

SNAF: A SERVICE NAMING AND ADDRESSING FRAMEWORK FOR SerAN PLATFORM

D. Royo, A. Reyes, R. Messeguer, R. Martínez

Technical University of Catalonia, Spain

Keywords: Ambient Networks, Web services, service management, Pastry.

Abstract

One key component of an Ambient Network are the services it offers to the users, services like calendars, agenda, object localization, network information, etc. In this paper, we propose a Service Naming and Addressing Framework (SNAF) and their implementation into the SerAN (Service advanced management for Ambient Networks) platform.

SNAF is based in a Decentralized Distributed Hash Table (DHT) and a P2P Pastry system.

The SNAF framework allows managing services using Web services technology. It implements two main functions: by one hand, the *name service function* which allows users to publish services in the SerAN platform; and by other hand the *address service function* which allows users to access the services. The framework offers services to heterogeneous platforms and also an interactive interface to help end users on the parameter selection for the service they have chosen.

1 Introduction

Ambient Network [8], [14] is a new model to provide accessibility and distributed services through the dynamic composition of heterogeneous networks based on self-organization on demand. This composition will enable end-users and network operators to jointly exploit available network resources for a broad range of services. We are developing an advanced platform called SerAN [5] that provides services offered by heterogeneous Ambient Networks. In this paper, our goal is to describe SNAF, the Service Naming and Addressing framework for SerAN. The framework allows managing services based in Web services also it allows users and computers to publish, un-publish and access services. The framework has been designed, implemented and tested using a simple Web service (See Annex 3).

We briefly describe some related work developed by other groups like Pairot et al. [6], who propose a middleware for collaborative learning implementing some of the Ambient Network features but it does not provide an easy-implementation of proxies to interconnect different Ambient Networks neither the capability of accessing and manipulating resources belonging to different Ambient

Networks. Moreover, their framework [6] has other restriction, because it is RMI, which is tied to java.

Bamboo DHT (Distributed Hash Tables) [10], [16] is a platform that allows Web service access from a peer-to-peer system. It allows multiple proxy registries to publish and delete service information from their UDDI (Universal Description, Discovery and Integration) registries, as well as queries for the services from all registries.

Our research tries to overcome these problems creating an environment for Ambient Networks where users will have simple connectivity to a big set of heterogeneous services independent of the underlying networks used.

The paper is organised as follows: Section 2 gives a general view of the SerAN platform and explains the need to have an implementation of the Naming and Addressing framework for the Ambient Control Space.

Section 3 gives a general vision of all components related to the SNAF platform and section 4 concludes the paper with remarks on future research.

2 SerAN Architecture

SerAN is an advanced management platform, which provides services from different Ambient Networks in a transparent way to the end users.

SerAN implements its functionalities using Web services technologies to provide application services to the end users and also network services required by different Ambient Networks.

Pastry [3], [7] forms a Distributed Hash Table (DHT), self-organizing overlay network of nodes, where each node routes client request and interacts with local instances of one or more applications. Any computer that is connected to the Internet and runs the Pastry node software can act as a Pastry node, subject only to application-specific security policies.

Each node in Pastry peer-to-peer overlay network has a 128-bit node identifier (nodeId), so this nodeId is used to indicate a node's position in a circular nodeId space, which runs from 0 to $2^{(128)}-1$, which is the Pastry ring. The nodeId is assigned randomly when a node joins the system. It is assumed that nodeIds are generated such that the resulting set of nodeIds is uniformly distributed in the 128-bit nodeId space. As a result of random nodeIds assignment, it is high probable that nodes with adjacent nodeIds were diverse in geography, ownership, jurisdiction, network attachment, etc.

SerAN identifies data with a key, so our Pastry platform routes messages via looking for the closest nodeId to the given key.

Pastry performs application-level routing and object location into an overlay network of nodes connected via Internet. The network provides a variety of peer-to-peer applications, for example: data storage, data sharing, group communication and naming.

Figure 1 shows the SerAN ACS developed over Pastry. It provides a robust scalable architecture to support an Ambient Network environment.

The ACS controls the establishment, maintenance and release of service-specific overlay networks. These overlay networks are tailored to the specific needs of media delivery services.

The Overlay Control Space determines the topology of the overlay network during its establishment phase and reconfigures existing overlays when occur changes into the underlying physical resources that make necessary the reconfiguration. During lifetime of an overlay, this function manages data routing as well as adaptation of media content.

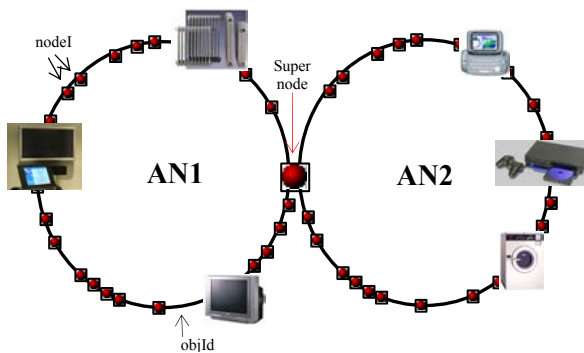


Figure 1 Ambient Control Space over Pastry

An Ambient Network in SerAN is managed by a Pastry ring where we design special nodes called super-nodes that allows the creation of new pastry nodes in the ring and acts as a gateway with other Pastry rings related to other Ambient Networks. Initially only the super-node exists.

The Pastry node registries services in the Pastry DHT table in order to make them accessible. When a service is published a key is assigned to it, that key index the DHT table and the object description is allocated in the proper position. Once the object is published it can be found by any application.

The SNAF framework contributes on registering, discovering and addressing services in SerAN platform.

3 SNAF Framework Architecture

In this section we describe the scenario where SNAF is integrated. SNAF allows managing services within the SerAN system. Basically, SerAN is an Ambient Network platform that proposes to implement Ambient Control

Space (ACS) functions over a DHT peer-to-peer Pastry system. In SerAN, the description of a service is expressed as a WSDL (Web services Definition Language). So, our research group developed the SNAF framework to publish, unpublish and access services over SerAN.

Our WSDL is a XML format description that includes any information related to control, events and presentation of the service.

Web services [13] define Simple Object Access Protocol (SOAP) [18] to execute remote procedure calls over services. Much like a remote procedure call SOAP is also used to deliver control messages to devices and return results or errors back to control points.

From the point of view of modus operandi, three main blocks compose SNAF:

- The block that is in charge of publishing and unpublishing services. (What services the Ambient Network offers?)
- The block that allows service discovering (who has the service?)
- The block that manages the access to published services. (What my service does?)

The first and second blocks were built over Pastry system. Services are identified through Pastry-Ids, which are assigned according their service names. Only the service name is needed to (discover) know where a service is hosted.

WSDL is used to describe a service, so, the third block is basically built, using Axis SOAP engine. Third block allows users to interactively execute all offered services from the platform. The system shows a screen to the user with the description (types of parameters, results) of all offered service functions. So, the user chooses the function and enters by keyboard the necessary parameters. We developed a parser to analyse the WSDL service description. The parser is based on a local SOAP client which we program using the Axis libraries. Remote service invocations are created on the fly and after it, users choose the service function and introduce the parameter values.

Following subsections describe, in more detail, the important pieces needed to understand SNAF in order to answer next questions: what SNAF is part of?, what SNAF uses for?

3.1 Web services Policies

The Web service Policy Framework (WS-Policy) [19] provides a flexible and extensible grammar for expressing the capabilities, requirements and general characteristics of entities in an XML Web services-based system. WS-Policy defines a model for the expression of these properties as policies.

In the SerAN platform, all published services are completely public. However, a user could require being aware of the reliability capabilities that are supported or required by a service. So, we will save the Web service description and a policy file advertising these capabilities

in the decentralized Distributed Hash Table of the SNAF framework. In this way, WS-policies will allow clients (or other Web services) to easily obtain information about specific capabilities such as guaranteed delivery, message ordering, application-specific service options, reply patterns of a specific Web service, etc. before calling the service.

A policy, for example, might assert that a particular service requires some form of reliable messaging or security, or it may require both reliable messaging and security.

At the abstract level a policy is a sequence of one or more rules, where each rule represents an acceptable alternative. A rule contains a number of policy assertions. All of the assertions need to be satisfied for the rule to be satisfied. However, only one of the rules can be satisfied for the policy to be satisfied.

Policies can be based on comparisons other than equality, allowing policies to depend on fine-grained attributes such as time of day, cost or network subnet address.

The following example illustrates a policy [20]:

```
(01) <wsp:Policy
xmlns:sp="http://schemas.xmlsoap.org/ws/2005/07/securitypolicy"
xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy">
(02) <wsp:ExactlyOne>
(03) <sp:Basic256Rsa15 />
(04) <sp:TripleDesRsa15 />
(05) </wsp:ExactlyOne>
(06) </wsp:Policy>
```

Lines (01-06) represent a policy for performing cryptographic operations with symmetric or asymmetric key-based security tokens.

Lines (02-05) illustrate the ExactlyOne policy operator. Policy operators group policy assertions into policy alternatives. A valid interpretation of the policy above would be that an invocation of a Web service uses one of the assertions (Lines 03- 04) specified.

3.3 Axis technology

Apache Axis [9], [15] is an implementation of the SOAP ("Simple Object Access Protocol") submission to W3C.

Axis is essentially a *SOAP engine* -- a framework for constructing SOAP processors such as clients, servers, gateways, etc

Axis 1.1 has proven itself to be a reliable and stable base on which to implement Java Web services. There is a very active user community and there are many companies who use Axis for Web services support in their products.

SNAF uses Axis written in Java for developing the service access side of our framework. We use Axis engine to build a generic SOAP client

3.4 SAX technology

The Simple API for XML (SAX) [2], [17] is a serial access parser API that provides a mechanism for reading

data from an XML document. The SNAF framework uses SAX to parse the WSDL file to obtain the service methods description: method name, parameters name, parameters type and result type. This information is sent to the SOAP client in order to generate the SOAP request.

4 SNAF Implementation

In this section we describe in detail the SNAF framework implementation.

First there is only a Pastry ring with a super node, when a new node is added to the pastry ring then the SerAN platform shows next menu:

- Publish Service
- Use Service
- Unpublish Service
- Service List

SNAF comprises the first two operations. Unpublish Service and the list of service actions have been created to help end users on service utilization.

The SNAF framework follows a five step protocol. First of all the service has to be published (see section 4.1), after publication there are four more protocol steps which are deeply described in next sections.

4.1 Publish Service

Given a service, the first step in SNAF framework is service publishing to make services accessible to other computers in the ring.

For publishing a service, a Pastry identifier has to be assigned to it. In our system, the service name is used to get this identifier. We apply the buildId method which returns a Pastry Id, which in the SNAF framework is the service identifier (Sid).

For example, if we consider into our Ambient Network a "tv" resource type, the code to get the Sid is:

```
Environment env = new Environment();
PastryIdFactory localFactory = new
rice.pastry.commonapi.PastryIdFactory(env);
Id Sid = localFactory.buildId("tv");
```

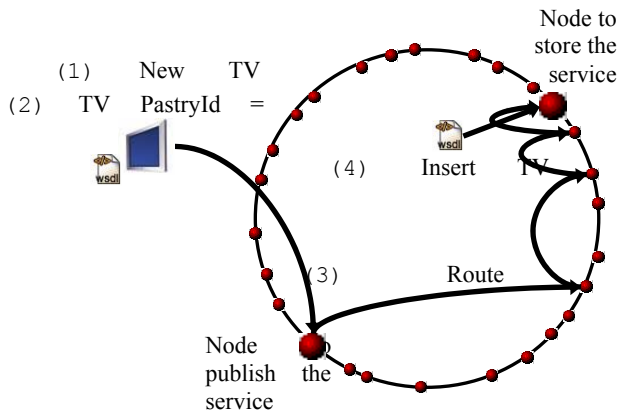


Figure 2 Publishing a service

Once you have obtained the Sid, the resource is inserted in the DHT table. Now the service is published.

Finally, SNAF assigns the URL of the WSDL service description and the URL of the computers which offer the Web service to the entry of the DHT table.

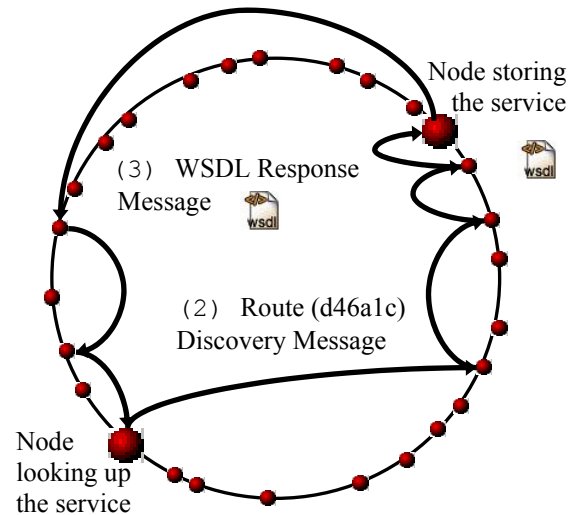
The node, which wants to publish a service, creates a thread. The thread obtains the service Sid, routes a message (Publish type message) to the Pastry node which controls the DHT table containing the service Sid, and waits the response message. When the response message arrives, the thread is wakening up and finishing its execution. The thread is in charge of managing any error generated when the service is published.

4.2 Service Discovery

On the previous step we get a Sid for the service, so now we have to discover where the service is.

SNAF creates a thread to do following actions:

1. Routes a discovery message to Pastry network addressed to the node which controls Sid and wait for the response to this discovery message. The thread waits in a semaphore.
2. When the response to discover message arrives, the thread is wakening up. The response contains the URL's service WSDL and the URL's client SOAP service to interact with the service.



(1) TV PastryId = d46a1c

Figure 3 Looking up a service

4.3 Service Description

Third step is service description. After a computer has discovered a service, the computer still knows very little about the service.

In order to learn more about the service and its methods, or to interact with the service, the computer must retrieve the service's WSDL description from the URL provided by the pastry node in the discovery message.

The SNAF service description is expressed in XML and includes a list of commands and actions needed to get a service answer and a set of parameters and arguments for each action.

The service description also includes a list of variables; these variables define the state of the service at run time, and are described in terms of their data type, range and event characteristics.

4.4 Control

The next step in SNAF framework is control. After a computer has retrieved a description of the service, the computer can send actions to the Web service. This is the traditional way to access a Web service. Control messages are also expressed in XML using the SOAP Protocol. Like function calls, in response to the control message, the service returns any action-specific values. The effects of the action, if any, are modeled by changes in the variables that describe the run-time state of the service.

SNAF offers another way of using Web services, where users interact directly with the Web service methods. The next step describes how to do it.

4.5 Presentation

SNAF allows to end users want sending actions to the Web service. Once, we have the WSDL's URL, the proper file is downloaded locally. The following step consists in parsing the WSDL file and obtains all information related to Web service functions. The description for each Web service function: Web service function name, number of parameters, parameters name, parameters type and result type, is shown to the user. That way, the user can choose the functionality and can introduce through the keyboard its parameters.

With all these information, the client SOAP creates a query on the fly and forwards it to the server which hosts the Web service.

The WSDL parser is written in java and uses SAX (XML parser). In Annex 1 is shown part of the parser's code.

The SOAP client is written in java too and uses Axis classes. It receives the method name, the list of parameters, type of parameters and response type, with all information a SOAP query is sent to the Web service server. Finally, the response message is obtained and showed to the user.

In Annex 2 is shown the code of a very simple SOAP client which allows to query a calculator Web service (Annex 3). The methods different methods have two parameters and one result, the code considers integer type parameters and results.

5 Conclusions and Future Research

In this paper we have described how a Pastry peer-to-peer system based on Distributed Hash Tables is extended to provide a robust scalable architecture to support Ambient Networks based on Web services.

We have implemented one of the key components of an Ambient Networking environment: the Service Naming and Addressing Framework (SNAF).

SNAF allows publishing Web services in a P2P DHT system. The Web services published can be accessed by other computers in the traditional way but also the framework offers to the end users an interface to access Web services interactively. This interface shows the Web service method list including the parameters and result type. So, users can introduce parameters values through the keyboard and the framework generates on the fly SOAP requests to the Web services.

We have tested the performance of our system with a simple Web service example. The code is shown in annex1 and annex2.

Another contribution is that SNAF does not need an extern UDDI system due that it is an UDDI by itself, which saves into their DHT tables the WSDL information and the client who gives access to the registry

In a future work, the SNAF framework will be extended with more features to improve the capacity of using Web services belonging to different Ambient Networks (Pastry rings) through Axis proxies.

Because until now, all our Web services offered are public; the capability of defining security access policies will be deeply developed.

Finally, we implement a Web service to offer functions of a calculator. We will extend the system adding support for new complex Web services and advanced policies.

This work has been partially funded by the Spanish Ministry of Science and Technology through project TIC-2004-07739-C02-01.

ANNEX 1

```
public void startElement(String uri, String
localName, String qName, Attributes attributes)
throws SAXException
{
    if(qName.substring(qName.length()-
9,qName.length()).equals("operation") &&
inPortType){
        op = new Operation();
        op.setName(attributes.getValue(0));
    }elseif
(qName.substring(qName.length()-
8,qName.length()).equals("portType")){
        inPortType = true;
        porttype = new PortType();
        porttype.setName(attributes.
getValue(0));
    }elseif (qName.substring(qName.length()-
5,qName.length()).equals("input") && inPortType){
        op.setInput(attributes.getValue(0).
substring(attributes.getValue(0).
indexOf(":")+1,
attributes.getValue(0).length()));
    }elseif (
        ...
        ...
        ...
    }
}
```

ANNEX 2

```
import org.apache.axis.client.Call;
import org.apache.axis.client.Service;
import org.apache.axis.encoding.XMLType;
import javax.xml.rpc.ParameterMode;

public class MyClient {
    public static void main(String [] args) throws
Exception {

        String endpoint;
        Integer op1;
        Integer op2;
        Service service = new Service();
        Call call = (Call) service.createCall();

        // setting the address where Web service is
        call.setTargetEndpointAddress( new
java.net.URL(endpoint) );

        // setting the method name to invoke
        call.setOperationName(methodName);

        // setting method parameters
        call.addParameter( "in0", XMLType.XSD_INT,
ParameterMode.IN );
        call.addParameter( "in1", XMLType.XSD_INT,
ParameterMode.IN );

        // setting return type
        call.setReturnType( XMLType.XSD_INT );
    }
}
```

```
// method invocation
Integer result = (Integer) call.invoke( new
Object [] { op1, op2 });

// printing result
System.out.println("The result of the method " +
args[1] + " is: " + result);
}
}
```

ANNEX 3

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions
xmlns:impl="http://localhost:8080/axis/services/C
alculatorWS"
<!--WSDL created by Apache Axis version: 1.4
Built on Apr 22, 2006 (06:55:48 PDT)-->

<wsdl:message name="sumaResponse">
<wsdl:part name="sumaReturn" type="xsd:int"/>
</wsdl:message>

<wsdl:message name="sumaRequest">
<wsdl:part name="in0" type="xsd:int"/>
<wsdl:part name="in1" type="xsd:int"/>
</wsdl:message>

<wsdl:portType name="Calculator">
<wsdl:operation name="suma" parameterOrder="in0
in1">
<wsdl:input message="impl:sumaRequest"
name="sumaRequest"/>
<wsdl:output message="impl:sumaResponse" name="sumaResponse"/>
</wsdl:operation>
</wsdl:portType>

<wsdl:binding name="CalculatorWSSoapBinding"
type="impl:Calculator">
<wsdlsoap:binding style="rpc"
transport="http://schemas.xmlsoap.org/soap/http"/>
<wsdl:operation name="suma">
<wsdlsoap:operation soapAction=""/>
<wsdl:input name="sumaRequest">
<wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/en
coding/" namespace="http://DefaultNamespace"
use="encoded"/>
</wsdl:input>
<wsdl:output name="sumaResponse">
<wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/en
coding/"
namespace="http://localhost:8080/axis/services/Cal
culatorWS" use="encoded"/>
</wsdl:output>
</wsdl:operation>
</wsdl:binding>

<wsdl:service name="CalculatorService">
<wsdl:port binding="impl:CalculatorWSSoapBinding"
name="CalculatorWS">
<wsdlsoap:address location="http://localhost:8080/axis/services/Cal
culatorWS"/>
</wsdl:port>
</wsdl:service>

</wsdl:definitions>
```

References

- [1] Box, D., et al. Web services Policy Attachment (WS-PolicyAttachment). March 2006. <http://schemas.xmlsoap.org/ws/2004/09/policy>.)
- [2] Brownell, D. SAX2. Ed. O'Really. January 2002 <http://www.oreilly.com/catalog/sax2/>
- [3] FreePastry, <http://freepastry.org/>
- [4] Kersch, P., Szabo, R., Lajos Z. Self Organizing Ambient Control Space. An Ambient Network Architecture for Dynamic Network Interconnection *International Conference on Mobile Computing and Networking. Dynamic Interconnection of Networks*. ISBN: 1-59593-144-9. 2005
- [5] Messeguer, R., Royo, D., Reyes, A., Parra, R. SerAN: Service advanced Management for Ambient Networks base don PASTRY. *The 2ND IET Internacional Conference on Intelligent Environments*. July 2006
- [6] Pairot, C., García, P., Gómez S. Dermi: A New Distributed Hash Table-based Middleware Framework. *IEEE Internet Computing Magazine*. Vol 8, No. 3, pp. 74-84. ISSN: 1089-7801 May/June 2004
- [7] Rowstron, A., Druschel, P. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. *IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, Heidelberg, Germany, pages 329-350, November 2001.
- [8] Schieder, A., Eggert, L., Papadoglou, N., Pittmann, F. Components and Concepts of the Ambient Networks Architecture. *Wireless World Research Forum (WWRF)*. 2004
- [9] Tarak Modi. Axis: The next generation of Apache SOAP. 2002. <http://www.javaworld.com/javaworld/jw-01-2002/jw-0125-axis.html>
- [10] Zitterbart, Z. A flexible and scalable peer-to-peer multicast application using Bamboo. *University of Karlsruhe, University of Cambridge*. June 2004 http://www.cl.cam.ac.uk/research/srg/netos/futuregrid/disc_hinger-report.pdf
- [11] Understanding UPnP™: A White Paper. UPnP forum. 2000
- [12] About the OSGi Service Platform. Technical White paper. OSGi Alliance, 2004
- [13] Web services. <http://www.w3.org/2002/ws/>
- [14] <http://www.ambient-networks.org/>
- [15] <http://ws.apache.org/axis/>
- [16] The Bamboo Distributed Hash Table A Robust, Open-Source DHT. <http://bamboo-dht.org/>
- [17] SAX, the Simple API for XML. 2004 <http://www.saxproject.org/>
- [18] Latest SOAP versions. <http://www.w3.org/TR/SOAP>