ソフトウェア工学　89−21
（１９９2. １2. １6）

INにおけるサービスロジックが伝達網を制御する部分に着目した検証方法の提案と実装

堀 正弘、伊藤 光恭、福永 勇二、永石 勉
日本電信電話株式会社　ソフトウェア研究所

インテリジェントネットワーク（ＩＮ）では、公衆網サービスを不特定多数の開発者が作成するようになるため、サービスの信頼性を高めることが非常に重要である。ＮＴＴが提案しているＩＮモデルでは、サービスは、物理的リソースを抽象化したオブジェクトである仮想リソース（ＶＲ）を操作することによって実現される。本文では、サービスを記述する、ＶＲの操作順序列の正しさを検証するために、正当性、完全性、非冗長性の、３つのサービスの性質を提案する。また、これらを静的に検査する方法と、これを実現したツールについて報告する。この検証により、信頼性の高いＩＮサービスが、短期間で開発できることが期待される。

A VERIFICATION METHOD FOR SERVICE SCRIPT OF THE INTELLIGEN T NETWORK

MASAHIRO HORI, MITSUTAKA ITO, YUJI FUKUNAGA and TSUTOMU NAGAISHI

NTT Software Labratories
NTT Shinagawa TWINS Bldg. 1-9-1 Kohnan Minato-ku Tokyo 108 Japan

The current verification methods used for the Intelligent Network (IN) are based on communicating process technologies. In this paper, an IN model is examined to study verification technologies which come from the characteristics of the model. In the proposed model, the IN architecture abstracts physical resources in the transport network and a Service Logic Program (SLP) controls the virtualized resources (VR) to achieve IN services. The characteristics of SLP control over the virtualized resources are examined, the verifications are classified, and a verification system is developed. Using this verification system, IN services high in quality and reliability can be achieved even by inexperienced programmers in a relatively short time.

1. Introduction

1.1 Importance of Verifications of IN services

Currently, the IN concept is being extensively studied with the aim of developing more intelligent network services and providing high quality intelligent telecommunication service in the near future. Against this background, verification technologies have emerged to play a very important role in the quick development of high quality, reliable IN services. The verification technologies currently in use are based on general communication models and so do not use the special properties of IN models. In this paper, we report an improved verification technology that uses the characteristics of an IN model, making it a better verification technology for IN application to service.

1.2 The IN model

In this section, we describe the proposed IN model based on [3],[4].

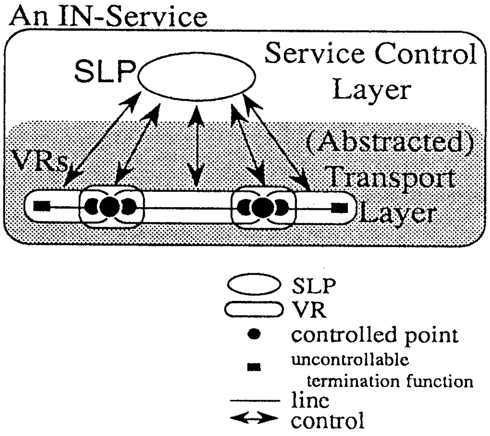


Fig.1 Relationship between SLP and VRs

1.2.1 Abstraction of Physical Network

In this model the concept of virtualized resources (VR) is introduced. VRs are abstractions of control points in a transport network. The relationship between VRs and on an SLP is illustrated in Fig.1. Each VR is represented by objects which are defined by the finite state machine (FSM). The SLP describes services by controlling these VRs. One SLP existing in a logical network and control VRs are basically executed in one process to achieve one service. In other words, an SLP will not interact with another SLP to achieve the service.

1.2.2 Properties of VRs

VRs are conceptual resources which include the physical resources in the transport network. From the definition of VRs, we can conclude that VRs have the following properties.

P1: VRs are defined by FSM.
P2: State transitions are not initiated through the communication path that exists between two or more entities.
P3: SLP and VRs communicate asynchronously.

2. Classification of verifications in IN

2.1 Special Properties of IN services

In describing IN services, an SLP creates, joins, splits and frees many VRs. The most difficult problem in writing an SLP is managing the status of VRs, which changes according to service status. Inexperienced SLP programmers who do not know the VRs' state transition are especially prone to making mistakes in using VRs when writing an SLP. In the IN concept, since an SLP is open to a customer, errors are likely to be committed frequently. Therefore, a verification technology which takes the SLPs' control part into consideration is most effective for achieving IN services. More concrete, under the condition that the FSMs of VRs are correct, we check that the state transition of all VRs resulting from the SLP control information are consistent with the VRs' FSM definitions.

2.2 Verification

Taking the properties described in 2.1 into consideration, we classify the verifications which are difficult to check in writing SLPs and which the services must obey. The results can be summarized as follows:

2.2.1 Consistency

This property refers to the fact that all the signals which the SLP sends to VRs must be accepted by the appropriate VR. In our verification technology, we check whether the SLP sends the signal to a VR that can process the signal according to the state transition map.

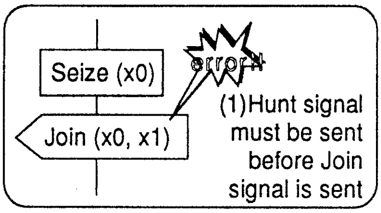


Fig 2 (1) Consistency

2.2.2 Completeness

This property refers to the fact that all the signals which VRs send to the SLP must be processed by the latter. In our verification technology, we check whether all the signals the VRs send to the appropriate SLP processes are defined in the SLP.

2.2.3 Non-Redundancy

This property refers to the fact that each processes described in the SLP must correspond to some signal sent to it by a VR. In our verification technology, we check whether or not the SLP has any non-executed codes.

3. Strategies

3.1 Assumptions and Limitations for Efficient Verification

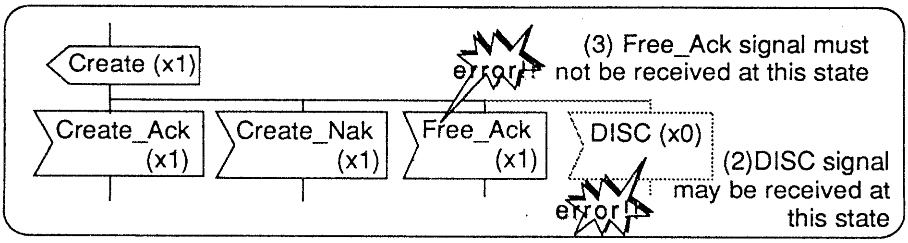
For practical verifications, we assume the following properties of VRs are added to the three properties described in 1.2.2. These assumptions make the model precise, but do not affect its practicability.

P4: The signals which are sent by an SLP process are never lost.

P5: The VRs are controlled by only one SLP.

For further discussion, we define the states of an IN service as follows:

Def: The state of an IN service is defined as an n-tuple (s1, s2, ..., sn), where s1, s2, ... , sn are the states of all VRs controlled by the SLP in the service. The state of an SLP transits to the next state when the SLP executes a control to VRs.



Create (x1)
Create_Ack (x1)
Create_Nak (x1)
Free_Ack (x1)
DISC (x0)
error!
(3) Free_Ack signal must not be received at this state
error!
(2)DISC signal may be received at this state

Fig 2 (2) Completeness and Non-Redundancy

To avoid the rapid increases in the number of states of the IN service when the SLP has loops, the service to be verified must satisfies the following condition:

P6: When the SLP of an IN service has loops, the states of the IN service, i.e., the n-tuple of VR states, at any points in the SLP are always exactly the same independently on the number of iterations of loops to be executed.

From the P6, we can ignore the increase of states caused by existence of the loops in the SLP and the verification procedures become easy.
In practice, this constraint results in no serious loss in the ease of describing the SLP of IN services.

3.2 Verification Method

VRs are entities that communicate to the SLP. They are considered a kind of process with the properties outlined in P1 to P5. Thus, the verification of service can be considered as the verification of a protocol shared among several individual processes. This, however, is not a practical method because the number of global states increases sharply if the number of VRs is increased.
We therefore use the following method to verify IN service. First, we pick up the point at which the control flow branches and point at which the SLP executes control to the VRs, and make them a state transition tree. Second, we trace the tree and compare it with the state transition maps of the VRs. Finally, we verify the correctness of the SLP using the criteria described in 2.2.1 to 2.2.3.

Table 1. Functions of Parts of Verification System

part	functions
front end	interprets SLP written in SDL/PR, and creates state transition tree
back end	traverses the state transition tree, compares it with state transition maps of VRs, and verifies it.
DB	stores data used by the front and back ends, such as state transition maps, different sorts of control signals, and so on.

4. System Overview

Our system for verifying the properties described in 2.2.1 to 2.2.3 is shown in Fig.3. This system, which is described in C language and runs on Unix, consists of three individual parts : the front end, the back end, and the DB. The front end interprets a given SLP program written in SDL/PR and creates a state transition tree. The back end traverses the state transition tree, compares it with state transition maps of VRs stored in the DB, and achieves verifications. The DB stores data used by the front and back ends, such as state transition maps of the VRs, different sorts of control signals, and so on. With this structure, we can easily deal with any other SLP representation, such as C language, by replacing the front end corresponding to that representation.

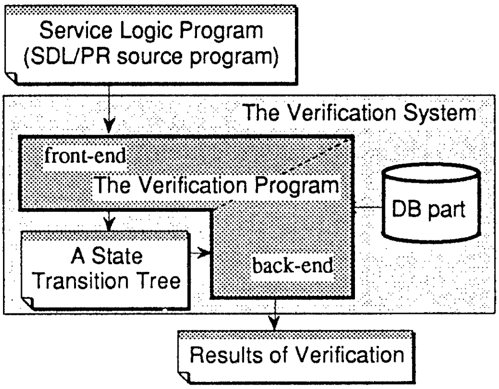


Fig.3 Overview of SLP-VR Verification System

4.1 The Front End

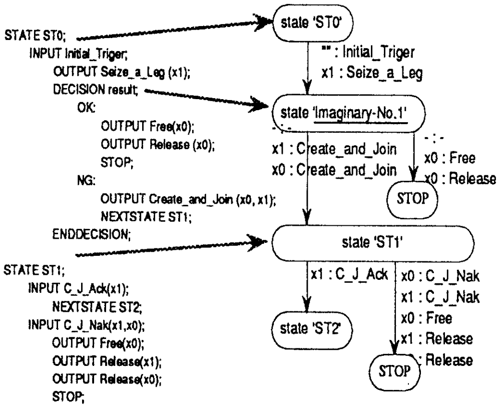
The front end interprets an SLP program written in SDL/PR and creates a state transition tree. This tree consists of state nodes, including a pointer to the state to be transit to and a list of control signals sent to VRs at that state. In this phase, both the VR control points and the SLP control flows, such as branches and loops, must be considered.

4.1.1 VR Control Points

The control signals sent to VRs are represented as either OUTPUT clauses or INPUT clauses in SLPs written in SDL/PR. The front end picks up the signal names that follow those keywords OUTPUT and INPUT and appends them to the signal list if they are control signals to the VRs.

4.1.2 The Control Flow of the SLP

The control flow of the SLP may branch either when the SLP receives the signals from the VRs or as a result of internal variable conditions. The former is represented as INPUT clauses, and the latter as a DECISION clause. The latter corresponds to transitions of the state of an SLP, and it is in this case that the state of the SLP is clearly described in the STATE clause as an SDL program. In the former case, it is not. In either case, state nodes are created.



```
STATE ST0;
    INPUT Initial_Triger;
        OUTPUT Seize_a_Leg (x1);
        DECISION result;
            OK:
                OUTPUT Free(x0);
                OUTPUT Release (x0);
                STOP;
            NG:
                OUTPUT Create_and_Join (x0, x1);
                NEXTSTATE ST1;
        ENDDECISION;
STATE ST1;
    INPUT C_J_Ack(x1);
        NEXTSTATE ST2;
    INPUT C_J_Nak(x1,x0);
        OUTPUT Free(x0);
        OUTPUT Release(x1);
        OUTPUT Release(x0);
        STOP;
```

Fig.4 Transformation from SDL representation to State Transition Tree

4.2 The Back End

In this part, the state transition tree is traversed, compared with state transition maps of VRs stored in the DB part, and verified.

4.2.1 Traverse of State Transition Tree

The traverse of the state transition tree created by the front end part starts from the initial state of the SLP and proceeds by the depth-first-search method. One of the branches is selected to be traversed next in order of appearance in the SLP. If either the end of the tree or a loop structure is found, the system backtracks to the nearest branch. Every state node has a flag which holds the state to traversing the subtree. The flag may be in one of three states : 'not-traversed', 'in-traversing', or 'complete'. Loop structures of SLPs may be found when the flag is either in the 'in-traversing' or 'complete' state in traversing the tree.

4.2.2. The Verifications

In traversing the tree, all VR states are traced and stored in the list included in the state node. At every state node, the following verifications are carried out.

(1) Consistency

In traversing the tree, the consistency of the SLP is verified by comparing the current state of every VR and the state transition maps of VRs stored in the DB part. If a signal that can not be accepted in the state transition maps is sent to the VR, it is reported and the verification is continued.

(2) Completeness

The completeness of the SLP is verified in the following way. First, all the signals that may be sent to the SLP by every VR in the current state are picked up from the state transition maps of VRs. Second, the tree is checked to verify whether or not all branches corresponding to the signals exist. If all the branches are not found, it is reported and the verification is continued.

(3) Non-Redundancy

The non-redundancy of the SLP is verified in the following way. All the branches of the tree in the same state are picked up and compared to the state transition maps of VRs. If branches which were never executed are found, it is reported and the verification is continued.

4.3 DB part

The DB holds the data of the state transition maps of VRs, different sorts of control signals, and so on. It is invoked by both the front-end part and back-end part. The state transitions of VRs are defined so that the next state of the VR to be transit is precisely specified corresponding to a control signal. The separation of the DB part from other parts ensures that we can easily deal with the modifications to VR specifications, additions of new sorts of VRs, and so on.

5. Conclusion

We have proposed an efficient method for verification of IN service. A tool utilizing the method has significantly reduced the number of mistakes in SLP programming and the amount of effort put into off-line debugging.

6. Acknowledgments

This work would not have been completed without the support provided by Yoshihiro Niitsu, Takashi Arano, and my colleagues.

7. References

[1] G.Bregant, R.Kung, "Service Creation for the Intelligent Network", Proceedings of the XIII International Switching Symposium, 1990.

[2] A.Okamoto et.al., "A Verification Scheme for Service Procedures, IEICE Technical Report, IN91-109,1991 (in Japanese).

[3] S.Esaki, T.Omiya, J.Kuwabayashi, "Abstraction of Transport Networks and Control MEssages for Intelligent Network", Proceedings of IEEE INFOCOM 1990.

[4] S.Esaki, T.Omiya, N.Shigematsu, "'Abstraction and Control Concepts of Transport Network Resources for Intelligent Network", IEICE, Nov. 1991 (in Japanese).

[5] K.Okada, K.Sata, Y.Kondo, "Hierarchical Software Definition Structure for Intelligent Network", GLOBECOM, 1989.

[6] Peng-Teng Peter NG, "Supporting Service Development for Intelligent Networks ", IEEE JOURNAL ON SELECTED AREAS IN COMMUNICATIONS, Vol.8, No.2, Feb. 1990.