8

The Three Level Approach for service creation within Intelligent Networks

Mikko Kolehmainen
Nokia Telecommunications
Hiomotie 5, FIN-00045 Nokia Group, Finland,
Phone: +358-9-511 23959
Fax: +358-9-511 23339
Email: Mikko.Kolehmainen@ntc.nokia.com

Abstract

Conceptually, the task of service creation can be divided into three separate major levels that together provide the full set of functionality needed for efficient service creation. Each one of the major levels can be implemented as a stand-alone solution or they can be integrated together. This paper introduces the main points of the Three Level Approach that is applied by Nokia as the conceptual framework in the area of IN service creation. The paper does not take a stand on how components of the approach can be implemented, instead the emphasis is on the identification of the basic requirements and capabilities.

Keywords

Intelligent Networks (IN), Service Creation, Service Creation Environment (SCE).

Intelligent Networks and Intelligence in Networks D. Gaiti (Ed.)
Published by Chapman & Hall

1 INTRODUCTION

. One of the main goals of the concept of Intelligent Networks (IN) is to reduce the time needed for introducing new telephony services (Thörner 1994) (Capellmann 1996). Therefore, it is quite important that an efficient approach for tasks related to service creation is identified. However, even though the process of service creation contains several phases and requires many tasks both consecutive and parallel to each other, one basic division should be made. The service creation can be conceptually divided into two major parts: development of service logic and development of service management applications.

Both of these parts are of significance to the service implementation and to the usability of the actual service. Usually the interfaces of the logic part of the service are defined quite well by circumstance, since the context of the logic is always tied to the physical implementation of the chosen SCP (Service Control Point) platform. From the SCE (Service Creation Environment) vendor's point of view, the SCP platform is always accurately specified. The role of the management application is a more complicated issue, since the possible execution environments are usually quite heterogeneous.

In this paper, the focus is on the definition of the logic part of the service implementation. The aim of the paper is to introduce the concept of dividing service logic creation into three levels. The paper does not take a stand with regard to different architecture options that could be chosen when implementing tools for service creation. However, in (Kolehmainen 1996) there is one example of how an instance of Service construction level can be implemented. The issues related to the provisioning of services within the approach are only briefly described.

2 RATIONALE AND CORE CAPABILITY REQUIREMENTS

When considering the requirements set for the capabilities of the service creation process, a clear trend of diversity can be identified. This is mostly due to the variety of the positions of telecom operators in their operational regions as well as their business models. In practice this means that different operators need different properties from the tools they are using for the task of service creation. The rationale of diversity in operators' requirements is the fact that resources allocated for and therefore also expectations of the service creation vary greatly.

If the issue is studied from the service creation tool vendor's point of view, the first choice to make is to decide whether the tools the vendor is going to provide are intended to be used by all operators or by some specified group with common characteristics. For example, private second and third operators have different service creation needs and resources than established PPT's. After this decision, the second task for the vendor is to identify what service creation functionalities

are needed and of use within the chosen target group. When these strategic decisions have been made, the next step is to identify the actual service creation portfolio within the IN product line.

The process of identifying a product portfolio is understandably a complicated and demanding process that is closely tied to overall strategic guidelines. However, there are certain basic capabilities that can be considered essential in specific tasks related to IN service creation. The ideal approach in defining the service creation tools would then cater for these capabilities and be included in the service creation tool portfolio.

The necessary capabilities identified are presented in Figure 1.

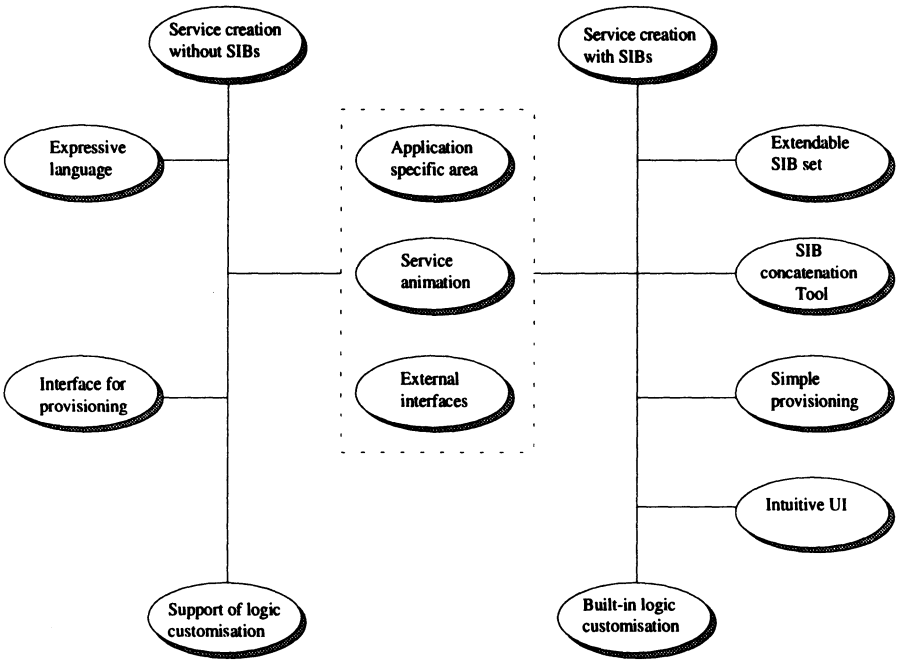

Figure 1 Core requirements set for service creation approach

There are two main lines that differ by their basic presumption. In the first line, the service creation is based on the solution that does not contain the notion of SIBs (Service Independet Building Block). Strictly speaking, this means that services are then constructed with a manner that is not aligned with guidelines presented in Q.1213 (ITU-T 1993). The actual implementation language can then be almost any procedural or object oriented programming language, a common or proprietary one. However, it is always possible, and naturally even recommended, to apply structural programming principles during the service development phase (Kinnunen 1997), (Laakso 1995).

In this context, the more interesting line and the one this paper concentrates on is the one that applies the concept of the SIB within the service creation process. In this approach the core capabilities are more intricate, since the service creation process must then fulfil basic requirements that are not present in the non-SIB approach. The differentiation in the requirements suggested in Figure 1 is mostly due to the differences in the service creation process. The main difference in the process is that before SIBs can be utilised, they have to be implemented in a method specific to the SCP platform.

3 OVERVIEW OF THE THREE LEVEL APPROACH

In the Three Level Approach the main idea is to identify different levels of service creation within the whole concept. The two factors which may be used as varying parameters between different levels are the ease of use and the rapidity of the actual task of service logic definition. As both of these factors are by nature hard to use as a basis for some metrics, it is not reasonable to define accurate identifiers for each of the levels. The more rational approach is to identify the different roles that are then represented by the levels. Figure 2 illuminates the basic idea of the approach.

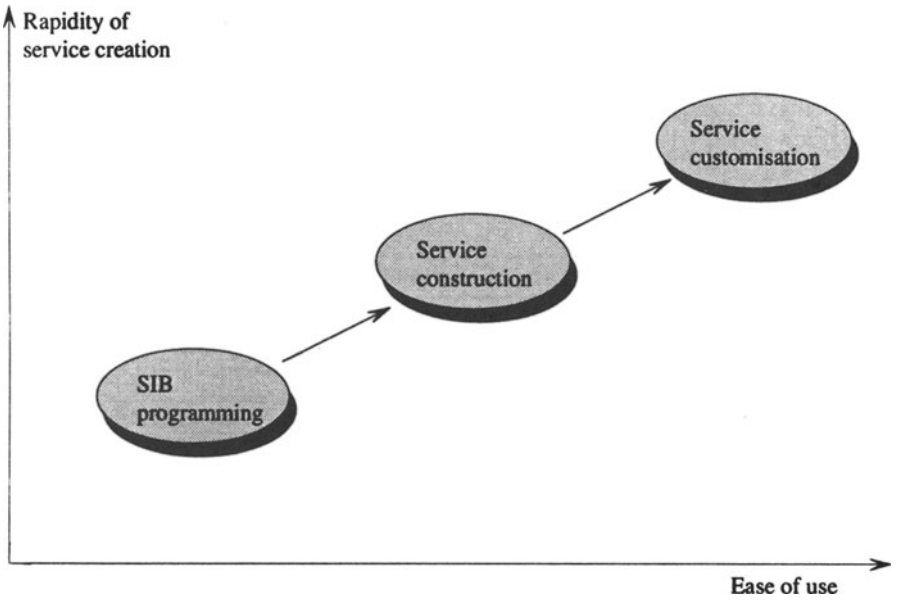

Figure 2 The three level approach

As Figure 2 suggests, the Three Level Approach consists of adjacent levels that can be hierarchically positioned in relation to each other. Each of the levels is targeted towards one specific task in the continuous activity of service creation. From each level there is a possibility for continuation to the upper adjacent level. In practice, this means that the outcome of the lower level in order can be utilised by the upper level in order. However, the levels can be considered independent so that one level can exist without support from tools positioned in some other level. In this case, it simply means that the functional capabilities provided by the nonexistent level are just missing. In any case, the tools of any single level can thus be considered stand-alone by nature.

Each of the levels represent a certain task, not specific tools. Thus, the approach does not bind or require any specific implementation to one single level. Furthermore, since only tasks are represented, it is possible that some tool implementation can provide the capabilities of all levels. On the other hand, the instantiation of the approach can be implemented so that one level can be mapped to one independent tool, which then has a well-defined interface to the neighbouring levels. The important thing is that in order to be independent, each of the levels must be able to provide some means for the service animation, albeit a modest one.

Every level requires specific core expertese from the personnel that are using the tools for their work. At the SIB programming level the core capability required is that of software engineering and programming. The net result of tasks carried out during SIB programming is an increase in the functionality of the platform that is available at the Service construction level. Service construction then requires expertese and understanding related to telecom networks. The outcome of this level is the actual service that can be utilised in the SCP platform. Finally the Customisation level requires expertese and understanding of subscriber needs. The subscriber specific and customisable parts of the service are stored in the service database that is accessed by the service. The data in the database is then used for guiding the behaviour of the service.

In the approach presented, the emphasis is on the definition of the service logic, except of course for the Customisation level that is by definition oriented towards service provisioning. The reason for this is that the relationship of service logic definition as well as service management and provisioning is not bijective by nature. Service management and provisioning must always be integrated with the operator's management and operation support system applications. From the service creation tools' point of view, this requirement is difficult to meet, since the interfaces of those systems are usually unknown to the SCE implementators. Therefore, adaptation work is nearly always needed and thus a generic and functionally adequate method for automatic management application generation is difficult to define.

This approach does not consider physical network configuration as part of the service creation process as (Turner 1995) suggests. The assumption is that the

network platform is already defined in the network design phase, which then can be seen as a prior phase before application-oriented IN service creation. In any case, it is important to realise that the network platform capabilities must be considered during some phase of service creation. In fact, in the Three Level Approach the most natural place to do this is the SIB programming level that conceptually provides the interface to the INAP (Intelligent Network Application Protocol) protocol.

4 THE SIB PROGRAMMING LEVEL

The lowest level of the Three Level Approach is responsible of the task of SIB creation. In this level, the idea is that the components being part of the service logic and considered reusable are implemented in a modularised manner so that they have a well-defined interface, when utilised in the next upper level. Basically the idea is that the SIBs are implemented in a programming language that is SCP platform specific. The service defined in the Service construction level is then generated as a program of this language. Therefore, there is always a linguistic mapping between SIBs at the SIB programming level and their representation at the Service construction level. Since the SIBs are the building blocks of the service, it is essential that their role within the service is understood and modelled correctly. Also, the SIBs being the nearest component to the SCP platform and INAP protocol interface towards the SSP (Service Switching Point), the possibilities for defining the SIB behaviour are in principle those provided by the actual SCP platform. In theory, the functionality of the largest possible SIB set that can be implemented at the SIB programming level should be equal to that provided by the actual SCP platform. However, in practise the case may not be so, since this requires that the SCP specific programming language supports modularised modelling and implementation of software components, e.g. SIBs. Of course, on platforms in which the service software is inherently defined with some object oriented language (e.g. Java) the mapping is quite bijective.

As suggested earlier in Figure 1, it is possible to omit the concept of SIBs in the service definition. In this case, the requirement is obviously that the programming language of the SCP platform or the implementation of the SCE does not compel the service designer to follow paradigms of software layering, so that the service could not be implemented without dividing the service software into modules. Of course, a separate issue is, whether unstructured programming is a desirable practice or not.

One of the goals of the IN concept is to reduce the time needed for introduction of new services. Implicitly this means that, in some stage of the process, the service implementation should be, if not an easy task, at least one that requires only limited skills in the area of software engineering. However, it cannot be denied that the complexity of software engineering is present in some of the parts

of service implementation. The complexity can be hidden behind automated tools, or it can be located in some defined place in the cycle of service creation.

The SIB programming level can be considered consisting of three separate fields of interest as Figure 3 suggests. The first of them is the task of SIB logic implementation, the second is the deployment of the SIB, and the third one is the support for SIB set administration with specific administrative application. In this context the deployment of a SIB should be understood to contain all the tasks that are required in order to get the SIB available in the SIB library.

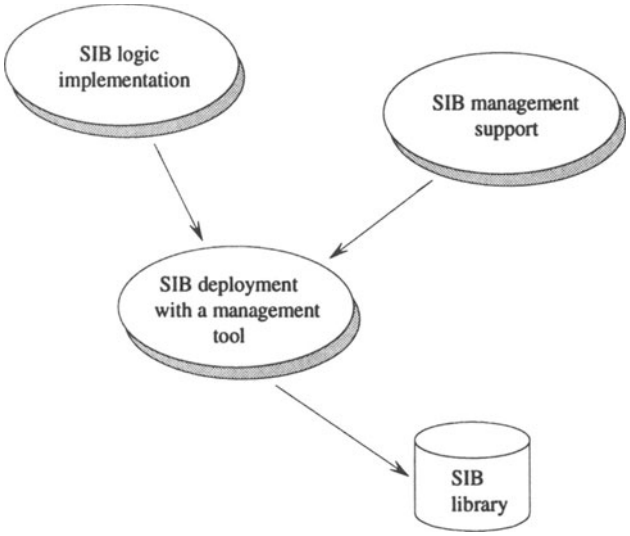

Figure 3 The task areas of SIB programming level

In the Three Level Approach, the SIB programming is the task that is considered to require the most expertise. This is mostly due to the fact that the SIBs access the platform components directly. Also, the concept of reusability proposes that it pays off to invest effort in the optimisation of SIB components. Since SIBs are software modules applying basic engineering paradigms depending on the execution platform, it is quite reasonable to require that people with software background are allocated to this task.

The tools used in this level must definitely provide a possibility for SIB execution, animation, and debugging. Implicitly this requires that there must exist a certain kind of test environment that makes it possible to test the SIBs with the principles of both white-box and black-box testing. This requirement does not preclude that simulation and testing environments could be the same both in SIB programming and Service construction level. However, since different components of the Three Level Approach are conceptually separate, the environments may well be independent level-specific implementations.

After the SIB component has been implemented, it needs to be introduced to the upper level. One applicable approach is to use a specific administration tool for the task of creating a library entry. Since one SIB can be seen to consist of several components, the administration tool must be able to gather the required components together and provide a defined access interface for the tools of the next upper level.

Another issue is the way the management of SIB components is arranged in the SLP (Service Logic Program) context. There are two basic alternatives. One possibility is to ignore the SIB management and provisioning at the SIB programming level. This means that these tasks are left as the responsibility of the service designer. The approach is reasonable, since from usability issues' point of view, the service management and provisioning applications should be considered one entity. Now, if the service management application is drawn together from several independent SIB specific components, it is not likely that the result is optimal.

On the other hand, the task of implementing a management application requires quite a lot of effort. Therefore, there can be an economical motivation, for the management part of the service to be implemented from pre-existing component specific modules. Again, it is reasonable to assume that at the SIB programming level the context and the environment of the management application is already known.

5 SERVICE CONSTRUCTION LEVEL

The actual service definition and implementation takes place at the Service construction level. This level is intended to be aimed at design personnel who are not very familiar with software engineering paradigms, but instead have knowledge about service ideas and network capabilities. In any case, it is required that the personnel whose responsibility it is to design, specify, and implement services must be familiar with relevant protocols, network capabilities, and some basic concepts of programming. Lots of complexity can be hidden behind automated functionalities provided by the service creation tools, but it is mandatory that the user understands the basics of the application area and how the hidden part is functionally positioned within the concept and, of course, the reasons for this positioning.

At the SIB programming level discussed earlier, the main emphasis is on the programming capabilities of the tools, i.e. that the tools intended for SIB implementation provide a complete instruction set that provides access to required SCP and network platform resources. In contrast, at the Service construction level one significant capability is that the tool used for defining the SLPs is easy, intuitive, and efficient to use. In fact, the solution at this level contains the user interface and the functionality that is normally provided by what is usually

understood by the concept of an SCE, cnf (Bihain 1994), (Knight 1994), (Locher 1997).

The Service construction level is based fully on the concept of SIBs. As in most SCEs, the service is defined graphically by concatenating SIBs together to form the service logic program. The main idea is that all the services that are constructed according to the three level approach utilise the SIB components implemented in the SIB programming level. In this phase, the key interface is the one provided by the SIB library. In effect, the SIB library must be administrated in a manner that makes it possible to specify different views of the contents of the available SIB set. For this purpose, an administration tool, either separate from or integrated to some already existing tools, is needed. The most important capability of the administration application is the possibility to configure the tools used in the Service construction level. The minimum requirement for configurability is that the available SIB set can be dynamically changed, possibly with or without recompilation of service editing tools.

The task of service testing at this level can be divided into two parts, the first being the simulation and testing of that part to be executed in the SCP platform. The second part is the task of modelling of the kind of behaviour the service execution generates in the network. The difference in emphasis when compared to the SIB programming level is that, when SIBs are tested and simulated, SIBs can be seen only as modules whose interaction to the calling context can be modelled only via careful design and well-chosen test cases within an existing test bed. In the Service construction level testing the SIBs and their functionality can be seen only via SIB interfaces and only as a functional part of the service implementation. Of course, the SIB's interaction with the world outside must be modelled with network simulation, probably with the same implementation as in the construction level.

Conceptually, there is a logical separation between Service construction and SIB programming levels so that the lower level is always tied to some specific SCP platform. In contrast, the Service construction level could be independent from the target platform. For example, one possible net result of this separation is that in a multivendor environment the SLPs need to be implemented only once. The approach makes this possible, if the target SCP platform provides a programming interface that is open enough and a programming language that can be used both as a source format and as a target format for linguistic compilation. This precondition is essential, otherwise the SCP platform remains closed as far as the Three Level Approach is concerned. In Figure 4, there is an illustration of how the separation of levels has an effect in cases where the tools should support this diversity.

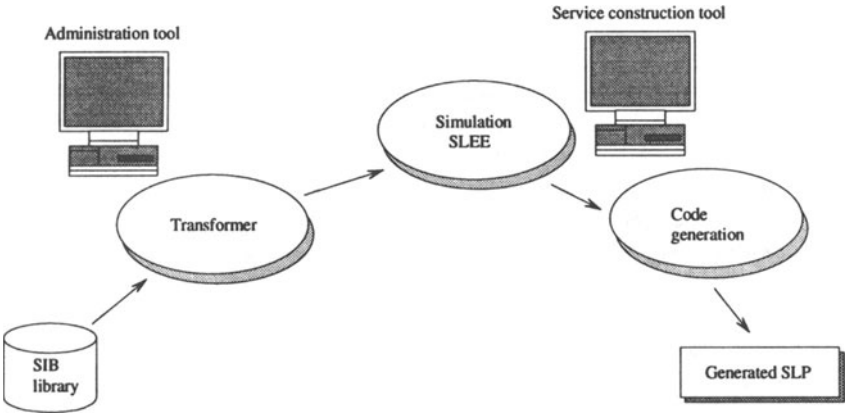

Figure 4 Different language transformations needed between separate levels when component diversion is applied

In Figure 4, the SIB library acts as an interface between SIB programming and Service construction levels. The administration tool can be positioned as a wrapper around the library and thus be seen as a part of the inter-level interface. In order to get the functionality of the SIB available in the SLEE (Service Logic Execution Environment) of the simulation environment, the SIB must be translated into the implementation representation of the Service construction tool. The transformation can be e.g. from an SCP specific language to C++. After the service has been implemented and simulated, it can be generated into the source representation for the target SCP platform. The reason why linguistic transformations of SIB implementations between SIB programming and Service construction levels are needed is the requirement that level implementations are independent of each other. In order to utilise Service construction level tools for generating services for several SCP platforms a generic service simulation SLEE must be used. This then means that the SLEE of the Construction level cannot be tied to the SLEE of some specific SCP platform.

The result of the service logic definition related tasks carried out in the Service creation level is a service logic program that consists of SIB invocations and commands that are needed for representation of a logic flow between those invocations. However, the produced service can hardly be considered as tested, since service logic animation and network behaviour simulation in a workstation is not enough. Therefore, a thorough testing phase in a laboratory environment must take place.

Service provisioning and management is an issue that should be considered at this level. This is because at this stage the service can be seen as an entity and the functional requirements set to the provisioning can be evaluated. When the service consists of SIBs, there are two possibilities: if those SIBs that require provisioning

data have necessary provisioning interfaces defined, the combination of these existing routines can be used. If the service constructed requires optimised provisioning and management screens or the operator's service management system has proprietary interfaces, the best solution is to implement the management part as a separate software project.

6 SERVICE CUSTOMISATION LEVEL

Strictly speaking, service customisation is not actually a part of service creation. Rather, it relates more to the service provisioning. However, the difference is quite vague, since in some approaches the logic which is the outcome of customisation can be quite freely defined. This is strongly dependent on the implementation, though.

The purpose of the service customisation is to provide a possibility to attach some parts of the logic of the service after the actual service has been implemented and deployed. It is worth noting that service customisation is always subscriber or subscriber group specific. The idea is therefore that the runtime database is used as a storage of subscriber or subscriber group specific parameterisable data that is used for guiding the behaviour of the executed service. As Figure 5 suggests, there are two main approaches in the service customisation. The first one bases on the concept of storing logic scripts into the database. These scripts are then accessed and executed by a specific script interpreter. The second approach utilises the idea of providing values for predefined database objects, which are then accessed directly by the service.

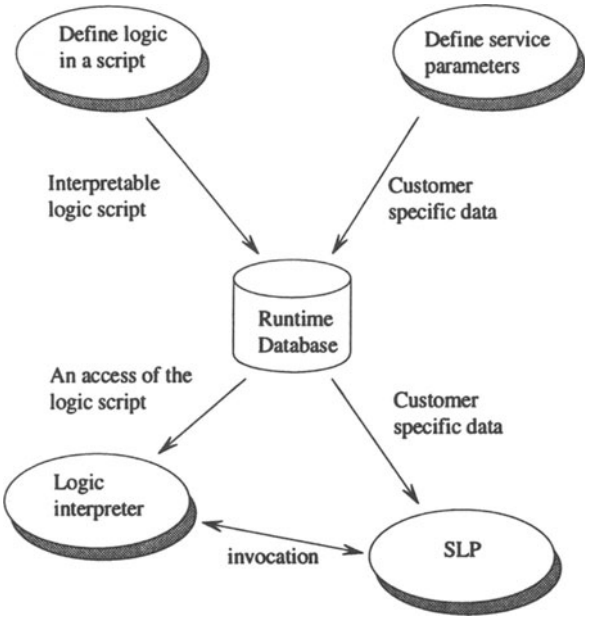

Figure 5 The two possible approaches for service customisation

The two parts of the Customisation level are targeted to two different roles. Since the first approach, script based logic definition, is more demanding than the other, service parameter based customisation, it should be used by personnel whose expertise is close to that of service designer's. Since the scripts can be defined by e.g. graphical editor or as ASCII scripts, their structure cannot be verified as an online task during the editing session. This then suggests that the customisation tools must provide a possibility for simulation of the script logic produced. Therefore, this approach can be seen as a direct extension of Service construction level.

The second part of the Customisation level is utilised according to a different scenario. In this approach, the service customisation data is given through a separate screen, e.g. a decision table that provides only a limited set of combinations all of which can be verified before service deployment, during the implementation of the management application. This approach is then clearly aimed to be used by sales personnel during or immediately after the sales situation.

The important feature of service customisation is that it can be used to extend the life cycle of the actual service logic program. For example, if the service consists of certain basic functionalities and invocation of the logic script interpreter, the service behaviour can be tailored afterwards with capabilities that weren't identified during service requirement analysis. The obvious implementation method for this extension is to define the new functionalities as subscriber or subscriber group specific logic scripts.

7 CONCLUSIONS

In this paper we discussed how the tools used for service creation can be divided into three levels according to their functional capabilities. Each of the levels is essential if the IN service creation is to be utilised in full scale. However, each of the levels can be implemented so that they are instantiated as a stand-alone tool with the support of lower levels as ready-made or tailored solutions.

The SIB programming level is intended to provide the basic framework for creating the SIB components which are then utilised at the Service construction level. As the task of SIB implementation is the critical part of service creation, it is probable that most of the effort needed for introducing services is spent during SIB requirement analysis, design, and implementation. Furthermore, it is important to realise that SIBs must not be seen as single, indvidual components. Instead, they should be considered together as an integral library that then can be used as modules for service logic program implementation.

The Service construction level is intended to provide the editing functionalities of tools that are normally considered as "traditional" SCEs. The service implementation bases directly on the concept of SIBs and their functionalities defined in the SIB programming level. It is anticipated that in this level key factors are the ease of use and how much service introduction time can be speeded up, if all the required SIB components have already been deployed into the SIB library.

With the service customisation, the basic idea is to tailor the behaviour of the service after the actual deployment. There are two basic approaches: in the first one the customisation can be based on scripts that are interpreted by the service logic program. The other possibility is to define dedicated screens that then are used for defining the input parameters for the service. The former approach is suitable for subscribers or subscriber groups that require flexible behaviour and the requirements cannot be determined accurately during service implementation. The latter approach provides only a finite set of different possibilities and therefore testing of customised behaviour can be carried out during service implementation tests.

9 REFERENCES

Bihain, A., White, J. (1994) Service Creation Environment as a software development platform, in *Intelligent Networks* (ed. Harju, J., Karttunen, T.), Proceedings of the IFIP workshop on intelligent networks 1994, IFIP, Chapmann & Hall, Padstow.

Capellmann, C. et al (1996) The P103 Service Creation Environment Model, in *Intelligent Networks and New Technologies* (ed. J. Nørgaard, V. B. Iversen), IFIP, Chapman & Hall, London.

ITU-T (1993) Q.12xx- Series Intelligent Networks Recommendations, ITU-T.

Knight, C. (1994) Service creation from IN to mobile and broadband, in *Intelligent Networks* (ed. Harju, J., Karttunen, T.), Proceedings of the IFIP workshop on intelligent networks 1994, IFIP, Chapmann & Hall, Padstow.

Kinnunen, J. (1997) Intelligent Networks (IN) Service Creation Language, M.Sc. Thesis, Lappeenranta University of Technology, 1997.

Kolehmainen, M. (1996) Nokia SCE - An Architecture for a Lightweight SCE, in *Intelligent Networks and New Technologies* (ed. J. Nørgaard, V. B. Iversen), IFIP, Chapman & Hall, London.

Laakso, R. (1995) Object Oriented Implementation of Intelligent Network Service Logic Programs, M.Sc. Thesis, University of Helsinki, Dept. of Computer Science, 1995.

Locher, M. (1997) Total Service Creation within IN, Intelligent Networks Summit 14th - 16th May 1997, London.

Thörner, J. (1994) Intelligent Networks, Artech House, Norwood .

Turner, G., D. (1995) Service Creation, *BT Technology Journal*, Vol. 13, No. 2, Ipswich, April 1995.

10 BIOGRAPHY

Mikko Kolehmainen was born in 1966. He received his M.Sc. in CS from the University of Helsinki, Department of Computer Science in 1992. He joined Nokia Research Center in 1993 and Nokia Telecommunications in 1996. During his time with Nokia he has been working with issues related to IN and especially to the service creation. Currently he is also a Ph.D. student with the University of Helsinki, Department of Computer Science.