CHAPTER 7
ARCHITECTURE FOR PROVIDING SENSOR BASED INFORMATION SERVICES TO USERS

7.0 CHAPTER INTRODUCTION

In previous chapters, methods of hierarchical assessment of situations in complex environments by fusion of information from heterogeneous sensors were developed. In this chapter, the last link in making sensor data available from sensing fields to end user sitting anywhere in the world has been dealt with. It focuses on design of Internet compatible access constructs to make sensor data discoverable over web. The concept of providing software as web service proliferates providing software application as a paid or free service by its developer. On similar lines, the concept of WSN- as – service has been proposed in this chapter. The problem of discovering and selecting the most suitable web service is challenging in conventional web services [207]. Semantic Web Services (SWSs) represent the technology developed for Internet wide automated machine-to-machine interaction [208]. It has thus been proposed to provide sensor data and abstractions based on it as semantic web based service to all registered users on demand.

An upper ontology for representing relation between sensor data and its abstractions (as per developed framework) has been designed. The ontology is then used to initialize semantic web service for particular application. Existing web standards of software modeling, web service and ontology description have been leveraged and customized to achieve this goal. Conceptualization of primitives for WSN based information service has been dealt with.

Methodology of semantic web services for sensor data and existing methods are subject matter of Section 7.1. Methodology involves deriving an upper ontology as per logical hierarchical data processing for proposed sensor fusion. Models of service-providing devices like end user device, context gathering devices and sensors have been modeled using Unified Modeling Language (UML). Formally represented UML models have then been recast to Web Ontology Language
for Services (OWL-S) standard for semantic web through intermediate conversion to Web Service Description Language (WSDL) format. This makes sensor web service ready for launch.

Section 7.2 deals with procedures for registering and accessing the developed services. Interfaces and data flow for bridging the developed web service to browser based access by users have been proposed. The access has been designed utilizing application layer web standards of Simple Object Access Protocol (SOAP) and Hypertext Transfer Protocol (HTTP). The service access is preceded by user registration. Flow diagrams for processes of registration for first time user and access for existing users have been provided.

The concluding Section 7.3 provides an overview of designed methods and their usability.

7.1 DESIGN OF SENSOR BASED SERVICE- SEN2SITU

Given the methodology adopted in earlier chapters, sensor data has been abstracted in form of situations hierarchically. Thus, the name ‘sen2situ’ has been given to the proposed service.

For development of such semantic web service based interface, following deployment assumptions have been made:

a) Sensor nodes are based on IEEE 1451 standard for smart sensors. It has a Transducer Electronic Datasheet where identification & calibration information of the node is stored.
b) WSN is organized in a hierarchical topology namely sensor nodes and intermediate mobile devices connected to dedicated gateway housed in sink nodes implementing situation recognition.
c) Conventional communication protocols of the existing WSN for routing, authentication and self-organization will be used [51].
d) The Gateway nodes are IP enabled and act as bridge between the data addressable sensor network and the Internet.

As a result of decade long research, the assumptions made are feasible [1]. The Gateway Nodes will host the semantic web based sensor services.

7.1.1 Methodology

In this section, workflow and main components of Sen2Situ service has been introduced. The domain independent upper ontology defining generic sensor to situation conversion has been
used for development of application specific service. The target service being developed here has goals of providing real time sensor data and its abstractions as paid or free service to a registered user. Proposed methodology for obtaining sensor to situation as a service is described in Figure 7.1.

![Diagram showing methodology for obtaining sensor service description]

**Figure 7.1: Methodology for Obtaining Sensor Service Description**

As shown in Figure 7.1, situations are the highest level of abstraction obtained from raw sensor data. Situations are output of local hierarchical processing from sensors to contexts to situations as described in earlier chapters. This information is used in upper ontology design for proposed service. Advantage of an upper ontology is that it can be extended to implementation or domain specific concepts and instances.

Conceptual software model of service describing required parameters for discovering and invoking the service is designed with feedback from service users and providers. The model is then formally represented using Unified Modeling Language (UML). In next step, the human comprehensible service description in UML is transformed into standard web service description language, WSDL using Model Driven Development (MDD) approach [210].
Semantic description of web service is then generated in two steps, first is generation of an OWL-S template with basic SWS structure and then use of designed upper ontology for semantic description of the service elements [211]. The final output is complete OWL-S description of desired sen2situ service. Details of each step are discussed in next section.

7.1.2 Representation of Sensor Data Fusion Model as Upper Ontology
The framework of processing sensor data to prevalent situations has been modeled as ontology for web based automatic access by programs. An ontology refers to the formal, explicit description of a shared conceptualization of a domain of interest that are often conceived as a set of entities, relations, instances, functions, and axioms. Ontologies play a pivotal role in understanding the semantics of the context services [154]. Ontologies are collection of application or domain specific concepts which may not remain valid across applications. An upper ontology, on the other hand, is top level ontology which describes very general concepts that are same across all domains.

A coarse upper ontology has been designed to capture the domain independent sensor data fusion hierarchy as proposed in earlier chapters. Ontologies are actually represented as formal languages that can be processed automatically, for example, description logics. However, for understanding and inspection of concepts the ontologies can be represented visually using techniques like indented tree and graphs. Graphic visualizations have been found to be more intuitive without visual redundancy. The proposed upper ontology is thus visually represented by graph in Figure 7.4.

For pervasive computing environments, an upper ontology bridging the concepts of contexts and situations has been proposed by Yau et al in [212]. The design of ontology presented here has been inspired by this one. However, in Yau’s ontology, there is an ambiguity in establishing the hierarchy between contexts and situations. Situations have been defined as atomic or composite, where atomic situation is same as context. They have also defined an entity class generating data related to context. However, physical or logical description of an entity is unclear. The concept of context has been defined by example. Although more contexts can be added to the concept, but that makes the ontology far from being a complete upper ontology.
The upper ontology of Figure 7.2 has essence of Yau’s ontology [212]. The maxim of 6W proposed in Chapter 5, has been used to define the intuitive context definition super class irrespective of specific application domain and set of target situations to be identified. The hierarchical structure of proposed ontology has been divided into three super classes namely Sensor, Context and Situation.

**Sensor Class:** Sensor class represents the data sources in monitored environment. These are conceptually at the lowest layer of ontology. The sensor class can have instances from domain of
wearable and ambient sensors. Each sensor emits values in binary, numerical or nominal form that can be stored and represented in simple data types like integers, floating point numbers or strings. Each generated value also has a time stamp. Time stamped sensor data is used in context extraction model.

**Context Class:** The context class is the super class for representing any contextual information. The context class has generic categories of context, that is, ‘Who’, ‘Why’, ‘What’, ‘When’, ‘Where’ and ‘How’ as disjoint sub-classes of Context class. Every instance of a context subclass possesses symbolic or nominal types as actual context values. Value of each context has been obtained by model based aggregation of select sensors. The generated context instance is also time stamped to consider context generated at same time for situation extraction.

**Situation Class:** The situation class is the super class of all target situations to be identified. Instances of situations are obtained by logical composition of context values of all categories generated at same timestamp. Situations have type of value as property. The data type of this property is symbolic values.

Besides the above three super classes, the ontology also defines the underlying Application and End User entities. The application defines the set of target situations to be identified from persistent monitoring through sensors. End user is an important and dynamic entity, as he operates on various sensors and data is generated accordingly. Each end user manipulates the sensors in his/her own individual way.

The proposed upper ontology instantiated for human activity recognition in an indoor environment has been shown in Figure 7.3. Implementation has been done in Protégé, an ontology creating and editing tool.

The contexts and sensor classes are instantiated for representing opportunity dataset described earlier. To accommodate further application specific contexts more types of contexts in the basic categories can be added.
Figure 7.3: Ontology for Activity Monitoring in an Indoor Environment
The designed ontology will be serialized in OWL form to merge with the basic OWL-template created from detailed sensor, context and situation abstraction model description.

### 7.2 FORMAL MODELING OF SERVICE

Unified Modeling Language (UML) is a standard language for specifying, visualizing, constructing, and documenting the artifacts of software systems. It has been used to express the content and behavior of web services manually. UML profile based model has been created for the envisioned Sen2Situ service. UML profile provides several extension mechanisms such as stereotypes, tagged values and constraints to represent conceptual model of the service formally.

#### 7.2.1 UML Models of Sensor, Context and Situation

**UML description of Sensors**

The Sensor class of upper ontology defined in Section 7.2 defined the taxonomy of sensor devices and their inherent properties like type of data, rate of generation and accuracy. In UML description of Sensors, their software interface for implementing sensor web service has been defined. The goal of interface is to design efficient processing and access to sensor data as per service requirement. Some sensors inherently provide data for a context and don’t require any processing. For example, a binary switch sensor provides data about usage of a device.

Some fundamental steps required before designing interface for sensor and its data transformation are following:

1) The sensors that are to be employed in particular context detection have to be selected and installed suitably to aid it.
2) For each sensor the data supplied by it have to be examined thoroughly, and a detailed understanding of their syntax and semantics has to be developed.
3) The sensor Application Programming Interface (API) has to be examined to understand, how data can be obtained from the sensor.
4) Need for persistent storage of data has to be established.
5) Those sensor data which can directly be mapped to context attributes need to be assigned directly to contexts.

6) Interpretation and enrichment of those sensor data which cannot directly be converted into context attributes has to be defined.

After these steps are done, sensor model exposes required context attributes to be consumed by context interface model. These attributes may be stored if persistent storage is required for future training purposes.

Mechanism to access sensors that constitute context has been defined as Sensor UML package in Figure 7.4. This package includes class diagram extensions for specification of sensors constituting contexts. These extensions are defined as follows:

- The << Sensor >> stereotype defines name, identity of sensor, data quality attributes and configurable values of each sensor. This class also defines the methods to access these properties. Basic methods of obtaining identity and description of each sensor, that is, getId() and getName() are publicly available. The methods described for a sensor are initialize_op(), start_trg(), start_context_coll(), process_data() and stop().
  - Initialize_op () process is about registration of sensor with other local service registry. On triggering this method, the sensor becomes ready for further sensing operations
  - start_trg () is a method to start sensor data transmission in its raw form towards the context device seeking sensor service
  - start_context_coll() is a method to transmit processed sensor data. Some data may have to be buffered locally for processing. This method makes use of private process () method to initiate the processing of sensor data before transmission.
  - process_data() is a private method that implements local processing of sensor data according to the steps described in Chapter - 4
  - stop_op() is a public method to release connections from registry and resources and to terminate its operation.

Active sensors are registered in local service registry maintained at base stations or other intermediate devices. Each new sensor is registered with its nearest registry. Sensors are managed in a service registry and therefore register themselves with a SensorRegistry object. The SensorInterface inherits from the RegisteredSensor interface. Each sensor may be transmitting raw or processed data mode at any time instant. The current mode can be made available to
registry by `getDescription()` method. In `SensorRegistry` object, all registered sensors are listed. It also provides public methods of `register()` and `unregister()` to be utilized by sensor interface on execution of its own `initialize()` and `stop()` methods respectively.

![UML Class Diagram Extensions for Modeling Sensor Mechanisms](image)

**Figure 7.4: UML Class Diagram Extensions for Modeling Sensor Mechanisms**

The `lookup()` protected method is utilized, when a registered sensor or particular data is searched in the registry.

**UML Description of Context Component**

Context class obtains context data from designated sensors and fuses it to generate and disseminate CI. The context gathering, representation, evoking and access are enabled by `ContextualInfo` interface shown in Figure 7.5. This interface makes following methods available.
for CI queries: -  *getId()* , *getDescription()* , *getContext()* and an overloaded *requestContext()* method. As name suggests, *getId()* and *getDescription()* methods are used to identify the particular CI by unique identification code or descriptive name.

Asynchronous contextual queries are enabled by an overloaded *requestContext()* method. It is passed *ContextSpecification* object representing the conditions specified by the client. The specification could be context description or any other event related condition. The *requestId* parameter identifies a particular request of a user, while *userId* parameter specifies the identity of the user. Thus single user can make multiple requests for different contexts. Other parameters are specification of sought information, which could be fused CI or underlying processed sensor data termed as “Context Attributes”. *contextAttributeSpecification* parameter specifies the description

---

**Figure 7.5:** UML Model of Context Class
or conditions on requested sensor data. The method calls `getContext()` method to actually access the current data by implementing fusion.

Contextual data is accessed by higher layers using the `getContext()` method. Identity of user whose context is sought is supplied as input parameter. The list of constraints on freshness and quality of service parameters that are to be fulfilled during the determination of the context are also passed. QoS list has quality of service parameter defined by its type and value. The values returned by methods of this method are `ContextResult` or `ContextAttributeResult` objects, depending on whether the corresponding request referred to a context or a context attribute. QoS objects representing the quality of service parameters measured for a particular context element are contained in these objects.

Clients may delete previously made requests by means of the `deleteContextRequest()` method containing the request identifier as a parameter.

The `ContextualInfo` interface extends the interface `ContextSource`. Clients make use of `ContextSource` interface to register them for CI service before making asynchronous context requests. This interface implements two methods of adding a context listener for a client and removing it when not required. The `ContextualInfo` interface fires a CI event when the conditions specified in a context request are met. The context listeners implement `onContext()` methods for receiving information as soon as the results returned by `getContext()` method matches the requested information by client.

In this manner, the context implementation devices locally maintain a registry service for enabling discovery and access of CI as well as processed sensor data as ‘context attributes’.

**UML Design of Situation Class**

Elements of situation and end user service are defined in the SituationExtraction package and are shown in Figure 7.6. The elements of the SituationExtraction package are as follows:
– The << Situation >> stereotype describes the situation type and its description as current context vector. It permits to determine the context sources that will be used to collect this information once the context-aware middleware has been chosen.

– The << SituAvail >> stereotype represents the elements necessary to collect the situation. Resolution of service, starting time of availing of service, number of samples sent (time interval between notifications) and freshness which specifies how recent the identified situation is. Resolution of service is for deciding whether user is interested in situations, contexts or most low level sensors.

![UML Model for Context to Situation Extraction](image)

**Figure 7.6:** UML Model for Context to Situation Extraction

This stereotype is abstract, and it is used only to define the common tags of two types of sensor based situation collections: abnormal or interesting event-based collection, defined by the << Alert >> stereotype and regular periodic collection defined by the << RegularFeed >> stereotype.
The event tag indicates the occurrence of situation that must be satisfied for event based data dissemination to the service. The period tag of regular feed indicates the rate at which situation notifications should be generated.

- The << AbstractionQuality >> stereotype represents the quality attributes that must be satisfied by extracted situation such as accuracy, precision, and confidence in sensor fusion computed as cumulative error in all steps.

- The << SituValue >> stereotype makes it possible to specify the relevant target situations that are of interest to a given application. The base tags of << CurrentSituValue >> stereotype represents the current extracted value of situations from protected methods and their description. The binary conjunction and disjunction specification of extracted situation as raw CI is presented using << And >> and << Or >> stereotypes.

Elements of situation UML are important for the collection of situation information and the identification of the constituent context values that are relevant to users of a given application.

### 7.2.2 Conversion of UML to WSDL Model

UMLs are good for manual modeling and understanding of web services. However, to make the web service visible at Internet interfaces, service description in form of Web Services Description Language (WSDL) has to be provided. WSDL is a platform-independent interface description of web services. UML models don’t contain details of port binding, TCP/UDP transport protocol and description of URLs. UML models are thus converted into WSDL documents to produce accessible service documents. In WSDL based description, these details can be easily configured by user in related XML configuration file.

As described in Section 7.1, a web service can be bound by HTTP Get and Post or through SOAP based remote procedure calls. The conversion of UML to WSDL has been explained for partial UML of Sen2Situ service in Figure 7.7. Few details like namespace information has been avoided to enhance conversion clarity.

The *Situation* class with the *type* and *description* attributes corresponds to a complex type in the type section of the WSDL file with the same name as the class and part elements with the same names as the UML attributes.
In similar manner, the complete UML can be mapped to a WSDL description to provide structure of situation web service. It is evident from partial WSDL code that it is not easy for human
understanding. Thus UML design precedes WSDL description. Generated WSDL description can be validated by web service development tools such as Apache Axis or .NET.

The WSDL description is transformed to web ontology language for services (OWL-S) based basic description. The transformation converts WSDL description to detailed profile, process and grounding information of OWL-S based service. A manual method for the same has been described in [213]. Furthermore, tools like OWL-S editor also can be useful for this purpose [211]. The OWL-S template is enriched by domain ontology developed in Section 7.2. A reference to this ontology is added to the template.

7.2.3 OWL-S Based Description of Sen2situ

An ontology refers to the formal, explicit description of a shared conceptualization of a domain of interest that are often conceived as a set of entities, relations, instances, functions and axioms [214]. Situation services described in ontologies possess explicit semantic representations, which makes the automatic selection services possible. OWL-S, the popular upper ontology for design of semantic web services has been used here. The concepts of this ontology have been described in first section. In Figure 7.8 and 7.9, the WSDL and OWL-S profile document of a situation reporting service that takes as input ID of the user using service and outputs current situation is explained.

```xml
<?xml version="1.0" encoding="utf-8"?>
<definitions name="getCurrentSituation"

targetNamespace="http://localhost/getCurrentSituation.wsdl"

xmlns:tns="http://localhost/getCurrentSituation.wsdl"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:owls-wsdl="http://www.daml.org/services/owl-s/wSDL/"
xmlns:getSituationGrounding="http://localhost/getSituationGrounding.owl"
xmlns:getSituationProfile="http://localhost/getSituationProfile.owl"
xmlns="http://schemas.xmlsoap.org/wsdl/"

<message name="getSituationQueryID">

<part name="User_ID" owls-wsdl:owl-s-parameter="getSituationProfile:ID"/>
</message>

<message name="getCurr_SituationOutput">

<part name="Situ_ID" owls-wsdl:owl-s-parameter="getCurr_Situation:Situ_ID"/>
</message>

<portType name="getSituationPortType">

```

184
The service description is of a dummy service that can be repeated in any real service. The profile of a service provides a detailed description of the service to facilitate its discovery. Figure 7.9 describes the OWL-S profile having basic information about the service like its name, contact information and other descriptions. The profile has been derived from upper situation ontology described in section 7.2 and WSDL description of Figure 7.8. It also provides the functional description about the types of input, output and other preconditions. UDDI schemas provide classification information and same is provided in the profile document.

Figure 7.8: WSDL document of the Situation Reporting Service.
Besides this, nonfunctional information like QoS (quality of service) is also described in profile document. profile:hasInput and profile:hasOutput are merely pointers to the input and output terms defined in process.owl.

Profile document is used until the service has been selected after which the process document is used by client to interact with the service. The OWL-S grounding document, together with the WSDL document, will give any tool or agent enough information to actually access the service. In WSDL documents a single port type can contain several operations where each is defined by unique URI. The OWL-S grounding ontology defines the initial connections for both parties know exactly which operation is under the grounding process. The OWL-S grounding document, together with the WSDL document, gives any tool or agent enough information to actually access the service.

Figure 7.9: Profile and Process Descriptions of Situation Reporting Service
The next necessary piece of software is a matching algorithm that any soft agent can use to discover the desired service by sifting through all the service entries in a given UDDI registry. It is described in next section.

7.3 REGISTRATION, DISCOVERY AND DELIVERY PROCESSES

In this section the process using which the service provider, service seeker, sensor based system and the Web infrastructure can communicate with each other during different stages of sensor based situation service discovery and dissemination has been described. In Figure 7.10, the flow diagram of inclusion and use of a new sensor in a service based implementation scenario is shown. The steps that take place in the process are:

- The sensor class returns the results of locally processed data as XML specification templates according to the formats described in UML modeling.
- Using XML/ WSDL to OWL-S parser, the service is functionally described in OWL-S ontology. This task can be done in sensor or context heads.
- The context head or the sensor communicates the service profile to current Aggregator Service Registry (ASR) implemented in Context Heads. ASR is based on UDDI standard and is a private registry available to sink and sensors local to a given system.
- Various context heads send their service profiles to gateway node, which records these in Gateway Service Registry (GSR). GSR is a public UDDI based registry service, discoverable for all users across the web.
- The sensor system gateway, implemented in sink, hosts this service by listening to HTTP traffic and providing details of the service in Gateway Service Registry (GSR). This registry is browsable by the end user through internet, using HTTP-get requests.
- The end user application upon finding service of interest sends a SOAP request in HTTP packet.
- The WSN middleware digs down to sensor node, to get requested service and sends to the user using a HTTP-Response packet.
Implementation of Aggregate Service Registry enables the sensor based system to be physically distributed also. As the information dissemination in all the devices is based on web standards, physical proximity of sensors mapping to a certain context is not required. It makes the system scalable to addition of new sensor devices also. Whenever a new sensor is added to the system, it connects to the context heads meeting its description constraints defined by contextAttribute specification of context class.

The flow diagram of discovery of service by potential users and delivery of service by service providers has been shown in Figure 7.11. Details of steps required from search to access of service would be as follows.

1. A user uses browser application to search the desired service.
2. The search engine looks up into public UDDI GSR and discovers the service.
3. The user accesses the service profile and a SOAP request message is generated based on it.
4. The Gateway node or sink receives the SOAP request as part of a HTTP POST request, and forwards this request to service request handler.
5. The request handler parses the SOAP message, formulates request for desired resolution (situation, context or context attributes).

6. Desired service is invoked through ASR and a SOAP response containing terms of access is created.

7. The Gateway formulates a HTTP response, which includes the SOAP response, message and sends back to the user.

![Diagram of Sensor Service Discovery and Delivery Process]

**Figure 7.11:** Sensor Service Discovery and Delivery Process

In the process, both context heads and sink use ontology based reasoning for finding most suitable match for requested data.

### 7.5 CHAPTER SUMMARY

In this chapter, the issue of wide availability of sensor data and abstractions based on it has been discussed. Access of this information by standard web techniques will make WSN based
installations versatile, more effective and more useful. The concept of sensor data and its abstractions as service has been elaborated in this chapter and reference architecture for implementation of such service has been proposed. An upper ontology describing the hierarchy of abstractions has also been described. The actual implementation process and additional software components required for WSN nodes, intermediate devices and sinks have been proposed. Flow diagrams of the processes for new sensor registration, discovery of sensor based service and finally delivery of the requested information has been proposed. These schemes and processes will make Sensing as a Service feasible.