HARDWARE IN THE LOOP: A DEVELOPMENT STRATEGY FOR SOFTWARE RADIO

Ehsan Azarnasab (University of Utah, Salt Lake City, UT, USA[+]; azarnasa@ece.utah.edu),
Peiman Amini ([+]; pamini@ece.utah.edu),
Behrouz Farhang-Boroujeny ([+]; farhang@ece.utah.edu)

ABSTRACT

A framework combining DEVSJava simulation engine and MATLAB is developed to implement a network of cognitive radios while using the real hardware among simulated modems. The process starts with one node and uses simulation in different stages of the development as a test and measurement tool. Based on the progressive simulation based design and development, we start from simulation and substitute the models with real modules step by step to build a full functional modem. Afterwards, to develop the network of the cognitive radio modems, we deploy the available real modems with simulated primary and secondary users in a central simulation. To simulate the effect of the primary users on real modems, an arbitrary signal generator is used to emulate the transmitted signal of licensed users on the channel. Using this method, we are able to develop our cognitive network while we do not have enough hardware. One real node is integrated in our simulation which performers channel sensing and data transmission. When this node detects the presence of a primary user on its current working carrier, it moves to an unoccupied carrier.

1. INTRODUCTION

Systematic design and implementation of modern systems goes trough several steps. Engineers often start with initial formulations and simulation of their design; then build several prototypes before the actual system is finalized. While modeling and simulation has been widely deployed in design of communication systems, the transition from simulation to development is considered to be a big gap. Facing discrepancies between simulation model and real system is a common problem. In addition, in applications such as Software Define Radio (SDR) where the complicated signal processing algorithms are involved, having powerful visualization tools as well as debugging techniques can greatly reduce the development time and related costs; this could explain why more engineers are using tools such as Simulink and Realtime Workshop for fast prototyping [1].

Furthermore, most of the software and hardware components of our system can be implemented independently. As a result, we have used simulation not only as a test and measurement technique in the development but also as an environment that real (implemented) and simulated (yet to be implemented) components work together. In the course of implementation, we gradually move from simulation to a system in which only real components exist. The process of bringing hardware into the simulation while satisfying real-time constraints is often called "Hardware in the Loop" (HIL) simulation [2]. In essence, the intermediate stage of HIL simulation makes it possible to add conventional simulation and real system experiment to form an incremental simulation-based design process. HIL simulation has also many other advantages such as reducing the complexity of the system and at the same time expediting the implementation phase by parallelizing the development of different components even if some modules are not ready. During this text, we use the terms co-simulation and HIL simulation interchangeably.

HIL simulation has been used in variety of applications including embedded systems [3], avionics [4] and robotics [5], [6]. However it is an engineering decision to which extent simulation should be applied and interweaved with the real components. One may apply the HIL simulation of a detailed version of transceiver consisting all modules. This resolution in modeling results in a higher level of confidence due to more accurate measurements of a real system inside the simulation. On the other hand, implementing a whole network comprising tens or hundreds of nodes in this way is not practical. When using HIL for network simulation, it is easier for the host - the machine that runs the simulator- to simulate a complete transceiver instead of individual internal modules inside the transceiver. Therefore, there is always a trade-off between the amount of simulation and hardware implementation in system design using HIL technology, when we are testing or developing an algorithm on a network of wireless nodes.

Simulink and Realtime Workshop are used in conjunction with different hardware platforms to develop embedded systems such as SDR. Simulink includes variety of blocksets that simplify the design process of SDR to a large extent. A developer in Simulink can also take advantage of the powerful MATLAB libraries for signal processing and communication algorithms. Realtime Workshop is designed for rapid prototyping and testing new algorithms. However, if we need to develop an optimized code for a particular DSP or GPP, considering limited resources, Realtime Workshop does not perform as well, unless many of the available general modules would be rewritten for the target machine, which is time consuming. To elaborate, since Realtime Workshop should conform to the state-flow of Simulink, it is able to generate code for various applications on different hardware platforms, but the generated code often needs extra memory and processing power. The optimization burden is thus put on the compiler only. In the realtime embedded system design era, the functionality of a module is defined not only by the outcome but also the memory and realtime requirements. Consecutively, it might not be a good option

to use Realtime Workshop for developing a complete radio with high processing and memory demand, while it boosts the productivity of developing individual algorithms.

Moreover, while Simulink and MATLAB are powerful tools for designing communications and signal processing systems, they are not suitable for network simulation and implementation. For network simulation, other tools such as OPNET [7] are used. For cognitive radio network HIL simulation, we use open source DEVSJava [8] platform together with MATLAB programming and Lyrtech libraries to design a SDR Development test-bed which can be used for development of a single wireless node as well as a network of cognitive radio modems.

In the first part of the paper, we run HIL simulation to implement a single transceiver. A complete transceiver is first simulated in MATLAB, then individual and combined signal processing and communications algorithms are substituted by those running on the board. To include the functionalities of Simulink in our system, we also combine MATLAB with DEVSJava, and test some parts of the radio in DEVSJava.

After the initial modem is implemented (as one secondary user), a higher level HIL network simulation with primary (PU) and more secondary users (SU) is performed to co-simulate a network of cognitive SUs. At first, a complete simulation of a cognitive radio system with SUs and PUs is developed in DEVSJava. The central simulation engine keeps track of the activities of channel assignments for SUs and PUs. Secondary Base (SB) station receives the sensing information from all SUs and compiles channel state information. In our system, we have distributed sensing in order to avoid hidden node problem as much as possible. A vector signal generator emulates the traffic of PUs and simulated SUs on the real channel. During the course of HIL simulation, we have replaced one, and are going to replace more, of the simulated SUs with the real cognitive radios in order to test the whole system. In our setup, the real nodes use filterbanks to detect the presence of the PUs [9], [10] and send their sensing information to the SB which is simulated in the host. In this way, we are able to develop our network while some hardware parts are missing.

2. Discrete Event System Specification (DEVS) Modeling

Simulation can be used as the feedback of the testing phase during system development. Building large and complex systems often requires defining intermediate goals, with conditions that should be satisfied before moving to the successive stages.

Modeling and Simulation starts with modeling the real system and builds simulators upon them. The event based nature of the radio network makes Discrete Event System Specification (DEVS) a suitable environment with enough power and more efficiency than time based modeling. Availability of the DEVSJava [8] package (written in Java as a portable language with many features including easy concurrency) made DEVSJava the prime choice for starting point. DEVSJava is used in many simulation problems before, including the simulation of Robot Convoy with behaviors [11]. DEVS is a theoretical modeling which can describe modular systems in hierarchial manner. Based on the formal definition of DEVS, it is able to model both time based and event based systems. In addition, other popular modeling techniques such as *Petri nets*, *Finite State Machines* and *Timed automata* have DEVS equivalents. The structure of a model may be expressed in a mathematical language called *formalism*. This formalism defines the way variables take values and the time these values should take effect [12].

The basic elements in DEVS formalism are *atomic* and *coupled* models [12]. The atomic model (Fig. 1) has internal states (S) which change by internal (δ_{int}) and external (δ_{ext}) transition functions. These transitions may produce output (λ). To describe the atomic model, suppose the system is in the state $s \in S$. If no external event happens, the system will remain in the same state for $\text{ta}(s)$ (resting time). The elapsed time is e. When the resting time expires ($e = \text{ta}(s)$), the system outputs $\lambda(s)$ and changes to the new state $\delta_{\text{int}}(s)$ (this transition is called an internal event). In the following figure, X and Y are incoming and generated external events respectively.

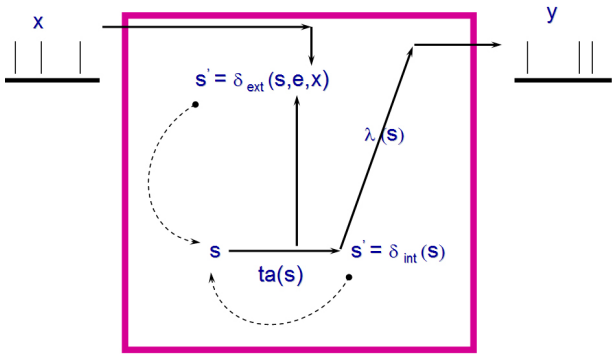

Fig. 1. DEVS atomic model in action

Coupled models consist of more than one atomic model. Generated events by one model can be fed to other models in a coupled model and act as external events for them. If an external event occurs before resting time elapses (system is in total state (s, e) with $e \leq \text{ta}(s)$), then the system changes to another new state $\delta_{\text{ext}}(s, e, x)$. The confluent transition function (δ_{conf}) determines the new state if both internal and external transition functions happen at the same time.

In practice, the internal events are used for scheduling upcoming events while external events are messages received at particular input ports of the model. Therefore one should implement δ_{int} to specify the timely behavior of the system

and δ_{ext} to build the interface between ports of the models by passing messages between models as in a network (with models being the nodes). The coupled model keeps track of all the components, components' influences, the set of input ports receiving external events and output ports sending those events. This model itself has input and output ports connecting to one or more of the components. The formalism is closed under coupling, which means a coupled model can serve as another DEVS model to build a hierarchical coupled modeling. The semantics of DEVS decouples inside of the model from outside by the concepts of internal and external events, when both can be present.

Another characteristic of practical real-time systems is the ability to work with concurrent real objects, having the outputs ready by a deadline and the ability of decision making based on computational processing units. Having that in mind, the real-time simulator objects in DEVSJava are implemented as concurrent *threads.*

In DEVS real-time simulation each atomic model is assigned to a real-time simulator that, based on the current time, decides handling of internal and external events. Therefore unlike conventional object oriented design, a DEVS object provides a mechanism for introducing time in the objects.

Models discussed in DEVS are implemented in some different programming languages such as DEVSC++ [13] (in C++) and DEVSJava [8] (in Java). In addition, based on the DEVS concepts, the designer just models the system while the simulator is constructed upon each model automatically. Therefore, one should design the basic models, from which larger models are built, and then connect them together (add couplings) in a hierarchical order. These models however, can change their structure and couplings dynamically during the simulation if needed (variable structure DEVS [14]).

3. Implementation of a Transceiver

We have used two methods for implementation of a cognitive radio node. First, Lyrtech provided libraries are used to interface MATLAB with the board. Second, DEVSJava is combined with MATLAB to incorporate hardware in a co-simulation engine and couple it with a host model. The second method is the starting point for developing the required interface for HIL cognitive network simulation.

A. HIL simulation in MATLAB

Based on the regression testing in the software engineering, we have divided the SDR problem into smaller units which are tested individually and also in groups. A complete simulation of the system has been first been developed in MATLAB programming environment. The simulation is broken up into functions which are then substituted by the embedded functions developed on an Small Form Factor (SFF) SDR hardware platform which is provided by Lyrtech and Texas Instrument as part of the support that we received for Smart Radio Challenge. Starting from the freeze mode (non real-time), the modules and algorithms are tested and optimized on the hardware platform one-by-one until the specified real-time requirements are achieved. In this way, it is possible to generate input for different individual modules while the preceding parts are not available yet.

In the next step, we test two or more modules on the board while the remaining parts are still being simulated. Hence, the dependency on the simulation is decreased gradually. The product of this development phase is the first version of a complete functional node implemented on the hardware with individual parts working properly. Using this technique, a cognitive radio transceiver is implemented on SFF SDR development platform [15] [16].

The interface between the platform and MATLAB is developed using MATLAB executable (MEX) files and SMSHELL API [17] provided by Lyrtech for the SFF SDR platforms. At each stage one part of our system is migrated from MATLAB script to the board. For each block the input is generated using MATLAB and passed to the board using the developed MEX function which writes into a predefined memory addresses in the DSP using SMSHELL and the network socket. The results of the algorithm and the values of variables at the intermediate stages of algorithm is monitored in MATLAB. For each algorithm, a C code is developed in Code Composer Studio and the generated output file is transferred to the board. A protocol is implemented to synchronize the transaction between the host (running MATLAB) and DSP.

Herein, we present the MATLAB script for test and implementation of convolutional coding and Viterbi decoding algorithms on the DSP core. We have tested the integrity of the results as well as the required processing time on the hardware. A simple code fragment for testing the convolutional coding in the MATLAB environment follows:

```
% set the monitoring addresses
setAddresses();

% select the target
h = targetSelect();
% load the bit file
targetDSPload(h,'c:\boz\TwoWay.out');

%----------- Convolutional Coding ----------------
%first control byte is the length of message
targetWrite(h,WRITE_MSG_LEN,int32(msgLen));
%pass the data
targetWrite(h,ADDR_WRITE_DATA,input32);
%Flag data ready
targetWrite(h,WRITE_HANDSHAKING,START_CNV_CODING);
tic;
while ((toc < 20) && _
    (targetRead(h,WRITE_HANDSHAKING,1) ≠ STOP_FN))
    %Wait for operation, at most 20 seconds!
end
%convolutional code length
K = targetRead(h,READ_CONV_LEN,1);
%number of double-words to read
n_dw = double(ceil((2 * (msgLen * 8 + K - 1) + ...
                    2)/3/4));
```

```
%read conv-coded interleaved result in double-words
cnv_coded_target_dw = ...
                targetRead(h,ADDR_READ_DATA,n_dw);
%convert to byte stream
cnv_coded_target = typecast(...
            cnv_coded_target_dw,'uint8');
%convolutional coding result
cnv_coded_target = ...
    double(cnv_coded_target(1: ...
        length(cnv_intr_coded_mex)));
%find if they differ
err = sum(cnv_coded_target-cnv_intr_coded_mex);
if (err ≠ 0)
    disp('Error in convolutional coding!');
end
%time taken in milli seconds
dly = targetRead(h,READ_DELAY,1);
disp(['Convolutional coding took :' ...
             num2str(dly) ' ms']);
```

In this script, first a connection with the board is established and the socket handle is used in the rest of the code for data transferring and synchronization. Note that the algorithms are previously implemented using Code Composer Studio and the output file is ready. Using the above technique we optimized our algorithms for 1kb input data packets and reached less than 1*ms* of time for convolutional coding and 20*ms* for Viterbi decoding which met our realtime requirement for our modem. In fact, for 20kbs (as proposed in the problem definition) this delay corresponds to 40% of the whole time available for input data processing before the successive input arrives.

B. HIL using DEVSJava and MATLAB

In the next step, we combine the functionality of DEVSJava and MATLAB for HIL simulation. This environment incorporates both event-driven environment of DEVSJava and powerful MATLAB toolboxes to be able to develop and test our wireless node. To elaborate, while in MATLAB programming we can generate arbitrary waveforms and pass it to the board, we do not have the capabilities of DEVSJava for event based simulation. The MATLAB functionalities in communications and signal processing are brought inside Java environment using MATLAB Builder for Java. MATLAB Builder for Java, can build Java classes from the MATLAB functions and have them ready to use in DEVSJava. The MATLAB m-files that were already developed for MATLAB simulation of the channel, the MATLAB visualization methods, and also the hardware-interfacing MEX files are compiled to Java classes that will be called later inside our DEVSJava simulation.

In Figure 2 we have presented a simple example of this environment which can be seen in the process of developing a node. We have developed a DEVS model to measure the performance of convolutional coding and Viterbi decoding on different channel models which are simulated in MATLAB. The blocks are atomic DEVS models and the couplings show the flow of data from output ports (at the right side of each block) to the input ports of other blocks (at the left side of each block). When an external event happens at a model, the signal (conveyed in the form of a message) generated at that block follows the coupling path associated with the output port of the event to the input port of the destination block. As it is shown in the figure 2, the data generator atomic model (*Data Gen*) sends random frames of data periodically (using external events or messages) to the DEVS model in charge of co-simulation (*DSP-cosim*) which then acts as a proxy to the DSP board. The co-simulation engine is responsible for handshaking and synchronization required when communicating with the board. When data is processed inside the DSP, an external event is generated for the DEVS model (*Fading channel*) in which the coded binary result is passed to our developed system in MATLAB. In the MATLAB routine, other parts of a transmitter such as baseband modulation, up-sampling, pulse shaping, and modulation are performed and then the signal is sent to a communications channel model and finally at the receiver down-sampling, carrier recovery, timing recovery, equalization, and demodulation are performed. In another cycle, the previous result data is again passed to the co-simulation engine for Viterbi decoding. The final results are then presented using MATLAB plotting commands in the DEVS model *TimeScope*.

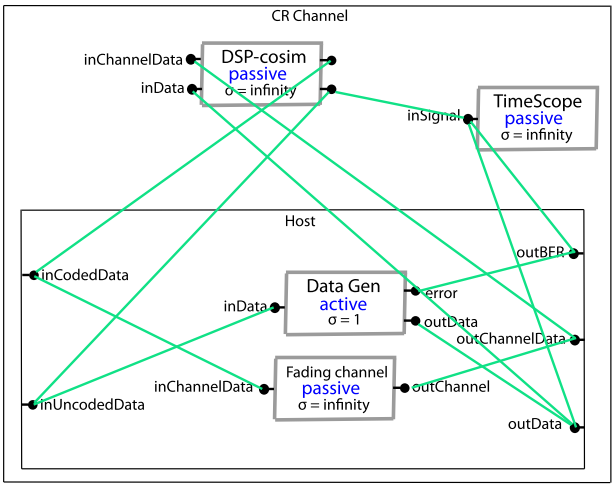

Fig. 2. HIL simulation with DEVS models

For more details, one movie to show the entire process is available at [18]. As can be seen in this movie, in each simulation run the data path begins at the data generator and passes the hardware (embedded in the DSP co-simulation engine) twice. The decoded data at the end of simulation is the same as the original one, which shows the robustness of our coding against noise.

4. Cognitive Radio Network

We have developed a simulation of a cognitive radio network using DEVS. Considering the discrete nature of data networks, DEVS perfectly matches the requirements

of our simulation scenario. Moreover, to test the cognitive radio system in the absence of a few radio modems, we have used the Hardware in The Loop (HIL) approach by plugging the hardware of the radio modem in a co-simulation engine as a secondary user (SU). The network design starts from an all-virtual simulation with only the models of the cognitive nodes in the simulation and adds real nodes to the system gradually. The intermediate system consists of real and simulated cognitive nodes talking to each other. This systematic approach for implementing collaborative systems has shown to be successful in revealing the underlying problems and devising high fidelity models [5].

Figure 3 shows a network model of primary users and cognitive secondary users with a base station. One of the secondary users (SU2), two primary users (PU1 and PU2) and also the secondary base (SB) station are simulated in the host. Only one of the secondary users (SU1) has hardware implementation thus is emulated using the co-simulation engine. Using the message passing mechanism of DEVS between the models, in which messages are external events, and also exploiting the time triggered message generation inside the modeled nodes, which are internal events, the simulation of the network was implemented. We used immediate messages for passing parameters between the models while time scheduled messages were used to pass the data-carrying binary signals (longer packet is scheduled for later time). For the simulated nodes, after a transmitter sends a packet of data, the DEVS model *Channel* passes the binary signal, along with the carrier frequency and other required parameters to a MATLAB code which simulates a complete transmitter, channel, and receiver. The MATLAB code for the transmitter includes source coding, base-band modulation, up-sampling and RF modulation, channel, down-sampling, etc. At the receiver, also the necessary functions are developed in MATLAB and the data which might have error is passed to the node in DEVS. The real cognitive nodes rely on their embedded software for data transmission among themselves and the co-simulation engine handles their interface to the simulation.

We use filterbank for sensing in the implemented nodes. The sensing information is passed to the Secondary Base (SB) in the host. The SB is responsible for channel assignment and also coordinations for sensing. All of the SUs, including the SB, sense the channel periodically. Afterwards, the SB compiles all of the sensing information and assigns channels to the users. To simulate the effect of the primary users, an Agilent ESG signal generator emulates a pseudo traffic on the bands in which the simulated primary users are occupying the bandwidth. The transmitted signal by the function generator is a multi-band waveform which is generated using a MATLAB script and uploaded to the device With Agilent's Waveform Download Assistant via network. SUs move to a new channel when a PU is occupying the bandwidth.

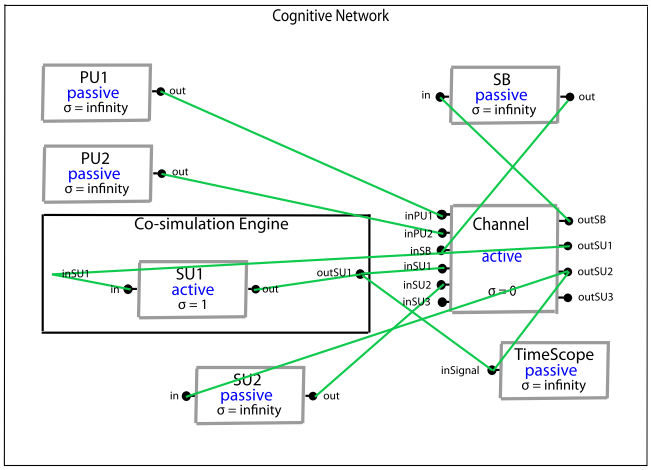

Fig. 3. A cognitive network with 2 Primary Users (PU) and 2 Secondary Users (SU) and one Secondary Base station (SB), SU1 is implemented inside an SFF SDR board and emulated along with the other nodes

In our simulation we have used a bernoulli arrival process for secondary users. The activity of a node is determined by the probability that a new packet is generated inside that node at each time slot. The primary users change the frequency bands that they are using randomly. Different primary networks need to be modeled differently deponing on their traffic type. For example, in a cellular network it is reasonable to assume that the channel is not freed during a conversation which takes at least a few seconds and lasts up to several minutes. However, the traffic model would be different for other networks such as wireless LAN or TV stations.

Progressive design starts with a traditional all-simulated network using virtual-time instead of the real-time to test the behavior of the design in the long run. The traditional simulation runs much faster than realtime and the deficiencies of the design strategies are revealed easier before going into the implementation phase. After the initial simulation (with only the simulated nodes) is finished, the HIL simulation with reduced realtime speed is run. The reduced speed of realtime simulation is often necessary to compensate for the overhead of co-simulation engine on the actual hardware. For the first HIL case study, when we have only one SFF SDR board the simulation scenario is as follows. In the first simulation there is only one PU and three SUs (including one SB simulated in the host). The PU is simulated using the signal generator to emulate the effect the simulated users on the wireless channel. Sensing data is collected by the real nodes and passed to the SB simulated on the host. Following this step, we increase the number of simulated nodes and then the real nodes.

5. STATUS OF IMPLEMENTATION, CONCLUSION AND FUTURE RESEARCH

A single cognitive radio node was developed using HIL simulation. Different signal processing and communication algorithms were developed and tested on the board and the performance of each routine and the required time was studied. The integration of DEVS with MATLAB gives designers the opportunity of decoupling the model and underlying (often very complicated) math. While it is possible to reuse the code available in MATLAB (including the MATLAB simulation codes), the event based nature of DEVSJava (adding the concept of time to the objects) adds to the power of normal object oriented Java language. In addition, the message passing structure of DEVS models fits naturally into the category of networking. As a result the developer can benefit from a well-defined simulation engine which is an alternate to some other very expensive solutions. The initial HIL network simulation is now ready and we can simulate the dynamic channel assignment in the presence of a simulated primary user. Currently, we are using a single functional SDR platform which acts as one secondary user that can send and transmit data. In the current setup, the real node can sense the channel and move to a new band when a PU transmit on the band that they are using. We are currently working on the cognitive radio network HIL simulation and as soon as more hardware is available, more of the models will be substituted with the actual nodes. In addition, so far a memory-less scheme is implemented in the secondary base during the channel access coordination. A more complicated AI method for resource management can significantly reduce the primary-secondary collisions and thus increase the link throughput. A flexible framework was developed to set up a cognitive radio network, and we are waiting for more hardware to progress towards the all-real cognitive radio network.

ACKNOWLEDGMENT

This project was done as a part of the SDR Smart Radio Challenge. We would like to thank the hardware and software support that we have received from SDR Forum, Lyrtech, Texas Instruments, Xilinx, Mathworks, Greenhills, Prismtech, Zeligsoft, and Syncplicity. It has been a challenge to learn this extraordinary collection of hardware and software, but, indeed a rewarding one. We are truly grateful to all supporting companies.

6. REFERENCES

[1] D. Burns and T. G. Sugar, "Rapid Embedded Programming in the Mathworks Environment," *Journal of Computing and Information Science in Engineering*, vol. 2, no. 3, pp. 237–241, September 2002.

[2] L. Li, T. Pearce, and G. Wainer, "Interfacing Real-Time DEVS models with a DSP platform," *In Proceedings of the Industrial Simulation Symposium*, 2003.

[3] M. Chiodo, P. Giusto, H. Hsieh, A. Jurecska, L. Lavagno, and A. Sangiovanni-Vincentelli, "Hardware/software co-design of embedded systems," *IEEE Micro*, vol. 14, no. 4, pp. 26–36, August 1994.

[4] J. Upton, *Boeing 777 (AirlinerTech Series)*, 2nd ed. Voyageur Press, August 1998, vol. 2.

[5] E. Azarnasab and X. Hu, "An integrated multi-robot test bed to support incremental simulation-based design," *In Proceedings of the IEEE International Conference on System of Systems Engineering*, 2007.

[6] Z. Papp, K. Labibes, A. Thean, and M. van Elk, "Multi-agent based HIL simulator with high fidelity virtual sensors," *In Proceedings of the IEEE Intelligent Vehicles Symposium*, pp. 213–218, June 2003.

[7] *http://www.opnet.com/*.

[8] B. P. Zeigler and H. S. Sarjoughian, *Introduction to DEVS Modeling and Simulation with JAVA: Developing Component-Based Simulation Models*, jan 2005.

[9] P. Amini, R. Kempter, R. R. Chen, L. Lin, and B. Farhang-Boroujeny, "Filter bank multitone: A physical layer candidate for cognitive radios," *2005 Software Defined Radio Technical Conference*, November 2005.

[10] E. Azarnasab, R. Kempter, N. Patwari, and B. Farhang-Boroujeny, "Filterbank multicarrier and multicarrier cdma for cognitive radio systems," *IEEE CrownCom*, August 2007.

[11] X. Hu and D. H. Edwards, "Context-Dependent Control of Adaprive Behavior Selection," *Proc. Workshop on Bio-inspired Cooperative and Adaptive Behaviors in Robots, in co-operation with The Ninth International Conference on the Simulation of Adaptive Behavior (SAB06)*, 2006.

[12] B. Zeigler, H. Praehofer, and T. Kim, *Theory of modeling and simulation*, 2nd ed. Academic Press, 2000.

[13] B. P. Zeigler, Y. Moon, D. Kim, and J. G. Kim, "DEVS-C++ : A High Performance Modeling and Simulation Environment," *HICSS (1)*, vol. 7, pp. 350–359, 1996.

[14] X. Hu, B. P. Zeigler, and S. Mittal, "Variable Structure in DEVS Component-Based Modeling and Simulation," *SIMULATION: Transactions of The Society for Modeling and Simulation International*, vol. 81, no. 2, pp. 91–102, 2005.

[15] "Lyrtech SFF SDR development platform technical specs," Lyrtech Inc., Available from `http://www.lyrtech.com/publications/sff_sdr_dev_platform_en.pdf`, Tech. Rep., February 2007.

[16] P. Amini, E. Azarnasab, S. Akoum, X. Mao, H. I. Rao, and B. Farhang-Boroujeny, "Implementation of a cognitive radio modem," *2007 Software Defined Radio Technical Conference*, November 2007.

[17] "Lyrtech SMSHELL technical reference guide," Lyrtech Inc., Tech. Rep., February 2007.

[18] *http://www.ece.utah.edu/commlab/sdrMovies.htm*.