Intelligent Networks for Rapid Introduction of Personal Communications

drs. Hans Bisseling ir. Erik van der Velden

Four years ago both authors finished the OOTI programme, after which they joined Ericsson ETM in Rijen to start working in the field of Intelligent Networks (IN). Hans Bisseling is active in the European research programme RACE as core task leader within the project Mobilise. The objective of Mobilise is to define a personal communication environment called PSCS (Personal Services Communication Space). Erik van der Velden works at the IN Application Laboratory of Ericsson ETM, where he is prototyping new IN concepts. This prototyping is done by using the symbolic programming language Distributed Erlang, specifically developed for telecommunication applications.

Intelligent Networks (IN) is a concept for rapid introduction of new telecommunication services into the public network. IN standardization work has been started the late 80s and in the coming years network operators will introduce IN platforms in their network. Then, new services can be introduced faster as they can be expressed by scripts that interconnect *service independent building blocks* (SIBs). These building blocks are in fact abstractions from network capabilities. By standardizing these SIBs, IN services can be specified independently from the underlying networks or vendor.

Personal Communications

The RACE project Mobilise (R2003) is a four-year project (1992-1995) and its objective is to define a concept for personal communication. In the Mobilise project IN is used as the basic framework to develop PSCS, a concept for personal communications. In this concept, service mobility and the personalization of service conditions are main concerns of a modern telecommunication environment. In addition, smooth co-operation among various services and a simple presentation form to the end-user are of extraordinary importance as well. Nowadays, people can communicate in many ways. They can communicate by using an ordinary telephone or a more fancy mobile phone, or by using a fax machine. They can alert someone else by activating his or her pager, or they can use modern communication techniques such as electronic mail and multimedia. But, one big problem remains: the management of their communications at any place and at any time.

Figure 1: The personal communications problem

Nowadays, people can adapt their communications to their personal needs. For example they want to do call management in which they can set conditions (e.g., time, caller's identity, location) in order to handle incoming calls (resulting in e.g., reachable, announcement, forwarding to someone else, voice mailbox). Important is that people can do this type of management in an integrated way for all their communications, this does not mean that the communications themselves need to be integrated. As people do not care about technology they care about their communications and how to manage it.

To fulfill all these requirements in new telecommunication systems, a user-defined environment called Personal Services Communication Space (PSCS) has been developed. PSCS can be regarded as a shell around the end-user that is available at any time and at any place, through which the end-user can manage his communications in an integrated way. The main features of the PSCS are the following.

- *Personal mobility*
  Personal mobility means that an end-user can use any network access point and any terminal while being identified through the same number and charged to his personal account. PSCS is considered to be an extension of UPT (Universal Personal Telecommunication) which is standardized by ETSI and ITU-T. The main feature of UPT is registration (similar to login) by which the end-user associates his personal number to a terminal address. After registration (PIN needed) this terminal can be used for incoming and outgoing communications, but charged to his personal account. UPT offers mobility across mobile and fixed networks as users can register themselves on mobile terminals as well as terminals connected to the fixed network.
- *Personalization*
  The PSCS concept for personalization is that end-users have personal working environments, that can easily be managed by subscribers and configured by end-users. Subscribers (those who are charged, e.g., companies) can control the service delivery to their end-users (e.g., employees), and define limitations on the service usage, depending on the situation the end-user is in at a certain moment. End-users are then allowed to configure their personal environments within these limits.
- *Inter-operability*
  Inter-operability describes the capability of the system to support effective interworking between different services, offered on heterogeneous networks.

The PSCS conceptual framework is primarily based on the Intelligent Network Conceptual Model (INCM) (ITU-T Q.1200) with extensions taken from Open Distributed Processing (ODP).

A prototype implementation of some PSCS aspects

In the Intelligent Network Application Laboratory at Ericsson, we have implemented some of the ideas from PSCS by using Intelligent Networks. For this we implemented a IN simulator. In the IN simulator we introduced the concept of Flexible Service Profile (FSP).

The main idea behind this exercise is to gain some insights in the effect of PSCS on the network elements and interfaces.

The Flexible Service Profile

To meet the PSCS requirements of user mobility and personal services we introduced the concept of Flexible Service Profile (FSP). Each mobile user has its own FSP, which not only contains all data for that user, but can also contain service scripts for the services to which the user is subscribed. This means that the user can really have his own unique services. To facilitate grouping of users with common service requirements, it is possible to call upon common FSPs from the FSP of an individual user.

When an FSP user registers on a telephone in the network he is reachable on that telephone, all the (personal) services of the FSP user are available, for both incoming and outgoing calls. So, the FSP ‘follows’ the user as he moves through the network.

The IN simulator

To be able to experiment with the introduction of the FSP in an IN network we implemented a complete set of IN nodes in a simulator.

The simulator executes on a network of SUN workstations. Each IN node can be started on a separate workstation. Furthermore, the simulator controls two hardware switches, which are stripped versions of a PABX. The switches are controlled via the serial interface of the workstations. Connected to the switches are six feature phones, two analog telephones, one 2 Mbit link connecting the two switches, and a DECT Radio Exchange. If no switching hardware is present the simulator offers the possibility to make calls using a graphical user interface. A screendump of the IN simulator in action is presented in Figure 2.

With the simulator it is possible to create personalized services for one subscriber in the Service Creation Environment Function (SCEF). Via the management interface (SMF) the FSP is then downloaded to the database (SDF). When this user reg-

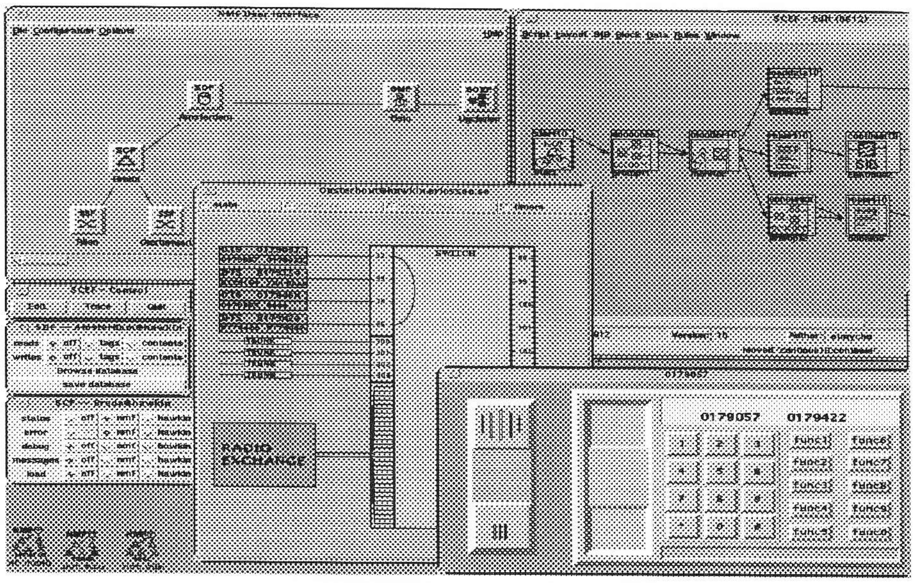


Figure 2: Simulating IN networks

isters on a telephone, his personal set of scripts are moved to a local Service Control Function (SCF) where they can be interpreted whenever the user invokes one of his services via a Service Switching Function (SSF). The new location of the user will be updated in the SDF.

When another user calls the FSP user, his current location is fetched from the SDF and the call is diverted to that location.

Implementation in Distributed Erlang

For the implementation of the IN simulator we used a concurrent declarative language developed by Ericsson called DISTRIBUTED ERLANG (see Example 1).

During the process of implementing the simulator we found that DISTRIBUTED ERLANG is very suited for this kind of applications. Some of the features of DISTRIBUTED ERLANG are the following.

- *Small but powerful*
  DISTRIBUTED ERLANG is easy to learn and a very high level language, which means that the gap between specification and implementation is very small.

- *Real time*
  Suited for the so-called soft real-time applications, with response times in the 5-15 ms range. DISTRIBUTED ERLANG has its own scheduler. But, when run under UNIX, no hard real-time is possible, since DISTRIBUTED ERLANG itself depends on the operating system.

- *Light-weight concurrency*
  Processes communicate with each other by sending messages. Message sending is asynchronous, a process sending a message does not wait until it has been received. Processes can select which messages they are prepared to receive. Light-weight concurrency allows you to easily cope with problems which are most naturally solved with massive parallelism, e.g., in the simulator we have one process per line in a trunk.

- *Transparent distribution*
  Distribution is part of the language. Distribution is transparent, the syntax and message passing is the same between processes on the same computer and between processes on different computers. Special interface languages for communication between computers are thus not necessary when DISTRIBUTED ERLANG is used, even if the computers are of a different type.

- *Robustness*
  DISTRIBUTED ERLANG provides three mechanisms for dealing with errors.

  - Monitoring the evaluation of expressions.

- Monitoring the behaviors of other processes.
- Trapping evaluation of undefined functions.

These mechanisms make it possible to design fault tolerant layered systems, e.g., having one layer of processes monitor a lower layer of processes and restart these processes when they terminate abnormally. Distributed Erlang's error handling mechanisms work transparently across distributed networks of processors.

- *Memory management*
  All memory management is done by the implementation of the language and not by the programmer.
- *Modules*
  Distributed Erlang has modules with import and export declarations. This is necessary to be able to structure large program systems into manageable building blocks.
- *Foreign Language interface*
  Distributed Erlang provides an interface for interacting with programs written in other programming languages.
- *Code changes in a running system*
  Many systems, like telecommunication systems, can not be stopped to install software corrections or amendments. Distributed Erlang provides mechanisms for loading or replacing modules in running systems. Code in modules being replaced can be phased out. Two version for a module can be present.

As an example the module of the prompt SIB is shown. The function of the building block is to display a text on the user's telephone and wait for digits. When these digits are received successfully, the result of this SIB is 'ok' and the entered digits are stored.

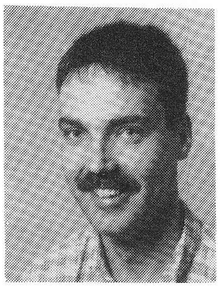
```
-module(prompt10).
-export([initial/2]).

initial(LogicData, Runlist) ->
   [Prompt, TermCond, Put] =
      bbsupport10:
      get_log_prms([prompt,termcond,put,[output]]),
   get(handler) !
   operation, self(), [input, Prompt, TermCond],
   receive
   result, ok,Digits ->
      dataman10:put_data(Put, Digits)
      ok, Runlist;
   result, nok ->
      failed, Runlist;
   Other ->
      throw({error,
      "rcvd unknown message during prompting"})
   end.
```

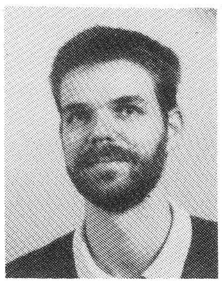
Example 1: A SIB implemented in Distributed Erlang

Distributed Erlang is already implemented on several machines. One of the architectures for which Distributed Erlang is available is the Support Processors in the Telephone Exchange. Support processors handle tasks independently of the central processor and can be used for, e.g., communication with hardware. Distributed Erlang demonstrates that it is not only possible, but profitable to use a higher level language in embedded real-time systems.

Conclusion

Personal mobility and personalized services will make their appearance soon. For the future Intelligent Network to handle these requirements some changes are needed. We think that the Flexible Service Profile will prove to be an answer. To make the ideas concrete we implemented this concept in a simulated IN. Distributed Erlang has proven to be an ideal programming language for implementing this simulator. □





Drs. Hans Bisseling (left) and ir. Erik van der Velden (right) completed the post-masters programme Software Technology in 1990. They are currently employed at Ericsson ETM in Rijen. Hans Bisseling is core task leader in the RACE project 'Mobilise'. Erik van der Velden is researcher at the IN Application Laboratory. Both authors are members of XOOTIC.