2009 International Conference on Parallel Processing Workshops

SenSORCER: A Framework for Managing Sensor-Federated Networks

Sujit Bhosale
Computer Science, Texas Tech University
SORCER Research Group
Lubbock, USA
e-mail: sujit.bhosale@sorcersoft.org

Michael Sobolewski
Computer Science, Texas Tec*f*h University
SORCER Research Group
Lubbock, USA
e-mail: sobol@sorcersoft.org

Abstract*—Despite many technology advances, the limited computing power of sensors encumber them from taking part in service-oriented architectures. In recent years, the sensor-networking subject has been extensively studied, with the focus on efficient usage of energy in sensors and efficient usage of sensor node processing power. However, providing sensor nodes with the relevant computing power and network sensor management is seldom addressed. With the availability of federated meta-computing environments, sensors can participate in service oriented computing, by exporting sensor probe data with relevant processing as a network service. The framework introduced in this paper allows for building highly scalable and dynamic sensor networks, providing sensors with corresponding federated computing environment. A novel approach using the SORCER metacomputing model to integrate several different sensors and sensor networks into a dynamic service-oriented sensor network is presented. The framework makes use of provisioning mechanism, which autonomically allocates required resources to sensor services. Furthermore, the dynamically typed language, Groovy, provides the runtime computing mechanism involving variables of sensor services. The framework is generic and extensible and allows incorporating different standard and non-standard sensor technologies. It also helps expedite the development process for service-to-service applications involving sensor data.

Keywords-Sensor Networks; Service Oriented Computing; Metacomputing

I. INTRODUCTION

Recent advances in integrated circuit technology led to the construction of very capable and yet inexpensive sensors, radios, and processors. That makes it possible to have powerful spontaneously networked and mobile systems. No doubt, the next wave of networking includes sensors and controllers [6]. In recent years, sensor network topics have been increasingly studied and researched, especially in the electrical engineering and computer science areas, with the focus on efficient usage of energy in sensors and efficient usage of sensor node processing power. However, providing sensor nodes with the relevant computing power issue is largely unaddressed. A few frameworks proposed we will review in Section 3, but those lack forming composite sensors and autonomic provisioning of sensor services.

Our research deals with the flexible management approach to sensor network formation problems. With the availability of the Service-ORiented Computing EnviRonment (SORCER) [1] sensors can participate in the metacomputing environment by exporting not only a probe data but a sensor data aggregation module as a service and providing sensor management through a user friendly service UI [20]. We can further extend this approach by exporting multi-sensor handling module as a service in the SORCER environment and handling the network of elementary and aggregated sensors.

The rest of the paper is organized as follows: Section 2 gives the motivation behind the framework and Section 3 gives the overview of related work; Section 4 reviews the background technologies; Section 5 presents design of the framework; Section 6 gives the implementation and experimental details, and Section 7 presents the benefits of presented framework. Finally, paper concludes with final remarks in Section 8.

II. MOTIVATION

Current data-aggregation models for sensors have many limitations associated with them. An insight into these limitations will give us better understanding of the sensor-distributed system requirements.

1. Large header overhead of existing communication protocols for relatively small sensor data. Sensor networks, consisting of many low power, low capability devices that integrate sensing, computation, and wireless communication, pose a number of novel systems problems. They raise new challenges for efficient communication protocols [14, 15]. The data generated from a single sensor at any instance is very small. To transfer this small amount of data over the network, header overhead of the current IP protocol is relatively high. This makes it very difficult for many real time applications to continuously collect data directly from large number of individual sensors efficiently in their applications.
2. Static topology of sensor locations and data collection points. The static topology of sensor locations and data collection points limit the rapid sensor data collection and maintenance; e.g., in agricultural area, where the sensors are located at different locations on the farms for various

measurements, the data collection specialist has to collect the data from the sensors, directly visiting those places. He has to connect to the sensor externally and collect the readings. In adverse weather conditions, there are no solid tools available for him, which can give the status information of the sensor in place.

3. Non-standardized sensor technology. The current sensor technologies are lacking a common open standard. There have been attempts to specify standards for sensor technology, but the use of such standards is very limited by venders. One example of such standard is IEEE 1451 [16], which gives comprehensive specification for many different sensor technologies. However, the adaptation of this standard has been very limited. Thus, the best approach to sensor networking should be inclusive of various sensor technologies and general enough to switch between the technologies transparently.
4. No efficient method of handling growing number of sensors. As the sensor usage grows in the future, the amount of data that these sensors generate would be enormous. Currently, there is no solid framework available for handling such large amounts of data that flows from clients to servers predominantly (data flow reversal).
5. Lack of easy sensor data availability to metacomputing applications. Sensors because of their limited computing ability cannot take part in meta-computing environments as fully "fledged citizens". As we are transforming from static architectures towards the dynamic architectures the sensors are required to take part in spontaneous networking as well. Currently, no mechanism is available by which metacomputing applications can get sensor data on-the -fly.
6. No uniform data-aggregation interface availability. Many applications need to collect information from several heterogeneous sensors and sensor networks, which are physically deployed at different places to provide comprehensive services. To reduce the complexity of querying from heterogeneous sensor networks, a uniform data aggregation interface should be provided for sensor-based applications. Currently, there is no such interface provided by any of the available sensor-networking technologies.

Analyzing all the limitations above, we have concluded that, the SenSORCER framework for sensor-federated networking is needed to allow sensors to take part in the metacomputing environments, to cope with data flow reversal issues along with high sensor data processing, aggregation, and distribution capacity. Furthermore, it should be extensible and general enough to accommodate different standardized and non-standardized technologies, yet contain enough concrete building blocks that developers can use to their benefits.

III. Related work

A. *A framework for a distributed sensor network based on Jini technology* [6].

This is a Jini **Error! Reference source not found.** service-oriented framework for the development of a distributed sensor network. The architecture relies on a three level data clustering. The sensors are connected to Terminal Communication Interface (TCI) that is able to virtualize the access to different kinds of sensors therefore allowing all the sensors to be accessed through consistent interface. At the second level a Sensor Service Provider (SSP) is used to contact different TCIs and to collect their data arranging them in a more structured way. This allows the third level, represented by the Application Service Provider (ASP) to access already pre-processed data easy to manage in order to extract more sophisticated data representation. The ASP is the only point of access to the system so that it can enforce specific policies [6].

The use of Jini allows this framework to provide the service oriented architecture power to the sensors, but the framework can only be used, as a data collection method for the sensors. TCI registers with a lookup service, and takes part in the data communication process and also, it is the only component communicating with sensors. TCI is heavily dependent upon the sensor technology used. The fact that, TCI is burdened with the lot many responsibilities, make its use difficult in real-time applications, where the fast sensor value reporting is necessary. It also does not add power to the computation part of the sensors. On the other hand, the framework presented here, makes use of the SORCER environment for data processing, whose Federated Method Invocation (FMI) [1] is used as the dynamic access mechanism of sensor data and exertion-oriented programming [2].

The ASP is similar to a composite service provider (CSP) in SenSORCER, but CSP is more flexible than ASP, as ASP is only used for data processing part. The CSP on top of that, allows a client to decide on which sensor services to use, and what computation to be done on their data. In addition, use of the Rio framework [4] makes dynamic provisioning facility available in SenSORCER, which is not possible in the framework with ASP.

B. *A framework to simplify software development for sensor network applications*

This is a component-based framework, where components provide the functionality of single sensors, sensor nodes, and the whole sensor network [7]. The framework uses sun's surrogate architecture [12]. The aim is the separation of functional blocks in order to increase flexibility. These blocks include, Node-specific Operating System, Driver Layer that contains at least one sensor driver and several hardware drivers. Node-specific Operating System handles device specific tasks.

The host middleware is the superior software layer, which organizes the cooperation of distributed nodes in the network. Also, the Middleware Management layer handles other components, which can be implemented and exchanged according to a node's task [7].

The use of surrogate architecture, in the above framework is well justified. However, sensor has a very small amount of computing power, so making sensor a direct part of network is not an effective solution. As most of the sensors generate data at a very fast rate, the service provided by the single sensor should be capable of storing data to the local store. By using the surrogate architecture, the sensors can be used in network applications, but the effective use of such sensor node is questionable.

On the other hand, SenSORCER is independent of the communication protocol and driver specific code of the system. It generically wraps the code into sensor probe. The elementary sensor service provider explained in Chapter 3, makes use of sensor probe but is independent of sensor technology used. This way, all the legacy sensors and their protocols can be part of a sensor network by wrapping them without any changes to underlying codes.

IV. Related technologies

In this section, we introduce related terms and technologies, which define federated metacomputing and provide building blocks of the SenSORCER architecture.

A. Service Oriented Architecture (SOA)

In general SOA is a paradigm for organizing and utilizing distributed capabilities that may be under the control of different ownership domains. It provides a uniform means to offer, discover, interact with and use capabilities to produce desired effects consistent with measurable preconditions and expectations [3]. SenSORCER is a sensor-distributed system built with SOA object-oriented concepts and federated method invocation [1].

B. Jini

In Jini **Error! Reference source not found.**, a service is essentially a Java interface that is implemented as a remote object. Therefore, any object implementing multiple interfaces could be turned into a provider of multiple services. The Jini service-oriented architecture has a concept of dynamic discovery and join of services whereby services are registered on the network and discovered in real-time via a unicast or multicast protocols on the network.

Jini provides a registry called lookup service (LUS), which is a service registry that allows service requestors to locate needed services by object types (interfaces) and associated complementary attributes. During startup, a service provider registers its services with the LUS. Service requestors use LUSs to locate the services they are interested in. The LUS itself is discovered through the discovery protocols by issuing multicast or unicast requests, as well as by receiving multicast announcements. Service requestors and providers use the discovery protocols to locate LUSs. When the services first enter the SenSORCR network they receive a lease from a LUS for a specific time period that is renewed periodically by their service provider. If the service gets disabled then the lease is not renewed and the service is deregistered from the LUS and thus leaves the network. This mechanism of leasing keeps the sensor network healthy and robust. New services entering the network become available immediately from LUSs and the existing services that are disabled are automatically disposed from the sensor network.

C. Rio

The Rio provisioning framework [8] provides a model to dynamically instantiate, monitor and manage service components as described in a deployment descriptor called an Operational-String. The Operational-String provides context on service requirements, dependencies, associations, and operational parameters. Rio provisioning services additionally provide pluggable load distribution and resource utilization analysis mechanisms to effectively make use of resources on the network [5]. The Rio framework allows enabling following capabilities for the sensor services in SenSORCER environment:

- Dynamically adapt to addition and removal of sensor resource on the network.
- Running sensor service on the compute resource available in the network that matches required QoS.
- Fault tolerance achieved by dynamically allocating the service to a different compute node (cyber node), if the original node fails.

D. SORCER

SORCER [2] (Service-Oriented Computing EnviRonment) is a federated service-to-service (S2S) metacomputing environment that treats service providers as network objects with well-defined semantics of a federated service object-oriented architecture. It is based on Jini [4] semantics of services in the network and Jini programming model with explicit leases, distributed events, transactions, and discovery/join protocols. While Jini focuses on service management in a networked environment, SORCER focuses on metaprogramming (exertion-oriented programming) and the execution environment for exertions [2]. SORCER uses Jini discovery/join protocols to implement its *exertion-oriented architecture* (EOA) using *federated method invocation* [1], but hides all the low-level programming details of the Jini programming model.

In EOA, a service provider is an object that accepts remote messages from service requestors to execute a collaboration. These messages are called service exertions and describe *service (collaboration) data, operations* and collaboration's *control strategy.* An *exertion task* (or simply a *task*) is an elementary service request, a kind of an elementary instruction executed by a single service provider or a small-scale federation for the same service data. A composite exertion called an *exertion job* (or simply a *job*) is defined hierarchically in terms of tasks and other jobs, and thus isa kind of a federated procedure executed by a large-scale federation. The executing exertion is dynamically bound to all required and currently available service providers on the network. This collection of providers identified in runtime is called the *exertion* *federation*. The federation provides the implementation for the collaboration as specified by its exertion. When the federation is formed, each exertion's operation has its corresponding method (code) available on the network. Thus, the network *exerts* the collaboration with the help of the dynamically formed service federation. In other words, we send the request onto the network implicitly, not to a particular service provider explicitly.

The overlay network of service providers is called the *service grid* and an exertion federation is in fact a *virtual metacomputer*. The metainstruction set of the metacomputer consists of all operations offered by all service providers in the grid. Thus, an *exertion-oriented* (EO) program is composed of *metainstructions* with its own *control strategy* and a *service context* representing the metaprogram data. The service context describes the collaboration data that tasks and jobs work on. Each service provider offers services to other service peers on the object-oriented overlay network. These services are exposed *indirectly* by operations in well-known public remote interfaces and considered to be elementary (tasks) or compound (jobs) activities in EOA. Indirectly means here, that you cannot invoke any operation defined in provider's interface directly. These operations can be specified in the requestor's exertion only, and the exertion is passed by itself on to the relevant service provider via the top-level `Servicer` interface implemented by all service providers called *servicers*—service peers. Thus all service providers in EOA implement the

`service(Exertion, Transaction) : Exertion`

operation of the `Servicer` interface. When the servicer accepts its received exertion, then the exertion's operations can be invoked by the servicer itself, if the requestor is authorized to do so. Servicers do not have mutual associations prior to the execution of an exertion; they come together at runtime (federate) for a collaboration as defined by its exertion. In EOA requestors do not have to lookup for any network provider at all, they can submit an exertion, onto the network by calling

`Exertion.exert(Transaction : Exertion`

on the exertion. The `exert` operation will create a required federation that will run the collaboration as specified in the EO program and return the resulting exertion back to the exerting requestor. Since an exertion encapsulates everything needed (data, operations, and control strategy) for the collaboration, all results of the execution can be found in the returned exertion's service contexts.

Domain specific servicers within the federation, or task peers (*taskers*), execute task exertions. *Rendezvous* peers (jobbers and spacers) coordinate the execution of job exertions. Providers of the `Tasker`, `Jobber,` and `Spacer` type are three of SORCER main infrastructure servicers.

V. SenSORCER Architecture

A. The SenSORCER approach

SenSORCER is essentially a federated SOA based infrastructure, which allows dynamic networking between sensor wrapping (SORCER) services. The SenSORCER approach can be described in three steps: Measure, Compute, and Communicate (MC^2). A sensor probe would measure the sensor data using sensor-specific technology for sensor connectivity. A sensor service employs the probe to connect to sensors allowing them to take part in SOA. SORCER's exertion-oriented programming allows sensor services to participate in collaborations, specified by exertions; to carry out requested computing tasks. In addition, the dynamically typed language Groovy [16] makes it easy to provide custom compute-expressions involving sensor service variables at runtime.

The federated method invocation carries the collaborative communication with sensor services. The service requestor incorporates vales from sensor providers as parameters that are communicated to other providers in the collaboration via the exertion's service context. SORCER defined service-to-service (S2S) communication allows relevant services to federate dynamically and assist in execution of exertions [2] provided by the requestor. If for any reason, a particular sensor service is not available, the request can be passed on to the equivalent available service provider. The SORCER infrastructure treats sensor providers as peers that implement a common *SensorDataAccessor* interface. Here are basic terms used in the context of the presented framework:

- Sensor Node: The entity that can take part in sensor network, maintaining a single sensor
- Sensor Service: The service (interface), implementing the value reading facility, connecting the single sensor
- Sensor Subnet: The logical grouping of sensor nodes
- Sensor Network: The logical grouping of sensor nodes and sensor subnets, by a specialized composite service
- Network Management: The facility provided by the specialized façade service, to add and remove sensor nodes, subnets, and create dynamic grouping

B. Organizational architecture

SenSORCER is a component-based framework, where each component is the sensor provider with façade services as multiple entry points to the sensor network. The SenSORCER framework is illustrated in the form of the UML component diagram [17] in Fig. 1. The main components along with their communication interfaces are shown. There are three main components in the framework:

1. Elementary Sensor Services
2. Composite Sensor Services
3. SenSORCER Façade Services

An *Elementary Sensor Service* (ESP) is the basic building block of this framework. As shown in Fig. 1, a *Sensor Probe* is the only sensor dependent component of the framework. It contains sensor specific driver code, which is used to communicate with the sensor. This component is dependent on sensor specific protocol and underlying technology. Communication with any sensor has many aspects like, synchronization, timing constraints, communication protocol, data calibration, etc. Sensor probe is dependent on all of these aspects of the sensor, but hides these details from sensor service providers. As shown in Fig. 1, ESP makes use of the common *DataCollection* interface to connect to the sensor and read sensor data through sensor probe. The sensor values are available to service requestors via the common *SensorDataAccessor* interface.

In sensor network semantics, the ESP service plays the role of node in the logical sensor network. When this service is started up it registers itself with the Jini service registry. There can be many elementary sensor services available in the framework, with every ESP working independently, providing single sensor connectivity. However, ESP can be used to connect multiple sensors, if sensors have the ability to connect themselves with other sensors, collaborate, and make collected data available to ESP via its *DataCollection* interface.

The aggregate sensor provider, *Composite Sensor Provider* (CSP), plays two important roles in SenSORCER. First, being the aggregate, it composes both ESPs and CSPs, processes service requests, collects the sensor data from its component sensor services, and makes its values defined in terms of component values available via the *SensorDataAccessor* interface. Thus, it performs processing on the collected data and returns the calibrated composite result to service requestors. In the second role, as a child, CSP can be a part of another CSP and in that case provides the calibrated result to its parent CSP.

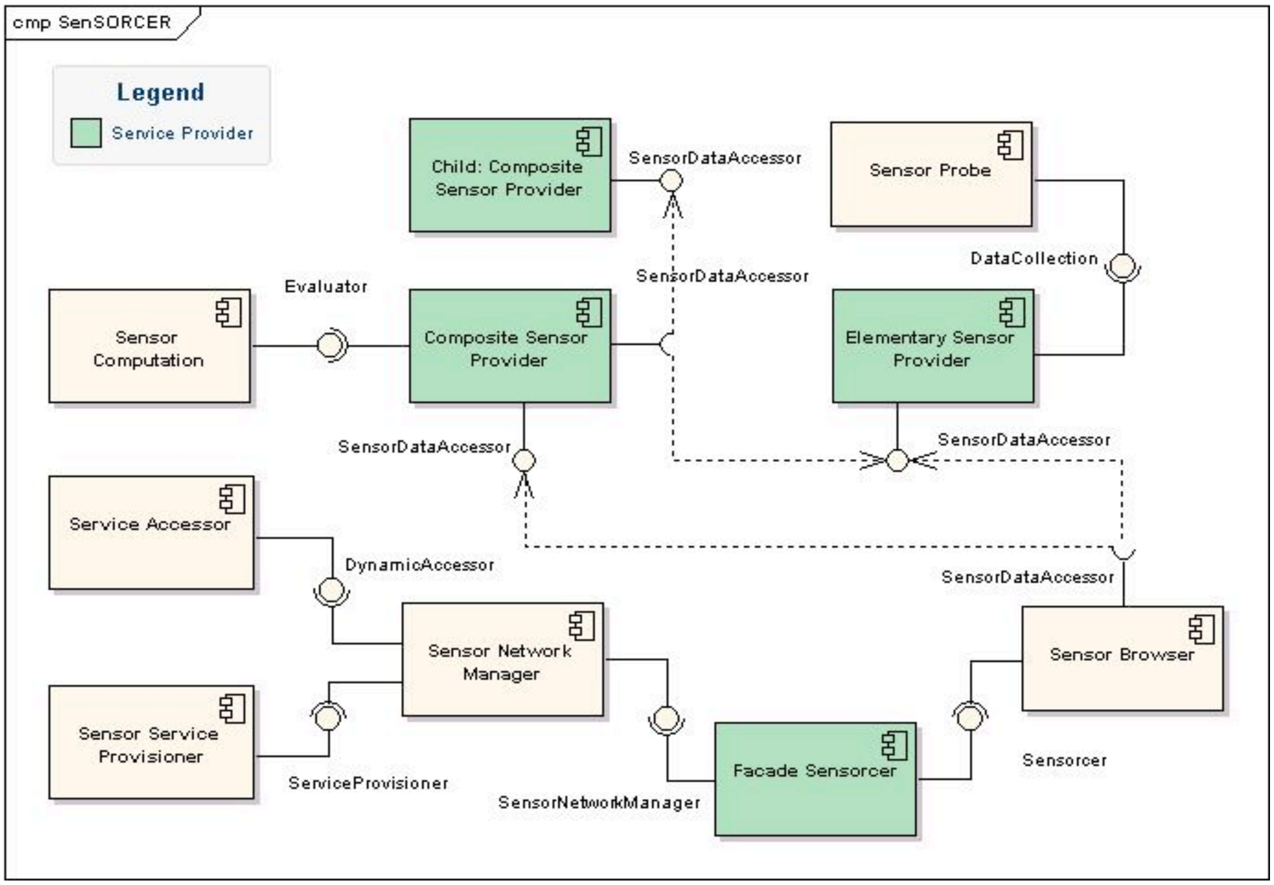


Figure 1. SenSORCER architecture - UML component diagram

CSP's ability to contain other CSPs along with ESPs makes logical sensor networking possible. Thus, the semantics of network management in SenSORCER is reduced to the management of a single CSP. The user of the system can add or remove sensor nodes from the network, by interacting and issuing requests to a single CSP. CSP management is sensor independent; one can change contained provider's implementation, and can use different sensors, without potentially affecting the CSP provider.

Sensor Computation provides capabilities of specifying required computing power to CSPs. It performs the user specified computation on to sensor data. The user can provide expressions, treating services as the variables inside the CSP expression. CSP replaces service variables with the actual values from component sensor services and then computes the expression as complex as required, which is then sent back to the requestor.

The user can interact with SenSORCER through a user agent attached to the *Sensorcer Façade* provider. When user wants to make use of any management functionality of the SenSORCER system, he opens zero-install user interface called a *Sensor Browser* in a Jini service browser, for example Inca X [18], as illustrated in Fig. 2. The design of the browser follows the MVC pattern [21]. Its model contains the data of the sensor network configuration, views display the data in appropriate format, and controller providers mapping between the network model and browser views.

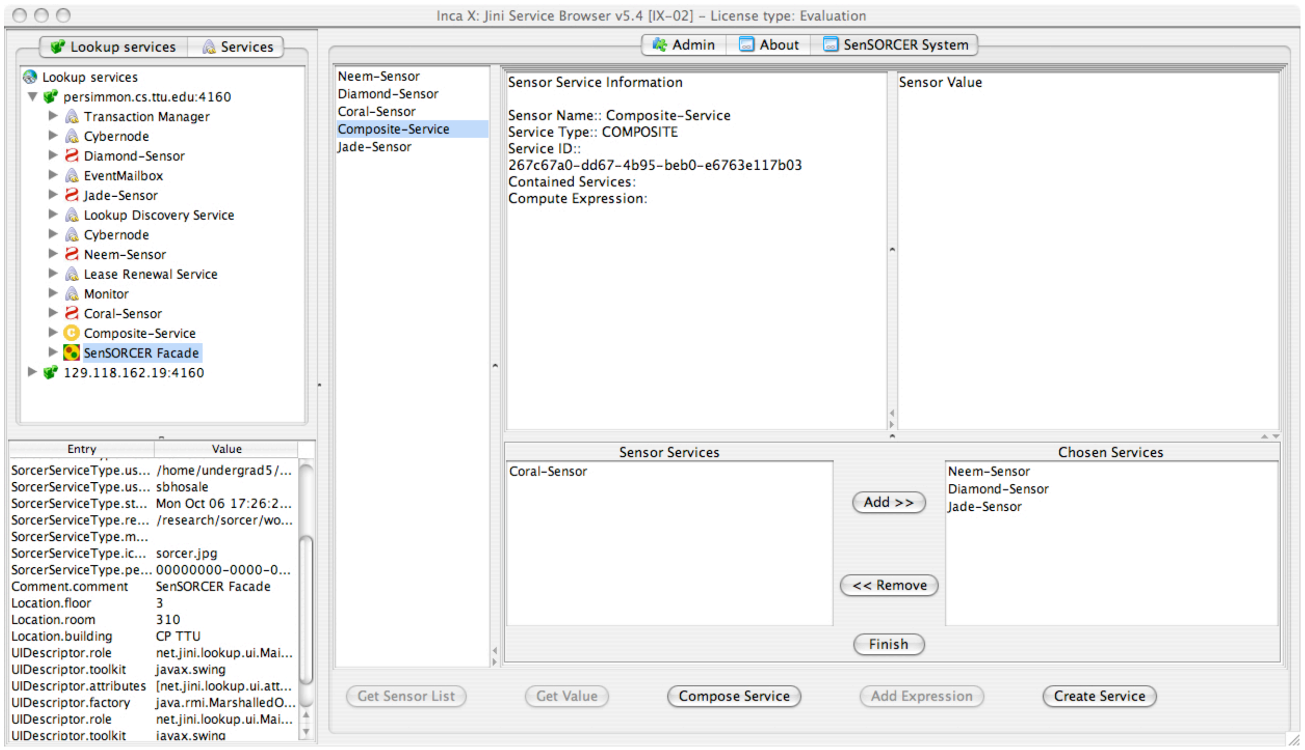
The *Sensorcer Façade* is the single entry point of the SenSORCER system. It provides a uniform access to the user through the *Sensor Browser*. The Façade uses a *Sensor Network Manager* to provide the CSP network management functionality. These network management functionality is carried out using *Service Accessor* and *Sensor Service Provisioner* components.

A *Service Accessor* finds service providers using the Jini Lookup Services. First, it discovers lookup services and then finds matching services specified by signatures in exertions. A *Sensor Service Provisioner* provides for provisioning of sensor services based on quality of service specified by requestors according to the Rio framework. Thus, dynamic network formation of sensors in SenSORCER dynamically allocates a CSP to the capable cybernode (the Rio compute node) with operational specifications provided by the requestor.

VI. IMPLEMENTATION

The SenSORCER framework has been implemented using Java, SORCER, Jini, Rio, and Groovy and deployed in SORCER Lab, Texas Tech University [19]. For the experimental purpose we used temperature sensors built in into SUN's Programmable Object Technology (SUN SPOT) device [10, 11]. Implementation also supports arithmetic expression evaluation, the sensor computation mechanism available for composite service providers. Fig. 2 illustrates the Inca X browser [18] displaying all currently available services in the sensor network. The notable are Jini infrastructure services (Lookup Discovery, Event Mailbox, and Lease Renewal services), Rio provisioning services (two Cybernodes and one Monitor - provisioning service). Four elementary sensor services are individually connected to four temperature sensors (Neem-Sensor, Jade-Sensor, Coral-Sensor, and Diamond-Sensor). In addition, one composite sensor service (Composite-Service) and one façade service (SenSORCER Façade) is also visible. The view shown on the right side of service list in Fig. 2 is the sensor browser interface attached the façade service. In Fig. 2 also the subnet formation with composite service is shown. The steps to carry out experiment are as follow:

1. As shown in Fig. 3, using composite service (Composite-Service), we formed a sensor subnet with three elementary sensor services (Neem-Sensor, Jade-Sensor, and Diamond-Sensor).
2. Associated a compute-expression to report average temperature of these three individual sensors ("(a + b + c)/3").
3. Provisioned a new composite service on to the network (New-Composite).
4. Using provisioned service, we formed sensor network with one composite (subnet formed in step 1) and remaining elementary sensor service (Coral-Sensor).
5. Associated expression with provisioned service to take average of two composed services from step 4 ("(a + b)/2").

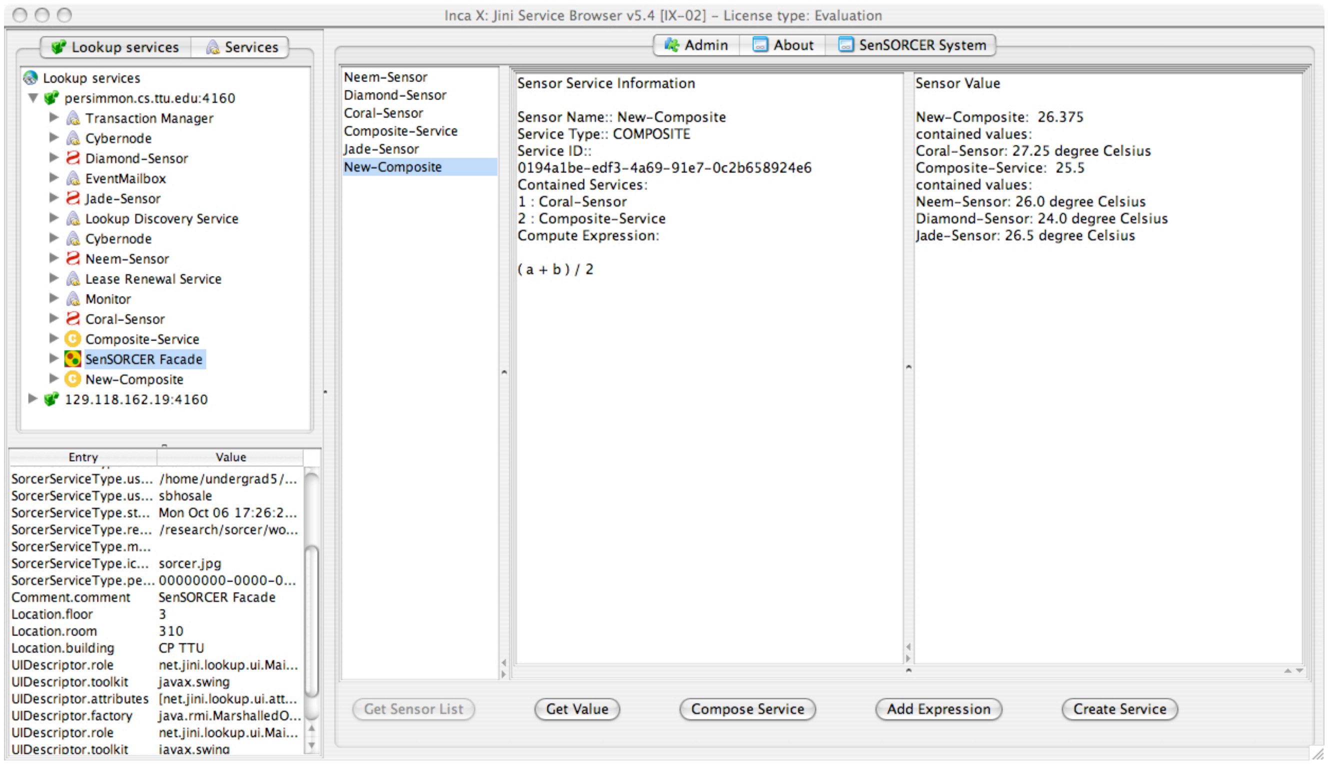


Sensor Service Information

Sensor Name:: Composite-Service
Service Type:: COMPOSITE
Service ID::
267c67a0-dd67-4b95-beb0-e6763e117b03
Contained Services:
Compute Expression:

Entry	Value
SorcerServiceType.us...	/home/undergrad5/...
SorcerServiceType.us...	sbhosale
SorcerServiceType.st...	Mon Oct 06 17:26:2...
SorcerServiceType.re...	/research/sorcer/wo...
SorcerServiceType.m...	
SorcerServiceType.ic...	sorcer.jpg
SorcerServiceType.pe...	00000000-0000-0...
Comment.comment	SenSORCER Facade
Location.floor	3
Location.room	310
Location.building	CP TTU
UIDescriptor.role	net.jini.lookup.ui.Mai...
UIDescriptor.toolkit	javax.swing
UIDescriptor.attributes	[net.jini.lookup.ui.att...
UIDescriptor.factory	java.rmi.MarshalledO...
UIDescriptor.role	net.jini.lookup.ui.Mai...
UIDescriptor.toolkit	javax.swing

Figure 2. SenSORCER services



```
Lookup services
▼ persimmon.cs.ttu.edu:4160
    ▶ Transaction Manager
    ▶ Cybernode
    ▶ Diamond-Sensor
    ▶ EventMailbox
    ▶ Jade-Sensor
    ▶ Lookup Discovery Service
    ▶ Cybernode
    ▶ Neem-Sensor
    ▶ Lease Renewal Service
    ▶ Monitor
    ▶ Coral-Sensor
    ▶ Composite-Service
    ▶ SenSORCER Facade
    ▶ New-Composite
▶ 129.118.162.19:4160
```

Entry	Value
SorcerServiceType.us...	/home/undergrad5/...
SorcerServiceType.us...	sbhosale
SorcerServiceType.st...	Mon Oct 06 17:26:2...
SorcerServiceType.re...	/research/sorcer/wo...
SorcerServiceType.m...	
SorcerServiceType.ic...	sorcer.jpg
SorcerServiceType.pe...	00000000-0000-0...
Comment.comment	SenSORCER Facade
Location.floor	3
Location.room	310
Location.building	CP TTU
UIDescriptor.role	net.jini.lookup.ui.Mai...
UIDescriptor.toolkit	javax.swing
UIDescriptor.attributes	[net.jini.lookup.ui.att...
UIDescriptor.factory	java.rmi.MarshalledO...
UIDescriptor.role	net.jini.lookup.ui.Mai...
UIDescriptor.toolkit	javax.swing

```
Neem-Sensor
Diamond-Sensor
Coral-Sensor
Composite-Service
Jade-Sensor
New-Composite
```

```
Sensor Service Information

Sensor Name:: New-Composite
Service Type:: COMPOSITE
Service ID::
0194a1be-edf3-4a69-91e7-0c2b658924e6
Contained Services:
1 : Coral-Sensor
2 : Composite-Service
Compute Expression:

( a + b ) / 2
```

```
Sensor Value

New-Composite:  26.375
contained values:
Coral-Sensor: 27.25 degree Celsius
Composite-Service:  25.5
contained values:
Neem-Sensor: 26.0 degree Celsius
Diamond-Sensor: 24.0 degree Celsius
Jade-Sensor: 26.5 degree Celsius
```

Figure 3. Logical sensor networking

6. As shown in Fig. 3, read Sensor Value from newly created composite service.

Fig. 3 displays the snapshot after step 6. The provisioned sensor service (New-Composite) is also visible as the registered service with the lookup service. On “Sensor Value” section we can see the average temperature values of all the sensors available in the system. The temperature values read from the other sensor services are also listed. “Sensor Service Information” section displays the composed service in step 4 and associated expression in step 5. The variables that are used in the expression are created dynamically, as the services are added into the composite provider. For example, in Fig. 3, variable ‘a’ and ‘b’ are created dynamically, when the two services are added into the provisioned composite service. (‘a’ is created for Composite-Sensor service and ‘b’ is created for Coral-Sensor service).

VII. SenSORCER Benefits

From the architecture point of view one can easily make out that the sensor probe is the only sensor dependent component. Applications written for sensor data are independent of the sensor technology used. In addition, sensor services themselves make use of sensor probes, so they are only concerned with sensor probe’s interface, which does not dependent on a particular sensor technology, thus, achieving probe independent service providers. One can easily change the existing implementation and technologies of the sensors used and still keep up with rapid evolution of the sensor technology.

One of the core challenges of sensor application design is balancing the resource usage of individual nodes with the global behavior of the desired network [15]. SenSORCER makes use of federated metacomputing environment, which offers dynamic federation of the services to complete complex and scalable tasks [9]. In addition, runtime network management makes the scalability and dynamic networking easier. There is no need to make any change to the physical network resources, as all we do is changing the logical network arrangements. The use of Rio provisioning allows for dynamic and failure resilient network creation on-the-fly. As the computing resources spread across the network and service providers can be associated with different compute resources, addition of new sensor services does not necessarily affect the performance of the system. The Rio framework allows for allocating the sensor service to the best compute resource (cybernode) from the available network resources at hand, alleviating the application design and development from usually required resource management.

The sensor network administration has two aspects, one is hardware related which has to be taken care at the particular location and with the sensor device physically. The other is software configuration and network management. For latter, all the sensor service administration is available for the user via the provided Sensor Browser (see Fig. 2).

The Sensor Browser is a very lightweight and zero-install service UI [20]; it does not contain any heavy processing components. For the most part, the service UI just takes the input from the user and gives back result from the SenSORCER network. The associated compute hosts run the available sensor services; nevertheless, the Sensor Browser can run even on mobile devices.

The user can collect the data from different sensor services directly or it can make use of composite provider, to make it go to the different providers. This service-to-service communication makes it possible to transfer data from node to node without any user intervention. If user wants to get data from different services and perform some computation, he can create a composite service provider to do this task.

This way, we can manage the data generated from different services, allowing various services to take part in both communication and computation processes.

The plug-and-play feature is very useful in sensor networks, as after the initial installation and configuration of the system, the addition and removal of sensors is more frequent. Plug-and-play of discoverable services with Jini lookup services allows any sensor service to appear and go away in the network dynamically. Since SenSORCER is based on the Jini infrastructure, the sensor services can come and go. If a service goes down the node is terminated and when it is up the node is immediately available in the network.

VIII. CONCLUSIONS

This paper highlights the issues involved in designing and implementing federated sensor systems and demonstrates the feasibility of such deployment for metacomputing sensor environments. The presented SenSORCER architecture shares the attributes of P2P systems, dynamic service object oriented programming, and inheriting the security provided by Java/Jini security services along with exertion-oriented programming. It is modularized into a collection of sensor providers (ESPs and CSPs) with multiple remote SenSORCER Façades. Façades supply with a uniform access points via their smart proxies available dynamically to service requestors. A façade smart proxy encapsulates inner proxies to federating providers accessed directly (P2P) by requestors. The experimental sensor services have been successfully deployed as SORCER services and we are planning for large-scale air vehicles distributed applications. The SenSORCER network scales very well with the Rio provisioning support to satisfy the needs of current users and service requestors. The system handles very well several types of network and computer outages by utilizing the Jini infrastructure and dynamic exertion-oriented programming model. It provides a zero-install sensor browser (service UI) attached to the SenSORCER Façade that provides sensor network management facility.

ACKNOWLEDGMENT

This work was partially supported by Air Force Research Lab, Air Vehicles Directorate, Multidisciplinary Technology Center, the contract number F33615-03-D-3307, Service-Oriented Optimization Toolkit for Distributed High Fidelity Engineering Design Optimization.

REFERENCES

[1] M. Sobolewski, "SORCER: computing and metacomputing intergrid," Proc. 10th International Conference on Enterprise Information Systems, Barcelona, Spain, 2008, pp. 74-85.

[2] M. Sobolewski, "Exertion-oriented programming," 2008, IADIS, vol. 3 no. 1, pp. 86-109, ISBN: 1646-3692.

[3] Definition of SOA from "Organization for the advancement of structured information standards". Retrieved April 12, 2009, from http://www.oasis-open.org/committees/tc_cat.php?cat=soa.

[4] W.K. Edwards, "Core Jini", 2nd ed., Pren-tice Hall, ISBN: 0-13-089408, 2000.

[5] Rio overview. Retrieved April 12, 2009, from https://rio.dev.java.net/overview.html

[6] M. Bertocco, S. Cappellazzo, C. Narduzzi, M. Parvis, "A distributed sensor network based on Jini Technology", VIMS 2002 IEEE International Symposium on Virtual and Intelligent Measurement Systems, 2002, pp. 68-71.

[7] J. Blumenthal, M. Handy, F. Golatowski, M. Haase, D. Timmermann, "Wireless sensor networks - new challenges in software engineering", Proceedings of Emerging Technologies and Factory Automation, 2003 (ETFA '03), IEEE Conference, 2003, vol. 1, pp. 551-556.

[8] Rio architecture overview. Retrieved April 12, 2009, from http://www.sun.com/software/jini/whitepapers/rio_architecture_overview.pdf

[9] M. Sobolewski, "Federated collaborations with exertions", 17h IEEE International Workshop on Enabling Technologies: Infrastructures for Collaborative Enterprises, Rome, Italy, 2008.

[10] Sun™ Small Programmable Object Technology (Sun SPOT) Developer's Guide. Retrieved 12 April, 2009, from http://www.sunspotworld.com/docs/Purple/spot-developers-guide.pdf

[11] Sun SPOT Overview. 12 April, 2009, from http://www.sunspotworld.com/

[12] The Jini Technology Surrogate Architecture Overview [PDF document]. Retrieved April 12, 2009 from https://surrogate.dev.java.net/doc/overview.pdf

[13] W. Heinzelman, A. Chandrakasan, and H. Balakrishnan. "Energy efficient communication protocol for wireless micro-sensor networks", Proc. the 33rd Hawaii International Conference on System Sciences (HICSS), January 2000.

[14] A.Woo T.Tong, and D.Culler, "Taming the underlying challenges of reliable multi-hop routing in sensor networks", Proc. the First ACM Conference on Embedded Networked Sensor Systems (SenSys2003), November 2003.

[15] G. Mainland, D. C. Parkes, and M. Welsh, "Decentralized, adaptive resource allocation for sensor networks", Division of Engineering and Applied Sciences, Harvard University, IEEE1451. Retrieved April 12, 2009, from http://www.completetest.com/IEEE1451_overview.htm

[16] D. Koenig, A. Glover, P. King, G. Laforge, J. Skeet, "Groovy in action", New York: Manning Publications, 2007.

[17] UML 2.0 infrastructure specification [PDF. Retrieved April 12, 2009, from http://www.omg.org/docs/formal/03-03-01.pdf

[18] Inca X service browser. Retrieved April 12, 2009, from http://www.incax.com/pdf/Jini-browser.pdf

[19] Laboratory for Service-ORiented Computing EnviRonment (SORCER). Retrieved April 12, 2009, from http://sorcer.cs.ttu.edu/

[20] The ServiceUI Project. Retrieved April 12, 2009, from http://www.artima.com/jini/serviceui/

[21] M. Grand, "Patterns in Java", Volume 1, Wiley, ISBN: 0-471-25841-5, 1999.