KanseiGenie: Software infrastructure for resource management and programmability of wireless sensor network fabrics

Mukundan Sridharan, Wenjie Zeng, William Leal, Xi Ju, Rajiv Ramnath, Hongwei Zhang, and Anish Arora

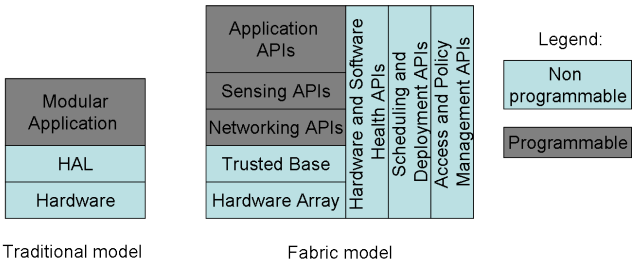

Fig. 1. Traditional network model and fabric model

***Abstract*—This chapter describes an architecture for slicing, virtualizing, and federating wireless sensor network (WSN) resources. The architecture, which we call KanseiGenie, allows users—be they sensing/networking researchers or application developers—to specify and acquire node and network resources as well as sensor data resources within one or more facilities for launching their programs. It also includes server side measurement and management support for user programs, as well as client side support for experiment composition and control. We illustrate KanseiGenie architectural concepts in terms of a current realization of KanseiGenie that serves WSN testbeds and application-centric fabrics at The Ohio State University and at Wayne State University.**

I. Introduction

DEPLOYED wireless sensor networks (WSN) have typically been both small-scale and focused on a particular application such as environmental monitoring or intrusion detection. However, recent advances in platform and protocol design now permit city scale WSNs that can be deployed in such a way that new, unanticipated sensing applications can be accommodated by the network. This lets developers focus more on leveraging existing network resources and less on individual nodes.

Network abstractions for WSN development include APIs for scheduling tasks and monitoring system health as well as for in-the-field programming of applications, network components, and sensing components. As a result, WSN deployments have in several cases morphed from application-specific custom solutions to "WSN fabrics that may be customized and reused in the field. In some cases, these fabrics support and manage the concurrent operation of multiple applications. Figure 1 compares the traditional WSN model with the emerging fabric model of WSNs.

Why programmable WSN fabrics? A primary use case of programmable WSN fabrics is that of testbeds. High-fidelity validation of performance sensitive applications often mandates rigorous end-to-end regression testing at scale. Application developers need to evaluate candidate sensing logics and network protocols in the context of diverse realistic field conditions. This can be done in the field, but in many cases locating the testbed in the field is inconvenient, so field conditions can be emulated by collecting data from relevant field environments and injecting those data sets into the testbed. Application developers also need to configure and tune system performance to meet application requirements. System developers, in contrast, can use testbeds to gain insight and concept validation by examining phenomena such as link asymmetry, interference, jamming behavior, node failure, and the like.

Sensing platforms that focus on people and on communities are another important use case for WSN fabrics. It is now economically feasible to deploy or technically feasible to leverage a number of connected devices across campus, community, and city spaces. In several cases, the cost of the facility is shared by multiple organizations and individuals; consider, for example, a community of handheld users who are willing to share information sensed via their privately owned devices. Assuming a programmable WSN fabric with support for these devices, hitherto unanticipated applications, missions, and campaigns can be launched upon demand. The existence of multiple sensor modalities and the benefits of fusion can be exploited, and programs can be refined based on data obtained from the field. Not least at all, non-expert clients and application developers can access and use fabric resources and data, while the responsibility of managing and maintaining the network will remain with expert domain owners.

GENI. The Global Environment for Network Innovation project [1] concretely illustrates an architecture where a WSN fabric is a key component. GENI is a next-generation experimental network research infrastructure currently in its prototyping phase. It includes support for control and programming of resources spanning facilities with next-generation fiber optics and switches, novel high-speed routers, city-wide experimental urban radio networks, high-end computational clusters, and sensor grids. It intends to support large numbers of individuals and potentially large and simultaneous experiments with extensive instrumentation designed to make it easy to collect, analyze, and share real measurements and to test load conditions that match those of current or projected

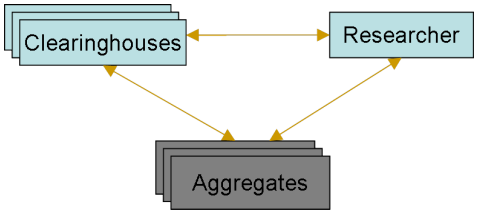

Fig. 2. GENI Federated Fabric overview

internet usage. To this end, it is characterized by the following features:

- Programmability - researchers may download software into GENI-compatible nodes to control how they behave
- Slice-based experimentation - each GENI experiment will be on a designated slice that consists of an interconnected set of reserved resources on platforms in diverse locations. Researchers, represented by slices in the GENI context, will remotely discover, reserve, configure, program, debug, operate, manage, and teardown distributed systems established across parts of the GENI suite.
- Virtualization - when feasible, researchers can run experiments concurrently on the same set of resources as if each experiment were running alone.
- Federation - different resources are owned and operated by different organizations.

Figure 2 depicts the GENI architecture from a usage perspective. In a nutshell, GENI consists of three entities: researchers, clearinghouses and the resource aggregates. A researcher queries the clearinghouse for the set of available resources at one or more aggregates and requests reservations for those that she requires.

Users and aggregates in GENI establish trust relationships with GENI clearinghouses. Aggregates and users authenticate themselves via the clearinghouse. The clearinghouse keeps track of the authenticated users, resource aggregates, slices, and reservations. Each resource provider may be associated with its own clearinghouse but there are also central GENI clearinghouses for federated discovery and management of all resources owned by participating organizations. GENI also relies on a standard for all entities to describe the underlying resource. This resource description language serves as the glue for the three entities because all interactions involve some description of resource, be it a physical resource such as routers and clusters or a logical resource such as CPU time or wireless frequency.

KanseiGenie. KanseiGenie is a GENI-compatible architecture that focuses on WSN fabric resource aggegates. GENI aggregates that are instances of this architecture include the Kansei [3] and the PeopleNet [8] WSN fabrics at The Ohio State University and NetEye [6] at Wayne State University. Since the effort to create and deploy a WSN fabric can be high, KanseiGenie installer packages are being developed to enable rapid porting of GENI-compatible programmability to other resource aggregates. The modular design of the KanseiGenie software suite lets testbed and fabric developers to customize the package for any combination of sensor arrays already supported by KanseiGenie or new sensor arrays.

Goals of this chapter. In this chapter, we overview requirements of next generation sensing infrastructures and motivate the various elements of KanseiGenie architecture for WSN fabrics. We then describe how the architecture can be extended and how it supports integration more broadly with other programmable networks.

II. Features of Sensing Fabrics

A Fabric is an independent, decoupled, programmable network, capable of sensing, storing, and communicating some physical phenomena [14]. Examples include networks of motion sensors, surveillance cameras, netted microphones, and so on.

Depending on the policy or platform ability, a fabric need not in general share state or cooperate with other fabrics. A fabric provides different services depending on its policy and hardware capability. Services are provided in the form of application programming interfaces (APIs). We classify the services as horizontal or vertical. Horizontal services are generic services that act as building blocks for more complex services. Vertical services are domain specific services that are usually optimized for specific application goals. Standardizing vertical services is desirable, so that applications can be readily composed and ported across fabrics geared to support a particular application domain. In general, a fabric need not make guarantees about its quality of service, delivering its results based only on a "best-effort".

A. Generic Services

We identify four types of generic services that WSN fabrics should provide:

- Resource management services: help researchers discover, reserve, and configure the resource for experimentations.
- Experiment management services: provide basic data communication and control between experiments.
- Operation and management services: enable administrators to manage the resources.
- Instrumentation and measurement services: enable the fabric to make measurements of physical phenomena, store the measurements and make them securely available.

1) Resource Management: To run an experiment, a researcher needs to discover, reserve, and configure the resource. Resource are allocated to a slice consisting of some sliver(s) of the fabric(s) that will serve as the infrastructure in which she runs experiments. For consistency, we follow the definitions of researcher, slice and sliver from GENI [1]. We distinguish the experiment creators and the experiments participants by calling the creators "researchers". The participants can be end-users or other researchers building their experiments on top of long running experiments. A slice is an empty container into which experiments can be instantiated and to which researchers and resources may be bound. All resources, be it physical or logical, are abstracted as components, such as Linux machines, or a network switch, or the like. When possible, a component

should be able to share its resource among multiple slices. A subset of the fabric occupied by a slice is called a sliver, which is isolated from other slivers. Only researchers bound to a slice are eligible to run experiments on it and a slice can only utilize the resource (slivers) bound to it. Resource management features include at least the following and possibly others.

- Resource publishing
- Resource discovery
- Resource reservation
- Resource configuration

For short-term experiments, resource discovery, reservation, and configuration may be performed at the beginning of the experiment in a one-shot fashion, assuming the underlying environment is relatively static in a short time window. However, for long-term experiments, resource management has to be an on-going process to adapt to network changes. For example, nodes involved in the experiment could crash or more suitable nodes might join the network.

Resource Publishing. A fabric shares its resources by publishing information about some subset to a clearinghouse. A clearinghouse may be held by some well-known site or the fabric itself. To promote utilization, a fabric can publish the same set of resources to multiple clearinghouses. This could result in multiple reservation requests to the same resource; in the end, the fabric decides which reservation(s) to honor, if any. Note that uncoordinated clearinghouses could make inconsistent resource allocations resulting in deadlock or live-lock situations. Global clearinghouses hierarchy and interaction architecture, as well as clearinghouse and resource provider policy should explicitly address this problem.

Resource Discovery. Sensor fabrics provide heterogeneous types of resources with diverse sensors, storage space, and computing and communication capabilities. Resource discovery is two-fold. First, resource providers must be able to accurately describe the resource, the associated attributes and the relationships between the resources. Second, researchers need to search for the resource they need with descriptions at different levels of details. Central to this discovery service is a resource description language for both the provider and the researcher. The resource request provided by the researcher can be concrete, such as which physical node and router should be selected, or abstract, such as a request for a 100 by 100 grid of fully connected wireless sensor nodes. In this case the discovery service has to map the request onto a set of physical resources by finding out the most suitable set of resources to fulfill the request.

Resource Reservation. Once a researcher discovers the desired resource, she will request the resource directly from the resource provider or from a third party clearinghouse to which the resource is delegated by the provider. Both the set of available resources and the permitted operations on it will vary according to the researcher's privileges. Note that if a researcher reserve the resource from a third party clearinghouse instead of directly from the provider, what the researcher has is a promise of the resource rather than a set of allocated resource. The reserved resource is allocated to the researcher's slice only after the researcher claims it from the provider. The success of the resource claim depends on the resource availability at the instant when the claim is issued as well as the provider's local policy.

Resource Configuration. The reserved resource slice needs to be configured in the beginning to meet the application's specification. Configurations range from software and run-time environment setups to setting transmission power or network topology. The resource configuration service needs to expose for each device the set of configurable parameters and the values these parameters can take on. Eliminating redundancy and performing other optimizations could be important. For example, if different experiments running on the same device require the same software library, this would result in wasted storage if duplicates of the library are installed.

2) Experiment Interaction: Experiments in wireless sensor networks take on many forms. Some run without human intervention whereas some adapt to human inputs and new sensing needs. Some run for months while some are short. We identify a set of features for experiment interaction as a basis for standardizing a common set of ways that researchers interact with their deployed experiments.

Debugging and Logging. In some cases, a network is so resource poor (in terms of memory, bandwidth and reliability) that is difficult or impossible to exfiltrate debugging data. But when it is possible to do so, standard services that permit the experimenter to pause part of the experiment, inspect state, and single-step the code should be provided. Logging is a related capability, which, when the resources permit, provides output of the experiment and can give after-the-fact debugging information. Typically a WSN application involves tens to hundreds of nodes, so the logging service should provide a central point of access to all the logging data.

Exfiltration and Visualization. Exfiltration, either in push or pull mode, allows researchers to collect the application output, possibly aggregated or transformed as instructed by users. Advanced exfiltration patterns such as publish-subscribe should supported. Standard visualizations of data exfiltrated by an experiment, such as a node map with connectivity information, should be provided, along with the option for the researcher to provide application-specific visualizations.

Data Injection. Data injection takes on two forms. Compile-time injection is derived from some injection file provided by the researcher that specifies when to inject what into which device, so the content of the injection is already determined at compile-time. Run-time injection is a part of client-side software that provides a way for the researcher to generate the data and inject it into the desired devices at run-time.

Key and Frequency Change. Long-running experiments evolve over time. To ensure security and reduce interference, a researcher may change the shared or public key or the communication frequency of the experiment every now and then.

Pause and Resume. As intuitive as it sounds, the difficulty to provide pause and resume services varies dramatically depending on the semantics of an experiment. The service boils down to the problem of taking a snapshot of the global system state and then reestablishing it later. Ensuring consistency of the system state after a pause just before a resume is a matter

of research interest.

Move. The move service is an extension on top of the "pause and resume" service. It enables the researcher to continue the experiment even when the resource allocated to experiment's slice becomes unavailable. A researcher can pause her experiment, remove it from the current resource, acquire and configure other available resource for the slice, and finally migrate and resume the experiment.

Experiment Composition. Sometimes the fabric owner might not want to directly run User executables on the fabric for security or other reasons. Experiments on such fabrics can be viewed as black boxes that take certain end-user input, such as sensor parameters, to compute certain outputs. On such fabric the designer might want to provide experiment composition libraries, which can be statically or dynamically recomposed to satisfy the Users needs.Also to promote the reuse of existing WSN applications, fabrics should provide a way to redirect the output from existing fabric applications to the input of another experiment, possibly the User application on another fabric. The redirected output can be viewed as a data-stream resource. In other words, the User rather than utilizing the physical resource of the fabric, uses configurable virtual (data) resources for experimentation.

Batching. This service enables a researcher to schedule a series of experiments, possibly with a number of controlled parameters. Instead of scheduling and configuring these experiments individually, the researcher simply give the ranges that each parameter will iterate through and the step size between consecutive iterations. Such batching service is especially useful when a researcher is trying to find out the set of optimal parameter for her protocol or to study different parameters' impacts on protocol performance.

3) Management and Operations Services: **Operational Global View.** In many cases, managing the fabrics requires a global operational view of the fabric or multiple federated fabrics. This service provides a portal through which researchers can find operational information such as node health and resource usage statistics. Since sensor nodes are much less robust than PC class computing devices, a user must know which set of sensor nodes worked correctly during the experiment in order to interpret the results. Thus generic node health information such as whether a node and its communicating interfaces function correctly during an experiment should be made available by this service. Resource usage auditing is necessary to check whether the user abides by policies and does not abuse privileges. The auditing includes such things as CPU, network and storage consumption. However, for mote class devices, detailed usage information on a per node basis may be unavailable due to resource limitations.

Emergency Stop. When a misbehaving experiment is identified, the administrator must be able to either isolate or stop the experiment so that its negative impact on the network is minimized.

Resource Usage Policy. It must be possible to limit the privileged operations that an experiment may invoke over the underlying fabric. Example privileges include the right to access certain sensors and the right of read and write fabric states. Both compile-time and run-time support should be provided for enforcing resource usage policies. For the run-time case, a administrator should be able to grant or revoke a slice's privileges the set of reserved resource.

Reliable Configuration. The fidelity of the experiment results depends on how well the actual run-time configuration of the experiment satisfies the researcher's specification. If they are different, it will help to filter out such abnormalities by providing a service that specifies the set of resource that are correctly configured. The complexity of providing this service can be increased if the configuration of the resource is performed in a wireless environment where the process is subject to more interference and thus has a higher failure rate. For example, wireless sensor nodes need to be programmed before the experiment can be run on them. In many cases, the sensing program is transferred to the sensor nodes in wireless channels. The fabric should provide some service to make sure that every sensor node is programmed correctly.

4) Instrumentation and Measurement Services: These services will support multiple, concurrent, diverse experiments from physical to application layer that require measurements. Inexperienced researchers can build their sensing applications by writing simple scripts on top of these services instead of learning a complex programming language, such as nesC, to write her sensing programs. Ideally, each measurement should carry the time and space information for future analysis. Also, given the nature of sensor network, each experiment can potentially generate huge amount of measurements. As a result, services for measurement storage, aggregation, and query are also required. Another critical requirement is that the instrumentation and measurements should be isolated from the execution of the experiments so that results of the experiments are not affected.

Traffic Monitoring. Traffic monitoring in the context of WSN includes both application traffic and interfering traffic. One unique property of wireless sensing fabrics is the complex interference phenomenon in their communications. Noise and interference can have a substantial impact on the experiment result. Therefore, collecting noise and interference data is necessary for the researchers to correctly interpret their experiment results.

Sensing. Wireless sensor nodes are equipped with multiple sensors, each of which capable of measuring different physical phenomenons with different accuracies. The sensing service should allow for controlling existing senors as well as adding new ones. Usually, wireless sensor nodes have very limited local storage. Thus, storage of sensing data should be explicitly addressed by the sensing service.

B. Domain specific services

In the previous section we described four classes of generic (horizontal) services that consist of basic building blocks for most wireless sensing fabrics. In this section, we shift our attention to domain specific or vertical services, which are tailored to the specific requirements of a given application. In the following section, we give two examples of vertical API, one of a search service designed in the security context and the other a rapid program development API for the KanseiGenie testbed.

1) Search API for security networks: In security networks (such as a intrusion detection network), the designer might want to provide the end-user with a flexible Search interface [14] that can search for objects of interest and can re-task the network as necessary. These objects can be various types of targets (such as humans, animals, cars, security agents) or sensor and network data objects (such as nodes that have failed or have low remaining battery life). These searches can be temporary (one-shot) or persistent (returns data at periodic intervals). While such an interface would be useful, it would be specific only to a this kind of network and hence we call it a vertical API. The key to providing such a search API is to design the sensor fabric with the service, which will interpret the user queries and re-task the network with appropriate programs or parameters

2) DESAL for KanseiGenie: DESAL [13] is a state based programming language developed specifically for sensor networks that is convenient for rapid prototyping of applications for sensor networks. While DESAL is primarily a programming language, a platform-specific DESAL compiler can be viewed as a domain specific service. Current DESAL compilers produce programs in nesC [5], which can be complied and run on testbeds like KanseiGenie. Thus DESAL is a vertical API for KanseiGenie, which allows users to write in high-level DESAL code than in more laborious TinyOS/nesC.

III. KanseiGenie Architecture

A. The Fabric Model

As we have noted, the architecture for KanseiGenie is based on a the idea of a fabric, a general networking model that takes the view that the network designer is not responsible for writing applications but for providing a core set of services that can be used by the application developer to create applications. The fabric model is agnostic about whether the application actually runs inside or outside of the fabric. In fact, depending on the capabilities of the fabric, it can be a combination of both. The generic services exposed by the fabric are analogous to those of a modern operating system on a PC. The user interacts through well defined APIs and commands to develop sophisticated applications, while leaving the low level management functions to the OS.

The fabric model clearly separates the responsibilities of the network designer from that of application developer (who is less likely to be a sensor network expert). A fabric manager, called the Site Authority(SA), takes care of implementing the services and exposing the APIs to the User, who might access the fabric APIs through any of the available communication networks (even possibly through another fabric leased to the User). Figure 3 shows the fabric model interaction between the User, Site Authority and Clearinghouse. Users could access the fabric through a specially designed user portal that implements and automates a number of user functions like resource discovery, reservation, configuration and visualization. A Clearinghouse(CH) arbitrates the user access to the fabric and users might be able discover the User APIs and fabric resources from the CH. The CH could potentially reside anywhere (even on the same machine as that of the SA) so long as it can securely communicate with the Site Authority. A single CH could manage access to multiple sensor fabrics and a User could seamlessly compose an application using one or multiple sensor fabrics. This aspect of the fabric model is especially suitable for the emerging layered-sensing applications [4].

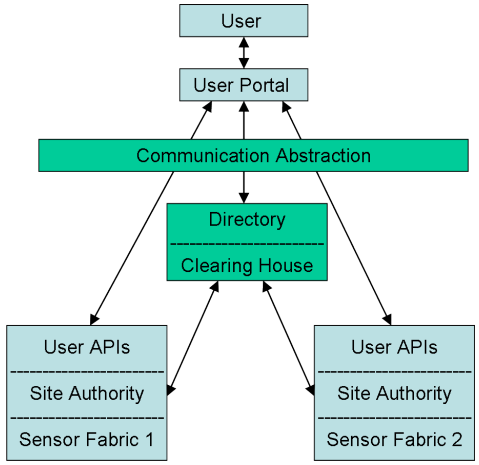

Fig. 3. Fabric Model

Depending on the policy or of platform ability, a fabric need not in general share state or cooperate with other fabrics. A fabric designer must attend to two key aspects; **Isolation.** The designer must make sure that, programs, queries and applications from one user do not interference with the applications of other users using the same fabric. Depending on the nature of the fabric this task might be hard, as in the case of sensor networks. Nevertheless, co-existence of multiple user applications is especially important in fabrics, which are full-fledged deployments where a production application is likely to be running at all times. While it is important not to disrupt the data from the production application, it is also important to let users test newer versions of the production application, short high value applications, and other applications

Sharing. In some fabrics, the designer might want to allow user applications to interact, while providing isolation if necessary. A classic example is the urban security networks, where a federal agency might want to access data generated by a state agency. But some times the trust relationships between two users is not straight forward when users who do not trust each other might want to interact for achieving mutually beneficial goal like that of a peer-to-peer file sharing service in the internet. The challenge of fabric designers is to evolve dynamic trust relationships which will enable such interaction, without compromising the security of user applications or the fabric itself.

B. KanseiGenie Architecture

The KanseiGenie is designed for a multi-substrate sensor testbed. Each of these substrates (also called arrays) can contain a number of sensor motes of the same type. This physical architecture is abstracted by a software architecture comprised of components and aggregates. Each sensor device is represented as a component that defines a uniform set of

interfaces for managing that sensor device. An aggregate contains a set of components of the same type and provides control over the set. It should provide at least the set of interfaces provided by the contained components and possibly other internal APIs needed for inter-component interactions. Given the collaborative nature of WSN applications, we believe that most users will interact with a sensor fabric through the aggregate interface rather than individual component interfaces. We hence forth denote the component and aggregate interface as Component Manager (CM) and Aggregate Manager (AM) respectively.

Three major entities exist in the KanseiGenie system (analogous to the fabric model), namely Site Authority (SA), Resource Broker (RB) (Clearinghouse in fabric model), and Reseacher Portal (RP) (User in the fabric model). RP is the software component representing the researchers in the KanseiGenie context. All entities interact with one another through well-defined APIS, divided into four logical planes, namely the resource plane, the experiment plane, the measurement plane, and the operation and management plane.

Resource brokers allocate resource from one or more site authorities and process requests from the resource users. A broker can also delegate a subset of its resource to some other broker(s).

Given that each sensor array is an aggregate, the KanseiGenie SA is conceptually an aggregate of Aggregates Managers (AAM) that provides access to all the arrays. The AAM provides an AM interface for each sensor array through parameterization. Externally, the AAM (i) administrates the usage of the resource provided by the site according to local resource management policies, (ii) provides the interface through which the SA advertises its shared resource to one or more authenticated brokers and, (iii) provides a programming interface through which researcher (using RP) can schedule, configure, deploy, monitor and analyze their experiments. Internally, the AAM provides mechanisms for inter-aggregate communications and coordination.

The RP is the software that interacts with the SA to run experiments on behalf of the researcher. It contains a suite of tools that simplifies the life cycle of an experiment from resource reservation to experiment cleanup. The RP could provide a GUI, command line or a raw programming interface depending on the targeted user. A GUI user interface should be the most user-friendly and and the easiest to use, though the kinds of interaction might be limited because of its graphical representation. Command line and programming interface would be for more experienced users that need more control and feasibility than the GUI can offer. For example, a researcher can write scripts and programs to run a batch of experiments with the provided command-line and programming interface that a GUI may not provide.

All these operations are based on trust relationships, and all entities must implement appropriate authentication and authorization mechanisms to support the trust establishments. For example, the SA may either directly authorize and authenticate resource users, or delegate such authorization and authentication to other trusted third party brokers. To invoke the AAM functions, a user will need to present forms of authentication and authorization during her interaction with the AAM. The Since all AM interfaces in the AAM are the same except for the parameter used to instantiate the AM interface for a specific array, it thus suffices to implement these APIs for a new sensor array to be supported by KanseiGenie.

C. GENI Extension to KanseiGenie

Figure 4 illustrates the KanseiGenie architecture in the GENI context. The KanseiGenie SA implements the interface for the aggregates. The SA interacts with the clearinghouse for resource publishing while the researcher interacts with clearinghouse to reserve resources. The researcher also interacts with SA to redeem their reservations and run experiments. The SA's interface is implemented as web-service for extensibility and compatibility reasons.

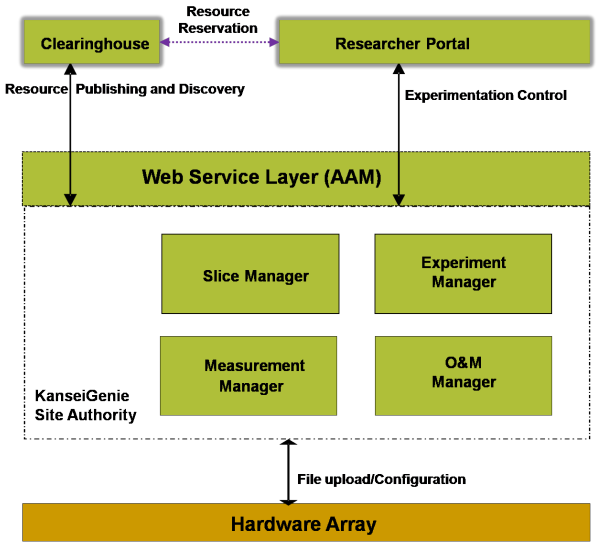

Fig. 4. KanseiGenie Architecture

KanseiGenie provides a web-based Researcher Portal implementation for easy access to its resource. A downloadable IDE-software (much like Ecllipse) version of the RP is under development and will be available for future users. The researcher portal provides the most common interaction patterns with the KanseiGenie substrate so that most researchers do not need to implement their client side software to schedule experiments in KanseiGenie.

KanseiGenie implements the Clearinghouse utilizing the ORCA resource management framework [7]. Utilizing a third party framework is motivated by the need of federated experimentation in the GENI. Although the interface for each entity is well-defined, using a common and stable resource management framework helps resolve many incompatibility issues and helps to decouple the concurrent evolutions of the SA and the broker.

D. Implementation of KanseiGenie

The KanseiGenie software suite consist of the three entities: the Researcher Portal software, the Orca-based resource management system and the KanseiGenie Site Authority.

1) Researcher Portal (RP): The Researcher Portal is implemented using the PHP programming language. The PHP web front-end interacts with the AAM through the web service layer. Such decoupling enables concurrent developments and evolvement of the RP and SA. The web-based RP abstracts the most common interactions with the testbed, that of one-time, short experiments. To conduct experiments that are long-running or repeating, the researcher can gain more fine-grained control by writing her own programs based on our web service API. The current RP portal allows users to interact with only one Site Authority. We are in the process of extending the RP to work with multiple Site Authorities in a federated environment.

2) Orca-based resource manangement system: The ORCA system consist of 3 Actors;

1) **Service Manager.** The Service Manager interacts with the user and gets the resource request, forwards it to the Broker and gets the lease for the resources. Once a lease is received, the Service Manager, forwards it to the Site Authority to redeem the lease.
2) **Site Authority.** The ORCA Site Authority keeps a inventory of all the resources that need to managed. It delegates these resources to one or more Brokers, which in turn lease the resources to users through the Service Manager.
3) **Broker:** The Broker keeps track of the resources delegated by various SA. It receives resource request from SM and if the resource is free, it leases the resource to the SM. A number of different allocation policies can be implemented using a policy plug-in.

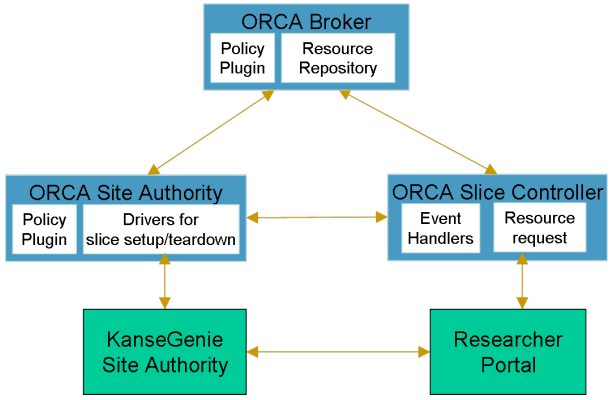

Fig. 5. KanseiGenie Integration with ORCA

To integrate ORCA for resource management, we modified the ORCA Service Manager to include an XML-RPC server that receives the resource requests from the RP. Similarly the ORCA SA was suitably modified to make web service calls to the KG AAM for experiment setup and tear down. Figure 5 shows this integration architecture.

Shared resource and trust relationships among the ORCA broker, SA, and RP are configured beforehand within ORCA.

3) KanseiGenie Site Authority: The KanseiGenie Site Authority in turn has three components, the Web Service Layer (WSL), the KG Aggregate of Aggregate Manager (KG AAM) and the individual Component Managers (CM) for each device type supported by KanseiGenie.

Web Service Layer. The WSL acts as a single point external interface for the KG Site Authority. The WSL layer provides a programmatic, standards-based interface for all the KG AAM APIs. We utilize the Enterprise Java Bean framework to wrap the four functional GENI planes. Each of the manager APIs are implemented as a SessionBean. We utilize the JBoss application server as the container for the EJBs partly because JBoss provides mechanisms for users to conveniently expose the interface of Session Beans as web services. Other reasons that motivated us to choose JBoss include the good support from the community, wide adoption, open-source licence, and stability.

KanseiGenie Aggregate of Aggregate Manager. The KG AAM implements the APIs for Kansei substrates. It keeps track of the overall resources available at a site, monitoring their health and their allocation to users. It also keeps track of individual experiments, their status, deployment and clean up. The status of the resources and experiments are stored in a MySQL database. A Schedular daemon (written in the Perl language) accomplishes experiment deployment, clean up and, after an experiment is complete, log files retrieval using the Substrate Manager of the individual substrates. The currently most of the generic APIs discussed in Section II are supported by the KanseiGenie AAM.

Component Manager. Each of the devices in Kansei has its own Manager (but for some primitive devices the Manager might be implemented on other more capable devices). The Component Manager implements the same APIs as that of the KG AAM and is responsible for executing the APIs on the individual devices. The logical collection of all the Managers of devices belonging to the same substrate form the Aggregate Manager of that substrate. Currently KG supports Stargates [9], TelosB [11], and XSMs [12]. The AMs for Imote2 [2] and SunSpots [10] are under development. The TelosBs and XSMs being primitive devices without a presistent operating system, their Managers are implemented on the Stargates. The CMs are implemented using Perl. A number of tools for user programming the motes and interaction with them are written in the Python and C programming languages.

E. KanseiGenie federation

The KanseiGenie software architecture is designed for sensor network federation. We discuss below the use cases and key issues in KanseiGenie federation.

1) Use cases: Although there are many use cases, key among them ore regression testing, multi-array experimentation, and resource sharing that is enabled by federation.

Regression testing. WSNs introduce complex dynamics and uncertainties in aspects such as wireless communication, sensing, and system reliability. Yet it is desirable to have predictable system behavior especially when WSNs are used to support mission-critical tasks such as safety control in alternative energy power grids. That is, it is desirable for system services and applications to behave according to certain

specifications in a wide range of systems and environmental settings.

Existing measurement studies in WSNs are mostly based on a single fabric such as Kansei and NetEye, which usually only represents a single system and environmental setting. To understand the predictability and sensitivity of WSN system services and applications, however, we need to evaluate their behavior in different systems and environmental settings. Seamlessly integrating different WSN fabrics together to provide a federated measurement infrastructure will enable us to experimentally evaluate system and application behavior in a wide range of systems and environmental settings, thus providing an essential tool for evaluating the predictability and sensitivity of WSN solutions.

Multi-array experimentation. Next-generation WSN applications are expected to involve different WSN substrates. These include the traditional, resource-constrained WSN platforms as well as the emerging, resource-rich WSN platforms such as the Imote2 and Stargate. Federation also enables experimentation with multiple WSN fabrics at the same time, thus enabling evaluation of WSN system services and applications that involve multiple WSN substrates.

Resource sharing. Federating different WSN fabrics together also enables sharing resources between different organizations. Resource sharing can enable system-wide optimization of resource usage, which will improve both resource utilization and user experience. Resource sharing also helps enable experiment predictability, predicting the behavior of a protocol or application in a target network based on its behavior in testbed. This is because federated WSN fabrics improve the probability of finding a network similar to the target network of a field deployment. Resource sharing is also expected to expedite the evolution of WSN applications by enabling more users to access the heterogeneous WSN fabrics at different organizations.

2) Key issues: To enable secure, effective WSN federation, we need to address issues in clearinghouse architecture, access control, resource discovery and allocation, as well as network stitching.

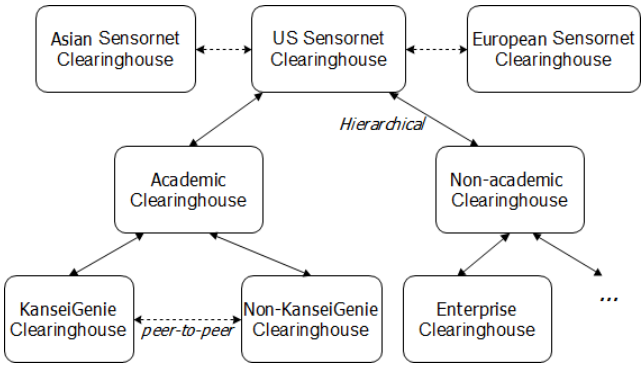

Fig. 6. Clearinghouse architecture in KanseiGenie federation

Clearinghouse architecture. For effective resource management among federated but autonomous WSN fabrics, we expect that the clearinghouses be organized in a hierarchical manner. As shown in Figure 6, for instance, several WSNs in the US may form one cluster, and this cluster interacts with a cluster of European WSNs. The exact form of the clearinghouse hierarchy depends on factors such as trust relations, resource management policies, and information consistency requirements. We are currently working with our domestic and international partners, including Los Alamos National Lab and India Institute of Science, to develop the conceptual framework for the federation hierarchy as well as the implementation mechanisms for supporting flexible adaptation of federation hierarchy as policy and technology evolve. This hierarchical federation architecture will be reflected by a hierarchy of clearinghouses in charge of the resource management within certain scope of the hierarchy.

Within the hierarchy of federated WSN fabrics, we also expect peer-to-peer interaction between different fabrics. For instance, the WSNs from the US academia may form a sub-cluster of the US-cluster, and these fabrics may share resources in a peer-to-peer manner without involving the top-level clearinghouse of the US cluster.

Access control. One basic security requirement of GENI is to only allow legitimate users to access their authorized resources. Therefore, one basic element of federated WSN fabrics is the infrastructure for authentication, authorization, and access control. For authentication, we can use the cryptographic authentication method via public key cryptography, and, in this context, adopt the hybrid trust model where the monopoly and anarchy models are tightly integrated. In this hybrid trust model, each fabric maintains its own certificate authority (CA) which manages the public keys for users directly associated with the fabric, and the distributed, PGP-style, anarchy trust model is used among different fabrics to facilitate flexible policies on federation and trust management. (Note that a fabric may also maintain an internal hierarchy of CAs depending on its local structure and policy.) Using this PKI, a fabric CA can vouch for the public key of every entity (e.g., a user or a software service such as a component manager) within its fabric, and every entity can verify the identity of every other entity through the public key certified by the local fabric-CA. The chain of trust formed across fabrics will enable authentication across fabrics. For instance, an entity B can trust the public key Key-A certified by a CA CA-A for another entity A in a different fabric Fabric-A, if the local CA CA-B of B trusts certificates issued by CA-A. The PKI will also be used in secure resource discovery and allocation.

For each legitimate user, the slice authority (SA) at her associated clearinghouse generates the necessary slice credential through which the user can request tickets to access resources of different fabrics. Based on their access control policies, the component managers (CMs) of the involved fabrics interact with the user to issue the ticket and later to allocate the related resources according to the authorized rights carried in the ticket.

Network stitching. After an experiment has acquired the resources from the federated KanseiGenie fabrics, one important task is to stitch together slices from different WSN fabrics. To

this end, the KanseiGenie research portal will coordinate with one or multiple clearinghouses to get tickets for accessing the allocated resources. Then the research portal will coordinate with the involved site authorities to set up the communication channels between slices in different fabrics, after which the experiment will have access to the connected slices within the KanseiGenie federation.

IV. KanseiGenie Customization and Usage

A. How to Customize KanseiGenie

KANSEIGENIE SA software uses a distributed architecture with hierarchical design. The KanseiGenie AAM implements the APIs of the four functional planes. The AAM redelegates the API calls to a specific Aggregate Manager depending on what substrate is being used for an experiemnt. The APIs of the Aggregate Manager are well defined and can be implemented on any platform/technology.

KanseiGenie currently supports the Stargate, XSM and TelosB sensor platforms. The AMs for Imote2 and SunSpots are under development. The architecture of KanseiGenie supports both a flat or a hierarchical arrangement of substrates and using platform neutral language like perl makes it customization quite easy. To use KanseiGenie to manage sensor subtrates already supported, an administrator only need to populated the MySQL database tables regarding the number and the physical topology of substrates. Also a testbed administrator can configure KanseiGenie for any combination of the already supported sensor platforms.

Customization of the KanseiGenie to a new sensor substrate involves three steps, one implementing the AM APIs for that substrate (either on those devices directly or another substrate, which can in turn control the new one being added). Two, updating the AAM resource and policy database about the new substrates topology and resources. Three, modifying and/or adding new GUI interfaces to the Researcher Portal to support the configuration parameters for the new platform.

B. Vertical APIs and Their Role in Customization

The KanseiGenie architecture supports additional APIs apart from the standardized (generic) APIs of the four functional planes. Basing the architecture on a Service Oriented Architecture (SOA) and dividing them into vertical (domain specific) and horizontal (generic) APIs provides a basis for customizing for different substrates. The vertical APIs provide a convenient way of providing APIs for specific application domains, while at the same time standardizing the APIs for a particular application domain.

Two of the customizations of the KanseiGenie architecture are the Peoplenet and the Search API for intrusion detection system, which we describe below.

PeopleNet. PeopleNet [8] is a mobility testbed at Ohio State university composed of about 35 cell phones and the Dreese building fabric. The Dreese building fabric provides services like the elevator localization, building temperature and light monitoring, along with a fabric for generic experimentation. The PeopleNet cell phone fabric supports APIs for instant messaging and buddy services (searching for buddies).

Search API. Search API [14] is a single API that provides the interface for an intrusion detection network. The single API lets the user to query the network for target objects as single shot queries or persistent queries. More over, the queries can be about real physical targets or logical data objects in the network and they can be confined to a geographical area in the network or the user can query the entire network.

The above two examples provides insight into the customization of the fabric architecture and how it can support multiple dissimilar fabrics. The flexibility of the architecture is because, we separate the horizontal APIs from the vertical APIs and because we leave the implementation of APIs to the specific Aggregate Managers, so the same API can be implemented differently by different substrates.

C. KanseiGenie Usage Step-by-step Run Through

The KanseiGenie Researcher Portal is designed to be an intuitive and easy way for a user to access the testbed resources. Here we give a short step-by-step run through of a typical usage scenario.

1) **Get access.** The first thing a user needs to do is to get access to the testbed resources. A user can do this by contacting the KanseiGenie administrator (say by email) or by getting a login from the GENI clearinghouse.
2) **Create a slice.** A user will first create one or more Slices(if a user wants to run more than one experiment concurrently she will need more than one Slice). A Slice represents the user inside the fabric. It is a logical container for the resources of the user.
3) **Choose the substrate.** The Portal displays all the different substrates available in the KanseiGenie federation. The user needs to decide which substrate she is going for this experiment.
4) **Upload the executable.** The user will next prepare the executable and/or scripts that needs to be tested for the particular substrate
5) **Create the resource list.** A user might want to test her program on a particular topology. The Portal provides a service that lets the user create any topology they want from the available nodes from a particular substrate.
6) **Get lease.** The user will next have to get a lease for the resources she wants to use. The Portal interacts with the Orca resource management system and gets the lease for the resources.
7) **Configure experiment.** Once the user has the lease, she needs to configure the experiment she wants to run on these resources. She will choose parameters such as the length of the experiment should run, which executable should be run, what logging and exfiltration service to use, what injection is required, etc.
8) **Run experiment.** Finally, once the configuration is done, the user can start the experiment from experiment Dashboard.
9) **Interact with experiment.** The Portal also provides services through which a user can interact with the experiment, while it is running. A user can inject pre-recorded sensor data into the Slice, view logs in real

time, visualize the network in realtime, view health data of resources. To enable most of these services, the user should specify/select the services in the configuration step.

10) **Download results.** Once an experiment is complete, the results and logs from the experiment are available for download from the Portal.

V. Evolving Research Issues in Next-generation Networks

In this section we will look at some of the emerging research issues in the fabric model.

A. Sensor fabric resource specifications

The sensor fabric resource specifications act as the language that all the entities in the architecture understand. It is important for the designers to come up with an ontology that is detailed enough for the users of the domain to take advantage of the fabric services and features, but should also be broad enough to enable interaction and joint experimentation with other programmable fabrics (such as other wireless networks and core networks).

Much of the complexity of sensor networks need to be embedded in resource specifications (RSpecs). Resource specifications will also be an extensible part of the federated KanseiGenie interface. As new resources and capabilities are added, these specifications will inevitably need to be extended. We expect the extensions to leverage a hierarchical name space. This will allow new communities that federate with KanseiGenie to extend the resource specification within their own partition of the name space, and components that offer specialized resources will similarly extend the resource specification in their own name space. Additionally, we need to consider the granularity of resource specification, which decides the level of details in resource description. In federated resource management, there is no unique solution and the implementation strategy is subject to both technical and administrative constraints. For instance, whether and how much information about resource properties should be maintained by clearinghouses will depend on the trust relations among the entities involved and may be encoded in resource specifications at different levels of granularity.

To enable reliability and predictability in experimentation, resource specification also needs to characterize precisely the reliability and predictability properties of sensornet testbeds, including the external interference from 802.11 networks, the stability of link properties, and nodes' failure characteristics in a testbed. Accordingly, an experiment will also use reliability- and predictability-oriented specification to specify its requirements on the allocated resources.

For the same experiment there may be different ways of specifying the actual resources needed. For an experiment requiring two TelosB motes and a link of 90% reliability connecting these two motes, for instance, we may define the resource specification to request two motes six meters away with the necessary power level for ensuring a 90% link reliability between these two motes, or we may define the resource specification to request any two motes connected by a link of 90% reliability. Both methods will give the user the desired resources, but the second method will allow for more flexibility in resource allocation and thus can improve overall system performance.

B. Resource discovery

For federated resource management, different clearinghouses need to share resource information with one another according to their local resource sharing policies. Two basic models of resource discovery are the push and pull models. In the push model, a clearinghouse periodically announces to its peering or upper-level clearinghouses the available resources at its associated fabrics that can be shared. In the pull model, a clearinghouse requests from its peers or upper-level clearinghouses their latest resource availability. We expect the pull model to be mainly used in an on-demand manner when a clearinghouse cannot find enough resources to satisfy a user request. The interaction between clearinghouses needs to be authenticated using, for instance, the PKI discussed earlier.

C. Resource allocation

We expect that federated WSN infrastructures will support a large number of users. Hence effective experiment scheduling will be critical in ensuring high system utilization and in improving user experience. Unlike scheduling computational tasks (e.g., in grid computing), scheduling wireless experiments introduces unique challenges due to the nature of wireless networking. For instance, the need for considering physical spatial distributions of resources such as sensor nodes affects how we should schedule experiments. To give an example, let's consider two fabrics S1 and S2 where both fabrics have 100 TelosB motes, but the motes are deployed as a 10 x 10 grid in S1 whereas the motes are deployed as a 5 x 20 grid in S2. Now suppose that we have two jobs J1 and J2 where J1 arrives earlier than J2, and J1 and J2 request a 5 x 10 and 5 x 12 grid respectively. If we only care for the number but not the spatial distribution of the requested motes, whether J1 is scheduled to run on S1 and S2 does not affect the schedulability of J2 while J1 is running. But spatial distribution of nodes do matter in wireless networks, and allocating J1 to S2 will prevent J2 from running concurrently, whereas allocating J1 to S1 will allow the concurrent execution of J2 by allocating it to S2, improving system utilization and reducing waiting time.

Wireless experiments in federated GENI may well use resources from multiple fabrics in a concurrent and/or evolutional manner. Scheduling concurrently-used resources from multiple fabrics is similar to scheduling resources within a single fabric even though we may need to consider the interconnections between fabrics. For experiments that use multiple fabrics in an evolutional manner, we can schedule their resource usage based on techniques such as "task clustering", where sequentially requested resources are clustered together and each cluster of requests is assigned to the same fabric to reduce coordination overhead and to maximize resource utilization. To reduce deadlock and contention, we need to

develop mechanisms so that an experiment can choose to inform the clearinghouse scheduler of its temporal resource requirement so that subsequent experiments do not use resources that may block previously scheduled experiments.

D. Data as Resource

One consequence of the fabric model is that the network is hidden behind a collection of user APIs and as long as the APIs are the same, a user can programmatically access it. In other words, it does not matter if APIs are provided by a huge sensor network or by a single PC: the user won't know the difference. Thus, a data base that can annotate and store results of experiments (or queries) and replay the data for similar future queries can now be viewed as a sensor resource. Alternately, a sensor network can be viewed as a source for a data stream and the user as a data transformation program. Under this unified view, a DataHub which can interpret quires and transform the stored data accordingly can fake a sensor fabric. Thus the fabric model provides a new unified model under which data (properly annotate and qualified) and sensing resources are inter-changeable and provides for inter-esting hybrid experimentation scenarios.

The architecture provides much research opportunities and challenges, as a number of questions need to answered before the architecture can be used beyond the most simplistic scenarios. Challenges include the following.

- How to automatically annotate and tag data coming from sensor networks to create a credible DataHub?
- It is common for the same experiment to produce multiple similar data sets in wireless networks. How does a user decide which dataset to use as representative of an experiment?
- Does the RSpec ontology need to be extended to represent data ?
- What range of queries can be answered with the current data? Should data be pre-processed to decided acceptable querries?

E. Network Virtualization

The fundamental aim of the fabric architecture is to virtualize and globalize the resources in a sensor network, so that in principle user anywhere in the world can request, reserve and use the resources. However, the more resource is virtualized, the less control the user has over it. Thus there is a trade-off between the level of access and virtualization. The challenge for modern network designers is to as much control (i.e., as low in the stack as possible) to the users, while retaining the ability to safely recover the resource and also making sure the resource might be shareable.

In a fabric multiple researchers will run their experiments concurrently on different subsets of an array of sensors of the same type. Usually, sensors are densely deployed over space. Such density provides the means for different experimenters to share the same geographical space and sensor array to conduct concurrent experiments that are subject to very similar, if not statistically identical, physical phenomenon. In such environments, interference is inherent between users due to the broadcast nature of the wireless communications; its effect is more prominent when the communicating devices are close to one another. The virtualization of wireless networks imposes a further challenge for the sensor fabric providers to ensure isolation between concurrently running experiments. Such interference isolation is usually achieved by careful frequency or time slot allocations. However, these solutions are quite primitive in nature and do not provide optimum network utilization. Even more important, these solutions are not suitable for sensing infrastructures where multiple applications from different users need to be run concurrently in a production mode.

Our recent research in this area using statistical multiplexing as the basis of virtualization is promising to provide better solutions, enabling much better network utilization and external noise isolation.

VI. Conclusion

In this chapter, we described the KanseiGenie software architecture for wireless sensor network fabrics. It enables slicing, virtualizaton, and federation in wireless sensor networks. We listed the features of next generation sensing infrastructures and illustrated how the KanseiGenie architecture meets the needs of such networks while also enabling collaboration with not only other sensor networks, but also with any programmable network in general, for example under the GENI framework.

We also described the benifits of basing the architecture on a network centric fabric model rather than on a node centric model. We gave a step-by-step run through of a typical usage scenario in KanseiGenie and illustrated how the software suite can be customized for domain specific applications using the Vertical APIs. Finally, we gave a brief overview of emerging research issues and opportunities in the area and outlined some preliminary solutions.

References

[1] Global environment for network innovation. http://www.geni.net.

[2] Intelmote2: High-performance wireless sensor network node. http://docs.tinyos.net/index.php/Imote2.

[3] Kansei wireless sensor testbed. http://kansei.cse.ohio-state.edu.

[4] Layered sensing. http://www.wpafb.af.mil/shared/media/document/AFD-080820-005.pdf.

[5] Nested c: A language for embedded sensors. http://www.tinyos.net.

[6] Neteye wireless sensor testbed. http://neteye.cs.wayne.edu.

[7] Open resource control architecture. https://geni-orca.renci.org/trac/wiki/.

[8] Peoplenet mobility testbed. http://peoplenet.cse.ohio-state.edu.

[9] Stargate gateway devices. http://blog.xbow.com/xblog/stargate_xscale_platform/.

[10] Sunspots: A java based sensor mote. http://www.sunspotworld.com/.

[11] Telosb sensor motes. http://blog.xbow.com/xblog/telosb/.
[12] Xsm: Xscale sensor motes. http://www.xbow.com/Products/Product_pdf_files/Wireless_pdf/MSP410CA_Datasheet.pdf.
[13] Anish Arora, Mohamed Gouda, Jason O. Hallstrom, Ted Herman, William M. Leal, and Nigamanth Sridhar. A state-based language for sensor-actuator networks. *SIGBED Rev.*, 4(3):25–30, 2007.
[14] Vinodkrishnan Kulathamani, Mukundan Sridharan, Anish Arora, and Rajiv Ramnath. Weave: An architecture for tailoring urban sensing applications across multiple sensor fabrics. MODUS, International Workshop on Mobile Devices and Urban Sensing, 2008.