Implementation of Objects Simulators and Validators using Program Generation Approach

Nikolay Baldzhiev[1], Velizar Bodurski[2], Vesselin Gueorguiev[3], Ivan. Evg. Ivanov[4]
Technical University of Sofia
Sofia, Bulgaria
baldzhiev@takt-ltd.com[1], vbodurski @takt-ltd.com[2],
veg@tu-sofia.bg[3] , iei@tu-sofia.bg[4]

***Abstract*— This paper describes an approach of implementing complex object simulator using program generator and flexible control hardware. A control algorithms validation and functionality simulation are the main target of this simulation.**

Keywords: component process simulation; industrial machines; program generation; real-time; control; behaviour validation.

I. INTRODUCTION

Simulation of various real systems has multiple advantages instead of experimenting and use of the actual system. Such an advantage is the possibility to train the personnel to operate various types of machines, or to use such simulations in preparations and optimizations of control algorithms. Around the world training or conducting initial tests of newly designed or improved control algorithms have always been a very challenging task. The machines are very expensive, and there always is a risk of accident during training or experimental work. Especially for cranes, where heavy lifts are taken, a single mistake can cause damages not only to the crane, but to the cargo and surrounding buildings. Also costs of operation of these machines are very high, including costs of fuel used, or in this case, high electrical consumption. The time needed for personnel training is also considered as a factor that can be optimized and improved on for companies. Another important aspect of simulation of working process is finding improved efficiency in ways to control the crane, using less energy and under all dimension limits.

There are a lot of very complex simulators for specific areas like flight simulators, power plant (and especially nuclear power plant) simulators. These systems are much specialised and built together with the real object, and in many cases – designed by the group designing the original object. Theoretical approaches are well explained in [5].

To produce simulator for an object that is available in nature but not supported by appropriate mathematical model and design documentation is real challenge. Existing difference between control systems' computers and simulation workstations makes the translation from high accuracy numerical simulation to semi-natural really hard.

Combination of program generator with SCADA (supervisory control and data acquisition) system for visualization and controlling can simulate complex machineries, with certain degree of accuracy.

In this article is shown a specific approach. It includes combined use of a specific program generating framework (PRGEN) to build data-acquisition real-time application, design of MATLAB-based model and implementation of final real-time simulator of portal cranes by the same PRGEN framework. The designed simulator is targeted for off-site personnel training and to be realistic environment for control programs and equipment development and testing.

Figure 1. Harbor Crane

The paper is organized as follows: in section 2 is presented theoretical part of program generator based on extended automata approach; section 3 covers design approach of crane simulator. Section 4 concludes the presented work.

II. PRGEN ARCHITECTURE

The program generator used for establishing simulation model is designed for creating distributed real-time control systems. Each node of the distributed system is described separately. A graph representation of the control algorithms is chosen. The system can be expressed by its activities. Each activity is separate thread. The activity thread consists of two different graphs:

- 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 [1].

978-0-7695-4593-6/11 $26.00 © 2011 IEEE
DOI 10.1109/DeSE.2011.105

- Signal Flow Graph (SFG) – a graph model representing signal transformation flow (the dataflow [2]). 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's 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 should start the execution of the STG when the system is started. For each state of the STG can be attached one or more SFG. For each state one or more transitions should be defined. A transition to the same state is acceptable. To make a decision which transition should be executed in each state should be built a Binary Decision Diagram (BDD) using values from predicates defined in SFGs of the node.

For each STG execution period is defined, defining how often the graph activates and executes its current state. The execution period can be modified if needed during runtime (e.g. when the system is in idle/power-down state, scanning inputs and refreshing outputs at a high rate is 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 state, until the execution period expires. When an asynchronous transition is activated the task executes directly the next state pointed by the transition, without waiting the period to expire.

B. *Signal Flow Graphs*

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 don't have activating inputs. After their execution remaining blocks are checked for activation. The blocks that have all their activating inputs set, start execution and so on 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). The activating inputs should be connected to an output of another FB in the same SFG. The non-activating inputs can be linked to an output of any FB from any SFG in the system node. In some specific cases link inputs can remain unlinked (they can use a preset default value)..
- Parameter inputs. The parameter inputs are always non-activating. They can also be linked to an output of another block (e.g. an adaptive controller, changing its parameter regarding the feedback) or they can be set by the developer to constant values during design.
- Internal state inputs. These inputs are also non-activating. They are typically used by FBs to store internal data (mainly to instruct the system how much memory should be spared for the internal needs of each FB). In very specific cases they can also be modified like the parameter .

Each input can be linked to only one source of data. An input which takes data from two different sources (e.g. from outputs of two separate preceding FBs) is not available.

Regarding the type of storage of data the inputs can be link inputs (data is stored outside the FB) or static inputs (data is stored inside the FB). Static inputs can also be linked to a special type of output – "point to point output".

Regarding the type of storage of data the inputs can be link inputs (data is stored outside the FB) or static inputs (data is stored inside the FB). Static inputs can also be linked to a special type of output – "point to point output".

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 can be linked to a static input and change its value.

Outputs can remain unlinked. Each static output (linked or unlinked) with a Boolean type can be used as a predicate of the BDD of the STG. Currently the BDD receives only a Boolean predicates. This is a drawback because if a continuous value needs to be checked, additional comparator FB should be added to the SFG that will produce a Boolean output.

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

Extended description of the model of the presented program generator can be found in [3][4].

III. DESIGN OF CRANE SIMULATOR

Possibilities to design and implement huge and complex object simulators are proven by different institutions, groups and researchers. Some well known examples are flight simulators [6] and production plant [7][8] simulators . Simulators of such a complexity and price are not in the scope of this paper. Nevertheless these examples motivated the authors to implement a harbour crane simulator. Before this simulator a number of simulators for small industrial and laboratory objects was designed and implemented.

Initial step in evaluating the model to be used for simulating the crane was detailed identification performed on site with a crane to evaluate performance and characteristics of the machine. Based on these two models were established:

1. discrete model – state machine describing various basic operation modes of the machine. The simulated machine has 3 different modes of operation and several sub-states of each mode.
2. continuous models – for each of the simulated continuous variables based on identification of the actual crane's performance in each discrete state.

The discrete model was implemented as a STG graph (shown on fig.2). Depending on the current state of the graph corresponding set of SFG graphs is activated simulating the behaviour of continuous operations (signal transformations).

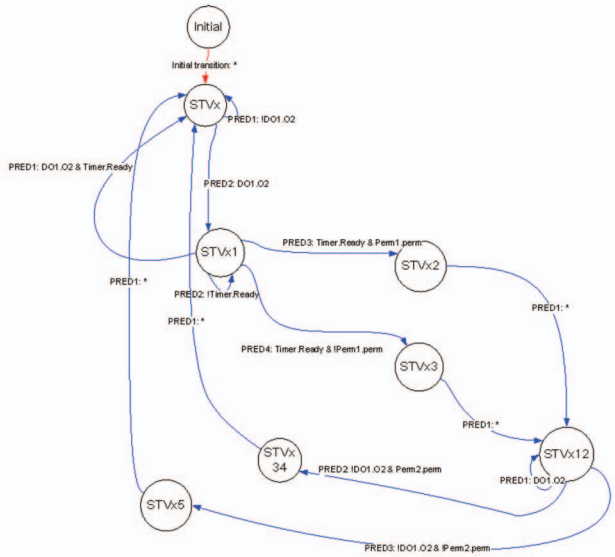

Figure 2. Part of discrete state graph emulating internal states of the simulated crane

The execution of the discrete graph starts from the Initial state, at simulation start, from there on we have non-conditional transition that the idle state of the machine. Depending on control signals obtained from the environment the graph changes states and adjusts its behaviour accordingly. At each cycle of execution of the graph associated SFG tasks with the current STG node are executed and as a result simulated continuous variables values are recalculated and generated.

One continuous graph for scanning inputs is displayed on fig. 3. In this case a number of complex FBs are listed for processing at each execution of the SFG. The operations performed by these complex FBs include: reading analog and discrete inputs of the simulator, thus providing source information of the predicates of the STG graph and used as feed-back information for simulation of analog signals.

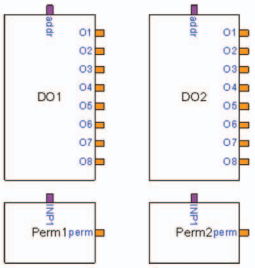

Figure 3. Inputs scanning SFG forming variables for STG predicates and feedback information for analog signals simulation

On figure 4 the SFG simulation of crane's boom movement is shown. This is macro-module model. It includes only three complex modules – message inputs, complex model of the crane and visual output module.

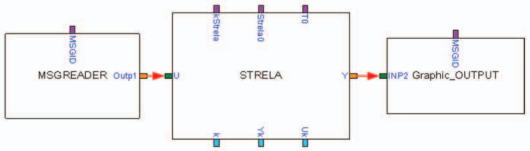

Figure 4. Analog simulation of crane's boom movement

On each execution of the boom simulating SFG the block scans all inputs information needed for the simulation, then current state is updated accordingly and new results are output for visualization and producing as simulation outputs.

External inputs to such crane include control signals from the controller which supervises system's operation. Combined with the full set of user interfaces (joystick, switches, signalizations) a full crane simulation is assembled. As the user operates its controls, information is passed to the controller by changing set points and modifying system's states. Based on this information the controller activates its control signals, which are forwarded to the simulator and as a result the simulator reacts to the control signals and returns responses similar to the real object. For visualizing internal states of the simulator and presenting logical information additional calculations are done and the results are displayed on a SCADA system to visualize crane's state. After initial model implementation additional functions that were introduced were various validation checks:

- validating current position to detect operator errors.
- validating speeds and accelerations of the crane to detect overloading of mechanics, due to incorrect/overzealous controller behaviour.
- validation of current system state and matches against control signals (under certain conditions some control signals should not be activated or should not exceed certain thresholds). – used for validating correctness of tested control programs.

The established simulation model allows adjusting of crane's behaviour. Depending on external conditions (winds, weather, load weight), the crane's behaviour varies significantly and thus both operator and control program should adjust their reactions to counteract possible complications. By providing such open model of the system development and release procedures for control programs can be significantly improved. By establishing a number of test benches simulated with this model the release and validation processes can provide significantly improved reliability analysis of the current version and make sure that no critical bugs are left unnoticed.

Hardware target platform for this simulation was Intel based industrial PC equipped with the proper analog, discrete and communication periphery. The decision to use this platform was based on the following:

- the co-processor of the MATLAB systems and the simulator were equivalent and any kind of hardware generated calculation errors were suppressed.
- real-time operating system had predictable time behaviour and did not inserted time-related errors.

From the planned test of simulation, behaviour of crane simulator was very close to actual real model. To compare our results, we've perform series of tests in real harbour crane at port of Varna at Black Sea (Bulgaria). The measurement from crane controllers was recorded in database, all inputs and

outputs was stored and used to compare with simulation results. The test with crane was performed on clear day, with no wind, or rain. Wind can dramatically change behaviour of crane. Load was empty container, used for calibration.

Four standard tests were performed.

1. Crane with no container, turning to container and lowering hook. Total 6 different movements to achieve desired position.
2. Crane with container, lifting the container, and turning to the ship. Total of 4 movements to achieve desired position.
3. Lowering the load, and positioning the container into the ship. Total 8 movements to achieve desired position.
4. Empty crane, from initial position (turned to the port). Turning to the ship, positioning above container, lifting load and unloading it into port. Total 32 movements to perform full unload of container from ship to port.

Table I contains generalized results of performed tests with the crane and the simulator. Number of measurement is number of sensor data recorded during real test and simulation.

TABLE I. COMPARISON BETWEEN REAL CRANE AND SIMULATION.

N.	MEASURE-MENTS CRANE	MEASURE-MENTS SIMULATOR	MIN. MOVEMENT ACC. %	MAX. MOVEMENT ACC. %	TOTAL ACC. %
TEST 1	680,000	890,000	95	98	**97**
TEST 2	1, 280,000	1,460.000	94	97	**95**
TEST 3	1,800,000	1,960,000	95	98	**97**
TEST 4	3,640,000	4,000,000	93	97	**96**

There was special software used to read data from controllers. Standard SCADA software was unable to record data from all inputs and outputs in 50ms time interval what was system's sample time. Maximum recording rate was 1 second, which was not enough. Specialized software on each 35 milliseconds interval scanned and read all data fields in controllers (two ABB controllers), and sent it to output buffer, where another thread was compressing the information and sending to database. Each second 50 MB of data was generated. We used SQL server standard operation to process these big amounts of data.

Each movement of crane was processed separately. Different movements have different accuracy, so in the table is listed minimum and maximum movement accuracy and comparison of real data and simulation data.

IV. CONCLUSION

Using program generator, user can simulate complex machines, using building blocks, graphs and other methods provided by PRGEN. There is no need of specialized simulators to be developed from scratch which is expense, and time consumption. An engineer working with PRGEN can create model of such a machine, and model of control algorithm, using library of pre-developed blocks (PID regulators, Analog and Digital IO periphery and many more). Also when there is a specific need, a block can be created as "C" program, and added to PRGEN's library. PRGEN fully supports OPC protocol for communication, so it can be used with SCADA software for visualization and control. Up to now, we've successfully used PRGEN in many different areas from building automation, to heavy industrial machinery.

REFERENCES

[1] David Harel, "Statescahrts: A visual formalism for complex systems" Science of Computer Programming 8 (1987) 231-274

[2] Krishna M. Kavi, Bill P. Buckles, "A Formal Definition of Data Flow Graph Models" IEEE Transactions on computers. vol. C-35, No. 11, November, 1986

[3] 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

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

[5] Eric Winsberg , "Simulations, Models and Theories: Complex Physical Systems and their Representations". 2001, *Philosophy of Science* **68**: 442–454.

[6] "Civil Full Flight Simulator Census". CAT (magazine). 2009, http://halldale.com/magazines/cat/issues/cat-42010.

[7] O. Benedettini, B. Tjahjono, "Towards an improved tool to facilitate simulation modeling of complex manufacturing systems". 2008, International Journal of Advanced Manufacturing Technology 43 (1/2): 191–9

[8] Gabriel A. Wainer , Discrete-Event Modeling and Simulation: a practitioner's approach. CRC Press, 2009.