The Scooby Pervasive Infrastructure.

Jon Robinson & Ian Wakeman
Network Lab, Department of Informatics, University of Sussex
Project Overview

• Part of the Natural Habitat project.
• Exploring ways to use natural language to perform service composition.
• Two approaches to this project:
  – Natural Language Perspective ↓
  – Middleware/Networking infrastructure ↑
• Both approaches to meet in the “middle”.
Scooby Research Challenges

• First prototype implemented and in place.
• Scooby is the infrastructure layer of the system. The major issues involved can be categorised as: -
  – User Policy Composition & Language Design.
  – Architectural.

• Scooby stands for: Service Composition Objects Ordered By You.
User Policy Composition Issues

“A policy is a collection of predicates and a consequent action”

• Users can define a policy from which some form of functionality can be inferred. Hence, policies can be thought of as a programme.

• Produce a policy centric language in which users can specify the functionality as to what their services do in an easy and understandable manner.
User Policy Composition Issues

• Provide mechanisms & abstractions which:
  – Allow the integration of existing services into the infrastructure.
  – Hide the details of the event transportation layer (Elvin).
  – Hide the discovery, interrogation and communication mechanisms away from the user.
Language Design Issues

• The language is required to be machine-generated and processed.
• The composition language itself will be in a human readable form.
How Scooby does Policy Composition in its current state

- Programmes can be thought of as polices.
- Provides an intermediate level language for producing Scooby services.
- Scooby compiler compiles policies into Java.
- Procedural based.
- Similar syntax to Pascal.
- Hides all messaging mechanisms from user.
- Handles publishing of features automatically.
- Execution occurs once an event has been received.
A Scooby Programme Example

Service example

Initial startup
Begin
  useSubscription “requires(service)” associate test()
End

Publish feature test result string
  interface
    string name
    string surname
  variables
    string newname
  capabilities
    define “type”, “silly”
Begin
  newname setto name plus surname
  result newname
End

Allows for the definition of the parameters that are to be accepted.
The local variables to be used within the code block
Definition of the capabilities which are used in the service
discovery and lookup mechanism.
Code block.
Scooby example for finding a feature

Feature example
Variables
  Feature remoteFeature
  Parameter parameterData
  Integer count
Begin

RemoteFeature setto findFeature("method:example;")
parameterData setto getParameters( remoteFeature )
count setto parameterCount( parameterData )

# in this example, we will have two parameters defined in the
# remote feature, one of type string, the other of type integer

if( count greaterthan 0 ) then
  begin
    if( isString( parameterType( parameterData, 0 ) ) ) then
      setParameter( parameterData, 0, “Hello world” )
    if( isInteger( parameterType( parameterData, 1 ) ) ) then
      setParameter( parameterData, 1, 100 )
    call remoteFeature( parameterData )
  end
end

• Features can be found in other services by using the
  findFeature method whereby the search criteria is
  passed in as a parameter.

• The feature which has been found can be asked to
  provide its interface by calling the getParameters
  method.

• Invokes the remote feature.
Architectural Issues

• Reuse of existing technologies.
• Use of an existing message system to provide the transportation layer for events (Elvin).
• Internal and External event structures.
• Must be flexible and reliable.
• Use of existing standards to form the basis for communicating between services.
Scooby Architecture

Input Channel
- SMS
- Email
- Sensor
- Internet

Scooby
- User Service
- Elvin
- Scooby Language
- Black Board
- Existing Service
- Existing Service

Output Channel
- SMS
- Email
- Client
- Internet

Channel Manager
Conclusion & Future Work

• Our approach is to allow for the user to be hidden from all mechanisms, protocols and abstractions provided by the infrastructure.
• The Scooby Composition language provides the user with a Scooby-centric language whereby knowledge of how the infrastructure works is hidden.
• It is the responsibility of the compiler to “fill in” the details regarding how the discovery, publication, RPC, and event subscription handling is to be accomplished.
• We are entering the second prototype iteration and need to re-address issues which have been highlighted which placed constraints on the initial prototype. For instance, the constraint introduced by adopting a context-based message system as the transportation mechanism.