17
Intelligent Network Service Creation: an ITU-T CS-1 view

Sehyeong Cho, Choongjae Im, JeongHun Choi
Electronics and Telecommunications Research Institute
161 GaJung, Yusong, Taejon, 305-606 Korea
Tel: +82-42-860-5630, Fax: +82-42-861-2932
E-mail: (shcho, cjim, jhchoi)@dooly.etri.re.kr

Abstract

ITU-T, the international standardization body, recommends a 4-plane model of intelligent network. This paper describes how we used the model for rapid prototyping, and discusses the merits and drawbacks of the intelligent network as described in the ITU-T Q.1200 series recommendations, in particular, CS-1, in terms of automated service creation. We focus on the middle two planes, the Global Functional Plane (GFP) and the Distributed Functional Plane (DFP). We also discuss formal constraints on SIBs and the relationship to automated service creation.

1 INTRODUCTION

ITU-T is currently standardizing the advanced Intelligent Network (IN) and also studying service creation. ITU-T developed the Intelligent Network Conceptual Model (INCM) to provide a framework for the design and description of each IN Capability Set (CS) and target IN architecture. With the INCM, IN is modeled by using four planes, each an abstract view of IN-structured network: Service Plane (SVP), Global Functional Plane (GFP), Distributed Functional Plane (DFP), and Physical Plane (PHP)[1]. We shall not be concerned too much with the service plane in this paper, and will focus mainly on the middle two planes. We shall be discussing the merits and drawbacks of the INCM, in particular, IN CS-1, in terms of modeling and implementation of an IN-structured network as well as service creation in the framework of INCM.

Section 2 describes our prototype implementations. Some SW engineering issues will be touched upon. Section 3 will discuss pros and cons with CS-1 in terms of service creation.

D. Gaïti (ed.), *Intelligent Networks and Intelligence in Networks*

Section 4 summarizes the findings and discusses the future of function-based service creation and control.

2 PROTOTYPING THE ITU-T IN

The prototypes described in this section represent part of a series of rapid prototyping activities, that are intended to give a comprehensive evaluation on the ITU-T recommendations, to give a hands-on experience with service creation and service control, and to create the actual implementation of IN as the last step in the prototyping. The prototyping started from the service plane, then moved to the global functional plane, then back to SVP, then to DFP, and so on. This is a very intuitive way of mapping the prototyping cycle to the INCM [3]. Experience and information gathered at lower planes are fed back to higher planes, in order to reflect to the local standard and to implementation.

Since recommendations for the service plane had very little to say about how to specify services, we used a conventional method of using a service requirement specification template (with the service name, the prose description, the usage, network capability requirements, and so on).

The GFP describes standard reusable units of service functionality, referred to as service-independent building blocks (SIBs), independent of how the functionality is distributed in the network. SIBs can be combined with global service logic on the GFP to realize services and service features. The concept of SIBs is of primary interest to service designers and serves as an abstract service modeling tool. These SIBs may be used in service creation processes, though IN CS-1 did not address this aspect and provides no standardized capabilities to support such processes [2]. In the global functional plane, all network functions are represented as if they are provided by a point entity. The way how these functions are actually performed in the network is not disclosed at this plane. Information hiding of this nature is extremely important because it implies those who work at this level don't have to have expert knowledge. One such class of people are service logic designers. Relieving them of the burden to be network experts enables quicker service design, as would be required in order to cope with the increasing number of services and tight time-to-market.

For the purpose of initial modeling of the intelligent network and putting the standard SIBs into test, the "global functional" concept helped us a great deal by reducing the complexity down to a manageable size. The reason is all the same: we didn't have to know the detail of communicating multiple entities. For instance, a user interaction SIB can be realized by a process in a computer (not a networked entities) by using audio device for announcement and keyboard for entering digit strings. In our case, we didn't even have to write a complex program, but instead we took advantage of the power of the inference engine of a production system to do the reasoning [10]. With a production system, the domain knowledge is encoded as assertions that represent contingent facts and rules that represent the "law of nature" in the domain of interest. The "programmer" does not specify how a task is performed. For instance, SIB firing rule can be represented as follows (with a little syntactic sugar):

> If SIB A finishes action at logical output X
> and an arrow connects from X to the logical start of SIB B,

then SIB B starts execution.

This rule fully encodes the semantics of SIB chaining, in other words, dictates the meaning of a SIB connected to another SIB by an arrow, and that, in a machine-executable form. The following is an excerpt from a Global Service Logic specification that uses Distribution SIB.

```
(make   SIB
        ^TYPE Distribute
        ^SIB-ID sib1
        ^state   inactive
        ^SSD-ID          ssd-1)
(setf   (distribution-type ssd-1) 'C)  ;;; time of day
(setf   (numberof_branches       ssd-1)   2)
(setf   (timeofday ssd-1)
        '(((0.0)(17.30))(17.30)(24.0))))  ;;; Midnight to 5:30PM -> branch 1
                                          ;;; 5:30 to Midnight -> branch 2
...
(make   GSL-CHAIN
        ^from-sib sib-1   ^to-sib sib-2
        ^logical-out 2)
(make   GSL-CHAIN
        ^from-sib sib-2
        ^to-sib sib-3
        ^logical-out Error
        ^logical-in  Clear-call-POR)
```

We built a graphic editor for creating GSLs and entering service data (see Figure 1) , and a translator that translates the graphical representation of GSLs into the form of rules and facts. These are fed to the rule-based GSL simulator. User-noticeable events such as ring, tone, announcement, etc. are given to the call/service animation engine to simulate the audio/visual effect. User interaction is input by simulated keypad and mouse, then fed to the translator, to be represented as a fact. Put together, they constitute a service logic design and validation tool.

In the distributed functional plane, the network is viewed as discrete logical groupings of functionality called functional entities (FEs). FEs realize service functionality by executing functional entity actions (FEA's). The distributed functional plane hides physical implementation details such as supporting protocols, or the physical location of functional entities, and the like, thus making it easier to analyze and to model. As services are represented by combinations of SIBs at GFP, each SIB is realized as one or more Functional Entity Action (FEAs) performed by Functional Entities (FEs). Some FEAs represent local actions, while others represent communication, or "information flows" among FEs.

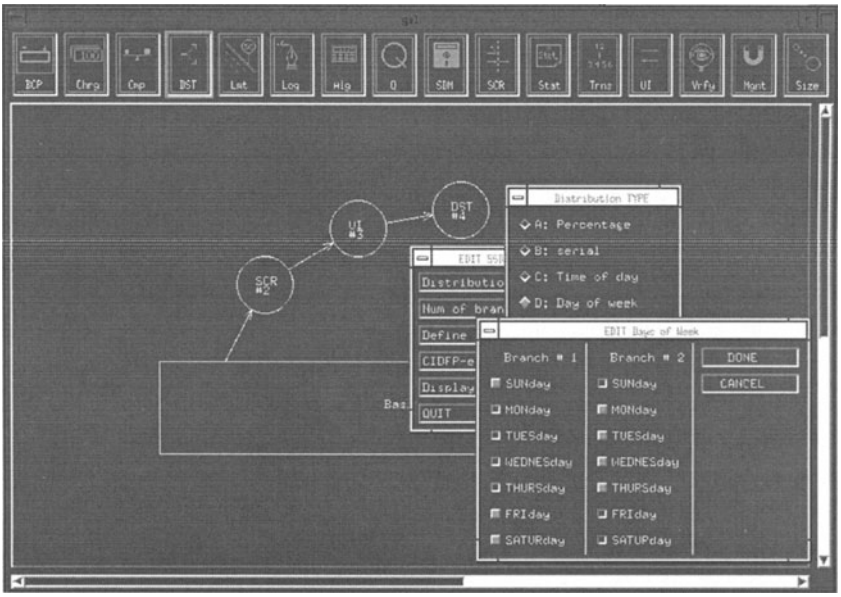

Figure 1 Editing a GSL.

A service logic at the DFP prototype, which we call a distributed service logic (DSL), is composed of instructions in a proprietary applications programming interface (API) that modeled the functional entity actions without the actual underlying transport, such as TCAP. The DSLs in turn are created by translation from GSLs, which are created by a service logic editor implemented in the first stage of the prototyping.

Most part of the translation went straightforward, but we did have trouble, which we shall allude to in the next section. Also, in order for service simulation, we needed some information that seem to pertain to the physical plane, such as database schema or SSP trigger information such as DP criteria.

Each DSL is statically and dynamically checked for verification. In static verification, we check for static errors of functional entity actions and the information element of information flow. After static verification, the distributed service logic is executed in the simulation environment with SCF simulator (a sort of virtual machine that executes the instructions in the DSL), a SSF/CCF simulator, SDF simulator and an SRF simulator. Figure 1 shows a snapshot from the simulation process of the Universal Personal Telecommunication (UPT) service.

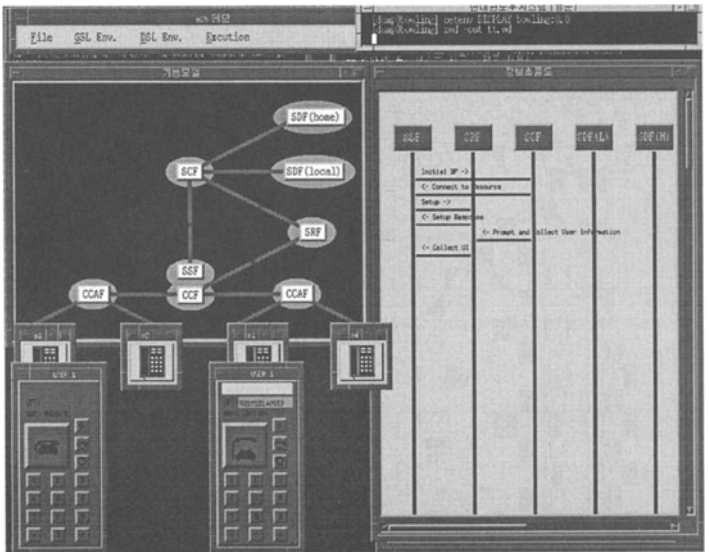

Figure 2. Service Simulation process of DSL.

3 ITU-T SIBS FOR SERVICE CREATION

As we alluded to in the previous section, the viewpoint distinction provides a easier way of service creation. A service designer draws a global service logic, without the need of knowing all the network detail - no information flows, no protocols, no database. All the dirty work is done by a machine - hopefully, and service logics and databases etc. are automatically produced. However, there are things to be resolved, before getting the dream come true.

One of the conceptual problems is that the basic call process is too much DFP-oriented. Even though basic call process is practically very different from other SIBs, the global functional plane should have hidden such detail. Currently, with CS-1, the global functions are conceptualized as "basic call process," which represents the functions provided by CCF/SSF, and "other SIBs," which represent the functions provided by SCF and other functional entities. That is, users of the recommendation Q.1213 must somehow understand the interaction between SSF and SCF, which is an obstacle for non-experts to using SIBs for service creation.

One way of getting out of this trouble is to reorganize the set of POI's and POR's into a set of SIBs. For instance, "Continue with new data" POR can be replaced by "Connect to a new party" SIB, depending on the context. This way, the viewpoint based on distributed functions can be replaced by the global viewpoint involving user(s) and network, where the network is viewed as a point entity that provides functions necessary for services.

POIs and PORs bear the name because it is presupposed that the BCP initiates (triggers) the service logic and the control will eventually "return" to BCP. However more than one POIs (or PORs) can exist, which entails there should be points at which the (suspended) service logic resumes execution. They differ from POIs semantically.

Then there is the problem of parameter mapping between GFP and DFP (to PHP, of course). Even though some information need not (and should not) be revealed at the global functional plane, there should be at least a source of information that can be used to make some inference to derive necessary information at the lower planes. For example, End-of-call POI in BCP indicates that "a call party has disconnected." In the GFP there is no way of distinguishing a service which is activated when the calling party hangs up from a service which is activated when the called party does (T_Disconnect vs. O_Disconnect). This might not be a problem if GSLs are only used for modeling purpose, but is a problem if they are to be used for formally specifying services. Physical details that should be filled in at deployment time or by a service management process are exceptions.

With current methodology, the SIB instances are specified by service support data (SSD) and Call Instance Data. The distinction between those two are unnecessarily rigid, restricting the flexibility of using SIBs. For instance, in a *Screen* SIB, the screen list indicator (or the list name in CS-1 refinement) is classified as service support data, therefore cannot be changed at runtime. However, there are many applications (e.g., UPT) which require that the database ID change dynamically. *Compare* SIB is another example, where comparison is allowed only between a constant and a variable. Rather than syntactically fixing the type of data (CIDFP or fixed value), it would be better for the user to be able to declare the type when instantiating a SIB.

Data transparency used to be a problem (GSL needed to use screen SIB twice, one for finding out the DB and the other for screening) but is hoped to be solved by using directory services based on X.500.

Error management is elsewhere pointed out to be not useful [5]. Some SCE implementors [6] make use of explicit handling of error cases, but we believe a few default error handling will suffice for all practical purposes.

In order to use GSL for service creation, a GSL should be a formal specification, even if it intentionally hides many network details. However, current SIB specifications lack the formality in data. This is especially true for SIBs that relate to data handling such as *Screen*, *SDM*, and *Translate*.

The DFP is supposed to be independent of physical plane, but sometimes it is not the case. In ITU-T recommendation [8], information flows for user interaction is shown to be varying according to the physical location of IP.

User interaction poses another problem, which is optimization problem. When *UI* SIB is used more than once, straight forward translation into DFP tends to show that connection to SRF can be either unnecessarily maintained or unnecessarily disconnected when another use follows. This is even more complicated when used in combination of if-then-else's such as screen SIB. Peephole optimization [9] should take care of simpler cases, but optimizing complex cases are for further study.

Perhaps the most significant potential obstacle to service creation is the non-monolithic nature of SIBs. Even though the SIBs are supposed to be monolithic by definition, some are only conceptually so. Figure 3 depicts two cases of combining SIBs into service logics, the first monolithic case, and the second non-monolithic. Mapping to distributed functional entity actions are described in simplified pseudo-SDL (state symbols omitted).

In the latter case, it is not possible to compose a service logic program from the specifications of the SIBs in a straightforward manner. The following definitions will be used for char-

acterizing certain properties of SIBs. We shall be concerned only with the service logic for SCF. We will assume, for the sake of simplicity, the SCF action for a SIB is defined as a linear sequence of FEA's (This will be true for each service instance).

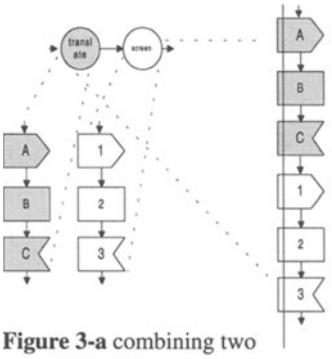

Figure 3-a combining two monolithic SIBs.

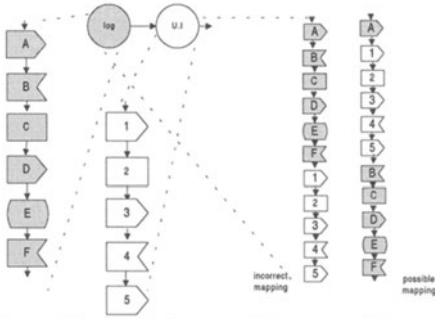

Figure 3-b combining two SIBs, one of them non-monolithic.

Definition 1 [Monolithic SIBs: type 0]:

Let a sequence of actions $\sigma_1, \sigma_2, \ldots \sigma_n$ define the (functional entity) actions necessary for SCF to provide the function defined for SIB *i*. SIB *i* is defined to be *monolithic* if in any combination of SIBs (that is, in a GSL) including an instance of SIB *i*, the sequence $\sigma_1, \sigma_2, \ldots \sigma_n$ will be executed contiguously (i.e., without any action δ_k defined for SIB *j* intervening). We shall call such SIBs as type 0.

If all SIBs are monolithic, a GSL specification itself is the execution sequence in terms of distributed functions, and therefore synthesis of service logic programs from a GSL specification is rather straightforward (see Figure 3-a). However, monolithicity is too stringent a constraint in order for the set of SIBs to define the network capability of interest. Actually some SIBs in ITU-T CS-1 are found non-monolithic. Among CS-1 SIBs, *Charge* and *LogCallInformation* are such examples. Figure 3-b illustrates the point. With definitions 2 and 3, we shall try to relax the constraint.

Definition 2 [Monolithicity of a part of a SIB]:

Let a sequence of actions $\sigma_1, \sigma_2, \ldots \sigma_n$ define the (functional entity) actions necessary for SCF to provide the function defined for SIB *i*. A contiguous sub-sequence $\theta = \sigma_k, \sigma_{k+1}, \ldots \sigma_l \ (1 \le k \le l)$ is defined to be *monolithic* if in any combination of SIBs (that is, in a GSL) including an instance of SIB *i*, the sequence $\theta = \sigma_k, \sigma_{k+1}, \ldots \sigma_l$ will be executed contiguously (i.e., without any action δ_m defined for SIB *j* intervening).

Definition 3 [Quasi-monolithic SIBs: type 1 and type 2]:

1. SIB i is defined to be *quasi-monolithic* if condition 1 or 2 holds: In any combination of SIBs (that is, in a GSL) including an instance of SIB i, all actions which are defined for SIB i are divided into contiguous partitions $\theta_1,...,\theta_k$, (i.e., $\theta_1=\sigma_1,\cdots,\sigma_{i_1}$, $\theta_2=\sigma_{i_1+1},\ldots\sigma_{i_2}$, $\cdots$, $\theta_k=\sigma_{i_{k-1}+1},\ldots\sigma_n$) such that: 1) each $\theta_m(m=1,\cdots,k)$ is monolithic, 2) θ_i precedes θ_{i+1} for all $i=1,...,k-1$ and 3) there exist syntactically identifiable preconditions for each θ_i , $i=2,...,k$. We will call such SIBs as type 1.
2. In any combination of SIBs (that is, in a GSL) including an instance of SIB i, all actions which are defined for SIB i are divided into contiguous partitions $\theta_1,\cdots,\theta_k$, (i.e., $\theta_1=\sigma_1,\ldots,\sigma_{i_1}$, $\theta_2=\sigma_{i_1+1},\ldots\sigma_{i_2}$, $\cdots$, $\theta_k=\sigma_{i_{k-1}+1},\ldots\sigma_n$) such that: 1) each $\theta_m(m=1,\cdots,k)$ is monolithic, 2) θ_1 precedes $\theta_2,\cdots,\theta_k$, and 3) $\theta_2,\cdots,\theta_k$ are mutually independent and can be performed concurrently, each driven by an asynchronous event. We will call such SIBs as type 2.

Translating a GSL into distributed functional entity actions are relatively straightforward, provided that SIBs are monolithic or quasi-monolithic. A type-1 SIB can be broken into k pieces and reordered according to the "syntactically" identifiable preconditions. Type-2 SIBs can be implemented by using $N+1$ service logic program processes, one for the leading monolithic sub-sequence, $N-1$ for the concurrently executing sequences (possibly after the SIB meets the logical end), and one for coordinating the N SLP's. Figure 4 depicts these two situations.

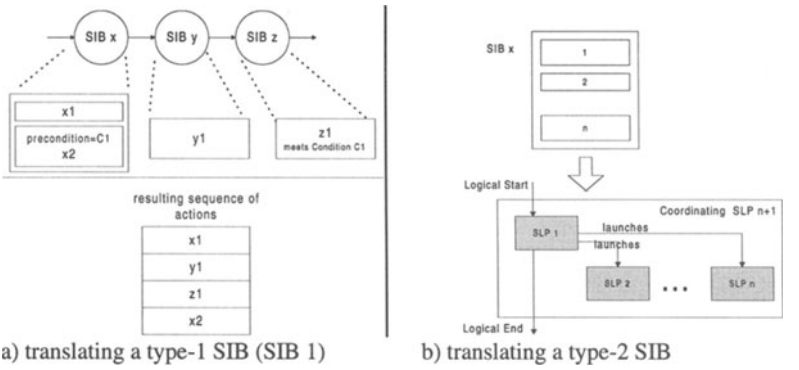

Figure 4 Translating SIB sub-sequences into SLPs.

Currently all CS-1 SIBs seem to be at least quasi-monolithic. However in the future when we extend the SIB sets or the capability sets, it would be advisable to be careful either to check such constraints, or extend the constraint beyond, within manageable complexity. Currently

the most obvious extension to the SIB constraint would be to allow θ_2,.. ,θ_k in definition 3 to be quasi-monolithic in turn. The runtime snapshot would look like Figure 5.

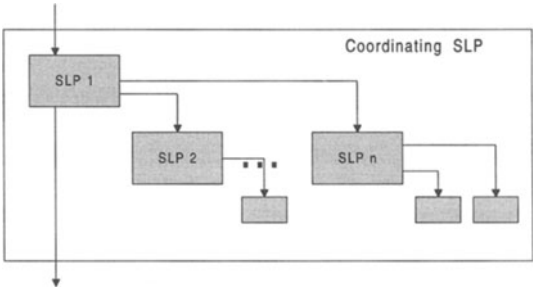

Figure 5. The most obvious extension of the SIB constraint

4 CONCLUDING REMARKS

The relation between INCM and rapid prototyping was described in this paper in terms of our experience in a series of rapid prototyping. We also touched on some issues in CS-1 (and beyond) in relation to service creation. Despite some drawbacks discussed in this paper and others, the 4-plane model provides a good basis for service creation in that it enables proper information hiding for people involved in, and a relatively sound basis for automatic translation from GSL to lower-plane representations, such as SLPs.

It must however be stressed that extension of SIBs and/or capability sets must take precaution as to the characteristics of those SIBs in order to use them for automated service creation. The foregoing classification of SIBs is by no means complete or exhaustive, but it merely suggests the need of such a formal analysis of the relation between SIB characteristics and the complexity of automated service creation.

Recently, SIB concept is being challenged by object-oriented school, since IN SIBs are not object-oriented, but basically functional. It might not look fashionable, considering that OO is such a buzz-word these days. However, unless OO is a panacea, there are domains that is suitable and domains that are not. In general OO paradigm applies well to domains where the world consists of "things," such as a management domain. In case of IN, the main concern is functions, not objects. Thus chances are, you might end up creating objects which are a bunch of functions in disguise.

Functional paradigm is not dead yet, but we should provide some formal guidelines as to how to safely extend the capability in order to provide a service-creatable platform for CS-2, CS-3, and beyond.

5 REFERENCES

[1] ITU-T draft Recommendations Q.1203, 1993.
[2] James J. Garrahan, Peter A. Russo, Kenichi Kitami, and Roberto Kung, " Intelligent Network Overview," IEEE Communications magazine, Vol. 31, No. 3, pp. 30-36, March 1993.
[3] Sehyeong Cho, "A Rapid Prototype of IN Using a Production System," Intelligent Network '94 Workshop, Ramada Renaissance Hotel, Heidelberg, Germany, Section 212.2, May 24-26, 1994.
[4] Masaya Akihara, Keiichi Shimizu and Shuji Ito, "An Implementation and Evaluation of Service Creation," Intelligent Network Workshop '95, Congress Center, Ottawa, Canada, May 9-11, 1995.
[5] Ty Chang, "Discussing the Weaknesses of the Standard Service Independent Building Blocks," 3rd International Conference on Intelligence in Networks, Bordeaux, France, pp. 67-72, Oct. 11-13, 1994.
[6] Martin H. Petruk, "Experiences in applying the ITU CS-1 Intelligent Network Conceptual Model (INCM) to service Creation," 3rd International Conference on Intelligence in Networks, Bordeaux, France, pp. 134-139, Oct. 11-13, 1994.
[7] ITU-T Recommendation Q.1213, May 1995
[8] ITU-T Recommendation Q.1214, May 1995
[9] Aho A. et al, Compilers - principles, techniques, and tools, Addison Wesley, 1986
[10] Lee Browston et al, Programming Expert systems in OPS5, Addison Wesley, 1986

6 BIOGRAPHY

Sehyeong Cho Received B.E. and M.Sc. from Seoul National University, Seoul, Korea in 1981 and 1983, respectively. He received Ph.D. in Computer Science from Pennsylvania State University, USA in 1992. He has been a Software Engineer at Electronics and Telecommunications Research Institute (ETRI) since 1984, and currently is a senior member of technical staff and project leader. His research interest includes Telecommunication Software Engineering and AI applications to Telecommunication.

Choong-Jae Im was born in Chungnam, Korea in 1968. He received the B.S. degree in computer science from Chungnam National University, Taejon, Korea in 1991 and 1993, respectively. Since 1993, he has been involved in Intelligent Network projects.

JeongHun Choi received the B.S. degree from KyeongBook University, and the M.Sc. degree from KAIST (Korea Advance Institute of Science and Technology), Korea, in 1985, and 1987, respectively. He is a Senior Member of Technical Staff of ETRI Intelligent Network Service Section. His current research interests include Service Creation Environment, Feature Interactions, and Formal Description Techniques.