Distributed Simulation of Heterogeneous Systems of Small Programmable Objects and Traditional Processors *

Ioannis Chatzigiannakis
Computer Technology Institute and CEID, University of Patras, Greece
ichatz@cti.gr

Christos Koninis
Computer Engineering and Informatics Dept. University of Patras, Greece
koninis@ceid.upatras.gr

Grigorios Prasinos
Computer Technology Institute and CEID, University of Patras, Greece
green@ceid.upatras.gr

Christos Zaroliagis
Computer Technology Institute and CEID, University of Patras, Greece
zaro@ceid.upatras.gr

ABSTRACT

In this paper we describe a new simulation platform for heterogeneous distributed systems comprised of small programmable objects (e.g., wireless sensor networks) and traditional networked processors. Simulating such systems is complicated because of the need to coordinate compilers and simulators, often with very different interfaces, options, and fidelities. Our platform (which we call ADAPT) is a flexible and extensible environment that provides a highly scalable simulator with unique characteristics. While the platform provides advanced functionality such as real-time simulation monitoring, custom topologies and scenarios, mixing real and simulated nodes, etc., the effort required by the user and the impact to her code is minimal. We here present its architecture, the most important design decisions, and discuss its distinct features and functionalities. We integrate our simulator to the Sun SPOT platform to enable simulation of sensing applications that employ both low-end and high-end devices programmed with different languages that are internetworked with heterogeneous technologies. We believe that ADAPT will make the development of applications that use small programmable objects more widely accessible and will enable researchers to conduct a joint research approach that combines both theory and practice.

Categories and Subject Descriptors

I.6.8 [**Simulation and Modeling**]: Types of Simulation—*distributed*; I.6.7 [**Simulation and Modeling**]: Simulation Support Systems—*environment*

*Partially supported by the EU within the 7th Framework Programme under contract 224460 (WISEBED)

MobiWac'08, October 30–31, 2008, Vancouver, BC, Canada.

General Terms

Design, Experimentation

Keywords

Heterogeneous Systems, Small Programmable Objects, Wireless Sensor Networks

1. INTRODUCTION

A particularly promising and hot research area has been the design and analysis of *wireless sensor networks* (WSN), which has attracted researchers from very different backgrounds, such as hardware, software, algorithms, data structures and evaluation, as well as researchers from various application areas. Advancements have been made in the physical hardware level, embedded software in the sensor devices, systems for future sensing applications and fundamental research in new communication and networking paradigms. Although these research attempts have been conducted in parallel, in most cases they were also done in isolation, making it difficult to converge towards a unified global framework. We envision that future research on wireless sensor networks and the Internet will exploit its theoretical and practical dimensions in parallel.

1.1 Coding for simulation and deployment

From a practical point of view, a joint approach of conducting research that combines both theory and practice must deal with considerable difficulties. To name a few, wireless sensor network application developers should acquire skills regarding embedded software engineering (manipulate timers and interrupts), dealing with low-level hardware devices (cables, hubs, and programming boards) and understanding the peculiarities of the wireless channel (hidden terminal problem, power versus distance model) etc. Still, even if all these skills are acquired, the cost of setting up and maintaining an experimental facility of significant size can be very high. Deploying the experimental network into a realistic environment requires iteratively reprogramming dozens of nodes, positioning them throughout an area large enough to produce an interesting radio topology, and instrumenting them to extract performance data.

An alternative to experimenting with actual sensor networks in order to validate and evaluate the performance of sensing applications is software simulation. Simulators provide a number of advantages over real world deployments, such as, e.g., the ability to easily test a great variety of parameters in protocols and deployment settings, including repeatable scenarios, isolation of parameters, exploration of a variety of metrics, and, of course, ease of development by leaving out all the details of deploying a real testbed. Currently, a number of simulators have been developed, most notable examples are **ns-2** [23], **OMNeT++** [24], **OPNET Modeler** [5], **GTNetS** [4], **YANS** [20] and **Shawn** [17].

Still, in order to use simulators as a basis, researchers make many simplifying assumptions that lower the overall simulation complexity, in order to conduct experiments with a lot of nodes in reasonable time. At the same time these same simplifications, be it in the simulation setup, the simulation scenarios, or the simulator itself, weaken the credibility of such research works. If the setup of the simulation and the selection of models and simulation scenarios are done improperly, this is a strong indication that the whole study is severely crippled from the beginning. An analytic and meaningful discussion of common pitfalls and shortcomings in the simulation is included in [19].

It is therefore essential to follow a combined approach with both software simulation and experimental evaluation. In such an approach, after testing a sensing application in a software simulation, it then has to be re-implemented for the particular software framework used by the real device. This is because simulators model hardware devices only in a generic, abstract way and simulation code is written either in a general-purpose programming or a dedicated scripting language. Furthermore, simulators often oversimplify the implementations of certain protocols that are widely used in networks, such as TCP/IP, leading to less accurate results. On the other hand, wireless sensor network devices usually execute a special-purpose operating system that may be written in a dedicated programming language and expose very different interfaces than those of the simulators. Thus, the benefits of following such a combined approach is somewhat lessened by this duplication of effort.

Some attempts to bridge the gap between simulation and real-world performance and to make the transition smoother have been proposed. The *NCTUns network simulator* [27] is an emulator for wireless ad-hoc networks, which uses the TCP/IP protocol stack available at the operating system to generate simulation results and allows real-world application programs to run on simulated nodes in order to generate realistic traffic. A direct approach to allowing both simulation and deployment using the same code base appears in the *Player/Stage platform* [13]. *TOSSIM* [21] takes advantage of TinyOS's structure and whole system compilation to generate discrete-event simulations directly from TinyOS component graphs. *SOS* [7] is an operating system for mote-class sensor nodes that takes a more dynamic point on the design spectrum than TinyOS. *Avrora* [26] is a cycle-accurate instruction-level simulator for the MICA2 sensor device. *Shawn* [17] focuses on the iSense platform and whole system compilation to generate simulations directly from the Shawn code. Recently, SUN Microsystems introduced the SPOT platform that is accompanied with *SunSpotWorld* software [9] that allows the emulation of such devices at Java Virtual Machine level.

1.2 Small programmable object technologies

There exist various different technologies of wireless sensor devices targeting low energy consumption, sensing modularity and reliable communication. Crossbow [3] offers a number of WSN platforms based on Atmel AtMega 128 controllers, feature low bandwidth radios or high data rate IEEE 802.15.4 radios and can be combined with a number of sensor boards. ETH Zürich developed the BTnodes [1] that combines an AtMega controller with Bluetooth radio. ScatterWeb GmbH [6] offers two lines of WSN hardware based on a TI MSP430 radio and a proprietary rate ISM band radio. The first lacks a voltage regulator but includes onboard sensors. The later features a regulator but no sensors. Coalesenses GmbH [2] offers the iSense platform, the first WSN hardware based on a single chip solution comprising an IEEE 802.15.4 radio and 32 bit controller. It features a software controllable voltage regulator and can be extended by a number of sensor and other interface boards.

Recently Sun Microsystems developed the SPOT platform [9] that attempts to overcome the challenges that currently inhibit development of tiny sensing devices. The device is built upon the IEEE 802.15.4 standard. The device runs the Squawk Java Virtual Machine (VM) without an underlying OS. The novelty is that this VM acts as both operating system and software application platform, thus allowing programming of the devices in the Java Micro Edition platform and allowing all the standard Java development tools to be used to create sensing applications.

We anticipate that in future sensing applications several types of sensor networks will coexist in the same structure alongside traditional processors. For example, embedded wired sensors and wireless sensor nodes will provide complementary functionality for Internet servers to process and present. These differences also imply different capabilities, operational range and requirements in resources, thus may also require applying different protocols in order to be more efficient. Such heterogeneous designs have been proven to be able to make a sensor network more computationally powerful and energy efficient [18] and result in improved performance [16]. The idea has been applied in many existing deployments [22], and is an important research direction for future sensor network architecture.

Developing applications to run on such a diverse set of devices, however, is a daunting task. Coordinating multiple languages, especially very different ones like hardware and software languages, is awkward and error-prone. Additionally, implementing communication mechanisms between different devices unnecessarily increases development time. Simulating such a system is complicated because of the need to coordinate compilers and simulators, often with very different interfaces, options, and fidelities.

Auto-pipe [12] is a toolset developed to address these difficulties. Auto-Pipe applications are expressed using a data flow coordination language. The applications may then be compiled and mapped onto complex sets of devices, simulated, and optimized. In [28], a system that simulates a complete sensor network is presented that focuses on heterogeneous systems comprised of simple (base level) sensor devices that are compatible with TinyOS [25] and more powerful Intel Stargate devices [8] that are used as intermediate gateways. The system consists of two major components for simulating the two classes of devices, respectively. A similar system is presented in [15] that supports the simulation,

emulation and deployment of heterogeneous sensor network systems and applications that are based on TinyOS. This framework uses TOSSIM to emulate motes and EmStar [14] to emulate "microservers" (a general term for platforms like Stargate). The authors employ a wrapper library to glue the two simulation systems together. All applications must be recompiled and linked to the EmStar library if they are to be emulated by the system.

1.3 Our contribution and justification

In this paper we present ADAPT (*Advanced Distributed Algorithms Platform and Testbed*), a heterogeneous environment, for the implementation, simulation, and testing of any distributed algorithm. ADAPT focuses on implementing distributed algorithmic ideas developed either for wired or for mobile wireless network systems. It is an evolution of the DAP system [11]. The design goals of the ADAPT environment are:

- to allow the user to develop the algorithm in a common, well-known programming language,
- to allow the same code to be used both for simulation and for real-world deployment,
- to support heterogeneous simulation environments,
- to be open and extensible for new application domains,
- to be of substantial help to the application designer by providing means to monitor the execution of the system and to extract detailed statistical information.

These goals reflect observations on the state of the embedded systems field. The field of embedded systems is witnessing three important trends. One is the availability at reasonable prices of powerful devices with sufficient performance characteristics to carry out meaningful tasks instead of being used as "dumb" sensors. Furthermore, this has resulted in the widespread deployment of such devices in many forms (e.g., smartphones). The networking capabilities of these devices make a lot of interesting applications possible. Lastly, although tools for programming these devices are available, these tools differ depending on the embedded platform (e.g., nesC for the TinyOS, J2ME for SunSPOT, etc.). The integration of these powerful embedded devices, possibly together with conventional computers acting as gateways or information servers, can produce very exciting applications. However, integration is very challenging due to the enormous diversity of the domain: the entities involved have different performance characteristics, networking capabilities and use different programming languages. A simulation environment capable to encompass this diversity is needed and we believe that ADAPT can fulfill this role.

Many systems (e.g., `ns`) are too generic and incorporate much more detail than it is needed for distributed algorithm simulation. In ADAPT all the less relevant lower levels are abstracted in a Topology domain. In such a simple but powerful model it is easier to capture some important aspects of a distributed environment such as asynchrony and failures of computing entities or communication channels. The user is free to customize every aspect of the simulation environment according to his own needs by setting the properties of every simulation object to constant or *variable* values (by selecting a random variable implementing some probability distribution). Furthermore, the user may define *scenarios*, which are series of actions (like node failures, interrupts, environmental conditions etc.) to be executed at specific points (also random variables, if needed) during the simulation, thus allowing vigorous testing of particular situations. In addition to the simplicity of the model, ADAPT provides a graphical topology and scenario editor for constructing the required environment.

Another disadvantage of the majority of the available systems is that they require the developer to code his algorithm in a language and a style that are specific for the system. In this case, the programmer is faced with a steep learning curve and there is a lot of wasted effort until he is able to utilize the full potential of the system. If, in addition, the language is inefficient (e.g., an interpreted language) the development/testing cycle becomes too long and the developer cannot assess easily the quality of his algorithm. In ADAPT the developer can program in a natural style (as in any other program) using `C++`, Java or any other language and the corresponding array of available tools. In fact ADAPT allows to implement the components of the application using different languages that are simulated in a unified environment. Furthermore, there is another interesting perspective: after an algorithm has been evaluated using ADAPT, the same code can be deployed on the *real* distributed environment, without serious modifications and without the use of the simulator, just by linking with libraries we provide.

ADAPT uses a very modular architecture where every simulated process is actually implemented as a "real", in the operating system sense, process. Each process communicates through the ADAPT *client* library to a central core, called *Engine* that manages the simulation and provides the relevant functionality. The communication between the core and the processes is done with a well-defined neutral protocol specified in XML. Beyond the basic functionality for developing and simulating distributed algorithms, ADAPT supports the creation of customized environments and complex simulation scenarios involving dynamic events (e.g., node failures, obstruction of node movement, etc.).

The use of ADAPT is rather intuitive and does not distract the developer from his main task, that is, the design, development and testing of the application.

2. THE ARCHITECTURE OF ADAPT

The main difference between ADAPT and similar systems is that every simulated process is actually run by ADAPT as a different "real" (in the operating system sense) process. The design decision of separating the core from the processes may add some amount of complexity to the system. The gain however is that it introduces some crucial advantages:

1. The developer can code each process without having to follow any specific programming model (e.g., event-driven).
2. It allows distributed execution of the simulation itself. This can be achieved using a number of technologies such as grid or overlay computing.
3. It provides a straightforward way to adapt the system to new simulation domains and technologies by writing new libraries to handle the communication with the core. If changes to the communication protocol

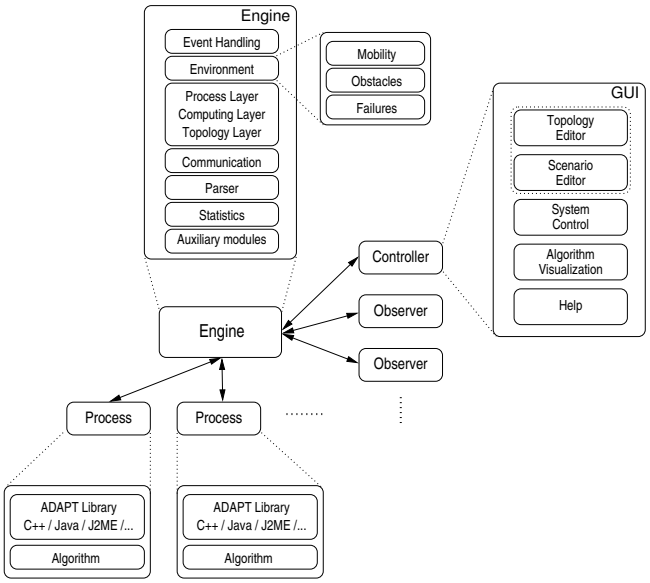

Figure 1: The architecture of ADAPT

are required, the architecture of ADAPT ensures that these will be localized and simple to implement.

4. A process may block without affecting the others. This observation is important: under this model it is easy to implement the *blocking receive* directive, a common challenge for distributed simulators.

On the basis of this design choice, ADAPT consists mainly of three components (see Fig. 1).

The *Engine* is the component that actually simulates the distributed system. It handles the simulation set-up, the processing of simulation events, the gathering of statistics, the communication with the graphical user interface, etc.

The *Graphical User Interface (GUI)* is used by the user to set initialization parameters, to control the simulation and generally to monitor its execution in real time.

The third component is the *ADAPT library*. The library is linked to every executable that a developer produces and allows it to communicate with the simulator engine. The ADAPT library implements all the directives that are needed for the implementation of a distributed algorithm (e.g., send / receive). We currently provide two versions of the ADAPT library: one implemented in `C++` and another in `Java`.

An analysis of the components of ADAPT follows.

3. THE SIMULATOR ENGINE

The Engine is the component that performs the core of the simulation. It holds the characteristics of the simulation entities (the simulated environment) and uses them to manage the interaction with the processes and the GUI.

The Engine follows the discrete event simulation model, that is a model where the system can change its state at only a countable number of points in time, and where these changes are represented by well-defined events. Thus the central module of the Engine is an event queue handler. Any action in the system, such as received communication from the simulated processes, is transformed into an appropriate event with a specific timestamp. The queue handler executes the events at the approriate time. The events have complete control over the simulation environment, thus implementation of complex events is possible (such as events altering the topology of the simulation).

An *action management* framework complements the event handling. The Engine has to manage communication coming from different sources (the simulated processes and the GUI). This communication is varied and may represent, e.g., simulated message transmissions, commands from the GUI to alter the simulation environment, etc. Each communication is encapsulated in an *action*. Upon execution, an action may generate events (e.g., a change in the value of a node property) or act directly on the state of the simulation (e.g., by pausing or resuming it).

The way the simulated environment is encapsulated in the Engine code makes the implementation of new types of actions and events very straightforward. Also, since the communication between the Engine and the other entities is specified in well-defined XML, the communication aspect of implementing new types of events can be handled easily. This is another point that contributes to the flexibility of ADAPT.

The Engine also gathers metrics and statistics about the simulation. Some types of metrics, such as the count of transmitted messages, are built in the platform. Additionally the ADAPT client library provides the programmer with directives for logging extra metrics that can later be extracted and examined.

3.1 Modeling of the environment

The most important feature of the engine is the way the simulated environment is modeled. Instead of representing the full detail of the environment (a situation that would place an unnecessary burden on the developer), its whole complexity is abstracted in the following layers:

Topology layer. It describes the location on which the computing nodes reside. The layer consists of topology nodes, representing positions, that may be connected.

Computing layer. It represents the computing nodes of the simulated system. Each computing node resides on a topology node and may move depending on the simulated environment. Computing nodes may have connections between them.

Process layer. It consists of the processes that make up the simulated algorithm. Each process is running on top of a computing node. Depending on the simulated environment a process may be able to alter the characteristics of lower levels (e.g., cause its computing node to move to another topology node).

The user is free to assign the exact meaning on the elements of each layer in relation to the application domain being simulated. For example, in a wireless sensor network the connections between topology nodes may represent paths that the sensors can use to move, or model communication obstacles. *Any* environment element may have properties associated to it (e.g., local temperature for topology nodes, speed for computing nodes). Properties can be constant or follow random distributions. These properties are available to the processes through the client library.

The environment may be static or may evolve over time according to user scenarios or in response to actions originating from processes. The user may tune the properties in order to achieve the required level of simulation detail.

The modeling and its dynamic nature make ADAPT capable of simulating a wide range of networks. The description of the environment and the scenarion is done in well-defined XML, so the user can benefit from third-party XML-aware tools for constructing complex simulated environments.

3.2 Defining Scenarios

The engine has the functionality to support user-defined scenarios, that is, a series of (simulation) actions (like node failures, interrupts, etc.) to be executed at a specified point in time during the simulation. The main use of scenarios is to allow repetitive simulation of particular situations. It is technically possible that actions are sent to the engine at any moment. Indeed, since some actions can also be triggered by user actions on the GUI (like node failures), actions related to scenarios use the same protocol as the ones coming from the GUI. However, the user may also group a series of actions into a scenario file that can be loaded by the engine. In all the actions, time, interval, duration and probability values can be specified to be random.

In addition to this user-defined approach, ADAPT implements functionalities that allow to model complex environments where the application will be simulated. The engine is supported by the following three domains.

Mobility domain. It encapsulates the required functionality for implementing different mobility patterns. The domain may control the movement of the nodes in a simulation or allow the ADAPT library to set the mobility pattern. Encapsulating mobility context and mobility behavior in different classes allows each node to use a different mobility model that can be selected at runtime.

Obstacles domain. It implements the framework for simulating obstacles that affect the area where the network nodes will move and communicate. The domain supports two types of obstacles: physical obstacles and communication obstacles. The physical obstacles prevent the physical presence of the nodes while communication obstacles cause disruption to wireless communication media. Since a physical obstacle may affect the movement of the mobile nodes, the domain communicates the geometric structures that represent an obstacle to the Mobility domain.

The actual simulation of an obstacle model is implemented in a separate sub-module which holds the state representing a single obstacle. An interesting feature of this architecture is that it allows the simultaneous simulation of multiple (and possibly different) obstacle models. Also note that since obstacles may evolve over time, an obstacle sub-module may interface to the action management framework and recalculate its state. Essentially this means that obstacles are not necessarily present throughout the duration of a simulation experiment, but may appear in a random or deterministic fashion for a period of time and then disappear. ADAPT already defines a wide range of obstacle models (e.g., rectangular shaped, circular, crescent, ring shaped, stripe shaped etc.) and it is easy to extend it by defining other obstacle models. This allows operating auxiliary obstacle simulators that advance in parallel to the primary network simulation.

Failures domain. It simulates failures that affect the network nodes. It supports two types of failures: node based and location based. The node based failures affect specific network nodes while location based failures affect all network nodes that are positioned within a particular region of the simulated area. Node failures constitute only halting errors, not Byzantine errors. Since failures may also result from the appearance of a physical obstacle, the domain observes the state of the Obstacle domain. This integration of the two domains further improves the realism of the simulator.

The actual simulation of a failure model is done in a way similar to the *Obstacles Domain*, and is implemented by a failure sub-module. ADAPT already defines a variety of failure models (e.g., random uniform, non-uniform etc.) and it is easy to extend by defining other failure models. As in the case with obstacle models, it allows the simulation of multiple (and possibly different) failure models. Thus we can operate auxiliary failure simulators in parallel to the primary network simulation.

Finally, it is possible to combine ADAPT with external tools that provide information about the environment. For example, the ADAPT Engine tracks the location of nodes and can query the external tool about environmental conditions (e.g., temperature or location accessibility) when such information in needed (for the previous example, when a process wants a readout from a simulated temperature sensor or wants to move to a new location resp.).

3.3 Auxiliary modules

In addition to the above, there are a few modules for a manifold of small but important tasks.

Statistics. Gathers the most commonly requested statistics from a distributed algorithm, such as number of messages sent/received, node/channel failures, message delay, etc. Also provides a mechanism that allows the user to easily define and display his own (additional) statistics that depend on the particular algorithm being implemented (e.g., memory usage per node). The module is available to all the event handling routines and to the processes through the ADAPT library.

Recorder. Used to record the whole simulation run and replay it later by running only the engine. The recorded information can be stored and used for further analysis.

Logger. The Logger is a low-level module that stores detailed information about the simulation run. The module is available to all the other components and processes and can be used for debugging or detailed analysis.

4. THE CLIENT LIBRARY

The client library is the part that handles the integration of user programs with the ADAPT Engine. The library provides all the needed directives for implementing distributed algorithms, such as directives for sending and receiving messages (blocking or non-blocking). It also exposes the functionality for interacting with the environment with directives for node movement, getting and setting properties, etc. In addition it provides directives related to the simulation, such as changing the colors of nodes and channels in their representation on the graphical user interface.

By design, the library is relatively simple since it offloads all processing to the Engine. This simplicity has the advantage of not imposing any unnecessary load on the programmer or the execution time of a program linked to the library. Depending on the application domain some directives of the library may not be needed (e.g., node movement), thus reducing the load even more.

The library can be ported to more programming languages or application domains in a straightforward way. Since the Engine is handling all the processing, the programmer wish-

ing to extend the functionality of the library has only to write the appropriate action and event as mentioned earlier. All the communication is done in simple XML, so the client library needs only to format and send an appropriate XML message. We are currently investigating ways to generate the relevant code automatically given the description of the functionality in XML.

Currently ADAPT provides a client library in two languages, `C++` and `Java`. `C++` was chosen as it is a well-known general purpose language. The `C++` library may be used e.g., for developing applications for embedded devices running a reasonably complete linux operating system or some smartphone models. The `Java` version of the library is available in two flavors. The flavor using the Java 2 Standard edition of the language may be used for applications intended to run on conventional hardware while the one using the Java 2 Micro edition can be used for applications targeting smartphones or the SunSPOT platform.

The processes participating in the simulation need not all be using the same client library. This makes it possible to simulate heterogeneous networks. We believe that is is feasible to mix "real" devices with simulated ones through an application acting as a proxy and we are currently investigating this possibility.

5. THE GRAPHICAL USER INTERFACE

ADAPT provides also a graphical user interface for monitoring the simulation in *real-time*. The GUI is a separate executable from the Engine and communicates with it over the network. The GUI can connect and disconnect at any time. It is thus possible to use it to setup a possibly long-running simulation on a fast server computer, disconnect, and then connect periodically to check on its progress. Multiple GUIs may be connected simultaneously.

The GUI includes a topology and scenario editor that can be used to specify in a user-friendly way all aspects of the simulation environment. The user has control over the simulation execution and can pause it and resume it at any point. The user may also, through the GUI, alter the simulation enviroment by changing the properties and status of nodes, or transmitting updated scenarios to the Engine.

As ADAPT may be extended for application domains that we could not have anticipated in advance, it is possible that the needs of the user for monitoring the simulation outgrow the capabilities of the GUI. The communication between the GUI and the Engine is done using an XML protocol that can be extended. On the Engine side this communication is handled from specific action objects (as mentioned earlier). If a programmer developing a new GUI needs more functionality than currently provided by the Engine, she needs to write only new appropriate actions. The modular architecture of ADAPT ensures that such code changes are localized.

6. SUNSPOT INTEGRATION

The SPOTWorld application supplied by SUN gives the user an easy way to emulate SPOT devices and run simple MIDlets on them (Java applications targeting the Micro Edition virtual machine). Through the SPOTWorld GUI the user can interact with the sensor panel of the devices, changing the values of various sensors. This approach allows the user to write and test the code in the SPOTWorld suite and finaly deploy it at the real SPOT without making any

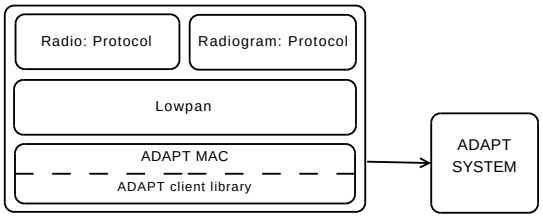

Figure 2: The emulator's radiostack

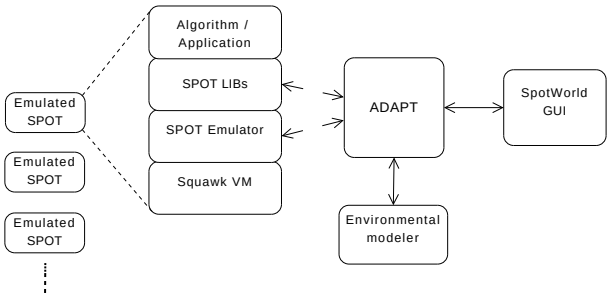

Figure 3: SPOTWorld emulator connection to ADAPT

change to his source code. The tool is very helpful for developing applications and evaluating the performance of the components at the device level. However, at the current version the emulator lacks complete topology networking. The developer can only simulate networks where all the devices are within transmission range of each other. This is a major drawback when dealing with sensing applications that cover a wide area. To overcome this problem, we integrate the SunSPOT emulator with the ADAPT system.

The SunSPOT emulator supports radio communication between the SPOT nodes through a MAC layer using multicast sockets (called `socketMAC`). In this implementation all the nodes are one hop away form each other. To achieve more complex network topology we have modified the `socketMAC` code to forward all radiogram packets through the ADAPT system using its java client library. The radiostack of the emulated spots is shown in figure 2.

As part of the initialization of the emulator's MAC layer we make an instance of the `AdaptSystem.java` which connects and registers with the ADAPT engine. Then all the radiogram packets that arrive at the mac layer from higher protocols are passed to the adapt system and vice versa. The MAC layer is also responsible for generating ack packets when it receives a data packet and for informing the lowpan layer of transmission failures when it does not receive the proper acknowledgment. In our implementation we use a feature of the adapt communication protocol to tag the packets according to their type (broadcast, unicast, ack, etc.). Using this policy we can easily extract useful information about the network traffic.

In order to run an application in a number of emulated SunSPOTs, ADAPT is responsible to start a Squawk VM for each application specifying the jar file that contains it. The Squawk engine first runs the SPOT emulator which deploys the jar and starts the execution of the application. Additionally the emulator will setup a connection with the ADAPT system to feed the SPOT various environment properties

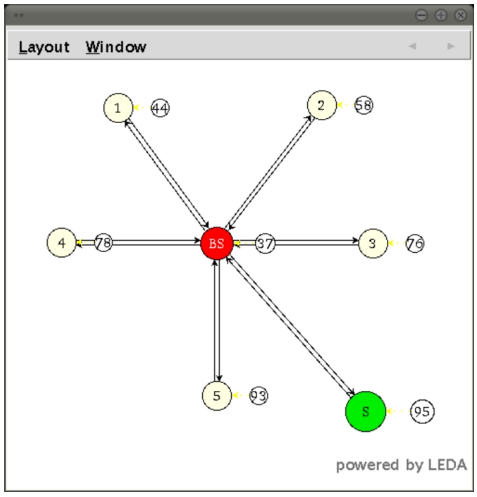

Figure 4: A simple star topology including a base station and a server. Each computing node has a process attached to it.

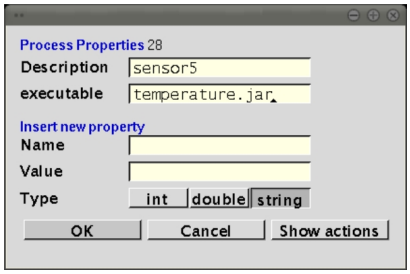

Figure 5: Specifying the executable file for a process. Additional properties can be added.

and update the status of its sensors values accordingly. Furthermore it will send back any change of its sensor panel made by the application (values of output pins/leds etc.). ADAPT can forward this information to the SPOTworld GUI so that it can display in real-time the reading of the sensors to the user and give her the option to change on the fly the values from the graphical user interface provided by SUN. The above architecture is shown in figure 3.

Using a Squawk VM for each application of the user is a trade-off. The drawbacks are mainly a performance penalty and increased memory consumption. However the advantage is that this design does not require any modifications to the user's code that will finaly run in the real SPOTs. Hence any application that was developed and tested in our simulation environment can be deployed in the real SPOTs and have the same behavior. Furthermore, preliminary simulations have shown that the impact in the simulation performance is acceptable and scales linearly with the node count.

7. AN EXAMPLE APPLICATION

We consider an application involving the deployment of several SunSPOT devices for gathering temperature measurements. The measurements are transmitted to a SunSPOT base station that uploads them to a central server (where they could be used for e.g., central air conditioning control).

First we must construct the topology. We will use a star

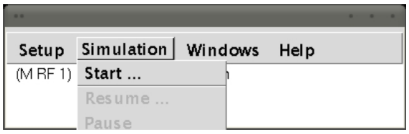

Figure 6: Ready to start the simulation.

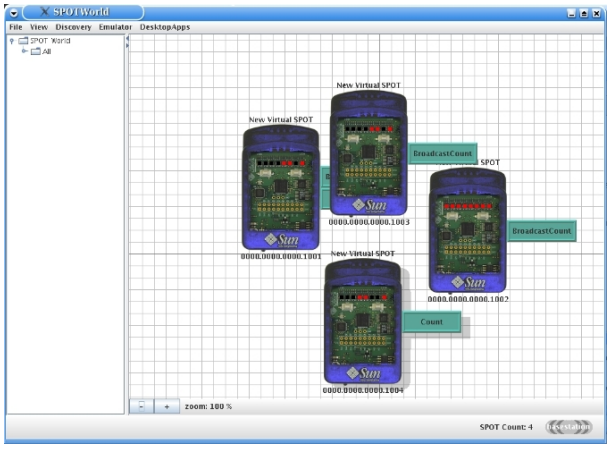

Figure 7: The SunSPOTworld tool.

topology with the SunSPOT base station at the center (the topology is simplified for the example). The topology is shown in figure 4. The figure depicts the computing nodes and the associated processes running on them. For each process we must specify as a property the executable to be run along with other properties that may be needed. This can be seen in figure 5. After the simulation environment is specified, some basic checks on the consistency of the specification are made. The user is now ready to start the simulation by issuing the apropriate command to the ADAPT system, as can be seen in figure 6. As the simulation is running the user may follow the progress through the SPOTworld tool provided by SUN and modified to communicate with ADAPT, as shown in figure 7.

8. CONCLUSIONS

We have presented the architecture and the design of the main components of the Advanced Distributed Algorithms Platform and Testbed (ADAPT), and discussed several distinct features along with their implementation and functionality, as well as extensions upon which we are currently working. We also report on the integration of ADAPT to the SPOT platform and how to extend the functionality of the development and evaluation tools provided by Sun microsystems. The main strengths of ADAPT are:

- The user can program in multiple languages (e.g., `C++`, `Java`) of his choice making the programs easily and efficiently ported to a real distributed environment.
- It enables the simulation of heterogeneous systems of devices of different capabilities and networking technologies.
- Scenarios can be defined for repetetive simulation of particular situations.

- It allows the execution of the simulation to be distributed among several hosts (if required) and to exploit distributed computing technologies such as grids and overlay computers.
- It can merge simulation with emulation and with real-world devices.
- A user is able to control, monitor, and visualize the simulation of his algorithm through a Graphical User Interface.
- Many users can simultaneously monitor and visualize (through a lighter version of the GUI) the simulation of some algorithm from remote locations.

Currently, we are in the process of organizing ADAPT in a easy-to-deploy package. This will also include extensive documentation about the architecture and use of our platform. Moreover, we will resolve license-related issues, so as to make our framework publicly available to the community. In the future, we plan to refine the architecture of our system, improve its genericity and enrich the library of implemented modules with more realistic models. Additionally we are developing tools that will make extending the platform simpler, by auto-generating code based on specifications given by the user. We plan to develop the integration with the SunSPOT platform further and provide a simulation environment for all aspects of the SPOTs (including sensors and movement). We are also investigating ways to move the simulation/emulation barrier at will, letting the user decide between a simulation run that is accurate but slower or more abstract but faster.

Further information about the ADAPT system is available in the webpage of the project [10].

9. REFERENCES

[1] BTnodes. http://www.btnode.ethz.ch/.

[2] Coalesenses GmbH. http://www.coalesenses.com/.

[3] Crossbow Technologies. http://www.xbow.com/.

[4] The Georgia Tech Network Simulator (GTNetS). URL: http://www.ece.gatech.edu/research/labs /MANIACS/GTNetS/.

[5] OPNET modeller, OPNET technologies, inc. http://www.opnet.com.

[6] ScatterWeb GmbH. http://www.scatterweb.com.

[7] SOS embedded operating system. https://projects.nesl.ucla.edu/public/sos-2x/doc/index.html.

[8] Stargate. http://platformx.sourceforge.net/.

[9] Sun^{TM}SPOT. http://www.sunspotworld.com/.

[10] ADAPT. http://ru1.cti.gr/projects/adapt.

[11] I. Chatzigiannakis, A. Kinalis, A. S. Poulakidas, G. Prasinos, and C. D. Zaroliagis. Dap: A generic platform for the simulation of distributed algorithms. In *Annual Simulation Symposium*, pages 167–177. IEEE Computer Society, 2004.

[12] S. Gayen, E. J. Tyson, M. A. Franklin, R. D. Chamberlain, and P. Crowley. X-sim: A federated heterogeneous simulation environment. In *10th High Performance Embedded Computing Workshop*, pages 75–76, 2006.

[13] B. Gerkey, R. Vaughan, and A. Howard. The player/stage project: Tools for multi-robot and distributed sensor systems, 2003.

[14] L. Girod, J. Elson, A. Cerpa, T. Stathopoulos, N. Ramanathan, and D. Estrin. Emstar: A software environment for developing and deploying wireless sensor networks. In *USENIX Tech. Conf.*, 2004.

[15] L. Girod, T. Stathopoulos, N. Ramanathan, J. Elson, D. Estrin, E. Osterweil, and T. Schoellhammer. A system for simulation, emulation, and deployment of heterogeneous sensor networks. In *SenSys '04: Proceedings of the 2nd international conference on Embedded networked sensor systems*, pages 201–213, New York, NY, USA, 2004. ACM.

[16] Exploratory research: Heterogeneous sensor networks. *Intel Technology Journal: Research & Development at Intel*, 2004. http://www. intel. com/research/exploratory/heterogeneous. htm.

[17] A. Kröller, D. Pfisterer, C. Buschmann, S. P. Fekete, and S. Fischer. Shawn: A new approach to simulating wireless sensor networks. In *Design, Analysis, and Simulation of Distributed Systems (DASD05)*, pages 117–124, 2005.

[18] R. Kumar, V. Tsiatsis, and M. B. Srivastava. Computation hierarchy for in-network processing. In *WSNA '03: Proceedings of the 2nd ACM international conference on Wireless sensor networks and applications*, pages 68–77, 2003.

[19] S. Kurkowski, T. Camp, and M. Colagrosso. MANET simulation studies: The incredibles. *Mobile Computing and Communications Review*, 9(4):50–61, 2006.

[20] M. Lacage and T. R. Henderson. Yet another network simulator. In *WNS2 '06: Proceeding from the 2006 workshop on ns-2: the IP network simulator*, page 12, New York, NY, USA, 2006. ACM Press.

[21] P. Levis, N. Lee, M. Welsh, and D. Culler. Tossim: accurate and scalable simulation of entire tinyos applications. In *SenSys '03: Proceedings of the 1st international conference on Embedded networked sensor systems*, pages 126–137, New York, NY, USA, 2003. ACM Press.

[22] A. Mainwaring, D. Culler, J. Polastre, R. Szewczyk, and J. Anderson. Wireless sensor networks for habitat monitoring. In *WSNA '02: Proceedings of the 1st ACM international workshop on Wireless sensor networks and applications*, pages 88–97, New York, NY, USA, 2002. ACM.

[23] NS2. http://www.isi.edu/nsnam/ns/.

[24] OMNeT++. http://www.omnetpp.org/.

[25] TinyOS. http://www. tinyos. net/.

[26] B. Titzer, D. Lee, and J. Palsberg. Avrora: scalable sensor network simulation with precise timing. In *4th International Symposium on Information Processing in Sensor Networks (IPSN)*, pages 477–482, 2005.

[27] S.-Y. Wang and Y.-B. Lin. NCTUns network simulation and emulation for wireless resource management: Research articles. *Wirel. Commun. Mob. Comput.*, 5(8):899–916, 2005.

[28] Y. Wen, S. Gurun, N. Chohan, R. Wolski, and C. Krintz. Accurate and scalable simulation of network of heterogeneous sensor devices. *J. Signal Process. Syst.*, 50(2):115–136, 2008.