Study of Simulink MOC in Metropolis
Haiyang Zheng

Background
Simulink is the most widely used specification capture tool for embedded control. It allows designers to model discrete and continuous time controllers and has a stateflow editing facility.

However, because the modeling environment of Simulink is a mixed and heterogeneous one as the combinations of discrete semantics, continuous semantics, etc. Usually it is not very easy to understand a Simulink model. A better way to represent a Simulink model may be necessary. One possible method is to separate the model into sub-models which have relatively simpler, well-defined semantics. With the easier understanding the sub-models, the whole model becomes more intuitive and easier to understand.

Metropolis meta-model is a communication based platform. It has a general semantics which can be customized into more concrete semantics. Thus, we try to use the meta-model to study the semantics of Simulink.

Goal
1. Define the semantics of a reasonable stable subset of Simulink in Metropolis, the timer-triggered semantics (TT) using the meta-model facilities (await, communication media, constraints, etc.)
2. Create a tool which transforms the Simulink models into generate appropriate MMM netlists using the above defined semantics.

Approach
A non-trivial simulink model is usually a heterogeneous model consists of both discrete and continuous semantics. More precisely, it has a discrete (periodical time triggered) model of computation as the top level, and a continuous model of computation embedded inside. A pure continuous model could be regarded as a model with the discrete period as infinity.

Due to the time limitation, the goal of this project is to realize a discrete (timer triggered) semantics as a refinement of the general communication semantics defined in Meta model. The project has two parts: one is the implementation of the timer-triggered semantics (TT) in Metropolis, the other one is to write a tool to automatically import and transform simple Simulink models into TT models in Metropolis and do simulation and analysis.

The key part of the TT semantics is to implement a scheduler which mediates the execution order of the different processes and ensures the completion of the execution. The key part of the tool which bridges the Simulink models and Metropolis models is the preservation the data dependency.
Progress made and future work
A package for the timer-triggered semantics is implemented.

The package implements discrete processes (dtprocess), discrete channels (dtchannel),
and discrete scheduler (dtScheduler), which uses statemedium dtStateMedium. A special
interface, dtStateMediumInterface has only one method, getPeriod(). The dtStateMedium
implements this interface such that the scheduler could access the periods of processes.
The dtScheduler assumes the order of the processes defined in the MMM netlist preserve
the data dependency, which is guaranteed by a automatic transformation tool.

The scheduler constructs an array of the invocation rates associated with each process by
normalizing the periods of the processes with their biggest common divider. (For
simplicity, the assumption is that there is one process having a smallest period while
other processes's periods are multiples of it, and the BCD is the smallest period.) With an
integer counter variable indicating the time increment, the dtScheduler iterates the
invocation rate array to schedule appropriate process to run. The scheduler ensures the
process finishes execution by checking a predefined block label. The package limits the
data type as integer only.

A simple netlist example which contains three blocks: ramp, gain, and subtract, is
implemented. With the help from Guang Yang about the SystemC backend tool, C code
has been generated for the simple example and compiled successfully. However, the
simulation fails with some segment fault error and I am currently trying to figure out the
bugs.

Once the simulation of the simple example gives the expected results, the next step of
automatic transformation of Simulink models into MMM netlists will be launched. I have
already found a tool from Vanderbilt University which can transform the text format of
the Simulink models into XML formats. And the XSLT tool can be used to translate the
XML files into MMM netlists.

Models of Computation for the Robot Diffusion Problem
Sarah Bergbreiter

Background
Distributed control systems provide an interesting domain for embedded system design
due to the reactive nature and complexity in timing inherent in these systems. Distributed
control can be defined as a control algorithm characterized by multiple sensors, actuators,
computation elements, or combinations thereof distributed through space. For example
in a building HVAC system, temperature sensors and actuators controlling the heating
system are distributed physically. It is natural to think that these distributed control
algorithms could be implemented on small, wirelessly networked nodes similar to those
built by the TinyOS group at UC Berkeley [1].
This project will examine one particular application of distributed control, robot diffusion. In the robot diffusion problem, $N$ robots reside in a given area $A$. The goal for these robots is to diffuse outwards to maximize their distance between each other. Robot diffusion had distributed sensing (between robots), actuation and computation making it an excellent candidate on which to study distributed control. One application for robot diffusion would be to deploy a mobile sensor network in order to cover the largest possible area with sensors [2]. A search and rescue operation would be a perfect venue for such a task. Yet another application more specific to robots is “diffusion mapping”, where line-of-sight connectivity determined while the robots diffuse outward is used to determine a map of an unknown area [3].

**Project Goal**

Naturally, as with most control problems, some of the key parameters that we are interested in proving are the stability and convergence of the algorithm. Many algorithms in the field of distributed control put communication synchronicity assumptions in the control algorithm to guarantee stability in convergence. However, due to the wireless network connecting the nodes of our system, it is likely that communication will not be synchronous at all. For example, in [4], stability of vehicle positions in a one-dimensional problem is directly tied to the communication sampling period. If this period is too large, the system will become unstable.

The goal of this project is to formalize and abstract the robot diffusion problem and to examine an algorithmic solution under various conditions of synchronicity. For example if the system is considered entirely synchronous (where all robots share the same notion of time), it is likely that several key properties such as stability will be provable. However, given increasing asynchronous behavior, I would like to determine where and how these properties fail. This will be accomplished through simulation and on real robots.

**Approach**

The first step in this project will be modeling the robot diffusion problem and a controller in the context of a reactive embedded system. Reference [4] describes the one-dimensional problem of robot diffusion as a discrete-time system. However, the goal of this project is to pay much closer attention to the time properties of the system. Therefore, an interesting test arises when modeling the system as a network of FSMs that communicate synchronously versus a network of FSMs that communicate asynchronously (the GALS – globally asynchronous, locally synchronous – philosophy).

After the controller has been modeled as a network of FSMs (where an FSM resides on each robot), I will write Esterel modules to describe each FSM. Simulating this network of modules in parallel will give a good idea of the performance for a network of robots who share the same notion of time. By modifying the simulation slightly to include various degrees of clock skew in the communication between the modules, I can model varying degrees of asynchronicity. Various stability tests will be run on the system to test performance under various timing constraints (for communication between robots).
Finally, because Esterel compiles into C, I can synthesize code from the Esterel model to run on real robots. The robots use the networked sensor nodes built by the TinyOS group at UC Berkeley [1] along with a toy RC car base to move and communicate. Due to the complexity of the localization problem however, I will only demonstrate a 1-D diffusion model in real hardware.

**Work**
The first work needed was to formalize the problem specification and to develop the controller for robot diffusion. To date, work has been concentrated on this point and for the one-dimensional case based on work done in [4] and a Simulink model has been built and its effectiveness demonstrated. In this example, the only information shared is the position of the neighboring robots.

However, an even simpler hardware implementation (and a slightly less restrictive requirement) would require that the controller know only the distance between the robots instead of absolute position (which is often more difficult to measure in reality). Therefore, this controller has been modified to share information on distance between the robots instead of position information. This controller has also been implemented in Simulink and tested there, with the next step being to translate this into a Finite State Machine representation.

Once the controller was designed, I have done an extensive stability analysis of the system. A large part of the project is to find how moving from a synchronous system to asynchronous effects the stability of the distributed control system. By adding delay in the feedback path (the equivalent of making the system more asynchronous), the stability analysis shows that stability decreases (or the range of acceptable gains decreases) as delay increases. However, now that the math is done, it is desired to test this on a real system.

The next order of business was to translate the Simulink specification into Esterel. Simulink has no real model of computation associated with it, and therefore, its concepts of time are not really well understood. On the other hand, Esterel is purely synchronous and we can synthesize C code from the Esterel model. Several Esterel modules have been written to simulate different parts of the system. For the control itself, everything is kept very abstract and distributed across the robots (there is an Esterel module for the control on each robot). For example, the controller uses the distance input to set the speed and direction of the robot. This is done simply by emitting setSpeed() and setDirection() signals. Nothing needs to be known about how this is actually implemented on the robots. For simulation purposes however, a separate Esterel module is used to capture these setSpeed commands and output approximate positions that the robot will move to (by integrating the velocity).

Once the Esterel models are complete and properly simulated, the next step is to synthesize the C code from Esterel. From here, some code may be added to test what happens under more asynchronous conditions. In addition, the emit setSpeed() and emit
setDirection() commands may be connected through the robot API in order to test the controller on real robots (which will behave asynchronously).

Low Power Network Discovery Algorithms
Abhijit Davare
David Nguyen

Background
An ad-hoc sensor network is a network of numerous wireless nodes where each node has sensing, storage, computation, communication and possibly actuating resources. Each node has a limited power source that determines the lifetime of the node. Therefore, energy is the most critical factor in sensor networks in all of its design levels (i.e. hardware, network layers and applications). Numerous studies in sensor networks have already demonstrated that communication between nodes is the dominating source of energy consumption. While traditional wireless network architecture has been based on systems of static base stations, it appears that a multi-hop network, where each node communicates with a few close nodes, is most efficient in terms of energy savings and bandwidth reuse. In multi-hop networks, each node communicates with other geographically distant nodes using intermediate nodes to build a communication path.

Goal #1
The goal of this research introduces and evaluates new power saving techniques for wireless ad-hoc sensor networks. We first develop efficient and localized coordination protocols to schedule the activities of the nodes in the network and put the idle nodes in the standby mode. We also address the problem of finding an efficient and localized information gathering algorithm that minimizes the total number of radio communications during the task of gathering complete information from all of the nodes within a network or a specified part/subset of a network. The assumption is that each node has all the information about itself (including its geographic coordinates and its data) and all of the information from the nodes within its communication range.

In order to present the flow of the algorithm within the network, we introduce the notion of a token. Token(s) defines a node that is currently active and is executing the algorithm. The procedure involves electing the next active node and giving the control of the algorithm (token) to that node. The token node can only send control to the nodes within its communication range (neighboring node). This algorithm can utilize single or multiple tokens. We start by demonstrating our approach using a single token and then show how we can generalize the algorithm to use multiple tokens.

Goal #2
Since many nodes in the network can be idle, we can save power by turning the radios of these nodes off and putting them in standby mode. In the first part of the project, we describe and evaluate a localized technique for reducing the energy consumption of the
network by putting the maximum number of nodes in standby mode without affecting the connectivity of the network, {Xu01, Che01}. Each node determines its status based on its available energy and the connectivity of the network within its neighborhood. We have developed necessary and sufficient conditions for the local procedure for checking the connectivity of the network. Interestingly, the same mechanism is the basis for efficient power management techniques. Our power management technique finds the provably minimum power state of the nodes within a local neighborhood without affecting the connectivity of the nodes, while a subset of the nodes are staying active and are communicating.

Each node that changes its status to standby mode broadcasts its status to its neighbors before going to sleep. The second part of the project is the design and evaluation of a low power localized procedure for gathering information from the active nodes in a network while some nodes are in standby mode. By information, we mean a type of meta-data that can be different for different applications. In the simplest case, this information is just the status (active/standby) of all the nodes within the desired area. In order to find the information contained in a node, either the node or one of its neighbors should be visited. This type of data gathering is NP-complete. Since the data gathering procedure is localized and the network is ad-hoc, at each active node the important part of the procedure is to decide where next to send the token. We provide an efficient, yet complete, low-power heuristic for data gathering. By complete, we mean that we have, provably, all the required information at the termination of the algorithm.

**Approach**

The information gathering procedure is built on the observation that different network shapes dictate different traversal strategies. The procedure starts by exploring the shape of the network, using a geometric perimeter routing (outer face routing) technique {Bos99}. The nodes on the perimeter are assigned a rank of zero, while all of the other nodes in the network rank themselves according to their minimum distance from the perimeter nodes. The next active node is selected based on its rank, number of neighbors whose status is unknown, number second level neighbors whose status is unknown and the area constraints. The area constraint is the component of the algorithm that ensures all nodes of an area are visited before leaving that area. Otherwise, the procedure will encounter the additional energy overhead of visiting redundant nodes to return to unvisited nodes from an already visited area. We illustrate the important of articulation points in this area based approach. Articulation points partition the graph into a number of subgraphs. The procedure uses the bounds of each of these subgraphs as the area constraints. We illustrate how tuning the different parameters of the procedure can yield a more stable method for traversing the graph. Also, we study the performance of the algorithm for different shapes of the network. We evaluate the performance of the algorithm against instances with a known optimal solution to study the efficiency of this approach.

We calculate the total power consumption in terms of the number of transmission and receptions in the network. If the energy cost of a transmission is A and the energy cost of
a reception is B, the total energy cost would then become: \( A \times (\text{Number of transmissions}) + B \times (\text{Number of receptions}) \).

**Status of Project From First Update**

We have formulated the target problems, discussed, and outlined the pseudocode for them. The simulator for the localized network topology coordination procedure and the localized information gathering procedure is written in Visual C++. We have simulated the network by uniform random placement of the nodes and using a unit graph model for the nodes. In terms of implementation, a lot of progress has been made for the implementation of information gathering. The perimeter routing and relaxations to find the rank of the nodes (i.e. shortest distance of the node to a perimeter node) are already implemented. We identify the perimeter nodes and divide the space into subregions accordingly. We have also used the three components (i.e. energy, unvisited neighbors, and area) to find the next active node and go through the nodes in the network. Currently, we are implementing the generator of instances with known optimal solution to test the performance of the algorithm against it.

**Current Status of the Project**

In developing a method to determine which nodes go to sleep, it is useful to know the amount of area uniquely covered by a node. Since we are interested in a complete network discovery algorithm, we do not want nodes that uniquely cover some area to be put into sleep mode. The exact approach to determine the amount of unique area covered by a node can be solved by geometric formulations. We have found, however, that this can be very computationally intensive, and may not be well suited for constrained sensor network nodes. The alternative we have chosen is a grid-based approach. To accomplish this, we overlay a grid onto the section of interest. With this, we can discretize the coverage of nodes into grid cells. In our implementation, a matrix is associated with the area of interest. Entries in the matrix indicate the nodes that cover each of the cells. Finding nodes that cover unique area is then approximately equivalent to finding matrix entries that are covered by only one node.

To accurately evaluate the performance of low power sensor network discovery algorithms, it is useful to have an optimal solution to compare against. The optimal solution consists of a network and a path through that network that gathers information from all of the nodes while visiting the fewest number of nodes. Finding the optimal path for a given network is difficult, even with global information. In order to develop benchmarks, we develop a method to generate networks, keeping track of the optimal solution during generation. The procedure is to first initialize the algorithm by placing a node at a random point in the area. Then perform the following two steps iteratively:

Place a unique neighbor, node A, in the range of the previous node, node B, but not in the range of any nodes prior to node B (if any).

Place a new node, node C, in the range of node B, but not in the range of any other nodes. This procedure continues until the desired number of nodes have been placed on the optimal path. The optimal path consists of the initial node and all of the “C” nodes in the iteration. After the optimal path has been generated, other “filler” nodes may be added to the network as long as they are in range of the path nodes. Since any number of these
“filler” nodes may be added once the optimal path has been chosen, the required network density can always be obtained.

**Pupil Detection and Tracking System**

Lior Zimet
Sean Kao

**Background**

Eye tracking using computer vision techniques has the potential to become an important component in future human-computer interfaces. Information about the eyes can be used to detect and track human faces and bodies, which has many applications in face recognition, monitoring human activity, stereoscopic displays, and more. The purpose of the system is to find the two-dimensional location of a person’s eyes using multiple light sources and an image sensor.

When a light source is aligned with a camera’s optical axis, the camera is able to detect the light reflected from the interior of the eye and the pupil appears bright in the image. This is often seen as red-eye in flash photography. The system consists of an image sensor and two IR sources that are placed around the camera’s optical axis. One source is close to the camera optical axis generating the bright pupil image and the other is sufficiently far to generate a dark pupil image. The light sources are synchronized with the sensor frame-sampling rate such that odd frames are illuminated by the close sources while even frames are illuminated by the off-axis source. We can find the location of the pupils by finding the difference of odd and even frames, followed by a threshold and tracking techniques.

![Figure 1: System Functional Block Diagram](image)

**Project Goal**

The goal of this project is to exercise the entire process of embedded system design from functional specification to implementation alternatives. We will describe the system by using a formal specification language, apply an appropriate model of computation to capture system functionality, and explore the architecture space. This includes examining different implementations considering software/hardware partitioning. Furthermore, the project will produce a viable embedded device or model of one that is capable of finding...
pupil locations. The design will focus on aspects of robustness and real-time performance.

**Approach**
We plan to design the system according to the design methodology outlined in the course. Formal specifications will utilize Unified Modeling Language to describe the functional characteristics of the system with a Use Case Diagram, Sequence Diagrams, and a Class Diagram. We will then use Simulink to create a synchronous dataflow model of computation of the system. The model will be fully functional and we will simulate the system using previously captured images from various environments. This verification process can eliminate specification and functional errors before any implementation. There are no available tools that are capable of automatically synthesizing HDL from Simulink or Matlab models with video frame data structures. Therefore we will explore system architecture options manually and implement one of these options. Given enough time, we would like to compare performance of different hardware/software partitions. The system will be implemented as HDL for hardware portions and standard programming languages for software portions. Available hardware programmable platforms will execute hardware components and a possible combination of microcontrollers or a PC may execute software components.

**Work progress**
The following work has been done so far:
- Writing a detailed system specification in English as a base for the formal specification language description
- Describing the system in UML high level description language:
  - UML use-case diagram for the system
  - UML Sequence diagrams
  - UML class diagram as depicted
- Preparing Simulink model of computation for the complete system.
- Capturing frames from a system in IBM Almaden center to be used as inputs to our simulation model in Simulink and Matlab
- Writing and simulating the detection algorithm in Matlab for design and verification purposes
- Identifying major hardware components that will be used for the implementation of the system according to the system specification and constraints:
  - Sensor will be a Zoran CMOS sensor
  - Programmable FPGA board for the main dataflow and control logic

**Progress Update**
We have accomplished the following tasks:
- Simulink model has been completed and debugged with half-scale test vectors. The model produces two-dimensional coordinates for a short video of forty 320x240 pictures. Figure 2 shows the top-level of the Simulink model.
- Simulink model is a completely synchronous system and is designed to follow the semantics of a synchronous dataflow model. This is a purely functional model and has been critical in refining the functional specifications of the system.
Explored architectural space with various hardware and software implementations for specific portions of the project, and performing the mapping as follow: Sensor controller is implemented in software on a PC. It is controlling the sensor through the USB connection.

All major components of the system that calculate the coordinates of the pupil locations shall be implemented in hardware. Data bandwidth on the software-hardware communication is too slow to be able to efficiently process large amounts of data on a microprocessor.

Some hardware components must utilize shared resources and communicate when they are employing certain resources in order to minimize memory usage. Decided on communication protocols to link the overall system: I²C bus protocol to control the sensor. USB serial communication between software on the PC and hardware on the FPGA board for system control and output results display. FPGA registers to communicate between hardware components on the FPGA. Completed writing major hardware portions in Verilog including: Memory controller (memory is a SRAM FIFO) Subtraction and thresholding of current and previous frames Calculating the size and properties of objects in the frame that are possible pupils Aggregating possible pupil objects in order to calculate their center position. Used ModelSim to test and debug Verilog components and tested various components together. Using Matlab to refine and verify the detection algorithms. Verified that the hardware is compatible. We have been able to communicate to the FPGA board through a USB port as well as communicate to the sensor. The sensor may be connected and send data to the FPGA board.

Figure 2: Simulink Model
Processor Modeling with YAPI
Sam Williams

Philips uses process networks to model their multimedia applications through a toolkit. YAPI, a UCB project, is an extension to these models, and is implemented as a Metropolis library. There are many advantages in using Kahn process networks to model systems including determinism, flexibility, concurrent processing, concurrent programming and explicit communication. All of which can be useful when modeling a processor. Furthermore, the object-oriented style allows for creation of extensible and configurable classes to represent anything from instruction sets to pipeline stages.

The goals of this project are to accurately, effectively, and easily model processor microarchitectures using the YAPI library. Of course, to model a processor, YAPI is not necessary. However, to accurately model timing, YAPI channels, which are basically unbounded FIFO's, are extremely useful, and flexible. Basic unpipelined processors are trivial so it was skipped. More advanced microarchitectures using process networks will be designed to implement pipelining, hazard detection, branch prediction and resolution, out of order execution, and decoupled microarchitecture. Furthermore, the flexibility afforded by the Java like language semantics allows for parameterized microarchitectures. For example, an execution pipeline depth could be modeled via a YAPI unbounded queue with two additions: flow control to ensure it maintains a fixed length, and initial filling with garbage up to the pipeline depth.

In order to accomplish these goals an series of microarchitectures will be constructed using YAPI, metropolis, and SystemC. All of these are required to run programs since the compilation flow starts with the YAPI extensions to metropolis, compiles it into java, converts it into systemc/c++, and final into machine language. The first step was to specify an instruction set architecture to be used throughout the examples. With this in hand, example microarchitectures can be conceived, implemented, simulated, and analyzed. Implementation can vary from a single process, which performs all computation, and a feedback channel designed to mimic the execution pipeline and thus latencies, to a multi-process/fixe-length multi-channel network for out of order execution, to a fully asynchronous decoupled microarchitecture with unbounded FIFOs. Furthermore, the parameterization of all classes allows for exploration of the microarchitectural space with the ability to quantify the performance of various microarchitectures. Thus a parameter like execution pipeline depth could be used to add latency to instructions, but could also allow a cycle time to be estimated from the depth. Additionally, an area function could be derived to provide additional feedback in analyzing the design.

So far, not only has the instruction set been defined, but 5 examples have also been created. The instruction set contains only RISC like operations such as load, store, add, subtract, boolean, etc... In addition two branches (if zero, if not zero) are defined. The architecture defines a single 256x32b register file, and always includes a Harvard style L1 cache.
The first example is a 5+n stage (modeled with 4 processes) in order statically scheduled processor. It is a very conventional looking DLX design with the additions of configurable execution pipeline depth, and configurable memory pipeline depth. As the first example, the most notable deficiency in process networks became visible - the possibility of deadlock. Without very careful planning of the architecture and scheduling within an iteration of read and write operations, it is quite easy to deadlock the machine. The second example was an abstraction of the first with the addition of a semantic for read/write operations - at the beginning of each iteration, write data from the last iteration is written (thus avoiding initial deadlock), then all ports are read. After that, the computation is performed. Thus each iteration is in the form: write, read, execute. Additionally, this example was abstracted into a single process with a feedback channel. 

The length of the FIFO represents the depth of the processor. The third example takes the previous single process DLX and adds hazard detection. The previous example was statically scheduled by the compiler, which highly simplifies the hardware at the expense of predictable execution models. The fourth version adds branch prediction. In this case, the processor is still an inorder design of depth n. A function (which could have been extended to a BTB) predicts branch targets and allows execution to continue. The branch is sent down the pipeline, and when it is time to commit, the branch direction is resolved and compared to the predicted direction. If a mispredict occurred, then the next n instructions which commit must be nullified, and the program counter is reset based on the correct branch direction. Any number of predicted branches may be in flight.

The most recent example is that of an out of order single-issue machine. In this case the design was partitioned into the following processes: Fetch, ReadWriteIssue, MUStation, and a parameterized number of integer execution units IUStation. Each station is a reservation station with a parameterized number of stations, and execution pipeline depth.

The design is based on virtual register allocation and renaming. Branches stall, since inorder commit has not been implemented. The design, although it has an arbitrary number of execution units each of which must write through a channel to all other execution units, and the commit stage, is not particularly complex since ports may be arrays. Thus a for-loop is used to access all ports. Additionally, this design uses a ReservationStationEntry derived from class Object to pass down all channels, greatly reducing the complexity of the design. Memory aliasing is avoided via compiler support with memory sync instructions designed to avoid hazards. Additionally, memory latency can be a stochastic process by simply inserting bubbles into the memory execution pipeline to mimic the unknown latencies required to access various levels of the memory hierarchy in the presence of TLB, L1, L2, L3, page misses.

The vast majority of the functionality in an out of order functional unit is common to all functional units. Thus it is quite easy and useful to create a Station Class which performs all reservation station management, as well as issue and dispatch. It is possible to extend this class to create an IUStation or MUStation by calling a user defined function which performs the functionality of the functional unit (i.e. integer operations, memory operations, etc…) Of course, memory units must also create an maintain data cache access. This simple out of order processor can easily be extended to a arbitrary wide
super scalar architecture by simply fetching multiple instructions per cycle, and using a YAPI channel to act as a instruction buffer. The ReadWriteIssue process will read an instruction, 1 at a time, from the instruction buffer, and dispatch them to the appropriate functional unit. Each dispatch will mark the functional unit as occupied. This process continues until the instruction level parallelism specified in the constructor has been achieved, all functional units have had instructions dispatched to them, or all functional units are busy. Although this is not a method applicable to hardware, given that the purpose of the project is processor modeling, it is perfectly acceptable to sacrifice direct ties to hardware in favor of cycle accurate behavior and parameterizability. The most recent examples are based around evolution of a pipelined ring. Where as a token ring passes a token to solve arbitration of the bus, this design segments the bus into stages with FIFO’s (i.e. YAPI channels). These designs reuse the virtual register allocation and renaming methodology so successful in previous examples. However, in order to solve the global knowledge and state stored in the ReadWriteIssue process, this example takes a “Lazy Susan” approach. In this case, the rotating table on top of the main table is replaced by a unidirection ring that dispatches issued instructions to all functional units. A microprocessor can issue an instruction to the ring only if there is space directly in front of it – i.e. a waitress can place a new dish on the table if there is space. On the reverse side, instructions can be dispatched to functional units from the ring if the functional unit can accept that instruction, and the instruction is in front of the functional unit. The analogy holds in that a person can remove the dish from the table if he wants it, it is directly in front of him, and he has space for it. Thus if there is no space for the instruction in the reservation station, it simply cycles around the ring. When an instruction completes in a function unit, it can be sent back to the processor on a separate commit ring. In addition the commit token must visit all functional units. Where as instructions can cycle forever (they must be executed exactly once), commit tokens have a finite lifetime. As tokens on the issue ring pass tokens on the commit ring they can transfer results (i.e. forwarding/renaming resolution/etc…). In this example, a RingInterface module is extended into Station and Processor classes, which in turn can be extended into specific functional units. Latency can be avoided by partitioning functional units and organizing the network topology based on the probability of data dependency between instructions. For example, it is unlikely an integer instruction is dependent on a floating-point instruction. As a result those functional units could be on opposite sides of the ring. The next concept to notice is that although there can be only 1 issue ring, there can be any number of commit rings organized into any ordering of functional units – thus minimizing latency further. Furthermore, any number of tokens can be on each segment of the ring (width becomes depth). This allows a super scalar vector processor to be implemented. Vector operations can either be decomposed into scalar operations and queued and dispatched to the ring, or can be nibbled on by each functional unit. If each functional unit can process a single element of a vector, it can queue its own subop, change the renames in the instruction and send it to the next functional unit. When all elements have been processed, the instruction on the issue ring is consumed. Rings can provide high frequency at the expense of latency. Parallelism, and proper design of the network can yield high performance.
Applying Ulysses to Bluetooth
Alvise Bonivento

Background
In the embedded systems design the problem of IP reuse and communications between different components is becoming more and more a primary concern. Traditional design approaches to select network resources are rather informal especially for the communication domain, therefore resulting in incorrect or inefficient solutions. For this reason methodologies and tools for system design based on formal MoCs have been recently proposed. A limitation of these approaches is that they impose an early partitioning of the system behavior and many errors may occur during this phase especially when the system operates under many but correlated scenarios. To avoid this problem it is necessary to switch to a design paradigm that is based on the following flow:

The designer specify separately all the scenarios that describe the behavior of the system. Tools derive an equivalent executable model from which an hw/sw implementation can be synthesized.

Ulysses is a new approach to protocol design based on synthesis from a scenario-based specification. Ulysses is based on the following principles:
Separation of communication and computation: the interaction among protocol entities is specified independently from data processing protocol functions.

Separation of function and architecture.
Protocols scenarios are specified using sets of related MSCs, called Message Sequence Nets (MSN), that are PNs whose transition are labeled with basic MSCs. The PN model allows the representation of concurrency and provides a formal model to check relevant properties like deadlock and race conditions.

Use of design patterns to simplify the design process. Often happens in protocols design to have the same function or procedure repeated many times, it is therefore useful capture this aspects in special patterns. Rules of composition are then provided in order to guaranteed consistency and correct by construction design.
Tradeoff architectures to minimize the implementation cost.

Project Goal and Approach
The project consists of applying the Ulysses methodology to the Bluetooth protocol. This will require describing the Bluetooth protocol in MSCs, formalizing the relevant patterns and deriving the consistent PNs models. The Bluetooth protocol suits particularly well for the test of this new methodology since it offers an overall structure which is different from the usual implementations of the OSI stack layer, offering parallel sub-layers and interactions between non adjacent layers. In such a singular approach (also due to the necessity of optimization for power concerns) the application of a formal design methodology like Ulysses can provide significant benefits. Furthermore the increasing need of communications between devices that can be very different between each other
will eventually result in protocol solutions that will offer aspects more and more distant from the classical OSI model and Bluetooth is just a step into this direction. Like all the new tools, Ulysses needs to be developed since there are some particular aspects of the protocol for which specification via MSNs do not seem to be the most natural approach. Another interesting dimension to be explored is the idea of building a fractal model for patterns (each pattern is the composition of other sub-patterns). It is actually our hope to encounter as many problems as possible on the way and to find consistent solutions for them in order to enhance the model and provide a wider range on patterns and eventually creating a pseudo-library of PNs models of the most common functions.

Where did we get so far (1st update)
We already developed an almost complete MSNs model for the Link Manager, the Host Control Interface and evaluated solutions for most of the remaining functionalities of the protocol. We are also on the way to start deriving the PNs model for a complete communication session at least for the layers above the Link Controller.

Where did we get so far (2nd update)
The PN model for the communication session has been developed and a mapping of those PNs into the Metropolis Meta-model is now under consideration. In the meantime some problems faced during the PN covering procedure have led to refinements and improvements of the covering algorithm of Ulysses.

Starting from the analysis of the Bluetooth example, some other ideas have come out on how to approach the protocol design via a Platform-Based-Design methodology that will enable automatic synthesis of the protocol starting from the specifications (services + constraints) without passing through the procedure of automatic layers refinement which has shown being not very productive.

It might also be the case that the overall idea of structuring the design on the OSI reference model might be the wrong approach, especially if projecting the scenarios of future network services. As a matter of fact choosing an OSI fashion layering is already an implementation choice, which showed great results until now, but it might fail in capturing the functionalities of networks where the users characteristics might be extremely diverse, like in the case of the “beyond 3G networks”. In order to face these shortcomings we might try to approach the problem from a higher level of abstraction and from that point on try to build a methodology enabling automatic protocol synthesis starting from a formal description of the specifications (top-down part), a formal characterization of implementation performances (bottom-up part), and a model of computation where these two aspects can meet in order to map functionalities onto architectures (meet in the middle part).
Effective Models for Network Application Design
Mel Tsai

Background
There are currently a wide variety of tools for network design, simulation, and application prototyping. The most popular software-based modular router is MIT’s Click Modular Router language, which allows users to quickly prototype and develop network router applications & protocols. By stringing together a series of pre-written network elements in the Click language, a network routing application can be built with useful functionality. Furthermore, the Click runtime environment can integrate this router application into the network stack of a Linux machine, allowing users to physically implement the network application on commodity hardware with relative ease.

Unfortunately, as with some other modular router packages, the Click Modular Router has several shortcomings. In my original project proposal, I proposed that the basic push-pull semantics of Click were “too restrictive” for many network applications. While this may still be true to a certain extent, after studying the semantics further, I no longer believe this is the primary weakness of Click as it relates to network application design. Rather, I have made the basic observation that Click designs are too fine-grained, and designing at a higher level of abstraction is required.

To illustrate this point, the diagram to the right shows the Click implementation of a basic IP router with two Ethernet ports. It contains 38 elements and 50+ connections between elements. Furthermore, the connections between elements are relatively inflexible in the face of changes; adding or removing functionality to this basic router design is cumbersome, especially if connections between existing elements must be manually removed and replaced with new connections. (Simply looking at the large text file describing this router will convince you of this problem.) In addition, this 38-element router does not include many useful router functions commonly found in commercial routers. These functions include ingress packet filtering policies, support for Ethernet VLANs and VLAN tagging, spanning tree algorithms, QoS or bandwidth limit policies, higher-level routing protocols such as RIP or OSPF, and the list goes on. Thus, a user who wishes to add functionality on top of a basic router must first implement the router, which could easily require more time than the application design time. A final point here is that the 38-element router in the figure has only two Ethernet ports, whereas the most commercial layer 2/3 routers and switches have 8+ ports. Adding more ports will cause the Click description (in terms of elements and connectors) to grow dramatically, especially if each port gets its own set of particular policies and
filters (it is reasonable for a router to have variable policies per port).

A basic question to be asked is, is the low-level fine-grained nature of Click useful? As it turns out, most software-based router design languages have such a fine-grained nature, so this basic question is not unique to Click. One possible advantage is the WYSIWYG principle: there is no ambiguity of what a Click router is doing. Click’s semantics and network elements are well-defined. On the other hand, recall that the overarching goal of Click is to facilitate the rapid prototyping of a useful network router application. If a user must connect 38+ network elements to achieve even basic functionality, this represents an unacceptable trade-off for an unambiguous description. Indeed, I claim that almost every useful network router application has the basic structure of a real router. As long as the router can be easily customized (via extensions or reductions) to suit any new application, you have achieved the basic goals of Click without resorting to a fine-grained approach, but also gaining the important advantage of high-level design without sacrifices to performance or functionality.

**Project Goals & Approach**

The basic goal of this project is to develop a new paradigm for a software modular router, based on the idea that users want a basic router “out of the box” which can be customized. I developed this idea after working with several commercial routers and switches from Nortel networks. These routers can be powered-up and provide basic layer-2 switching without any configuration from the user. Setting just a few simple configuration parameters via a command-line interface allows the implementation of VLANs, spanning tree protocols, IP forwarding, and the like. Thus, my idea for a software-based modular router is simple. In software, implement a full and customizable software router and present the user with an actual router’s command-line interface. This interface supports:

- Enabling/disabling of major functions to suit the needs of the particular application. These major functions include layer-2 switching, layer-3 routing, VLANs, RIP, OSPF, etc.

- Implementation of *packet filters.* Typically, a commercial router allows the user to configure and enable one or more packet filtering policies. These policies are enforced by any port or VLAN in which the filter is enabled (via a simple command-line option). For example, a filter might say “deny all packets that come from the 10.x.x.x subnet.” Using filters, very powerful routing policies can be developed that can implement a large subset of network applications that users may need. Using Click, on the other hand, would require adding a separate “packet classifier” element to each port for every packet filtering operation. This filter cannot be modified on-the-fly, cannot share state among other classifier elements, and is a cumbersome way to include simple packet policies (much less complex packet policies).

- Implementation of “QoS filters.” These are “plug-in” modules that contain basic QoS policies such as RED, bandwidth limiting, statistics gathering, etc.
In order to implement arbitrary functionality on the router, the concept of per-port packet filters can be extended. This idea is the major contribution of this project, and I have discussed this idea with people from Nortel and they agree. Essentially, rather than writing new elements within Click, users can simply write a new packet filter module in C++. In order to implement network address translation (NAT), for example, the “NAT packet filter” could implement a table of address translation rules. This table can either be private, or shared arbitrarily between other NAT filters on different ports simply by naming the filter uniquely or giving it a shared name. If desired, the NAT filter can completely bypass the rest of the router, turning the software router into a dedicated NAT packet-translating device. This generalization of the packet filtering concept is an extremely elegant and easy way to implement arbitrary router functionality to an existing router. Furthermore, this concept does not appear to be used by any major router vendor.

In summary, I am essentially going to implement an abstracted Cisco router on top of a Linux box with software. This router can be extended using normal packet filters & QoS modules, but can be made very general by implementing new filters in C++ and disabling some of the basic out-of-the-box features. Essentially, with this approach you can write network applications at a high-level of abstraction yet dive down to the bottom layer if necessary. There are some interesting side-effects of this approach. Unlike Click, for example, the hardware model of the router can be maintained within software, allowing real-world router performance to be estimated. The software-to-hardware mapping between filters, L2 and L3 switching, etc. in this approach is very natural if the model of computation is similar to a hardware router. On the other hand, Click designs are very far from the hardware, and the mapping of Click elements to hardware is a complex topic. In fact, entire projects within the CAD group have been dedicated to this effort, with limited success.

I have discussed this new idea extensively with several others, and this seems to be a good direction for high-level, mixed-grained approach to software modular routers. It removes several of the limitations of Click while giving the user a powerful interface for rapid prototyping. This approach is not purely general, and I realize that some (very) exotic network applications may not be suitable for this paradigm, but I believe these will be the exceptional cases.

The figure above shows the hardware/software model of a port within this framework. Once again, the basic router application functionality is provided out-of-the-box, but can be quickly extended by packet filters and QoS modules.
Completed Work
I have begun work on the basic run-time of the “router-in-a-box.” Luckily, much of the infrastructure of this software can be taken right from Click, and I am using Click’s code wherever possible. I have investigated how Click uses the Linux pcap library in order to “take-over” a Linux machine’s Ethernet interfaces and turn it into a router. I have nearly finished, in C++, the classes that will form the basic elements of this software router: packet filter prototypes, layer 2 switching with ARP, IP routing with a simple trietable implementation, and simple bandwidth-limiting QoS. By the time the final paper is due, I will have a working proof-of-concept of this idea with two or three working examples. Unfortunately, I do not think I’ll have time to implement a full command-line interface (CLI) to this router, nor will it contain many of the complex control-plane routing algorithms (such as OSPF) that a final implementation may have. These will be items for future work. On the other hand, Click does not have these features either.

Communication Based Analog Design
Yanmei Li

Background
Communication Based Design (CBD) has been actively studied in the last few years as a basic design paradigm to provide design reusability and system integration at the highest possible way. Its basic principles can be applied to analog design as well, i.e. CBD-A. At the analog level, communication deals with electrical properties of single port connections rather than with complex behaviors that represent protocols between communicating blocks. As is known, a crucial point in CBD is the synthesis of communication adapters. Incompatible protocols between two communicating blocks are captured using some formal specification mechanism and an adapter is possibly synthesized and inserted without further modification of the incompatible blocks. Such adapters enable the interaction between different blocks which have behavior mismatch. In CBD-A, a similar approach is considered. A formal description of the analog interfaces has to be determined. Considering an analog port as a two-wire connection, a set of electrical properties has to be determined to capture the issues involved in the adaptation process. As an example, port features may include DC level, dynamic range, driving/loading capabilities, biasing requirements, temperature dependency, and so on. However, it is evident that these electrical features of ports alone are not sufficient to determine the adapter to be inserted between analog circuits. Since each analog circuit acting as an adapter will insert some degree of degradation on the signal (e.g. SNR degradation), some knowledge on the system itself is required to determine the specifications on the adaptation block in terms of inserted noise, distortion, CMRR, PSRR and so on.

Goal
This project aims to explore CBD-A methodology based on an available hardware platform, Anadigm Field Programmable Analog Arrays (FPAAs). For the analog adapters, we choose to focus on a particular class that performs linear transformations of
the electrical signal, such as amplification, filtering and translation (DC offset). This class of adapter is indeed very useful, covering most of the signal conditioning circuits that are used in sensor read-out paths and a broad range of linear circuits. From the implementation point of view, we further restrict our implementation to OpAmp-based circuits and, more specifically, to switched capacitor circuits. Actually, very many of analog circuits can be implemented with switched capacitor structure, which can achieve high performance (e.g. it is easy to realize stable and accurate time constant) in small area. Thus, we will use Anadigm FPAAs whose pre-built blocks are switched capacitor circuits to implement analog interface adapters. To achieve that, some simulation strategy based on FPAA models will also be developed to ensure fast and accurate estimations of the non-ideal effects introduced by specific FPAA configuration.

Investigative approach
First, since we do not know enough about the FPAA performance and its signal degradation in terms of noise, distortion, dynamic range, etc, we need to get necessary data through direct measurements on suitable example circuits. Having these important data, we will develop behavioral models for the FPAA platform with some tools like Matlab, Simulink, Verilog-A. Then, with the idea similar to the formalism in CBD, we will formalize the analog interface specification from the electrical point of view and also formalize signal degradation. After that, to implement the analog adapters with FPAA, a synthesis mechanism will be developed to map analog system description and interface specification onto FPAA platform. Finally, this proposed CBD-A methodology will be tested by applying it to some simple analog design.

Progress (till the first project update)
Literature search. But it seems that few related work was done. (We still keep searching useful papers.)
Got familiar with the architecture and design environment of Anadigm FPAA. Its unique feature is to implement analog functions in reconfigurable architecture. So, it allows us to integrate analog circuits into programmable systems, thus gives us the analog equivalent of FPGAs. Consequently, time to solution can be drastically reduced and design flexibility is improved compared with an analog ASIC or a discrete implementation. Another important feature is, without interrupting the operation of a system, FPAAs can be under real-time control of the system. That is because an analog design implemented with FPAAs can be translated into C-code such that it can be controlled by a microprocessor. The basic architecture of an FPAA consists of pre-built Configurable Analog Modules (CAMs), which provide common analog elements, such as filter stages, amplifier stages, summing/difference stages, voltage multiplication, rectifiers, oscillators, references, etc. The design of complex analog systems is simplified since the design process is moved from the component level to the functional level. With FPAAs, designers can describe analog functions like gain stages and filters without considering the lower level of components such as Op-Amps, capacitors, resistors, current mirrors, etc. Physically, this Anadigm FPAA is based on a CMOS-based fully differential switched-capacitor technology with an analog switch fabric. Here, RC-equivalent networks are provided via switching capacitance. And the FPAA can achieve a very
broad range of RC-based functions. Its unique feature, analog programmability, is also provided by this switched capacitor technique.

Studied the analysis methods for switched capacitor circuits and associated non-ideal effects (e.g. noise generation).

To estimate non-ideal performance of FPAA (such as noise, distortion, etc), I built several example circuits and performed measurements on them. The measurements are downloaded directly from the digital oscilloscope such that the data can be processed and analyzed with the help of Matlab or in Labview environment. So far, some measurement data have been obtained, and they are being processed with Matlab.

**New Progress**

Thinking of approaches to build the behavioral model for the FPAA

Since the first update, I was focusing on the chip measurements, processing and analyzing the data in Matlab. Originally, I planned to finish this first phase of the project within two weeks. Unfortunately, it costs more time due to lots of unexpected problems. So far, measurements are almost completed although one more measurement is needed on several filter circuits with different Q. The FPAA model development will be based on the results of these measurements, so it is an important part of work.

Basically, an inverting gain stage is adopted as my measurement circuit. My measurement is focused on its noise, distortion, and dynamic range. One of the problems is that the noise is too big, which is far away from the claimed SNR. To check if the noise coming from the input signals, I measured the input signal (coming from the signal generator). Also, the signals at J9 and J10, which are the differential inputs to the AN220E04 are also measured. No big noise is observed here, so the noise does not come from the input signal. That also means, using single-ended input to the board is fine with the measurement. Another possibility is that the noise may be related to the FPAA system clock. So, measurements are done with different clock frequency. The results from FFT analysis show that, when CAM clock= 2.0975MHz, measured noise is at 2.008MHz (Fig.1 & Fig. 2); when CAM clock= 8.390MHz, measured noise is at 8.072MHz (Fig. 3 & Fig. 4). So, the frequency of noise is almost equal to the working frequency of the amplifier. Eventually, it is found that the big noise is caused by the on-board oscillator, not from the chip. To deal with this case, I disabled the on-board crystal acting as ACLK. Some waveforms like Fig. 5 & Fig. 6 are obtained when system clock and CAM clock are both 10MHz. Noise is dramatically reduced!

So far, the dynamic range, noise, and distortion information of FPAA have been obtained when it is configured as an amplifier.

**Figure 1 & Figure 2.**
Modeling continuous-time sub-systems in Metropolis
Gabriel Elrea

Background
Metropolis is intended as a design environment for heterogeneous embedded systems. In particular, for many applications it is very useful to model a continuous-time (CT) sub-system using the same representation model as the rest of the system. This would enable not only simulation, but also the use of analysis and synthesis tools over the whole system.

Following, we identify some scenarios in which the modeling of CT sub-systems would be valuable:

1) Sampled-data systems. The most common form of interaction between the discrete and the CT domain arises from the use of computers to control some physical process. In most cases, this is achieved by sampling the continuous variables and writing control
actions to a zero-order hold circuit at discrete time instants, usually in a periodic fashion.

2) Hybrid systems. Some CT system can't be described by ODEs, because the dynamics change abruptly when the state variables reach certain values (e.g., a collision), or when an external command is applied (e.g., a switch relay is activated). Therefore, they are modeled as a finite state machine, in which each state represents the evolution of the system as a set of ODEs, but transitions between discrete states may occur.

3) Event driven systems. In some cases, the system should be able to detect when a certain condition is satisfied by the CT variables, in order to execute certain tasks (e.g., temperature greater than 25 degrees). An event is generated as a consequence, and processed by the discrete system. While in hybrid systems, these events could indicate a change in the CT dynamics, in event driven systems, the events are used simply as information to be processed, and don't have necessarily an effect on the dynamics.

Goal
The goal of the project is to develop a general methodology for modeling CT sub-systems using the Metropolis meta-model. The CT behavior is described by ordinary differential equations (ODE), where time is the independent variable. An interface between a CT block and the rest of the system will be defined, and several modeling examples will be generated.

The main challenge in this project is how to deal with the notion of time. While this is an essential notion in CT systems, in many models of computation it is not explicitly included, or the semantics refer only to the concepts of causality or ordering of events.

Approach
First, we will try to identify the most efficient way of representing the interface between the CT sub-system and the rest of the system, and define a medium that implements this interface. In principle, we will assume that whenever a value is written into the CT inputs, a zero-order hold will retain this value until the same input is written again. A problem to be addressed is how to determine the validity of an input when the CT process reads it. Moreover, the semantics of the interaction have to specify when the execution of the process thread is enabled. Two approaches could be used: the CT process is executed when an input is provided, or when an output is required. We will explore the use of the GTime quantity and its role in this interaction.

Next, we will study how to embed an ODE integrator into a process, trying to make this process as general as possible, so we could use different ODE solvers with minimum changes in the code. If time permits, we will study how to make the ODE solver detect events and communicate them to the rest of the system thru the interface.

The final outcome at the completion of this project, is expected to be a formal definition of the interface, and its implementation as a medium object, plus a (probably simple) ODE solver coded into a process. Both together would conform a "template" that could be used as a reference for future models. Several examples will be developed to show the
use of this template, and simulations will be carried out to demonstrate its performance.

**Work done until the first project update**

We worked on ideas for the definition of the interface between the CT sub-system and the rest of the system. We concentrated our efforts initially on sampled-data systems, but keeping in mind potential extensions to more sophisticated systems. Different possibilities were considered for the semantics of the interface, without converging to a satisfactory solution. Most of the time was spent in trying to understand the semantics (and, to some extent, also the syntax) of the Metropolis meta-model.

**Work done after the first project update**

I. Formal analysis of the interface

We present in this section an analysis of the interaction between the CT sub-system and the DT sub-system, and impose some restrictions on the events in the interface between them.

For simplicity, we take the convention of considering the CT sub-system "passive" and the DT sub-system "active", in the sense that decisions are made in the latter, although this is not required for the validity of the results to be presented. With this convention in mind, we group the ports of the CT sub-system into inputs $u_i$ in $U$, $i=1...m$, and outputs $y_j$ in $Y$, $j=1...p$. The events in the interface are therefore grouped as write events $w$ (DT to CT inputs) and read events $r$ (CT outputs to DT).

On a side note, we mention that there may exist a causality relation $C$ in $U \times Y$ such that $(u_i, y_j) \in C$ indicates that output $y_j$ is "affected" by the input $u_i$. This could indicate, for example, if the CT sub-system is composed of several decoupled sub-systems. We will assume that all inputs affect all outputs for simplicity, but if that's not the case we could derive less restrictive conditions, as it will be mentioned below.

Let's assume that all events $e$ are associated with two tags: execution index $X(e)$ and a time tag $T(e)$. The first one indicates the sequential order in which the event takes place in a valid execution, while the second one indicates the "real time" at which the event happens.

Under these assumptions, we make the following analysis:

1) Eliminating deadlocks. Since the outputs of the CT sub-system could be used to write new values on its inputs, we will require an ordering relation that guarantees that when a read is executed, no writes will be executed later with a time tag less than that of the read event. More formally: for all write events $w$ and read events $r$ such that $X(w)>X(r)$, then $T(w)>T(r)$. If the CT sub-system doesn't have an instantaneous mapping from inputs to outputs, then the requirement could be written as $T(w)>=T(r)$, which is less restrictive and becomes compatible with the GTime quantity already built-in in the Metropolis meta-model.

2) Eliminating the need for unbounded queues at the inputs. Since we are using zero-order holds on the write operations, each value written remains valid until the next write event. In order to avoid the use of unbounded queues to store all write values and at the moment of a read event ordering them according to their time tags, we will require that
the write events have to be ordered. Moreover, we will also require the converse of the previous requirement, i.e., after a value is written, no read event is executed later with a time tag less than that of the write event. More formally: for all write events w and w' such that X(w')>X(w), then T(w')>T(w); and for all write events w and read events r such that X(r)>X(w), then T(r)>T(w). Notice that, with this restriction, every time a value is written in any input, we can update the state of the CT sub-system to the time tag of that write event, and discard the previous value of the input, so we only need a buffer of size 1 for each input.

3) Eliminating the possibility of computing the same trajectory many times. If we don't impose an ordering to the read events, on every read event we would need to integrate the CT sub-system from the time of the last write up to the time tag of the read; if we have many consecutive read events we have to repeat the procedure each time. To avoid this, we will require an ordering of the outputs such that we can update the state of the CT sub-system on every read. More formally: for all read events r and r' such that X(r')>X(r), then T(r')>T(r).

In case we have a causality relation C, these requirements can be easily extended by imposing the ordering between inputs and outputs that are on causality relation, the ordering among inputs that affect the same outputs, and the ordering among outputs that are affected by the same inputs. These will be, indeed, less restrictive requirements.

In Metropolis, the tags X and T are equivalent to the built-in quantities GXI and GTime respectively, assuming we are using the least restrictive requirements (substituting > for >=). Therefore, if we annotate all events in the interface with the GTime quantity, the axioms in the definition of this quantity will guarantee that the previous restrictions are satisfied. This provides a sufficient condition.

Another sufficient condition, much more restrictive, is to embed the whole system in time, i.e., annotating all events in the system (not only in the interface) with the GTime quantity.

II. Definition of the interface
We will model the CT sub-system as a process, which interacts with the rest of the system thru a set of interfaces implemented by mediums. We define four interfaces:

1) M2DT: used by the DT sub-system to request a read event from a certain output of the CT sub-system.
   update boolean haveReadReq();
   update double getReadTime();
   update void issueOutput(double value);

2) CT2M: used by the CT sub-system to indicate the value of an output requested by the DT sub-system.
   update void writeInput(double value);

3) DT2M: used by the DT sub-system to request a write event to a certain input of the CT sub-system.
   update void writeInput(double value);
4) M2CT: used by the CT sub-system to get the value of an input written by the DT sub-system.
   update boolean haveWriteReq();
   update double getWriteTime();
   update double receiveInput();

   We can see from these definitions that time is not present in the interface between the DT sub-system and the medium. In fact, it is the medium that resolves the time of the read or write request and communicates its value to the CT sub-system with the functions getReadTime and getWriteTime. The medium also provides the functions haveReadReq and haveWriteReq, which will allow the CT sub-system to synchronize with respect to the read and write requests from the DT sub-system.

III. A sampled-data system example

We are working on the Metropolis meta-model code that implements a sampled-data system consisting of a tank and controller that tries to keep the level of liquid in the tank constant.

As stated in the previous section, we annotate the events in the interface with the GTime quantity. The sampling time is imposed by including constraints on the read and write events of the DT sub-system. The CT sub-system synchronizes with those events by "awaiting" for the medium to indicate a new read or write request.

1) Imposing the sampling period.
   constraint {
     loc for all i: GTime.A(beg(Controller,M.read),i+1) ==
                     GTime.A(beg(Controller,M.read),i)+T;
   }

2) Imposing synchronous behavior.
   constraint {
     loc for all i: GTime.A(beg(Controller,M.write),i+1) ==
                     GTime.A(beg(Controller,M.read),i)+T;
   }

3) Synchronizing the controller and the plant.
   await {
     (M.haveReadReq() | M.haveWriteReq()); this.all; this.all);
   }

This approach can be extended to non-synchronous read and write events and non-periodic sampling. It is also possible to refine the interaction between the DT and CT sub-systems by introducing A/D and D/A converters with non-zero conversion time.

At the present, we are working on the syntax of the Metropolis meta-model in order to compile and simulate this example. The key point in this approach is to resolve the time quantity in the medium.
Implementing Run-Time Support for DRAFTS
Mark McKelvin

Background
The focus of this project is to implement a run-time support environment for the Distributed Real-time Applications Fault Tolerance Scheduling (DRAFTS) project. DRAFTS is a project that addresses the issue of automating the synthesis process from a fault tolerant data-flow (FTDF) model, that is based on the underlying classic data-flow model of computation, to executable code such that the implementation is fault-tolerant and meets real-time constraints. Unlike the classic data-flow model, the FTDF model has firing rules that allow an actor to fire when a subset of inputs is missing. It is based on assumptions on the distributed run-time environment that is characterized by its ability to perform message handling across distributed processes, static scheduling, and coordination between processors. Furthermore, the run-time environment will be used as a performance model to provide timing information about the FTDF. This will allow the designer to address the issue of scheduling computation and communication to meet real-time constraints.

Project Goals
The overall goal of this project is to implement a performance model that can be used by a designer to implement a FTDF model. The environment is to include a library of functions that may be used by the user or designer to implement a FTDF network. Primarily, my work is to assure the scheduling of computation and communication for a predefined set of actors that must operate according to the FTDF model specifications. One step to accomplishing the above goal is to model the communication of actors that may be placed on different nodes, or processors, and to model the communication of actors on the same processor. Keeping in mind platform-based design concepts and modularity, the communication scheme between actors used should ideally be able to be implemented on various architectures. Given that a library for specifying and simulating FTDF actors in Metropolis exists, the same interfaces between actors and communication media should be implemented. The final step is to implement a static scheduler that will schedule the communication and computation between actors.

Approach
The general approach taken to accomplish the aforementioned set of goals is by using a communication-based design approach. The actual implementation is similar to a micro-kernel structure that is modular and can be used on multiple, distributed processors and various architectures (ideally). For simulation purposes and a lack of appropriate resources, a single processor computer system running Unix or one of its variations will be used to run multiple copies of the micro-kernel. Each computer system represents a physical processor in the network. In addition, the actual implementation will allow the micro-kernel and its related library functions to run on multiple computer systems, so to fully test communication on the physical channel.

As stated before, actors are specified using a library called the FTDF library. This specification includes information such as the number of inputs, the number of outputs,
the computation, the destination and sending actors, and the firing rules for an actor. An
interface between the actors and the computer network (medium) will be created. This
interface will have the ability to send and receive data through the medium. To simulate
the communication between actors on different processors, the plan is to use User
Datagram Protocol (UDP) over the Internet Protocol (IP) to establish a physical
communication channel. UDP is chosen over the Transport Control Protocol (TCP)
because it is unreliable, but yet, it supports the ability to address data from source to
destination computers, thus creating channel end points. This is also parallel with the
FTDF model since its purpose is to be able to operate and respond to an unreliable
communication medium. Therefore, networked computers using UDP will serve as the
interface to the physical IP medium that will implement the communication between
actors on different computer systems. Actors on the same processor that share memory
will be implemented using a communication protocol that exploits the shared memory on
a single processor. Figure 1 below gives a very abstract example of how the interfacing
(shown in red), or micro-kernel, will cooperate with the user processes developed using
the FTDF library and the underlying network.

Work
The work done thus far has primarily been focused on formalizing functions that
interfaces the physical medium to the incoming and outgoing data between two computer
systems by using UDP and IP. I have created code that act as user functions that will
physically network multiple computers given an address using UDP over IP. I have also
begun work on communication between multiple processors on the same processor. I
have created a structure that can be used to initialize ports on the actors that will facilitate
the actual connection between the actor and a messaging task that will handle the routing
of data from a source port on an actor to the correct destination port of an actor.

Work Update
Since the last progress report on this project, I have added more functionality to the
micro-kernel code in the context of interfacing the transport of data from actor to the
physical medium. Up to this point, code has been generated that allows processors to
communicate with one another via channels. User functions are implemented that allows
the user to create a channel between processors via the UDP/IP protocol stack. Send and
receive tasks are near completion. Code has also been generated that allows the user to
specify a source and destination for actors to communicate via shared memory. This
functionality uses the actors’ ports and a driver to transfer data directly between actors
that share memory, or the same processor. Other components that have been
implemented include a set of functions that the user can use to schedule tasks and initiate
drivers and ports. The library uses the semantics of the embedded machine, a virtual
machine that mediates drivers, ports, and tasks in real-time. An application is also being
created to test the functions to aid in the refinement of the code.
User Application

FTDF Library

Micro-Kernel Library
- Scheduling Instructions
- Communication Tasks

Operating System

Physical Network

Figure 1: The micro-kernel library is an interface ports, drivers, and tasks between the FTDF library and the underlying network. The interface will offer communication tasks used to communicate between local and remote processes and a scheduling instructions to schedule all of its tasks.

**RTL Code Generation for Teepee**

Nathan Kitchen
Vinay Krishnan

**Background**
The Teepee framework is the design environment for the Mescal project. Teepee will generate both the simulator of a processor and synthesizable RTL code from a single formal description so that the simulation description and synthesis description of the design can be verified to be equivalent.

**Project Goal**
The project purpose is to implement RTL code generation for Teepee and show that the RTL description of the system is equivalent to the simulator.

**Approach**
First milestone
We aim to generate from the abstraction of the processor a very basic Verilog circuit description. This shall functionally be equivalent to the simulator. For this milestone, we will identify the functional units that are needed for each instruction and copy them so that there is a separate mini-datapath for each instruction. (This is the structure of the simulator.) We will select the output of a mini-datapath based on the current instruction.

Second milestone
The next step is (conceptually) to fold the separate datapaths back together into the original circuit topology. We will already know the necessary control signals for each functional unit from the first phase. We just need to read the right set of signals from a lookup table using the instruction tag as an index. One complication is that the processor may have multiple issue width, so we will have to find a way to combine sets of signals.

Equivalence
We will show that our code generation is correct by simulating the Verilog produced and comparing its outputs to the outputs of the simulator. We have many sample designs that we can use for this.

Work until First Update
We familiarized ourselves with the Teepee framework. We experimented with sample designs in the graphical. We studied the expression language and constraint language that is used to specify the datapath and control flow. We also inspected the flow of the hardware simulator generator and understood how it simulates the hardware. On the implementation language side, we familiarized ourselves with the Java language and the Verilog hardware description language.

Work since First Update
We began to write by hand the Verilog for a sample processor. In the process, we discovered that we did not know about all the constraints on the processor structure, but our mentor quickly brought us up to speed. Now we are fixing our example. We began writing the code to generate Verilog. Our code is modeled on the simulator generation code. It can already be invoked from the Teepee environment; we just need to finish the functionality. Our progress towards this was hampered for a few days by the crashing of a laptop used for the project and time spent on its subsequent reincarnation.

**Metropolis SystemC Based Simulator with the support of apped Behavior and Performance Indices**
Guang Yang
Daniele Gasperini

**Background**
To handle the increasing design complexity, platform-based design has been proposed for years. In this design methodology, a design can be viewed as two parts, the architecture and the behavior sitting on top of the architecture. The architecture provides some
software and hardware services. It also provides performance estimation for these services. The lower level an architecture resides, the more accurate performance estimation it can provide. The other part, behavior, defines pure functionality of the system. It has no information about how these behavior will be implemented, thus no knowledge about the performance of them. Then, designers can decide how the behavior would be implemented. This corresponds to choose or develop a particular architecture. Typically, to speed up the design process, designers often start from architectures described in high abstraction level. And then, successively refine the architecture down to real HW/SW implementations. One possible trace of such refinements could be transaction level, RTL level, gate level, physical level. No matter in which level an architecture is, after mapping behavior to that architecture (i.e. specify which piece of functionality is implemented by which service(s) provided by the architecture), it is necessary to evaluate how good the behavior is performed on that architecture under the mapping. This evaluation can be used to optimize the architecture, the behavior and the mapping. This project is targeting such an evaluation tool.

Goal
Currently, we are doing our work in Metropolis, a research project backed by the concept of platform based design. Our goal is to extend the Metropolis SystemC based simulation tool, and make the mapping and performance evaluation possible. In the new simulator, behavior and architecture will be run concurrently. By using special constructs provided by Metropolis MetaModel (mmm, design language in Metropolis) defined in both behavior and platform, behavior and architecture can be tightly synchronized. At each synchronization point, the behavior part can ask the performance indices from the architecture. This from outside seems like that behavior is mapped to the architecture. In previous tools, e.g. in VCC, mapping is done statically at compile time. And then performance numbers could be gained by either compiler or running the statically annotated program. Now, we are trying to do that dynamically. Designers can conveniently and independently change the behavior, architecture and the mapping, then simulate them again. In this way, the effect of the changes can be quickly gained without compiling the whole design. This is especially helpful in debugging the design, or fine tune the mapping.

Approach
In this approach, we describe both behavior and architecture in metamodel, and then simulate them as we may do for individual part. The difference is that we add more synchronization points for annotated events. Whenever comes to such a point in either behavior or architecture part, the corresponding simulation is suspended and wait for scheduling decisions made by simulation manager. To implement this algorithm, we follow the same simulation strategy in the current simulator. The things we need to add are a few filtering functions for mapped behavior synchronization and quantity resolution.

In order to achieve the above approach, we must incorporate following functions into the simulator.
Define two networks: the user-provided real network (including both behavior and architecture parts) and the automatically generated or user-provided scheduling network (to resolve performance indices for both parts)
Register synchronization points from both parts to simulation manager
Register performance indices’ requests to simulation manager
When coming to a synchronization point, resolve performance indices and reach a fix-point for all indices in the platform (also possible for indices in behavior if any)
Pass performance indices from platform to behavior part
Coordinate simulation to let it run one synchronized step each time
For all above work, SystemC code should be automatically generated. Also, there are newly defined syntax for mapped behavior and quantity manager stuffs, which have to be handled by compiler. These require quite some implementation work on metamodel compiler.

Work
We have decided the functions needed for simulation and the simulation strategy, which includes where, when and how to perform those functions. For each function, we devised sketchy algorithms and defined some basic supporting classes. A good part of functions have already been developed*.
Change some syntax rules in MetaModel compiler to support mapped behavior and quantity manager*.
Constructing a test case, which demonstrates the mapped behavior and quantity manager mechanism*.
The rest of our work would mainly be implementation and debugging.

(*: Work done since last update. )

Communication-Minimized Distributed Computation
Jie-Hong Jiang

Background and Previous Work
As an embedded system interact with its environment, which may span over a large area, it is sometimes necessary to distribute a computation task to some particular locations. In such cases, the communication costs among different locations may be quite expensive or unreliable. Therefore, it is important to derive algorithms distributing computation tasks under physical constraints with minimal use of communication resources.
Communication complexity has been intensively studied in the community of theoretical computer science for over two decades. Previous work focused on proving lower and upper bounds of communication complexity, assuming computation models have unbounded computation power and unrestricted communication depths. However, these assumptions usually violate realistic situations, especially for embedded systems. As a result, communication complexity in a design perspective is relatively new in the literature. Here we study communication complexity in the context of embedded system design.
Objective
Given a computation task and a geometric partition of its inputs and outputs, we study the required physical links and depths of communication among different blocks induced by the partition. In particular, we consider the communication between two parties who have bounded computation power subject to implementation constraints.

Problem Formulation
In this project we explore the solution to the problem formulated as follows. Let $C(I,O)$ be a computation task (either combinational or sequential) with sets $I$ and $O$ as inputs and outputs respectively. Suppose $I = I_1 \cup I_2$ (with $I_1 \cap I_2 = \emptyset$) and $O = O_1 \cup O_2$ (with $O_1 \cap O_2 = \emptyset$) are physically constrained such that $I_1$ and $O_1$ are separated from $I_2$ and $O_2$ with very precious communication resources between them. We are asked to fulfill $C$ under a communication depth constraint by using a minimum number of communication channels bridging such a separation.

Planned Approach
We tackle this problem according to the following steps.
Combination instances with single output.
Combination instances with two separated outputs.
Combination instances with two sets of separated outputs.
Sequential generalizations (more specifically, for finite state machines).

What’s New since the Last Update?
We notice that communication depths should be taken into account in addition to physical links because the performance of an embedded system is very likely dominated by the communication depth. If we restrict the communication depth to be one as in Figure 1(a), then the minimum communication bits can be derived using functional decomposition. In the depth-2 case in Figure 1(b), minimum communication can be achieved by generalizing non-disjoint functional decomposition. For the general depth-$k$ case in Figure 1(c), the exact computation is still open. With the notion of communication depths, the computational complexity of finding a minimum communication protocol in a...
Specific depth upper bound $k$ is exponentially harder than that in $k-1$. We are searching a heuristic approach that is scalable with respect to the depth upper bound.

Figure 2. (a) Topological separation of a computation task. (b) Functions of each computation block in (a), where $x$ and $y$ have two bits. (c) Signals of $x'$ and $y'$ under inputs $(x,y)$, where “T” and “B” denote the oscillation and the bi-stable behaviors respectively.

Figure 2 shows an example where combinational cycles are necessary to achieve minimum communication objective. Breaking the combinational cycle increases the required communication link. Previously we conjectured that the additional power of combinational cycles stems from the fact that it must have unobservable oscillation and/or bi-stable behaviors inside circuits. However, this is true only when the two parties $A$ and $B$ in communication do not use disjoint conditions to break dependencies. Once $A$ and $B$ share some information, they may possibly use it to eliminate the oscillation and/or bi-stable behaviors.

For sequential circuits, we can consider that two parties $A$ and $B$ in communication have the same copy of a state transition graph (STG) which specifies the overall specification of the entire embedded system. However, due to the physical IO separation, they have imperfect information about the inputs and about the current state of the other party. But remember that they originally have the same STG, we may extract some information to reduce the communication. With the separation of outputs, it induces different equivalence classes of states for the two parties. Fortunately, for one party, say $A$, according to its current state it can guess $B$'s possible current states. Hence $A$ may use this information to reduce the communication cost, and similarly for $B$. The sequential case differs from its combinational counterpart in that it has additional state space. However, one might break the cycles and treat state variables as primary inputs. This case one might solve the sequential instances using combinational solution. But whether or not this approach would cause the loss of optimality needs to be investigated. We are now looking for game theoretic analogies as a possible solution candidate.
Environment Modeling in Quasi-Static Scheduling
Donald Chai

Background
QSS is a technique to reduce overhead caused by communication and context switches for a network of processes. It takes as input a set of communicating processes and generates a single sequential program that reacts to various inputs by interleaving operations of the original processes. It does this by modeling the schedule as a game between the system and its environment.
To be able to schedule more interesting processes, the current assumption is that some inputs are fully controlled by the system (the system always wins the game). However, this assumption may be too strong and lead to deadlock in the actual composition of the system with its environment.

Project Goal
Investigate conditions for the successful operation of a schedule in an environment.

Approach
There are (at least) three approaches. They are:
- model the environment explicitly by extracting the necessary information from the environment.
- form a set of rules describing a subclass of Petri nets which are always safe to perform QSS scheduling on, but weak enough to allow interesting programs to be written.

Work
I have formulated a method in which to associate fireability of input transitions to marking of the Petri net or nodes in the reachability space.
I have tested the existing QSS scheduler on a set of examples to see how it behaves on different inputs. I am exploring the use of trace theory as a verification tool for modelling the environment, but it may turn out to be more than enough.
I will work on a larger MPEG2 example to see how partitioning affects the system.

A Methodology for the Computation of an Upper Bound on Noise Current Spectrum of CMOS Switching Activity
Haibo Zeng
Joshua Garrett

Background
In mixed signal designs, the switching activity of CMOS digital blocks is responsible for noise currents injected into the power supply system and substrate. These injected currents may affect the reliability and performance of sensitive analog components. To verify correct operation, we would like to quantify this noise. For practical reasons, a measure of the noise current cannot be done through exhaustive simulation; rather, we
would like to determine an upper bound for the spectrum of the noise current with respect to all possible transitions of circuit inputs.

Project Goal
The project goal is to verify and improve the accuracy and utility of the upper bound current spectrum algorithm in [1], as demonstrated through several benchmark digital circuits. In order to implement the algorithm, we will have characterized a set of CMOS standard cells with respect to the input timing and output load dependence of noise current. This approach is novel in that it does not require exhaustive simulation of the circuit under test (in fact, the algorithm complexity is shown to be only linear in the number of gates), and yields an upper bound in the frequency domain, which is the most useful point of reference for analog and rf circuit designers who wish to ensure robust operation of their sensitive components.

Approach
An algorithm for computing the upper bound of the noise current spectrum was proposed by Dr. Alessandra Nardi in [1]. This algorithm will be tested in this project, and improvements made where possible to increase accuracy or reduce computational complexity. To test the algorithm, we will need to characterize a subset of gates from the ST 0.13um process library, producing look-up tables for each of the $2^{2p}$ input transitions (for a gate with p inputs) which relate noise current values to output load capacitance and input transition times. The form of these look-up tables will have a one-to-one correspondence with those provided for delay calculation, since these templates represent the useful range of input timing versus output load for each cell in the library. Several benchmarks will be used for test cases, in order to compare the upper bound generated from the algorithm to that measured by full simulation. These benchmarks will be converted from native BLIF (Berkeley Interchange Format) format to VHDL, synthesized into gate level Spectre netlists using Synopsys Design Compiler, and imported into Cadence. These netlists will also be imported into SIS, so that the graph traversal and logic simulation necessary for the algorithm can be performed. Before exploring the algorithm, we need to verify the accuracy of the gate current spectrum model itself by comparing the current injection and the corresponding spectrum according to a circuit simulation and according a reconstruction from the characterized library. We need to choose the input transition times and arrival times of the primary inputs in order to generate on the internal nodes all the particularly critical cases.

Work to First Project Update
To date, Dr. Nardi has outlined much of the algorithm and characterization process, but scripts for cell characterization (targeted for an older 0.18um process technology) and implementing the algorithm remain unfinished. Our work to this point has been in studying the previous work in order to understand how to prove the proposed algorithm. In order to facilitate the characterization process, some preliminary tests of the simulation tools and a new process technology have been done.

Additional Work and Second Report Status
Since the library characterization is being migrated to a new 0.13um technology and standard-cell library, a few implementation details of the characterization approach have been updated. A test-bench circuit (my_adder.blif) that describes a 16-bit adder has been selected, and which synthesizes to four distinct gates in the new library (all three-input, single-output type: AO5HS, AO5NHS, EN3HS, and EO3HS). The bit-significant structure of the adder allows freedom in determining how many inputs and outputs to consider in testing the noise-current spectrum algorithm – verification of the algorithm can initially be done considering only the lower few bit positions, and eventually extended to a larger portion of the netlist. The previous noise-current extraction script is being brought back online. Out of date software references are being updated to match the current environment – currently, the characterization script does not complete, but these updates should take no more than a day or so. The output form of the noise current look-up tables are chosen to match those defined for the gates under investigation in the new library (as listed for gates in the hcmos9 CORE9GPHS.lib file). Several tasks need to be completed before the algorithm is running. The main file compiles properly, but gives several faults when testing the benchmark .blif file under SIS. For example, a segmentation fault is given while running the my_adder.blif and a floating exception is given while running the prova.blif. The “segmentation fault” maybe appeared as early as processing the first internal node after all the primary inputs are processed. The fault seems to appear while getting the netlist of the corresponding gate whose output node is the processed node. To recover it, we need to rewrite the CALCULATE_CT function; the input is the Composite Table of the primary inputs or the calculated internal nodes, and the output is the Composite Table of the processing node. The “floating exception” appears while running the function CHECK_SIMUL in utilities.c which is used to check whether the assumption in the extension from single input switching mode to multiple input switching mode is satisfied; to recover this arithmetic exception, we need to know clearly how the output transition time is dependent on the propagation delay of the logic gate. Constructing a final version of the algorithm will proceed in two stages. First, we need to perfect the algorithm for 2-input gates -- there are still some bugs in the programs, which are needed to be figured out and recovered; this issue is discussed as above and we are just working on that. Second, extension from 2-input gates to multi-input gates may require some elaboration on some issues, for example, how to deal with simultaneous events, how to update the phases of the current.