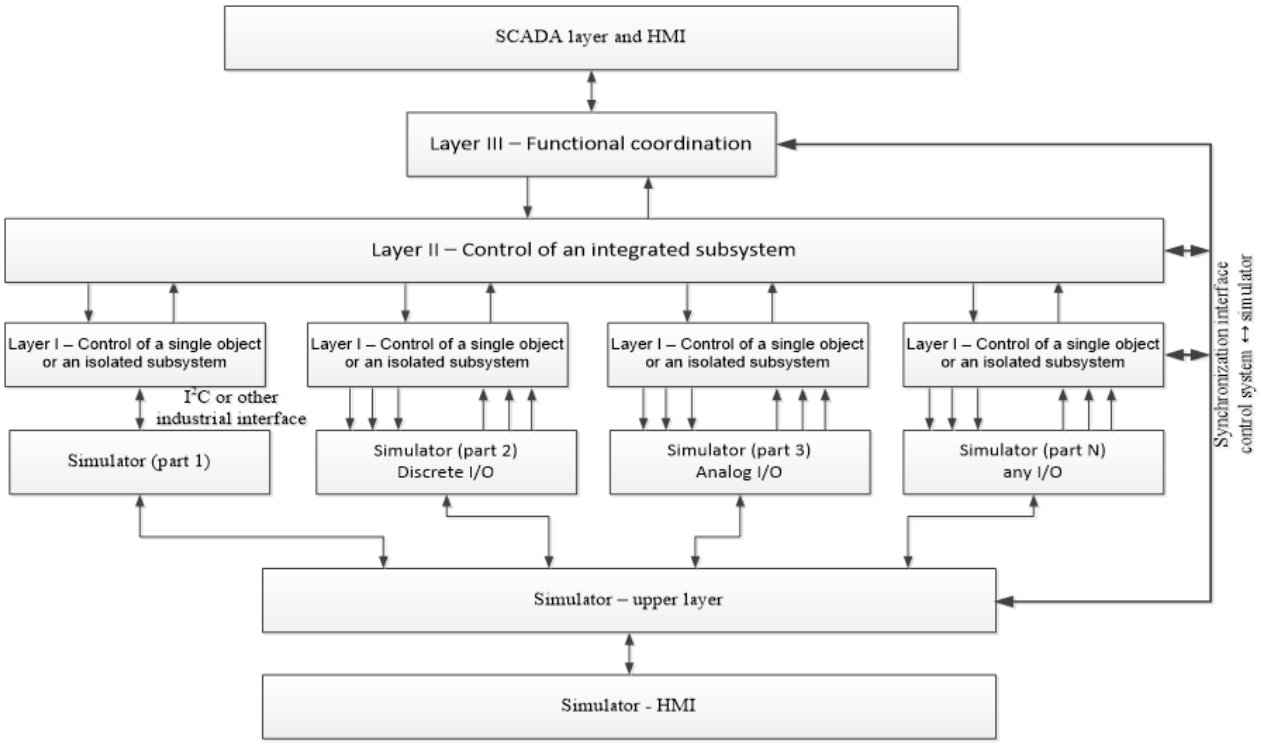

Fig. 1. General structure of a multi-layered simulator

The combination of the control system and the simulator in one complex system can be represented with the structure shown in Fig. 2.

According to Fig. 2, we can simulate the object in different levels of physical detailization:

- Full simulation. He is used physical interface. It is of the same type as that of the real object. It may include parts of hardware of the object;
- Partial simulation. Used physical interface emulates the real interface;
- Partial simulation with data exchange. This exchange simulated physical connections on a base of computer networking.

Different variants of simulation implementations are discussed in [1]. Only those including elements of physical hardware will be targeted here.

In any case, if some elements of Human-Machine Interface are presented, they are included "as is". This means that all protocols – commands and data exchange are included in their original versions.

When we have to simulate a distributed object, its "distributiveness" has to be simulate also. All know types of distributiveness have to be simulated – delays of any type, parameter distribution (like temperature), communication and software delays, transportation, etc. As usual, the difficulty and complexity of this simulation is higher than that of the control system. When we have to implement additional "synchronization" line between the simulator and the control system to control the simulated time flow, as shown in Fig. 2, the complexity increase even more.

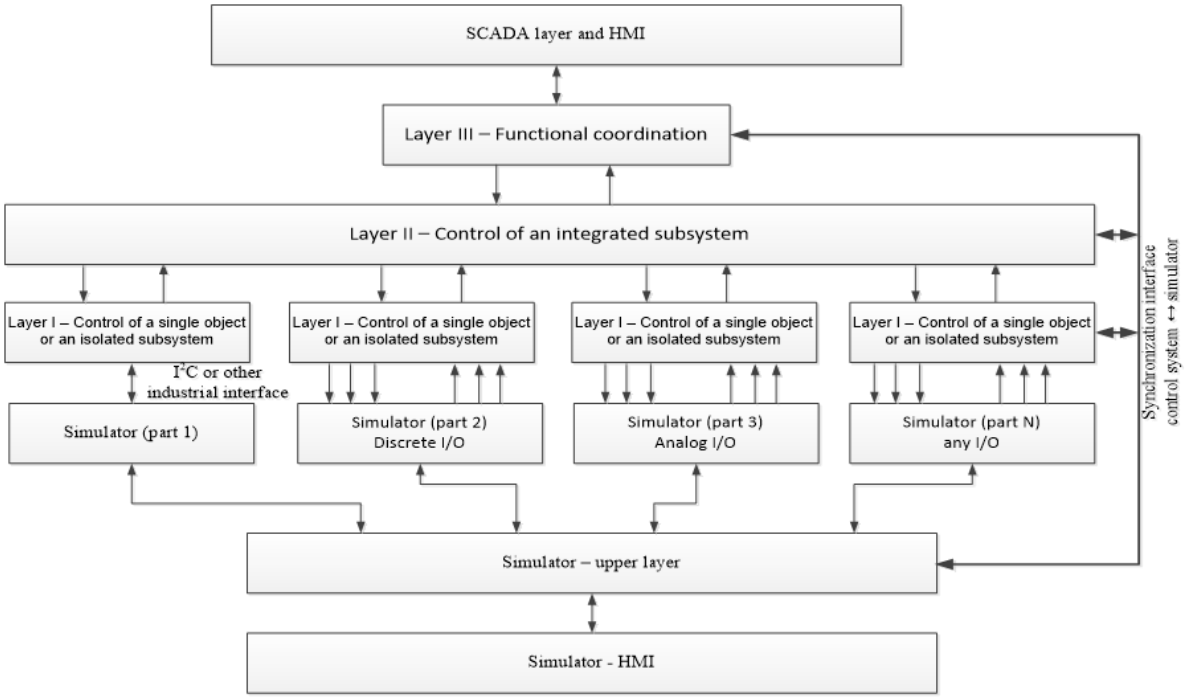

Fig. 2. Combined structure "control system ↔ simulator"

III. The PrGen – Program Generator For Distributed Control Systems And Object Simulators

The idea for program generation here is realized as a program generator table type supported with extended program library. It is adequate for both stand-alone and distributed systems. This program generator has many version over the time [10][11][12].

The basic idea of the control structure representation is an extended Moore machine. It assigns specific actions for every node of the state machine. Very specific property of this program generator and implemented by it extended Moore machine is the possibility to embed communication in the state graph. This approach makes possible to generate distributed systems operating as a "virtual mono-machine" or as an agent-based / component-based system.

Technically control algorithms are presented as graphs. Every designed system is described by its activities. Each activity is implemented by separate thread. The activity thread consists of two different graphs:

- A State Transition Graph (STG) – the description of the general behaviour of the activity thread is done by finite automaton modelling using graph model. The graph representation is as the described in [13] statechart.
- A Signal Flow Graph (SFG) – a graph model representing the signal transformation flow (the dataflow) [18]. Similar to Simulink® data model it consists of data paths and Function blocks. It represents data transformation paths (mostly continuous system's part), decision preparation based on numerical calculations (based on numerical calculations predicates used in transitions of the State Transition Graph), handles I/O and communication drivers.

A. State Transition Graphs

System behavior is represented by at least one activity thread for each node. Every activity thread is formally represented by its STG. This STG defines the logical behaviour of the thread. Some specific implementations contain a single state with an infinite loop to itself. Every STG has an entry point (initial node). This point only shows where the execution of the STG starts. Depending of the data transformations associated to some state, one or several Signal Flow Graphs can be attached to it. Every state has at least one output transition (the exception is the END state). Transition to the same state is normal but has to be done in different execution intervals (e.g. no infinite loops in one control period). Decision making for transition from one state to the other is based on an associated to each state Binary Decision Diagram (BDD). Values for the BDD's predicates are coming from the State Transition Graphs' of the node.

The execution period for each STG is defined explicitly. This period defines how often the graph is activated to execute this specific state. If necessary, the execution period can be changed in runtime, but only when this specific graph is not in its execution mode.

To implement both synchronous (time-driven) and asynchronous (event driven) control algorithms, two corresponding types of state transitions are defined. A synchronous transition means that the graph execution stops until next activation period arises. The associated activity task falls in sleep mode. When an asynchronous transition takes place, the activity task moves to the following state without stops or delays and starts its execution immediately.

B. Signal Flow Graphs

The SFG models the data flow of the system. Entry points of the SFG are usually Function blocks (FB) implementing some drivers – input drivers (or in some situations with delayed output – output drivers), communication drivers or data transmission blocks. Input data and data from previous execution moments are propagated to some FBs. These FBs

make calculations (e.g. data transformations), check constraints or conditions. Produced data are propagated to other FBs. Thus are generated outputs of different type – output to device drivers, communication, to the user interface, data logging, etc.

The core of this execution control is the data flow control. For a single activation of the SFG, each FB is activated only once. The FB activation is function of FB's input connections readiness. There are two types of connections: activating and non-activating. At the SFG's activation point the execution starts with FBs having only passive (non-activating) inputs. This means that this FB is always ready for activation. An example is input driver FB. It only generates data output but its input is from the environment. When all "always ready" FBs are executed the remaining FBs are checked for readiness. If all activating inputs of some block are "ready" this means that this block is ready for activation and it is queued for activation. In some moment more than one FB can be ready for activation. They are executed in the way they are found "ready". This does not influence final SFG results because in that moment executed modules are independent to each other. The SFG interpreter follows this logic and activates all FBs in the presented manner until all FBs are executed. In case of data loops all those paths are "non-activating" and cannot generate infinite calculation loops in a single execution period.

For every FB are defined three possible type of inputs. There are defined also two possible types of outputs.

The inputs are as follows [22]:

- Link Inputs (activating or non-activating);
- Parameter Inputs;
- Internal State Inputs.

Every input has a single data source. Links to more than one data sources (FB outputs) are impossible because this will be ambiguity.

Function Block outputs are as follows [22]:

- Static outputs – can be a data source of unlimited number of link inputs.
- Point to point outputs – forced target output. They cannot be data sources for data inputs.

All inputs and outputs can be scalar, vector or matrixes. This allows to generate complex system models.

The difference between this program generator's model and other is the fact that the communication subsystem is included in the FB library. There are several Function Blocks implementing logical and physical data exchange protocols for internal and external data exchange. Exploiting this specific approach one can generate stand-alone or distributed control system or simulator as a "virtual mono-machine". This makes the design process much easier and free of specific technical details. The communication bus is implemented hidden, but observable [14][15].

The extended description of the model of the presented program generator is available in [10][16].

IV. Implementation of Object Simulators

In this part of the paper will be discussed the control systems and the simulators of two different types of objects. Both those combinations "control system-hybrid simulator" were implemented using the presented before program generator PRGEN and implemented by it using systems generation approach. Both objects have analogue and discrete (logical) behavior. Also, their sizes as number of inputs/ outputs are different. The complexity is incomparable. The second one has a batch type of functioning.

The first object is a business building. The second one is an apparatus for extracorporeal plasma apheresis. The business building is huge distributed object. The medical apparatus is a relatively small stand-alone machine.

The discussion will cover the following elements:

- how the simulator will help;
- the structure of the real object;
- problems in simulator's program generation process and their solving.

A. Business building simulator

The first object that will be presented is a business building. It has restaurants, stores, several floors and other controllable objects.

Any attempt to implement control system for such an object without decomposition and vertical integration will fail because of the very huge number of control points, feedback signals, distributiveness in both parameters and geography, unobservable influences between system elements, etc. There are other reasons, which are as follows:

- Only limited experiments with the real equipment are possible. Collection of real data is expensive. After the object's exploitation beginning many experiments are impossible (because they need to collect data, property of the utility companies – water, heating, electricity);
- experimenting some dangerous scenarios is impossible after the official start of the object exploitation;
- periodic training of the personnel is needed;
- an experimental test-bed for analyses of specific situations is needed.

A list of some elements of control are:

- total electricity load measuring (and control);
- for every separate object – reading heat, water and electricity metering devices;
- heating and other equipment control;
- control for abnormal situations and functioning;
- unbreakable system log;
- etc.

Thousands sensors with different level of complexity were installed in the building. Hundreds of actuators, switchers and intelligent output devices had to be controlled. So complex object has multi-layered requirements for exploitation.

All of these technical, functional and security requirements needed a multi-layered control system. Without of an appropriate simulation the testing, tuning and normal exploitation of this control system was impossible.

System decomposition was provided first on the level of separate sub-objects. According to their similarity several classes were built. A single class template for every specific class was generated (PRGEN provides a possibility to generate templates with different levels of complexity and to store them as an extension to the basic FB library). These

templates were instantiated for every sub-object. This is a normal object-oriented technique but it is not very common in this area. The Fig. 2 represents an approximation of the final structure. The size of the final simulator had to have two upper levels for simulation of integrated functions; three logical low-level configurations; seven computational threads with three thousand function blocks included. This size makes the process very hard. To avoid this the size of the simulator was reduces ten times (all similar elements were presented in reduced quantity). Nevertheless, it included all computational and logical structures.

Additionally, Function Blocks library was extended with blocks simulating human streams and other specifics. Specific automata for abnormal situations were designed. They include malfunctions injectors in the electricity and water supply simulation, different behavioural scenarios and other. All simulation modules were implemented using ARM-based Single-Board Computers (SBC). They had the required physical periphery and communication.

As a conclusion we can say that the controlled object is a heterogeneous distributed discrete-event system. It has huge number of analogue inputs and outputs. It also has several sub-objects of analogue type of functionality. Many input and output devices communicate to the controllers using MBUS physical layer and MODBUS protocol over it. The structure of the MODBUS/MBUS connections is shown in Fig. 3.

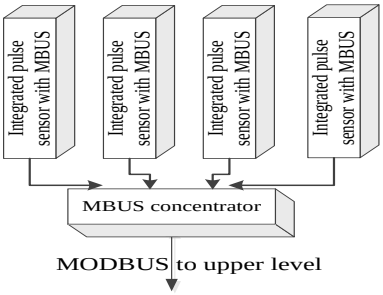

Fig. 3. MBUS to MODBUS connection

The upper control layer is implemented using SCADA system running on an industrial PC computer. Number of PLCs connected to SCADA implements the lower control layer.

For every control loop was implemented a simulation of the controlled object (sub-object). This simulation runs on SBC under Real-Time Operating System. The real-time interpreter of the generated by the program generator configuration runs as a RT OS task(s). The control system is implemented on PLCs. These PLCs and the SBCs were connected by real hardware interface. One of the most important features of the simulator is the ability to be re-loaded with different system internal statuses (contexts). This makes possible the re-execution of situations which have happened and have been logged.

Additional feature of the simulator is that it is able to operate in parallel of the real object. It is able to work in real-time or in fast (prediction) time for object behavior prediction.

The implemented Human-Machine Interface was used for building operators training and as an illustration of all systems, control approaches, functionalities and exploitation boundaries.

One of the specific problems, very hard for solving, was MBUS slave simulation. Because MBUS was only a carrier for MODBUS protocol, a MODBUS upper level simulator was included in the FB library. The simulator of the low-level pulse sensors was implemented. This simulator used I^2C instead of MBUS to simulate data input and transfer from pulse devices to MODBUS module. All upper layers remain unchanged.

B. Extracorporeal plasma apheresis machine simulator

The second object that will be presented is a relatively small medical apparatus – an extracorporeal plasma apheresis machine (a kind of perfusion pump).

Perfusion pumps are used in many different medical procedure. The discussed here apparatus is used for specific infusion or extra-thoracic circulation procedures. They require very high accuracy pressure control and flow metering and limiting throughout the process [17][18]. Schematics of the object is presented in Fig. 4.

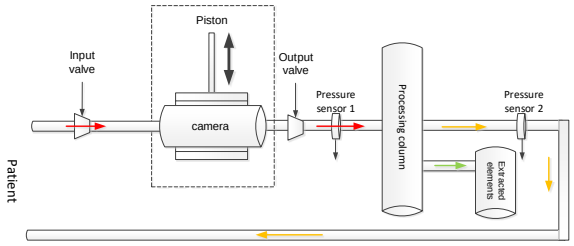

Fig. 4. Schematic diagram of an extracorporeal perfusion apparatus.

The system discussed here is an extension of the one presented in [19][20]. It operates like the left heart chamber. The main difference is in the fact that this system can be connected to a patient. It can infuse, process and return patient's blood in extracorporeal loop. Thus, the control object is an apparatus for extracorporeal biofluids processing. It is based on a process analogous to the left heart chamber. It collects biofluid from some reservoir or blood form the patient and implements diastole refilling "heart's" chamber. Closing input valve it starts to press the chamber. The collected fluid leaves it (systole) with controlled pressure or flow ad goes for processing in the nanofilter or chemical column. After this processing the blood is returned to patient. In case of some other biofluid processing it is collected in an output reservoir. A schematics of the circulationis shown in Fig. 4.

Simplified state chart of one of the implemented batch processes is shown in Fig. 5. It is implemented as a specific STG with number of SGFs as separate activity.

As before, this simulator is based on a Single Board Computer. The real object and the simulator have the same physical periphery. All simulation and exchange of analogue and discrete signals were implemented using Analog-to-Digital and Digital-to-Analog converters having high resolution and precision. All pulse devices – outputs and counters operate in 8 ns/pulse resolution. The model of the apparatus includes both analogue and discrete elements. They were implemented using program generator library modules. A

specific fast speed communication channel between the controller and the upper level of the simulator was implemented. An other fast speed channel for simulation control was included also. Using it the investigator can set different parameters of the model (low or high haematocrit, too much gas in the blood, etc.). Program generation and a component approach were used to build the simulator. This simulator covered near 100% of the machine behavior for selected control parameters. This made possible to design a controller meeting all security and process control requirements. Provided before design phase risk-analysis emphasized many possible problems. Most of them were built in the simulator. Managing them through HMI the investigator (and trainee in teaching process) can induce problems in operation and to see (or teach) how they are processed. This problem intrusion process is implemented by specially designed Complex Function Blocks. These specific Complex Function Blocks include inputs for external triggering for specific disturbances and for re-parameterization from the upper level of the simulator. Using external data lines similar to ones shown in Fig. 3 this was implemented.

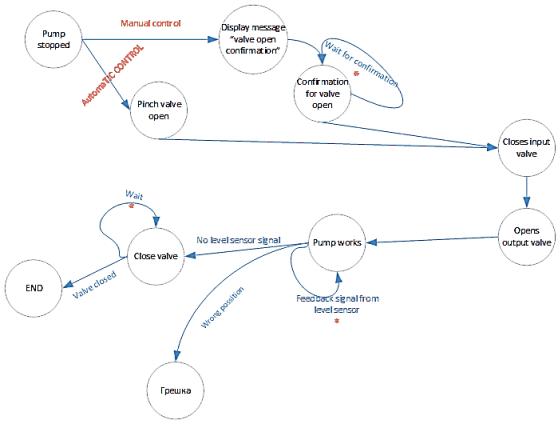

Fig. 5. State chart diagram of a specific simulated process

V. Analyses of the Presented Approach for Simulators Implementation Using Program Generation

Analysis of the presented simulators has to include the amount of work to implement the simulator, similarity between the real object and the simulator, possibility to operate in real time and in simulated (slow or fast) time, possibility for disturbance intrusion, flexibility and adaptability of the simulator to investigators needs.

Both presented simulators have analogue and discrete parts, operating simultaneously and dependent to each other. Both have very big nonlinearities. Both have user interfaces introducing significant problems with operational mode changes. Both simulators are of type HIL but especially hybrid simulators. They include general-purpose hardware but have also specific hardware extensions. They operate in real-time programming mode. These simulators are based on the same approach – a component design using an automated tool, a program generator. The program generator produces the system configuration. This configuration is loaded and executed by the real-time configuration interpreter over a specific RT OS.

For the plasma apheresis machine a specific problem was the fact that we do not have precise model of blood fluid and it changes under the procedure. Luckily, we do not need this precise model but only partial one covering changes is haematocrit, viscosity, filter clogging, spontaneous gas separation from the blood, sensors malfunction, uniqueness of each pump loop, etc.

Simulators were built using ARM processor based SBCs and specific process periphery of the type similar as the object's hardware interface. The plasma apheresis simulator was built using one SBC. The building simulator included more than ten SBCs. Connection to the control systems was via analogue, discrete, pulse and communication interfaces and has structure similar to Fig. 3.

All simulator can operate in real time and in fast and slow time.

The work for implementation of every simulator is tens of times lower than implementation by programming. Most of the elements were hard for fully numerical simulation and only the hybrid approach was adequate.

Complexity analyses of the simulators and the corresponding control system show that they are on similar levels.

Because the simulators and control systems are numerical in their core, it is possible to make situation analysis using detailed data logs for events, reactions, analogue signal levels, etc. More over, this makes possible to recover some situation starting from some moment using full data dumps.

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).

Comparison between simulators about modelling problems shows the plasma apheresis apparatus as harder even it is smaller. The building simulator is more time consuming because it had very big number of I/Os and physical elements or their simulations to be implemented.

VI. Conclusion

Here was presented an approach to designing and implementing object simulators using program generators. The main presented idea is to use the same toolset for the simulator and control system implementation. The presented approach enabled simultaneous design of the hybrid simulators and the corresponding control system. All those simulators are built using the PRGEN program generator and its Function Blocks library. Some specific hardware for object peripherals representation was designed according specific implementations. This approach reduces the investments and risks in the design and implementation phases of the control system design. The possibility to use implemented simulator for events and abnormal situations analyses makes them a helpful, versatile and relatively inexpensive tool with very big adaptability, flexibility and extensibility. The difference from other hybrid simulators is the ability to easily include parts of the real hardware together with the simulated one.

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, https://doi.org/10.1016/S0019-0578(07)60196-7

[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, https://doi.org/10.1016/j.jmsy.2013.12.007

[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, https://doi.org/10.1016/j.conengprac.2013.10.010

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

[5] MathWorks, Generate and verify embedded code for prototyping or production, http://www.mathworks.com/embedded-code-generation/, [last accessed: 25.07.2021].

[6] The Rapid Automotive Performance Simulator (RAPTOR), http://www.swri.org/4org/d03/vehsys/advveh/raptor/default.htm [last accessed: 25.07.2021].

[7] 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, https://doi.org/10.1109/WSC.2009.5429318

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

[9] 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: 25.07.2021].

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

[11] 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.

[12] 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

[13] 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

[14] D. Harel, "Statecharts: A visual formalism for complex systems" Science of Computer Programming 8 (1987), pp. 231-274, https://doi.org/10.1016/0167-6423(87)90035-9

[15] 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

[16] 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

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

[18] Longo, G. A., Mancin, S., Righetti, G., Zilio, C., "Flow dynamic and energetic assessment of a commercial micro-pump for a portable/wearable artificial kidney: Peristaltic vs. diaphragm pumps", Thermal Science and Engineering Progress, Vol 3, pp 31-36, 2017, ISSN 2451-9049, https://doi.org/10.1016/j.tsep.2017.03.006

[19] Kirk, R., Dipchand, A. I., "Continuous donor perfusion for heart preservation", Progress in Pediatric Cardiology, Vol 46, pp 15-18, 2017,ISSN 1058-9813,https://doi.org/10.1016/j.ppedcard.2017.07.007

[20] J. Kralev, B. Ivanov, I. Evg. Ivanov, A. Yonchev, D. Georgieva, An approach for perfusion pump control for nanofiltering, Proceedings of the Technical University of Sofia, Volume 67, Issue 2, 2017

[21] Ivanov B., A model of extracorporeal perfusion pump, 7th Mediterranean Conference on Embedded Computing MECO'2018, Budva, Montenegro, June, 2018, pp. 523-526, ISBN 978-1-5386-5682-2, IEEE Catalog Number: CFP18397-PRT

[22] Markov E., I. Evg. Ivanov, V. Gueorguiev, Program Generator Architecture, DESE 2011, Dubai, UAE, December 2011, https://doi.org/10.1109/DeSE.2011.104