A Model-Based Design Approach for Wireless Sensor-Actuator Networks

Anthony Rowe Gaurav Bhatia Raj Rajkumar
Dept. of Electrical & Computer Engineering
Carnegie Mellon University, U.S.A.
agr@ece.cmu.edu,gnb@ece.cmu.edu,raj@ece.cmu.edu

Abstract— **In this paper, we propose a model-based design approach for developing wireless sensor-actuator networks that can map multiple sets of application-level interactions onto a single networking substrate while still enforcing individual requirements. We use a top-down design approach where the functional requirements for each application are graphically modeled using a tool called SysWeaver. Sensor networking applications add unique challenges for model-based design frameworks because the system deployment view is tightly coupled to an installation-specific network topology and link characteristics. Wireless devices can also be mobile and hence may not easily map to standard deployment views. We introduce a SysWeaver plugin called *SenseWeaver* that is able to capture live toplogy data from an instrumentation deployment and feed the topology and link characteristic information to the system model. A developer can then use SenseWeaver specify the functional requirements of multiple applications, analyze communication and task scheduling requirements based on actual topology data, and automatically generate customized code for each sensor network node.**

I. Introduction

Wireless sensor networks provide a versatile and simple deployment platform for sensing and interacting with the physical environment. These devices can support multi-hop communication forming mesh networks capable of self-configuration, self-healing and automatic management. These properties make sensor networks suitable for various cyber-physical system applications like industrial control, critical infrastructure monitoring and building heating and cooling systems. Much work has been done in addressing a variety of challenging sensor networking topics including: network stacks, energy management, simulation, application task design etc. All of these components at each layer in the stack need to work tightly together for a deployed system to operate correctly and efficiently. Systems are now emerging where multiple sets of application requirements are sharing the same underlying infrastructure. For example, many home automation systems have transducers that should be shared with heating and cooling systems in order to optimize building energy consumption. In this paper, we present a plugin called *SenseWeaver* for the SysWeaver model-based design tool that helps model, synthesize, analyze and automatically generate code for complex wireless sensor-actuator applications.

One of the main advantages for using wireless mesh networking is the ability to rapidly deploy devices and have them automatically setup communication paths. For control applications, this makes it difficult to estimate communication latencies without having information about the underlying deployment topology which may be difficult to anticipate at design time. For example, an HVAC system might have a control loop designed around reading temperature and CO_2 values in multiple rooms that need to be processed in order to actuate heaters, coolers or blowers in different parts of the building. Without predictable system components and runtime information, it is hard to estimate reliability and to tune control loop update rates based on timing parameters. Model-based design of distributed applications provides the ability to use analysis frameworks that can ensure correct system operation while satisfying these types of para-functional requirements.

In general, model-based design holds the promise of (a) capturing rich behavioral descriptions along multiple concerns (or aspects), (b) offering an interoperable toolset to analyze and verify both functional and para-functional requirements of a system, and (c) supporting the ability to generate executable code directly from these models. While measurements from run-time environments may need to be fed back to calibrate the models (with parameters such as execution times, network topologies and system overheads), model-based design can in principle be independent from the specific hardware, operating systems or programming languages.

SenseWeaver uses a top-down approach where functional requirements of applications are graphically modeled using a tool called SysWeaver. SysWeaver enables the capturing of para-functional behaviors (e.g. timing, fault-tolerance, security, etc.) of an embedded real-time system and their interactions with the functional behavior of the system [1]. SysWeaver also has the ability to (a) analyze the para-functional properties of the system (e.g. timing properties) either internally or by exporting an appropriate subset of the model to external analysis tools, (b) automate design choices (e.g. mapping of software to hardware entities) [2], and (c) generate code for distributed embedded platforms [3]. This paper addresses the unique considerations required for applying this design cycle to sensor-actuator networks.

Modeling wireless networks has unique challenges as compared to existing systems that utilize component-based design. The physical environment and specific placement of devices heavily influences timing and reliability of communication links in sensor networks. In systems like automotive body electronics, the network topology and placement of hardware is well under the designers control. This is not always the case when deploying a reconfigurable control system using wireless components. Sensor networks tend to have highly redundant segments in the network layer. Instead of modeling this and other similar characteristics on an individual node-by-node basis, primitives are needed that capture aspects of the network as a whole. Finally, wireless networking models need to capture properties like mobility, self-healing and self-configuration.

To meet the unique challenges of sensor networks, SenseWeaver provides a WSN physical view, a set of sensor networking primitives, an analysis framework and deployment plug-ins for SysWeaver. We use an instrumentation phase to collect information about the environment from a deployed network. We introduce a primitive that represents the sensor network as a clustered component that allows deployment across multiple nodes with a single connection. We also provide the semantics to represent mobile nodes and how they can interface with the networking cluster.

II. Related Work

In the following section we will discuss various current approaches and related tools that aid in the design and deployment of wireless sensor networking applications. We will discuss existing programming language approaches, simulation tools and component based design modeling tools.

One approach to deploying sensor networking systems relies on using a high-level programming language with a single system-wide view of the application. TinyDB [4] takes a database-centric approach by accessing the network using SQL-type commands. A cross-layer design with an integrated tree routing MAC protocol facilitates communication optimized for database access patterns. This allows for energy-efficient network-wide querying of sensors with data aggregation. Though extremely efficient at accessing whole sensor networks, TinyDB does not support custom application-specific logic. The built-in routing protocol does not easily support arbitrary node-to-node communications. In many applications like control and automation, nodes need to communicate autonomously without explicit gateway control.

The Regiment Macro-programming System [5] is an example of a high-level programming language that describes an application as a set of spatially distributed data streams. Regiment contains primitives that facilitate processing data, manipulating regions and aggregating information across regions. The high-level program goes through a de-globalization process where code is compiled from a network-wide application into a set of node-specific executables. Regiment is compiled down to an intermediate token machine language that passes information over spanning trees constructed across the network. This approach provides great flexibility when it comes to application-specific logic; however, it is less efficient at providing short-lived queries like TinyDB. The token-machine-based approach does not easily lend itself to highly dynamic behavior with multiple modes of operation and changing data paths. The tight coupling between language and network protocol makes any-to-any communication as well as low-level adjustment of MAC protocols difficult.

Multiple research groups have developed wireless sensor networking simulators that tend to specialize in a particular layer of the system. ns-2 is an open-source discrete event simulator widely used in networking research. Primarily designed for simulation of IP networks, various projects like UCB Daedalus and CMU Monarch have extended the framework to support wireless communication and mobility. SensorSim [6] extends ns-2 by adding sensor network-specific models, supporting hybrid simulation and providing a graphical user interface. The OPNET Modeler wireless suite is a commercial tool designed for modeling various different wireless networking technologies ranging from 802.11 to mobile ad-hoc networks. The software focuses on the protocol stack with the ability to model RF propagation, interference, transmitter/receiver characteristics, node mobility and the interconnection with wired transport networks. OMNET++ [7] is an open-source discrete event simulator that shares many of the same features as OPNET. Tossim [8] is a discrete event simulator that emulates the lowest layer of TinyOS primitives. Tossim allows source to be compiled either for simulation or for real deployment on nodes. Em* [9] is a Linux-based framework that can run applications on embedded X-Scale or mote class devices. Em* software can operate in simulation or on real hardware. Simulators like ATEMU [10] and Avrora [11] attempt to simulate the network at the cycle accurate machine code level. Machine-code simulation allows any operating system to be simulated and is not limited to homogeneous source files. Most of these simulators are designed to aid users in developing and evaluating network protocols rather than looking at end-to-end application development. Our work is complementary in that it tries to generate the system using a top-down approach that would utilize an underlying network layer that can be fine-tuned using one of the many existing network simulators.

Various modeling tools have emerged in order to address the challenges associated with end-to-end application design for sensor networks. GRATIS [12] is a graphical framework built on top of the General Modeling Environment (GME 3) [13] that allows designers to connect different TinyOS components together. GME provides a meta-modeling framework where domain-specific models can be integrated with analysis and synthesis algorithms. It supports multiple views and supports most, if not all, phases of the development process. GME does not provide an automatic multi-view synchronization that reinterprets changes in one view in the semantics of the other as SysWeaver does. GME is based on meta-models that have offline interpreters, while SysWeaver provides modeling blocks (couplers) that validates the model as it is being built. GRATIS and its predecessor GRATIS II are able to statically analyze, validate and translate the models of TinyOS programs into NesC executables. It does not provide a way of modeling interactions between applications,

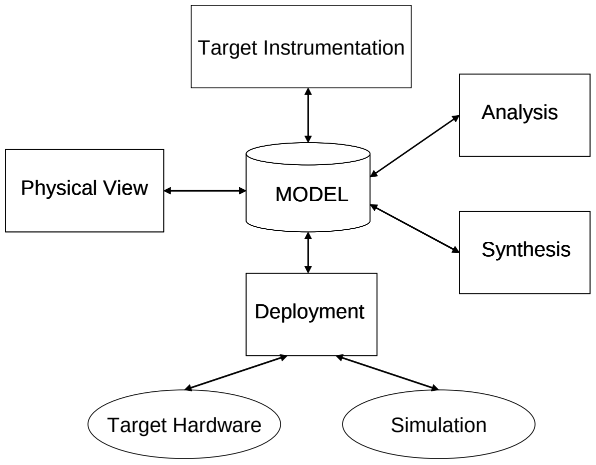


Fig. 1. The Components of SenseWeaver and their interactions.

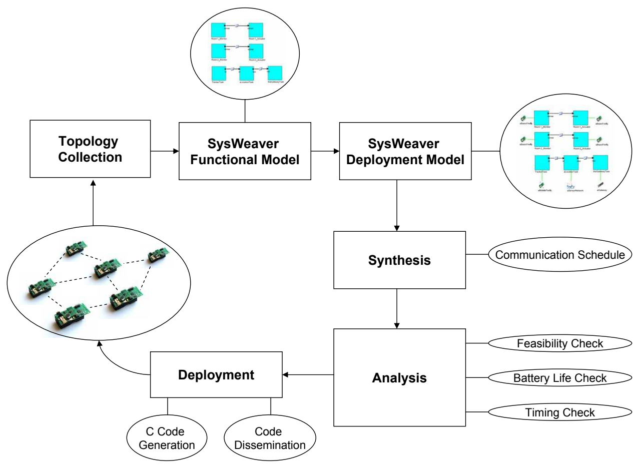


Fig. 2. Workflow that includes an instrumentation phase to collect live data from a network to enhance model parameters.

or the ability to analyze or simulate network interactions.

VisualSense is a modeling and simulation framework that builds upon Ptolemy II [14] for wireless sensor networks. Ptolemy provides models of computation with which the user can construct a system. Most of these models of computation support actor-oriented design. Actors are software modules that communicate with other actors through events. Actors have ports, and the port connections specify the communication parties. The execution of a model in a system is defined by a director. Each model has a director which specifies the semantics of the actor graph. A model can, in turn, be encapsulated in an actor by defining an interface. The execution of this model is then controlled by the director of the model into which it is inserted. VisualSense provides a means for defining the channels for sensor node communication as well as sensor node attributes. The framework permits the integration of additional node and channel models written in Java. Though useful for modeling sensor systems, VisualSense does not have a direct path towards code generation that can run on a real hardware platform.

Viptos [15] connects VisualSense with TinyOS and Tossim allowing graphical models of sensor networking applications to be automatically generated and deployed on real hardware. Viptos maintains the ability to connect Ptolomey II components with the TinyOS network which allows the introduction of non-TinyOS nodes. Viptos focuses on design and simulation of a single application system while SenseWeaver allows the modeling and composition of multiple applications that share a common network. SenseWeaver also introduces the notion of target instrumentation in order to provide its model with information from the real network.

III. Workflow

In this section, we introduce the steps in our proposed workflow for the design and implementation of a WSN system. Figure 1 shows the various components of SenseWeaver and their interactions. The workflow iterates between the model and various actions that iteratively add detail to the model. For example, a developer might capture toplogy information and then test to see if application timing parameters are met. If they fail, the designer can either re-visit the timing parameter, or try adjusting the topology by adding more nodes and then re-run the analysis tools.

At a high level, the SenseWeaver workflow consists of the following steps:

1) Model functional requirements of applications,
2) Introduce initial network topology data from the system into model,
3) Model physical attributes of network,
4) Synthesize system parameters which achieve or satisfy requirements,
5) Analyze the system based on user-input and synthesis output,
6) Repeat steps if necessary to satisfy specified functional requirements and system constraints, and
7) Generate Code and Deploy the application.

This design cycle is shown in Figure 2 with the main steps shown in rectangular boxes. The ovals show the supporting functionality that the SenseWeaver plug-in adds to SysWeaver.

A. Modeling

Model-based software design for wireless sensor networks aims to target those key areas of embedded systems which apply to a large-scale networks. Specifically, we look to achieve 1) composability and scalability, 2) multiple behavior encapsulation, 3) usability, 4) communication infrastructure and 5) correctness by construction. A model which can satisfy the requirements for these areas results in an efficient system design workflow and serves as the central component which interacts with other components in SenseWeaver. Visualization of the model is extremely useful for helping the designer navigate and interact with the model. In our workflow, we use the SysWeaver tool [16] to help us satisfy the above mentioned model-based design objectives.

SysWeaver provides abstractions to model both functional and para-functional behaviors into separate views whose interactions are automatically handled. Each of these views emphasizes a single concern enabling different domain experts (e.g. signal processing experts, control experts, real-time experts, fault tolerance experts) to focus on the concern of their expertise leaving the interactions with the other views to be automatically handled by the tool. The interactions among views are managed by maintaining a single internally consistent model, and treating each view as a partial projection of that model using a view-specific filter designed to only show elements and abstractions relevant to that view.

We now look at the requirements of the different modeling aspects as pertaining to Wireless Sensor Network design and how the primitives in SysWeaver enable us to model these requirements.

1) Functional Modeling: The functional model consists of the different applications in the system along with their interactions. This includes representations for periodic and aperiodic tasks, as well as a description of interfacing between the various tasks. We require a representation for the tasks, which can be defined as *blocks*, and the interfacing between them, which we call *links*. A system would consist of instantiations of blocks along with links ”wiring” them together. For wireless sensor networks, we would need to make sure that blocks can model periodic tasks, event-triggered tasks, as well as data-flow tasks. A block should have input and output interfaces and should be able to support multiple threads of execution. Links should contain information about what they communicate (eg. message sizes). Blocks should be composable and reusable so that multiple instances of the same type of block can be made.

SysWeaver uses the notion of a $Component$ to represent software modules. The main pieces of each $Component$ are $Ports$, $ApplicationAgents$ and $Couplers$ which are used to model the system and its interactions. A Port is the interface through which components communicate with each other. There are input ports which receive data and output ports which transmit data. A data transmission is represented as an *event*. Events are entities which are communicated across components and they can represent any data structure. The application code is represented inside an $ApplicationAgent$. An $ApplicationAgent$ is a set of functions which process and generate events. The $Ports$ contain entities called $ProtocolAgents$ which are responsible for communicating the data between components and contain the mechanisms to do so. For example, if components are on a single processor using shared memory, the $ProtocolAgent$ uses function invocation as communication. Conversely, if components are on different processors, the $ProtocolAgent$ uses inter-processor communication based on the network protocol property. $ProtocolAgents$ can contain multiple threads of execution as deemed necessary by the designer. $ApplicationAgents$ are reusable and can be hierarchically structured to satisfy the composability requirement.

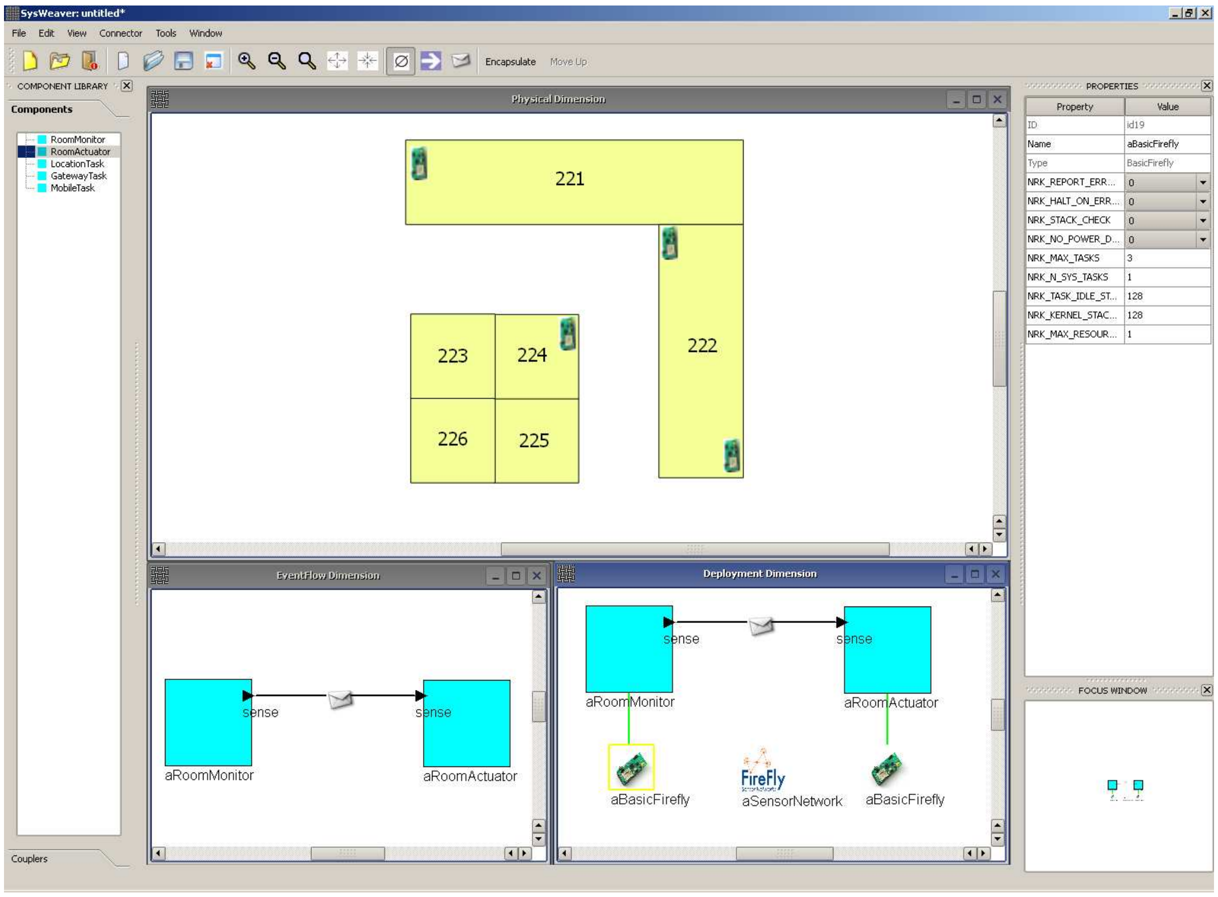
2) Physical Modeling: Wireless sensor networks have the property where changes in the physical environment greatly impact the system. Therefore, the system model needs to be aware of these physical properties and constraints and needs to capture these attributes. For example, node location, physical obstacles and infrastructure information play an integral role in how routes and communication schedules should be designed.

The Physical View in SysWeaver is used to capture the physical properties of the system. It can be used to model the sensor nodes and their location within the infrastructure as well as information about the infrastructure itself. An editor is used to create the infrastructure layout and nodes can then be positioned within the layout. This information is communicated to the underlying semantic layer. Any changes in the layout are automatically conveyed to the semantic layer to analyze the impact on the system. For example changes in how devices interact with the networking structure on the physical view would also be reflected as changes in the deployment view. The data in the Deployment View or the Physical View can also be populated by interfacing with external tools. Figure 3 shows a screenshot of SysWeaver with the different views and components of a model. Here we see the physical view on the top, an event-flow view on the bottom left and the deployment view on the bottom right.

3) Deployment Modeling: The Deployment Model describes the hardware used in the system and needs to have all of the important details associated with the target hardware and interconnections. The communication mechanism used in the network (eg. MAC Protocol, Link Layer Protocol) should be modeled and should support easy replacement. This applies to the underlying target platform which consists of hardware information as well as OS information, if any. The model enables architecture exploration for deployment purposes. The Deployment Model should also be able to communicate with the Functional model so that the appropriate communication and processing information as pertaining to the functional blocks is captured. This requires a mapping between functional blocks and the deployment blocks.

One unique property of WSNs is the notion of mobile devices that interact with the system and are also part of the system. These need to be part of the network, so they cannot simply be treated as system inputs, but do not have fixed locations. Many nodes have the property that the same application and networking code should run on multiple nodes within a network. The model should be able to easily support both of these properties.

Hardware information is easily captured by the Deployment model in SysWeaver using the notion of $Couplers$. $Couplers$ are primitives that express relationships between entities. For example, a $Network\ Coupler$ represents the network relationship between all the $Node\ Couplers$ connected to it. $Couplers$ also have properties associated with them. A $Network\ Coupler$ can contain information about the underlying MAC Protocol and any changes to this are propagated to all other $Couplers$. The Deployment Model also contains a $SensorNetwork\ Coupler$ which contains a graph and list of all the nodes on the network along with link information and topology information. This enables the



PROPERTIES

Property	Value
ID	id19
Name	aBasicFirefly
Type	BasicFirefly
NRK_REPORT_ERR...	0
NRK_HALT_ON_ERR...	0
NRK_STACK_CHECK	0
NRK_NO_POWER_D...	0
NRK_MAX_TASKS	3
NRK_N_SYS_TASKS	1
NRK_TASK_IDLE_ST...	128
NRK_KERNEL_STAC...	128
NRK_MAX_RESOUR...	1

Fig. 3. Screenshot from SysWeaver showing the different views of the same system.

designer to deploy a group of tasks onto the SensorNetwork Coupler. This connection denotes that the group of tasks should run on all nodes in the network in addition to other tasks assigned to individual nodes. SysWeaver also has a $MobileNode\ Coupler$ which it uses to represent nodes in the network that do not have fixed locations. Depending on the designer and the underlying target platform relationship, this Coupler can be used to represent unique nodes and the properties can be used for code generation and analysis purposes.

B. *Instrumentation*

This part of the framework is critical for maintaining a tight coupling between the system model and the actual runtime network properties. By being able to integrate real network data into the model, the system can be optimized to accurately reflect the desired functionality and requirements. Getting hold of this data is a difficult task and requires support from the underlying target platform. The Instrumentation Phase can occur on multiple occasions. It primarily happens at the initial step in the workflow, where we collect the initial network topology. After that, instrumentation can be done after deployment to collect data while the system is actually running. It is important that the instrumentation code be very concise and non-interfering since it may execute during normal system operation.

Information required for instrumentation in SysWeaver is collected either through help from the target platform, or through custom instrumentation code which can be generated from within SysWeaver. The data can consist of network connectivity graphs as well as link strengths between nodes. The data can be used to update corresponding components that exist in the Physical View and add components which might not have existed in the view. This closed in-the-loop design results in a tightly coupled system modeling and deployment. The instrumentation features in SysWeaver are extensible so that new data can be easily added.

C. *Synthesis and Analysis*

The power of model-based design is increased by the ability to do an efficient analysis of the system model. To support analysis, the model needs to encapsulate all the relevant information while effectively visualizing results. The kinds of analysis that a designer would be looking for are 1) node lifetimes, 2) end-to-end flow latencies, 3) network load hot-spots, 4) communication schedulability and 5) flow reliability.

The semantic model within SysWeaver gives us a way to store most of the information required for analysis. All Application Agents have timing and schedulability information such as deadlines and sampling periods. Based on the underlying platform, they can also be assigned priorities.

Each Application Agent has a Transition Table which defines the state machine within the component. Each entry in the Transition Table has a $Trigger$, an $Action$, and a Worst Case Execution Time(WCET) value for the entry. The $Trigger$ indicates what causes the transition to occur, which in most cases is a result of an event arriving through an input port. A special type of trigger called a $PeriodicTrigger$ indicates a periodic transition. The $Action$ describes the event produced from the $Trigger$. The $Action$ normally results in an event which is sent through an output port. The final event resulting from the Completion of a flow is designated as a $CompletionEvent$. Each Application Agent can have multiple entries in its Transition Table. The table entries have a many-to-many relationship to support all combinations of triggers and actions. Having this kind of structure within a component gives an indication of execution times within a flow as a result of interaction between components. To capture network latency, the Couplers which connect ports to each other have message sizes associated with them. This coupled with information about transmit and receive delays associated with the Node Coupler or the Network Coupler can be used to give network latencies and flow analysis. The timing information within each Application Agent along with the approximate size of receive and transmits done by each Application Agent gives us the resource usage of each Application Agent. By capturing this information in the Functional View, the Deployment View can then calculate node lifetimes since it has the mapping of the Application Agents deployed on each node. SysWeaver can export and import information from other tools which it can aggregate and provide to specific analysis engines.

Another property of the semantic model is that it can provide synthesis internally as well as by interfacing with external tools. Synthesis can be used to provide suggestions or estimate properties that the designer is trying to optimize. Coupled with the analysis engine, the synthesis engine can be a very powerful feature. It can provide insight into communication routes, node schedules, sampling periods for tasks and optimal locations for nodes. Being able to provide suggestions for system aspects can go a long way in helping the designer who may not be a WSN expert. The engine takes the model as an input along with the requirements which the designer is trying to meet and outputs the parameters that it can tweak to try and satisfy the requirements. Synthesis can invoke the analysis plug-in to verify if any of the constraints are being violated.

Figure 4 shows an example input and output file associated with a typical network toplogy that describes both the links and application-level communication requirements. ”$*$” is a symbol reserved to represent the body of nodes in the system. A flow generated from ”$*$” to a node, or from a node to ”$*$” is an upstream or downstream communication specifically from all nodes to one or from one node to all. In this example configuration, the mobile node RSSI data could be generated from any node and must be aggregated at a single point. The details of the actual communication scheduling are beyond the scope of this paper, but in general we assume an underlying TDMA MAC protocol. The analysis engine then relies on greedy graph searching heuristics that attempt to order flows while satisfying a two-hop interference constraints. Since each task in our model was given a priority and worst-case execution time, we can use well known real-time scheduling theory to ensure feasible schedulability of tasks on each node. With TDMA-based communication, we know all communication patterns ahead of time allowing us to determine the worst-case latencies in the absence of packet loss and pre-compute blocking times. Finally, by combining the worst-case execution times of tasks along with scheduled communication, we can accurately predict the worst-case energy consumption and hence the battery life of a node. The analysis engine can check these computed values against the parameters specified as properties in the design to alert the designer of inconsistencies. Even on a small system shown in our example, the scheduling complexity becomes difficult to manage by hand, making automated synthesis essential.

D. Deployment

The Deployment phase of the workflow involves gathering the implementation of the system as modeled. Different kinds of deployments can result from a single consistent model. Code for simulation as well as for the target hardware can be generated by adjusting deployment preferences. The implementation of the deployment phase involves building library blocks for the different kinds of deployment and for different simulators as well as different target hardware. Using the SysWeaver approach, each Coupler becomes a library block, wherein the Protocol Agent is the code that handles communication between components, the Application Agent code is the interface with the user application code, and there are $StateChangeEvent$ handlers which handle the relaying of events between the Protocol Agent and the Application Agent. SysWeaver is used to generate the ”sys-code” which glues together the coupler libraries with the generated communication code, and user-specific code. Having a library built for each type of deployment enables code generation for different platforms in the same system. Simulation is represented as a Deployment target which uses the coupler libraries built for the different simulators.

The SenseWeaver plug-in generates code that can be compiled to run on the $nano-RK$ real-time operating system (RTOS) described in [17]. Nano-RK is a fully preemptive RTOS with multi-hop networking support that runs on a variety of platforms. It supports fixed-priority preemptive scheduling for ensuring that task deadlines are met, along with support for and enforcement of CPU and network bandwidth reservations. Tasks can specify their resource demands and the operating system provides timely, guaranteed and controlled access to CPU cycles and network packets in resource-constrained embedded sensor environments. It also supports the concept of virtual energy reservations that allows the OS to enforce energy budgets associated with a sensing task by controlling resource accesses. Nano-RK provides various MAC and networking protocols including a low-power-listen CSMA protocol called B-MAC [18], an

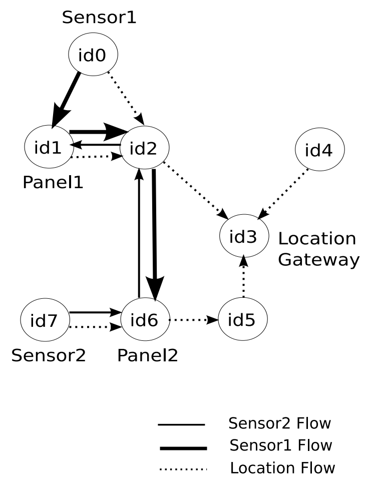
```
# <node idi> <link> <node id> :
#              [properties] <RSSI>
# = is a bidirectional link
# < is one way from right to left
# > is one way from left to right
links {
  id0 = id1 : -10
  id0 = id2 : -17
  id0 = id3 : -24
  id1 = id6 : -38
  id1 = id2 : -4
  ...
  id4 = id5 : -19
  id5 = id6 : -4
  id5 = id7 : -20
  id6 = id7 : -15
}


# <task id> : <start node> <link> <end node>
#            [ time bound, paths, ... ]
flows {
  room_sensor_1_1 : id0 > id1 [ 500ms, 1p ]
  room_sensor_1_2 : id0 > id6 [ 1000ms, 1p ]
  room_sensor_2_1 : id7 > id6 [ 500ms, 1p ]
  room_sensor_2_2 : id7 > id1 [ 1000ms, 1p ]
# Downstream flow from all to node_id3
  location_task : * > id3
}
```

(a)

```
node_schedule {
  id0
      TX: 1 Flow: location_task
      TX: 7 Flow: room_sensor_1_1
      TX: 7 Flow: room_sensor_1_2
  id1
      TX: 3 Flow: location_task
      TX: 8 Flow: room_sensor_1_2
      RX: 7 Flow: room_sensor_1_1
      RX: 7 Flow: room_sensor_1_2
      RX: 11 Flow: room_sensor_2_2
  id2
      RX: 8 Flow: room_sensor_1_2
      TX: 9 Flow: room_sensor_1_2
      RX: 10 Flow: room_sensor_2_2
      TX: 11 Flow: room_sensor_2_2
      RX: 2 Flow: location_task
      RX: 3 Flow: location_task
      TX: 4 Flow: location_task
 ...
  id6
      RX: 5 Flow: location_task
      TX: 6 Flow: location_task
      RX: 9 Flow: room_sensor_1_2
      TX: 10 Flow: room_sensor_2_2
}
```

(b)



(c)

Fig. 4. Example input and output from the SenseWeaver analysis engine. (a) shows sample input to the analysis engine that captures topology information as well as flows with their associated properties. (b) shows an example schedule output with the $room\_sensor\_1\_2$ flow in bold. The RX and TX followed by numbers denote slot numbers for an underlying TDMA communication protocol. (c) shows a visual representation of the topology with the various communication flows. The bold arrows correspond to the bold schedule entries in (b).

implicit tree routing protocol and a TDMA based protocol called RT-Link [19].

Due to the energy constraints and the desire for analyzable timing properties, we opted to use the TDMA network protocol, where all packet exchanges occur in well-defined time slots. Each node in the system must be given a time slot schedule that coordinates with its neighbors. Given a network link topology, using distance two graph coloring, it is possible to generate a schedule that is collision-free and avoids the hidden terminal problem. Given information about flows in the system, it is possible to further optimize schedules such that nodes are able to sequentially forward data within a single TDMA cycle.

IV. Conclusion and Future Work

In this paper, we introduced SenseWeaver, a SysWeaver plug-in that supports model-based design of wireless control applications. We showed how a top-down model-based design approach for building wireless sensor-actuator networks can manage complexity as well as enable the automatic integration of multiple applications. As component libraries become increasingly mature, system development will be able to cleanly reuse code. This will not only reduce the amount of hand written code required for an application, but also reduce development time and increase system reliability. Having a plug-in framework with a well-defined interface gives system designers more choices in composing, analyzing and deploying systems which will make future systems more structured and easily amenable to change.

In the future, we plan to incorporate existing simulation tools as well as develop more sophisticated synthesis and analysis engines. We also hope to expand upon the heterogeneous nature of SenseWeaver by supporting more platforms and networking protocols. For SysWeaver, we are developing an expert system which is a rule-based design engine that provides the designer with verification and analysis capabilities to compose systems. This involves the definition of a rules interface as well as support to encapsulate possible sets for rules which can be composed together. The interface of the Component Designer is being updated to add more custom properties as required by applications. This should help to increase the number of reusable components.

References

[1] D. de Niz and R. Rajkumar. Time Weaver. A Software-Through-Models Framework for Embedded Real-Time Systems. *ACM Language, Compilers, and Tools for Embeddded Systems (LCTES'03)*, 2003.

[2] D. de Niz and R. Rajkumar. Partitioning Bin-Packing Algorithms for Distributed Real-Time Systems. *Special Issue on Design and Verification of Real-Time Embedded Software*, 2005.

[3] D. de Niz and R. Rajkumar. Glue Code Generation:Closing the Loophole in Model-based Development. *Workshop on Model-Driven Embedded Systems (MDES 2004)*, 2004.

[4] S. Madden, M. J. Franklin, J. M. Hellerstein and W. Hong. TAG: A Tiny AGgregation Service for Ad-Hoc Sensor Networks. *Operating Systems Design and Implementation (OSDI'02)*, 2002.

[5] R. Newton, G. Morrisett and M. Welsh. The Regiment Macroprogramming System. *International Conference on Information Processing in Sensor Networks (IPSN'07)*, 2007.

[6] S. Park, A. Savvides and M. B. Srivastava. SensorSim: a simulation framework for sensor networks. *Proceedings of the 3rd ACM International Workshop on Modeling, Analysis and Simulation of Wireless and Mobile Systems*, 2000.

[7] A. Varga. The OMNeT++ discrete event simulation syste. *European Simulation Multiconference (ESM'01)*, 2001.

[8] P. Levis, N. Lee, M. Welsh and D. Culler. TOSSIM: accurate and scalable simulation of entire tinyos applications. *International Conference on Embedded Networked Sensor Systems(SenSys '03)*, 2003.

[9] L. Girod, J. Elson, A. Cerpa, T. Stathopoulos, N. Ramanathan and D. Estrin. EmStar: A software environment for developing and deploying wireless sensor networks. *USENIX Annual Technical Conference*, 2004.

[10] J. Polley, D. Blazakis, J. McGee, D. Rusk, J. S. Baras and M. Karir. Atemu: A fine-grained sensor network simulator. *IEEE Communication Society Conference on Sensor and Ad Hoc Communications and Networks (SECON'04)*, 2004.

[11] B. Titzer, D. Lee and J. Palsberg. Avrora: Scalable sensor network simulation with precise timing. *International Conference on Information Processing in Sensor Networks (IPSN'05)*, 2005.

[12] P. Volgyesi and A. Ledeczi. Component-Based Development of Networked Embedded Applications. *28th EUROMICRO Conference (EUROMICRO'02)*, 2002.

[13] G. Karsai, J. Sztipanovits, A. Ledeczi and T. Bapty. Model-Integrated Development of Embedded Software. *Proceedings of the IEEE*, January.

[14] J. Eker, J. W. Janneck, E. Lee, J. Liu, X. Liu, J. Ludvig, S. Neuendorffer, S. Sachs and Y. Xiong. Taming heterogeneity - the Ptolemy approach. *IEEE Special Issue on Modeling and Design of Embedded Software*, 2003.

[15] E. Cheong, E. A. Lee and Y. Zhao. Joint Modeling and Design of Wireless Networks and Sensor Node Software. *EECS Department, University of California, Berkeley, Technical Report No. UCB/EECS-2006-150*, 2006.

[16] D. de Niz, G. Bhatia and R. Rajkumar. Model-Based Development of Embedded Systems: The SysWeaver Approach. *IEEE Real-Time Applications Symposium (RTAS'06)*, 2006.

[17] A. Eswaran, A. Rowe and R. Rajkumar. Nano-RK: an Energy-aware Resource-centric RTOS for Sensor Networks. *IEEE Real-Time Systems Symposium*, 2005.

[18] J. Polastre, J. Hill and D. Culler. Versatile low power media access for wireless sensor networks. *SenSys*, November 2005.

[19] A. Rowe, R. Mangharam and R. Rajkumar. RT-Link: A Time-Synchronized Link Protocol for Energy-Constrained Multi-hop Wireless Networks. *SECON*, 2006.