TINA-IN Work Group RFP

IN access to TINA services & Connection management

(IN-TINA Adaptation Unit)

Response of Alcatel, Deutsche Telekom, France Télécom and Lucent Technologies

Version: Final, endorsed by the TAB (2.04)
9.11.1999

Table of Contents

0. Document information

Title: Alcatel-DT-FT-Lucent response to RfP “IN access to TINA services & Connection management (IN-TINA Adaptation Unit)”
Version: Final (2.04)
Date : November the 9th, 1999

1. General Description

1.1. Submitters

In the context of the TINA RFP, Alcatel, Deutsche Telekom, France Télécom and Lucent Technologies decided to join their initial submissions and to provide a common response to the TINA-IN-WG RFP related *to "IN access to TINA services & Connection management (IN-TINA Adaptation Unit)*".

Contributors: : Ulrich Bittroff (T-Nova), Carla Capellmann (T-Nova), Alban Couturier (Alcatel), Hessel Idzenga (Lucent Technologies), Christof Lorang (T-Nova), Warren Montgomery (Lucent Technologies), Jean-Marc Pageot (France Télécom), Frans Panken (Lucent Technologies), Stéphane Tuffin (France Télécom).

Submission Contact Points :

Carla Capellmann
T-Nova Deutsche Telekom Innovationsgesellschaft mbH, Technologiezentrum
FE13 Platforms for Value-Added Services, Network and Service Management
D-64307 Darmstadt
Germany
Phone: +49 6151 83-3070
Fax: +49 6151 83-4221
E-mail: capellmann@tzd.telekom.de

Alban Couturier
Alcatel Corporate Research Center
Ets de Marcoussis UAA/GSA
Route de Nozay
F-91461 Marcoussis CEDEX
France
Phone: +33 1 69 63 11 97
Fax: +33 1 69 63 17 89
E-mail: Alban.Couturier@ms.alcatel.fr

Jean-Marc Pageot
FRANCE TELECOM CNET
/DAC/ARP
Technopole Anticipa
2, avenue Pierre Marzin
22307 Lannion
France
Phone: +33 2 96 05 12 16
Fax : +33 2 96 05 37 84
E-mail: jeanmarc.pageot@cnet.francetelecom.fr

Frans Panken
Lucent Technologies
P.O. Box 18
1270 AA Huizen
the Netherlands
Phone: +31 35 687 4707
Fax: +31 35 687 5954
E-mail: frans@lucent.com

1.2. Background of the proposal

This submission merges the initial responses of Alcatel [1], Deutsche Telekom & France Telecom [2], and Lucent Technologies [3] into one common response to the RfP "IN access to TINA services & Connection management (IN-TINA Adaptation Unit)" of the TINA-IN work group [4]. Considering the initial submissions of the contributing companies clearly substantial parts have found their way into this submission. To reach a common point of view, however, the common response deviates in parts from some of the individual submissions.

The two most important points about this joined submission in comparison to the initial ones are
1. the common agreement on mandatory and optional interfaces of the adaptation interfaces, and
2. the use of Parlay interfaces to define the Call Control interface.

Let us briefly discuss both points.

Mandatory and optional interfaces. The motivation for an IN-TINA Adaptation Unit lies in the benefits expected from introducing TINA in IN. The basic goals identified in this RfP response are:

to enable rapid introduction of new services (1)
on top of different call-based networks (2)
(also) by 3rd party service providers. (3)

Please note that in (2) on the one hand we restrict the RfP answer mandatory part by aiming at call-based services, on the other hand we extend the requirements on the IN to TINA mapping by aiming at a service level that is independent from Intelligent Network Application Protocol (INAP). The RfP answer optional TINA usage adoption should covers any network architecture, whether it is call or session based.

These basic goals lead to the mandatory requirements for the adaptation unit are:
1. *Generic access.* For the introduction of a new service it should not be necessary to change the access part of the Adaptation Unit. As a consequence, the access part must take care of all different access cases. (1)
2. *Tight control of network mechanisms outside the generic access.* The question of how to design a service is left to the service provider. This means that the service specific code, like the SsUap, is not part of the Adaptation Unit. (1), (2), (3). This tight control is expected to assure the service independence in the IN TINA Adaptation Unit usage.
3. The optional service interface of the ITAU complies with the TINA service session model and requires the presence of the SsUap interfaces. The usage part of the service "hear and feel" is then implemented in the SsUap, which can eventually be downloaded by the AU.

To meet these two requirements the IN-TINA Adaptation Unit must have the following ***mandatory interfaces*** in addition to the INAP interface towards the switch (see Figure 1):
- **Access interface.** To meet the requirement *Generic access* we propose to use TINA Ret Access.
- **Call control interface.** For the requirement *Service independence* obviously we can not use TINA Ret Usage. Consequently an alternative interface is needed to control the call (see below).
- **Management interface.** Aside from general configuration issues, this interface is needed when a new service is introduced, e.g. to define which kind of access session should be used for the new service.

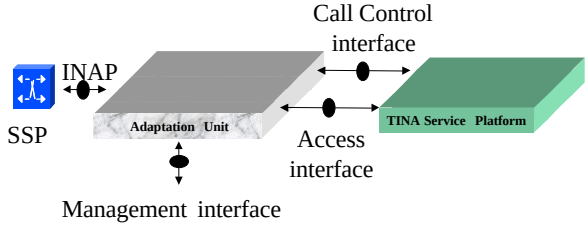

Figure 1. Mandatory interfaces of the adaptation unit.

Although we propose the use of a Call Control Interface in usage, this does not exclude the use of TINA Ret Usage. If a service designer wants to benefit from the service architecture as defined in TINA, the adaptation unit then comply in usage part with the tradition service session interfaces. In such a case the IN TINA Adaptation Unit is compliant with the TINA usage, which is its optional interface. However, the SsUap can still be developed on top of the Call Control Interface.

In the initial submissions the use of a Call Control interface has already been suggested and requirements have been listed. In order not to diverge from existing standards, we propose to be close from the Parlay generic call and CS-1 call control interface as they seem to meet the requirements. Of course, every interface begins with the usual TINA prefix “i_”, but the object model is preserved.

To summarise: To make TINA applicable in legacy IN networks we propose a pragmatic approach for an IN-TINA Adaptation Unit (ITAU) based on two main requirements, *generic ac*cess and *service independence in usage part,*, resulting in an Adaptation Unit with TINA Ret Access and call control interfaces inspired by Parlay as interfaces towards the TINA service platform.

2. Business model

In this chapter, the use of the Retailer Reference Point as the ITAU interface is explained, as well as the ITAU and services position, in business model terms. The Retailer Reference Point of the ITAU is composed of

- the Retailer Reference Point access part
- the Call Control Interface as a basic usage part, and
- the TINA Retailer usage part as an optional usage part.

2.1. Business roles

To identify the business roles involved, a simple 800-service is used as an example. In the Intelligent Network, a service user dials the 800 number, the service logic checks its identity and location, enabling some call restrictions, and finally redirect the call to the most appropriate destination. This service logic can be divided in two parts: first an authentication, and second the acceptation of the call and the routable number transmission.

Obviously, the behavior of the service user is quite similar to the behavior expected from a TINA consumer. Here, it must be kept in mind, that the service user's terminal does not fit the needs of a TINA customer premises equipment (CPE).

As stated above, the service can be divided in two parts: The part that deals with the authentication can be performed by a stakeholder playing the role of a TINA retailer, the other part dealing with the acceptation of the call and the routable number transmission can either be performed by a TINA Retailer or a TINA third party service provider.

A stakeholder playing the connectivity provider role might be requested to establish a communication relationship between the service user and the destination.

As previously mentioned, the terminal of the service user does not fit the needs of a TINA CPE. To fit its needs, the terminal must be able to request directly methods of other TINA stakeholders using a distributed processing environment (DPE). This direct request of methods is not feasible using a telephone in fixed or mobile networks. The service user's terminal is "non-DPE enabled". Therefore, the ITAU is used to enable the service user's terminal to access the DPE. The ITAU extends the service user's terminal to a TINA CPE. It offers a proxy of the service user to the TINA system.

2.2. Adoption of Retailer Reference Point

The retailer reference point is chosen as reference point between the ITAU and the retailer. Below, the reasons for this choice are explained in detail by comparing the high level requirements for the consumer and the retailer business role with the usage of IN services by service users. It is also explained, why the connectivity reference points ConS and TCon are not used.

The retailer reference point is normally divided into two parts: the access part and the usage part. The access part used by the ITAU remains rather unchanged. For details refer to section 3 "Information model" and section 4.2 "Access Interfaces". Instead of the usage part, the following interfaces are recommended:

- The Call Control Interface is defined in this document as a telephony dedicated interface. It is used instead of the Retailer Reference Point's usage part as it was defined in TINA. For details refer to section 4.3.

- The Retailer Reference Point's usage part corresponds to the reuse of the Retailer Reference Point's service session and an adaptation of the Retailer Reference Point's communication session. For details refer to section 5.2.

2.2.1. High level requirements for consumer and retailer business role

TINA defines the following high level requirements for the consumer business role [5]:

1. obtaining location of retailers, service providers and other consumers,
2. (de)registration at retailers,
3. initiating service relationships that include service providers and other consumers,
4. indicating availability to retailers (for receiving invitations),
5. accepting downloads from retailers to upgrade the interaction capability with the retailer.

And for the retailer business role, the following high level requirements are defined:

6. manage (de)registration to obtain various services (including person-to-person communication, if desired) by consumers,
7. manage (de)registration to provide various services by third party service providers,
8. authorisation prior to usage,
9. maintenance of session-level user service profiles and treatment policies,
10. session management, communication to establish and maintain the association list of parties and resources that partake in a session with session owners and session policy information for the purpose of establishing access to the session,
11. manage downloads to consumers and service providers to upgrade the interaction capability with them,
12. collecting accounting information for the purpose of billing, in the general, for each invoked service (including network connectivity) as well as for the services of the retailer (optional)

2.2.2. Use of IN services

If a service user wants to use a service, the Service Switching Function (SCF) has to decide whether it must invoke the Service Control Function (SCF) to provide this requested service or not. The criterias when to invoke the SCF are described in [15]. Assuming that sufficient criteria are met, the SSF queries the SCF to start a Service Logic Processing Program (SLP). This SLP controls and monitors the SSF in order to provide the requested service in a well defined way. Regardless of the functions implemented by the SSF, it is the service user who requests a service. To be honest, he might not know, whether the requested service is realised using the IN. And he might request the service explicitly by dialling an IN service number (+800 for example) or implicitly by calling a second user who has subscribed a service like black list or white list.

In this scenario, the service user obtains the location of the retailers (1), the service providers and other consumers by using the telephony system just like a TINA consumer. But, the technology to obtain the location is different. In the telephony system signalling systems are used, whereas in TINA a kernel transport network based on a middleware will be used. In IN, (de)registration at retailers (2) has been done by subscribing to a phone company and by getting a phone number (E.164 address). (De)registration at retailers in TINA is quite similar to book-in and book-out procedures in mobile networks such as GSM. The indication of availability to retailers for receiving invitations is denoted implicitly by having got a phone number or explicitly by defining black lists and white lists by the IN service user for example (4). TINA invitations can be mapped to an attempt to establish a connection. As described in the IN service scenario, the service user initiates a service relationship that might include other consumers or even service providers (3), e.g. tele votes. In IN, if user-network-interface is upgraded, the service user needs a new customer premises equipment. Downloads of software is not needed / not supported via the current user-network-interface (5, 11).

To sum up, the behaviour of the IN service user is quite similar to the behaviour of a TINA consumer. From a high level point of view, the IN service user mostly meets the high level requirements of a TINA consumer. There are only differences in the technology used to access the network (signalling system vs. kernel transport network) and in the intelligence and supported features of the terminal equipment (dump telephone vs. home computer).

In IN, the management of (de)registration to obtain various services (including person-to-person communication, if desired) by consumers (6) has been performed off-line yet. But the integration of this feature in the current IN

will be very advantageous. The management of (de)registration to provide various services by third party service providers (7) is out of scope here. It is part of the relationship and interactions between retailer and third party service provider. Authorisation prior to usage is one of the major advantages of TINA (8). In IN, there are also service examples where authorisation of service users prior to the usage of the service is mandatory. The credit card calling service is an example. Maintenance of session-level user service profiles and treatment policies (9) is also done in SCF for specific services (black and white lists). The SCF establishes, controls, monitors or cancels connections. Furthermore, a TINA retailer must support session management, communication to establish and maintain the association list of parties and resources that partake in a session with session owners and session policy information for the purpose of establishing access to the session (10). Both, the SCF and the retailer are invoked to establish communication relationships. The way, how to do that, differs absolutely. The SCF must collect accounting information if a premium rate service [14] is accessed. Also a retailer must be able to collect accounting information for the purpose of billing, in the general, for each invoked service as well as for the services of the retailer (12). SCF and retailer require similar accounting functionalities.

To sum up, the high level requirements defined for a retailer mostly fit the needs for an SCF. There are only differences in the technology how to access the service (signalling system vs. kernel transport network) and from whom it is accessed. In IN, the service is accessed via an SSF on behalf of a service user. In TINA, a service is directly invoked (after establishing an access session) by the consumer using the kernel transport network.

This leads to the following conclusions:

1. ***The service user fulfils the consumer role if some functionality is added like access to the kernel transport network and the user interfaces of the retailer reference point. These additional functions should be implemented in the ITAU.***
2. ***To integrate a retailer in the IN, an interface to the signalling system no. 7 is mandatory. This interface together with the necessary protocols and their state machines are provided by the adaptation unit. Then, the retailer remains independent of the underlying technology.***
3. ***Related to a specific call, the ITAU mimics the consumer. Its objects, the consumer domain computational objects, are also called End user Proxy (EUP). The main difference between the EUP and classical end user computational objects is that EUP does not communicate with the rest of the application in the end user domain, but controls the SSF and the Specialised Resource Functions (SRF) through INAP.***

2.3. Non adoption of connectivity Reference Points

In the previous chapter, basic rules have been mentioned to define the counterpart of the IN mechanisms in the TINA formalism, in order to describe mapping rules. The TINA Ret reference point has been clearly identified for a mapping of INAP to TINA. This chapter explains why only Ret RP is used and why ConS and TCon are not proposed in this document.

In the first approach, where a call control interface is proposed as the retailer RP usage, the connectivity functionalities are offered, and the service does not need the access to the general ConS RP.

In the second approach, based on the TINA service session adoption, the communication session is not strictly adopted, because of its heaviness in this very simple environment. The only thing to do is to send an E.164 to the ITAU, or even nothing, when the call just need to be continued. The richness of the communication session concerning the SFEP NFEP resolution is useless in IN, where the call is already initiated, which implies that the NFEP is already set. The use of ConS RP could be done after a classical communication session, but for maximum flexibility, it is foreseen that ConS RP should not be implemented directly in the ITAU: since the ITAU should only address the telephony network, only an IN layer network co-ordinator should be available on the ITAU. However, the proposition of optimisation considers the whole intelligent network as the TINA terminal. This idea comes from the fact that the user interface is performed by the phone set and the network nodes implementing the SRF, and that the end user's service components are in the ITAU. In this configuration, all hardware and software terminal components are distributed among the whole intelligent network as if the whole IN was a TINA terminal. Moreover, the ITAU just needs to get a simple address in the telephone network, in order to join the originating leg to this destination. This simple mechanism can be covered by a simplified terminal flow connection, connecting the SFEP, representing the IN control leg, to a Network Flow End Point, which is the routable number in the **`connect`** operation.
If the CSM detects the initiating party is in the IN, it will send back to the EUP the address where to redirect the call to, without accessing ConS. According to this new mechanism, TCon is obviously not used at the IN side.

2.4. Business scenarios for IN service

These observations lead us to propose the following model, where the adaptation unit complies with Ret-RP. Quite surprisingly, Ret-RP is used inside the intelligent network, far from the real domain of the end-user, which is the phone. This configuration is the tricky answer to the absence of the DPE in the terminal.

2.4.1. Openness of the IN-TINA Adaptation Unit

In this chapter the Ret RP means without distinction the Ret Access with the CCI or the TINA usage with the simplified communication interface.

Ret RP is in TINA the inter-domain reference point which interfaces the terminal to the retailer. It is a clear cut in the infrastructure, as it was designed to be technology independent and inter domain. Even if INAP is an open protocol designed to offer interoperability between services from one vendor and switches from another, it was not designed to open the operator's network to external service provider. This IN-TINA Retailer RP adoption solves this issue, because according to the use of Ret-RP, the network operator plays the role of end-users, and the service plays the role of another actor which retailer/service provider. In a more global view, the migration toward TINA gives the network operator the role of a TINA retailer, since the operator offers access to the service network, and the service accessible through Ret RP could be offered by an external service provider, equivalent to the TINA 3rd Party Service Provider, as shown in the following picture:

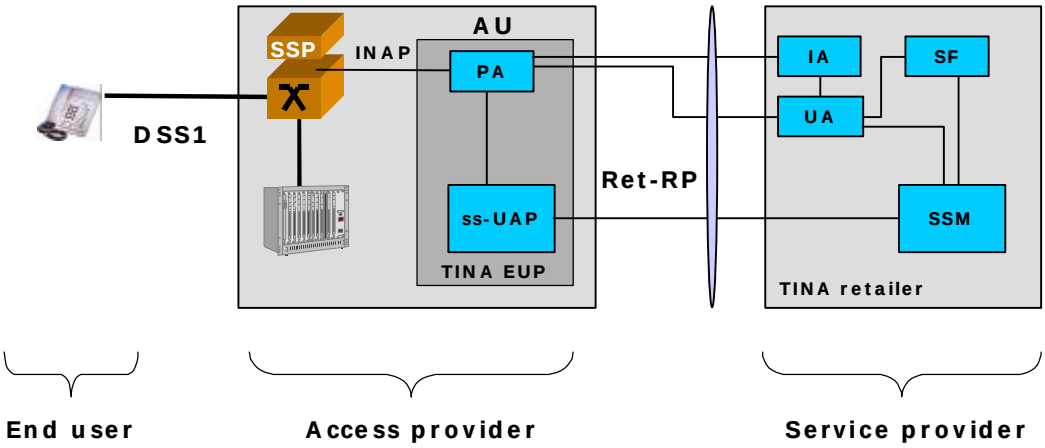

Figure 2. 3rd Party TINA access and service provisioning

In a nutshell, one of the most important benefits of the IN-TINA adaptation unit is to open the network to third party service provider, which means in TINA terms to bring the TINA business model in IN.
The most simple consequence is to offer a network control interface to new service providers, as Web server offer their information on the networks connected to Internet. However, this Ret RP interface between the ITAU and the service can also be used only as an internal interface in the IN operator domain, allowing a unified access and usage for different type of access network. Another business configuration is an IN operator providing access (authentication and billing) and outsourcing the service session to a 3rd Party Service provider, as shown in 3rd party service provisioning. This solution can be useful mainly for information services which are offered on the IN, with an "unified" access.

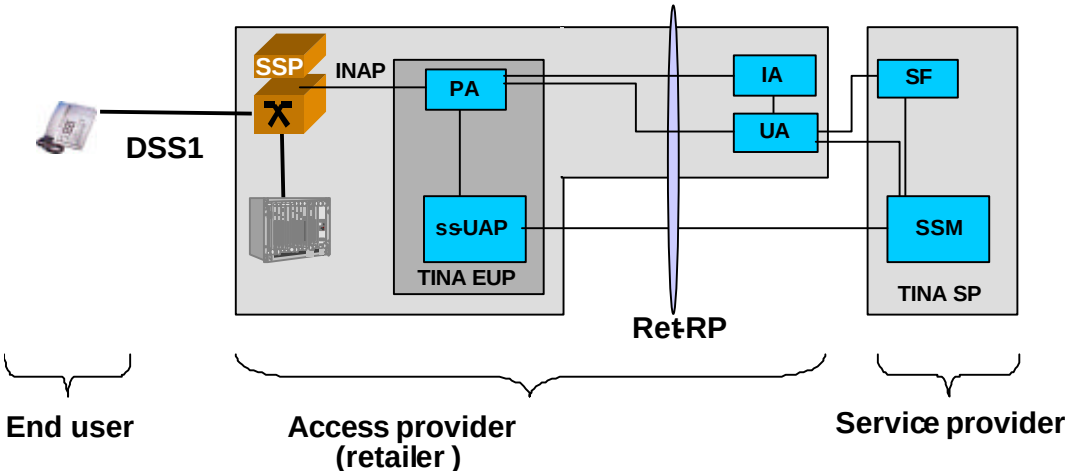

Figure 3. 3rd party service provisioning

2.4.2. Openness of the IN–TINA service

Another relevant issue is the use of the Retailer Reference point as the interface to connect the service logic inside the intelligent network with external advanced terminals, with systems such as Internet stations, or even with external information systems providing a part of the service logic.
Indeed, the TINA adoption in the service control brings a framework for the distribution of service logic.
Briefly, one argument for keeping the service (Service Session Manager) inside the IN operator domain and to use Ret RP between the ITAU and the service as an intra-domain reference point, is that the service role is to connect and organise the different parties and their actions inside the session while having access to the parties profile, and to the parties themselves. If an IN user (here TINA user, too) wants to be connected to another party (e.g. in case of some 800 services or in case of Internet call waiting), the acceptance of the call and the routable number must be transmitted to the SSP via the IN service, but the decision to accept the call, and the knowledge of the number where the called party can answer may be only performed in the called party domain. In classical IN, the acceptance of the call and the number translation is performed in the SCP, with data stored in the SDP and refreshed via the SMP management interface. These choices are more or less imposed by the lack of openness in the norms where the SCF holds the entire service logic, and the SDF is in the operator domain. Thus, the operator is obliged to store and maintain data or algorithms which should be in another domain, because of their complexity, or because constant update. Only the Internet surfer can decide if the Internet call waiting will be finally accepted, and the available phone device where an 800 call should be routed might only be known by the hot line company itself.
The interface between the TINA service provider, probably the IN operator, which enables this communication with the stakeholder who will provide the service information is the Retailer reference point, where the 800 service subscriber, as well as the Internet surfer, plays the consumer role, as shown in the picture entitled: IN service involving 2 TINA Ret-RP .

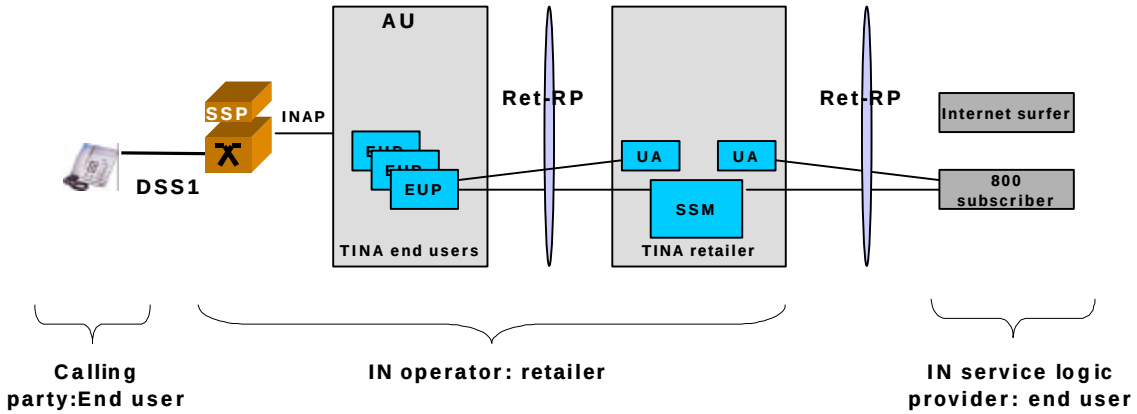

Figure 4. IN service involving 2 TINA Ret-RP

In conclusion, TINA provides a framework for new services and new services interconnections, enabling CTI IN integration and new IN-Internet services through the use of the inter domain reference point Retailer.

To sum up, the Adaptation Unit together with the user plays the Consumer role. It also realises some functions of a kernel transport network and offers a call control interface, and can also offer a TINA usage with a simplified communication session. Because of the non-secured signalling connection between the Service Switching Point and the Adaptation Unit, the Adaptation Unit should belong to the Connectivity Provider, namely the provider of the IN.

3. Information model

In the current section we explain the information flow between the ITAU and the TINA retailer. We focus in particular on the refinements needed in the current TINA information flows to realise the ITAU, i.e., to realise a services gateway between the IN domain and a TINA platform that enables the execution of the service logic in the TINA platform.

The ITAU enables end-users to contact their TINA retailer and subsequently consume TINA services via their telephone sets. The services restitution offered when consumers access their retailer via a telephone set could differ significantly when compared to the situation consumers access their retailer through e.g., a PC. This is natural and mainly resides in the type of terminal used. The retailer should be informed about the terminal used by the end-user to e.g., list the correct set of prescribed services. The reason is that the set of prescribed services may actually differ on the terminal used to access the retailer. From the perspective of the intelligent network, the ITAU enables the outsourcing of IN services to another platform. This is enabled via the call control interface and requires a business relation between both domains. From an end-user's perspective, however, the role of the ITAU is to bridge the user and its retailer transparently, allowing the end-user to list prescribed services, list the access sessions, start a service, etc. This must be performed in such a way that the retailer knows the restrictions that apply to the user's access session. This implies all kinds of restrictions, including the end-user's terminal, the restrictions of the access network and the corresponding transport capabilities that are used by the end-user, restrictions that apply to the ITAU and its business contract with the retailer, etc. The current subsection shows the refinements needed in the TINA specifications to make this bridging role of the ITAU possible.

3.1. User context information

A first observation is that the ITAU forms the mediator between telephone user and retailer, and therefore does not require more information in e.g., the operations requestNamedAccess() and/or requestAnonymousAccess(). At this point, the retailer does not need to know that the end-user contacts the retailer via the ITAU. However, as

soon as the access session has been established, the user context needs to be passed to the retailer, requiring refinements of the current TINA specifications.

Referring to [6], we obtain the following information regarding user context:

```
struct t_UserCtxt {
      TINACommonTypes::t_CtxtName ctxtName;
      TINAAccessCommonTypes::t_AccessSessionId asId;
      Object accessIR;
      Object terminalIR;
      Object sessionInfoIR;
      Object accessSessionInfoIR;
      TINAAccessCommonTypes::t_TerminalConfig terminalConfig;
};
```

`t_UserCtxt` informs the TINA retailer about details regarding the end-user domain, including name of context, the interfaces available during this access session and the terminal configuration information. In our case, the ITAU needs to perform this functionality on behalf of the end-user. The ITAU needs to come up with the ctxtName and can e.g. use it internally to distinguish between different access sessions with either the same or different end users. The objects mentioned in `t_UserCtxt` are references to the `i_UserAccess`, the `i_UserTerminal` and the `i_UserInvite` interfaces. Bear in mind that informing the retailer about these interfaces does not mean that the retailer must invoke operations supported by these interfaces. We will get back to this in a later stage.
The `terminalConfig` includes information regarding `t_TerminalId`, `t_TerminalType`, `t_NAPId`, `t_NAPType`, and `t_TerminalProperties`.

- `t_TerminalId` is an unsigned long and can e.g. be used for the telephone number of the involved party (recall that we are dealing with CS-1, so there is at most one party involved).
- `t_TerminalType` is defined as follows
  ```
  enum t_TerminalType {
     PersonalComputer, WorkStation, TVset,
     Videotelephone, Cellularphone, PBX, VideoServer,
     VideoBridge, Telephone, G4Fax
  };
  ```
  and fits our needs
- `t_NAPId` is typed as an unsigned long, which fits the telephone number.
- t_NAPType is typed as a string which is expected to be too general. We therefore suggest to use the t_AddressType (as described in Section 4.3) instead.
- t_TerminalProperties is of the type t_PropertyList, which is a list of properties that is on its turn typed as a name value pair. One of the terminal properties that is currently defined is the t_TerminalInfo, typed as follows:
  ```
  struct t_TerminalInfo {
        t_TerminalType terminalType;
        string operatingSystem; // includes the version
        TINACommonTypes::t_PropertyList networkCards;
        TINACommonTypes::t_PropertyList devices;
        unsigned short maxConnections;
        unsigned short memorySize;
        unsigned short diskCapacity;
  };
  ```
  Naturally, the t_TerminalInfo cannot be filled in by the ITAU, since the ITAU is not aware of this terminal specific information.

3.2. User context and the ITAU

As noticed in the subsection above, the ITAU cannot find out the details of the terminal used by the end-user. What the ITAU may know (or can find out) is the type of access network used by the end-user and the transmission capacity of this access network. Based on this information, the ITAU can make an educated guess about the terminal information. If the information regarding the access network used is passed to the service

provider, he can also exploit this information to provide a better service. As an example, consider the situation that the service provider knows that the end-user uses ISDN. This service provider may then instruct the ITAU to display certain information on the telephone set of the end-user. This is not possible in the case the end-user uses a POTS. Note that in the current TINA specifications the information regarding the access network used by the terminal is not necessary in Ret access because the terminal informs the network about the network addresses by means of Network Flow End Points (NFEPs).

Based on the above reasoning, we define the following structure for the access network used by the user who needs the TINA service via the ITAU.

```
struct t_AccessNetworkUsed {
      t_CalledAccessNetworkType accessNetworkUsed;
      t_CalledAccessNetworkBearerCapacity capacityAccessNetwork;
};
```

, where

```
   enum t_CalledAccessNetworkType {ISDN, GSM, IP, ATM, NotDefined}
```

and

```
typedef long t_CalledAccessNetworkBearerCapacity; // [bits/s]
```

This information can be passed as a parameter in the terminal properties. Note that the Plain Old Telephone Set (POTS) is not a different switching (addressing) end point than an ISDN terminal, and does not need to be considered as a separate network type. The use of POTS will be mapped on the NotDefined type.

As mentioned above, by invoking the operation **setUserCtxt**(), the ITAU can inform the retailer about the interfaces to use if the retailer needs information about the user or the terminal settings. Recall that the ITAU also has a business relation with the retailer. However, if the ITAU invokes the setUserCtxt() operation, it does this on the end-user's behalf, and not on its own behalf. The fact that the operation setUserCtxt() announces the interface and the corresponding operations that can be invoked, does not necessarily mean that the retailer also invokes all operations. An important operation in this context is the operation `getTerminalInfo`(). It is important that the retailer knows that the user accesses the retailer via the ITAU, and not directly. A deadlock situation, where the ITAU waits for the retailer to invoke the operation `getTerminalInfo`(), whereas the retailer waits for the ITAU to invoke an operation may occur. Since IDL does not support to enforce operations depending on a certain stage, there are three solutions for this problem:

1. Do not manifest this problem in the IDL code and rely on the common sense of both domains. Since it is the retailer who offers its services via the ITAU, this retailer may need to know the terminal used by the end-user and is therefore expected to call the operation `getTerminalInfo()`.
2. Introduce an exception e_TeminalInfoNotSet. This exception informs the retailer that the operation `getTerminalInfo()` needs to be invoked before any operation can be called.
3. Introduce an operation that allows the end-user to set the terminal info.

The second alternative is hard to realise, since an exception is coupled to an operation. This means that every operation that can be invoked by the retailer should be extended with this exception. In addition, the ITAU may be waiting for the retailer to invoke an operation in order to throw the exception. This solution does therefore not prevent a deadlock. The third option requires the extension of the TINA i_RetailerNamedAccess interface, demanding re-compilation. Although this is not a prime reason for rejecting this option, it is one to consider. By allowing the ITAU to invoke the operation setUserCtxt(), it can inform the retailer that the user is connected via the ITAU by exploiting a userCtxtName that spells "ITAU". Taken the pros and cons of each option, the first one was chosen.

Because we do not force the ITAU to inform the retailer about the access network used by the end-user, the operation getAccessNetworkUsed() is introduced in the interface i_UserTerminal. This is the same interface that supports e.g., the operation `getTerminalInfo`(). This operation comes in handy if the access network used by the end-user is not passed by the ITAU as part of the operation setUserCtxt(). For the discussion whether an operation is required to **set** the access network that can be used by the end-user, the same reasoning as discussed above can be applied. For the same reason as mentioned above, it was decided not to manifest the problem in the IDL code and to rely on the common sense of both domains. The operation getAccessNetworkUsed() is as follows:

```
void getAccessNetworkUsed (
```

```
      in AccessCommonTypes:: t_AccessSessionSecretId asSessionSecretId;
      out MgmntItauCommonTypes ::t_AccessNetworkUsed accessNetworkUsed;
) raises (
      AccessCommonTypes::e_AccessError
);
```

4. Computational model (IDL, MSC / state model)

4.1. Overview

In Figure 1 (see section 1.2) we positioned our Adaptation Unit in the IN and TINA environment. From an IN point of view, Adaptation Unit and Service Platform together realise an SCF. Other IN functionalities, like SRF (specialised resource function) and SDF (database functionality) that, again from an IN point of view, are needed, are not explicitly shown, but may be included in the service platform.
Here our special concern is the ITAU which connects the legacy IN world to the service platforms CORBA domain. An overview of the interfaces and components of the ITAU is shown in Figure 5. In the following, they are briefly introduced. Please note that we present here one possible design that illustrates already recognised functionalities.

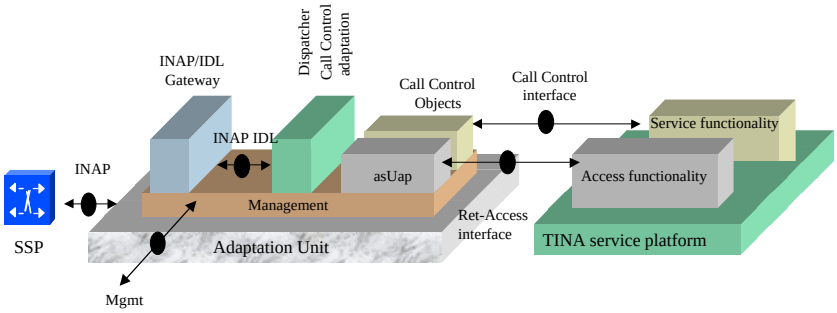

Figure 5. Functional design of the Adaptation Unit

The TCAP/***INAP interface*** connects the ITAU to the legacy IN domain. National INAP dialects are deployed here. For a syntactic translation of these INAP messages into INAP-IDL operations, the OMG Corba/INAP Gateway may be used. In that case the TCAP/***INAP IDL interface*** is available. The dispatcher's task is to distribute incoming INAP messages related to access establishment to the asUap and service related messages to the Call Objects.
When an IDP (Initial Detection Point) message arrives the instantiating of an asUap is required. If the information received with IDP is sufficient to establish a connection to the service platform, a secure access is requested and established using the ***Ret-access interface***. This interface is compliant to TINA Ret RP.
If a service session is needed during an access session the ***Call Control interface*** is required for further call processing. This interface is completely service independent and offers in a first step an access for telephony services to the underlying network infrastructure. It is providing direct access to the INAP in an object oriented manner. This abstract object design leads in a second step to a technology independent, service independent Call Control interface. Optionally, the ***Ret-usage interface*** can be used instead.
For configuration of the adaptation unit a ***management interface*** is introduced.

To summarise: our ITAU provides the following interfaces:

1.	TCAP/INAP:	access to SSP using a INAP dialect	**mandatory**
2.	TCAP/INAP IDL:	IDL interface, syntactic translation of INAP	**optional**
3.	Ret-Access:	access interface between ITAU and service platform	**mandatory**
4.	Call Control:	easy usable, low level telephony call control interface	**mandatory**
5.	Ret-Usage:	service usage interface between ITAU and service platform	**optional**

6. Mgmt: management interface **mandatory**

4.2. Access interfaces

To “bridge” IN and TINA, the ITAU needs one interface to communicate with an SSF and one reference point to be able to invoke TINA objects and to be invoked by TINA objects.
The interface between the SSF and the adaptation unit is defined by INAP. To meet the requirements of [4], we will strictly refer to the Interface Recommendation for Intelligent Networks CS-1 [16]. This interface recommendation defines both the interactions between SSF and SCF and the correct sequence of these interactions. The interactions are defined using ASN.1. The sequence of interactions is controlled by finite state machines.
The retailer reference point is chosen as reference point between the ITAU and the retailer (see section 2).

4.2.1. Access interfaces between adaptation unit and retailer

The following figure shows the interfaces in access part of the retailer reference point. It is assumed, that the retailer implements all provider interfaces. The adaptation unit does not need to implement all the user interfaces in behaviour. Which interfaces are absolutely necessary and which might be optional, is described below. Again, if the behaviour of an interface is not defined, the appropriate object exists, but this object does not implement any behaviour. It only throws a user defined exception.

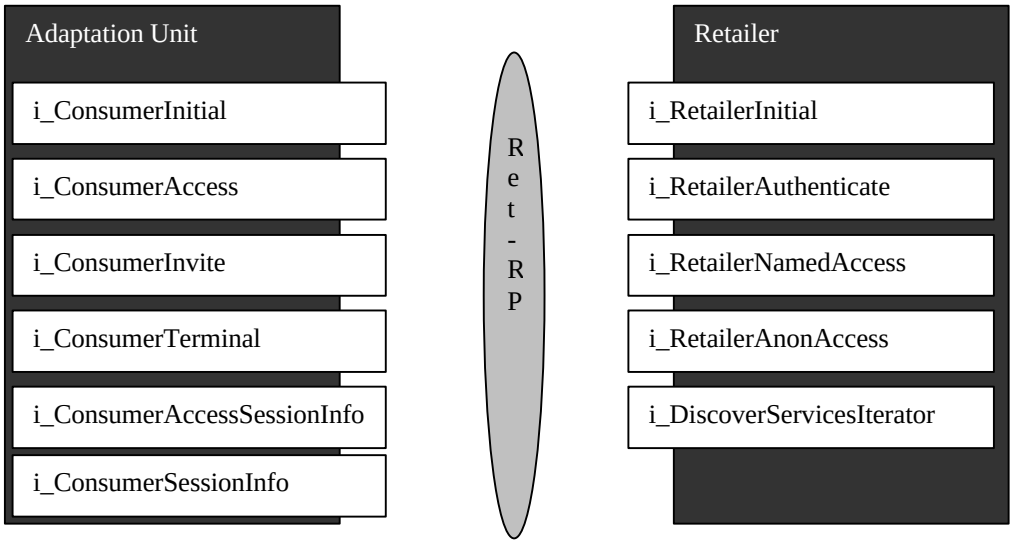

Figure 6. Access interfaces between adaptation unit and retailer

For more information, which interfaces are always available outside an access session, only available inside an access session and only available outside an access session, please refer to [6].

i_ConsumerInitial:
This interface is only needed if an IN service should be initiated using the INAP message InitiateCallAttempt. In order to initiate a service, the retailer has to invoke an i_ConsumerInitial interface. The reference of this interface is only known if it has been registered using the operation registerInterfaceOutsideAccessSession. Then, an access session can be established in a way conformant to [9].

i_ConsumerAccess:
This interface must be implemented in behaviour.

i_ConsumerInvite:
This interface need not to be implemented in behaviour.

i_ConsumerTerminal:
This interface must be implemented in behaviour.

i_ConsumerAccessSessionInfo:
This interface need not to be implemented in behaviour.

i_ConsumerSessionInfo:
This interface must be implemented in behaviour.

4.2.2. Interactions between adaptation unit and retailer

There are two possibilities to invoke a service in IN. The most often used is the initiation of services by the SSF sending an InitialDetectionPoint message. The other possibility is the initiation of services by the SCF sending an InitiateCallAttempt message. The messages InitialDetectionPoint and InitiateCallAttempt have been defined in [16].

These two cases for initiating services and their influence on the interactions between ITAU and retailer are investigated in the following.

Initiating a service using InitialDetectionPoint

The description below strictly refers to [6]. Several assumptions are made to simplify the description below:

- As in [6, section 4: Access Part] we avoid to specify how the consumer contacts the retailer in order to get the reference of the ***i_RetailerInitial*** interface.
- A secure context has already been set up between the consumer and the retailer before using the operations of the i_RetailerInitial interface to start a TINA access session. This secure context might be established using the CORBA security services. (If this secure context is not established before using any operation of ***i_RetailerInitial*** interface, the procedures defined in [6] shall be performed.)
- In the case described above, the service invocation can only be made through an SSF sending an ***InitialDetectionPoint*** message.

The adaptation unit has to invoke all operations to establish, maintain and end a TINA access session conformant to the access part of the retailer reference point on behalf of the service user's request. Therefore, each invocation of a service must be mapped into one and only one TINA access session together with or without an invocation of TINA service sessions. Therefore, the ***InitialDetectionPoint*** message is mapped into several operation calls to establish a TINA access session:

1. The ITAU contacts the retailer in order to get a reference to the ***i_RetailerInitial*** interface. (This operation can be called each time when an ***InitialDetectionPoint*** message is received or only once when the ITAU starts up.)

2. Dependent, whether the user only uses the requested service or owns a service that is provided by the TINA retailer or third party service provider, an anonymous or a named TINA access session must be established. An anonymous TINA access session is started using the ***i_RetailerInitial:: requestAnonymousAccess()*** operation. If a named TINA access session is needed, the ITAU has to call the ***i_RetailerInitial::requestNamedAccess()*** operation. After the operation calls are performed successfully, an TINA access session has been established and the ITAU has got a reference to the ***i_RetailerNamedAccess*** or the ***i_RetailerAnonAccess*** interface.

3. Then, the ITAU has to inform the retailer about its interfaces and terminal configuration calling ***i_RetailerNamedAccess::setUserCtxt()*** or a similar operation if an anonymous TINA access session has been established.

4. The ITAU can now call a subset of the operations the ***i_RetailerNamedAccess*** or the ***i_RetailerAnonAccess*** interfaces provide.

5. The ITAU can now establish, maintain and end one or more TINA service sessions. In IN CS-1 specifications [14] the axioms of "Single ended" and "Single point of control" are very important. CS-1 capabilities are intended to support only services and service features that fall into the category of "single ended", "single point of control" services. The following definitions apply: A single-ended service feature applies to one and only one party in a call and is orthogonal (independent) at both the service and topology levels to any other parties that may be participating in the call. Orthogonality allows another instance of the

same or a different single-ended service feature to apply to another party in the same call as long as the service feature instances do not have feature interaction problems with each other. Single point of control describes a control relationship where the same aspects of a call are influenced by one and only one service control function at any point in time. It is recommended, that the design of the adaptation unit strictly follows these two axioms. Therefore, it is recommended that one TCAP dialogue is controlled logically by one and only one instance simultaneously. How and where the control of TCAP dialogues will be realised, is implementation dependent.

6. When the TINA access session is ended, the ITAU has to send either ***ReleaseCall*** or a message, that the processing of the instructions of the SSF are completed, back to the SSF. ***ReleaseCall*** must be sent if a control relationship is currently active between ITAU and SSF. If only a monitor relationship is currently active, a message, that the processing of the instructions of the SSF is completed, must be sent. This message must be mapped in a ***TC_End*** message.

While calling the operation ***i_RetailerNamedAccess::setUserCtxt()*** or a similar operation if an anonymous TINA access session has been established, the adaptation unit gives references to some of its interfaces to the retailer. These interfaces might be used to call operations during a TINA access session.

A service user, who is registered at an E.164 terminal, can be called using the E.164 address of this specific terminal. Registration of a service user means, that the service user's address information can be resolved accessing some databases in the retailer. If a service is invoked in order to call this registered service user, that means, if a TINA access session has been established between the ITAU and the retailer on behalf of a callee, the registered service user can be called regardless whether he does or does not join a TINA access session at this time. That is why the retailer does not need any interface references for the called user.

For the complete IDLs refer to [6].

Initiating a service using InitiateCallAttempt

The description below strictly refers to [6, 9]. Several assumptions are made to simplify the description below:

- An i_ConsumerInitial interface has been previously registered by the consumer. The registration of this interface can only be done invoking the operation registerInterfaceOutsideAccessSession during an access session.
- In this case, the service invocation can only be made through an SCF sending an ***InitiateCallAttempt*** message.

To initiate a service by the retailer, the retailer and the ITAU have to perform the following operations:

1. The retailer invites a consumer invoking ***i_ConsumerInitial::inviteUserOutsideAccessSession.***

2. The ITAU accepts the invitation setting the out-parameter reply.reply of ***i_ConsumerInitial:: inviteUserOutsideAccessSession*** to success.

3. After that, the ITAU starts an access session like described above. The service logic located in a service session manager (SSM) requests the sending of ***InitiateCallAttempt*** by invoking appropriate operations on ITAU's interfaces.

For the complete IDLs refer to [6].

4.2.3. Named and anonymous access session

TINA has defined a common access that is independent of the requested service. This is reasonable because it clearly separates functions that are common to all services from functions that are specific to one. To benefit from TINA, this unified way to access retailers and request services shall remain unchanged as much as possible, in order to reuse as much software as possible.

The access session is necessary to identify a service user as a trusted party. Trusted parties can be named or anonymous users. Anonymous users have only restricted rights to use services of the retailer. As described above, two kinds of access sessions can be used between the retailer and the ITAU.

A named access session shall be used if the service user needs to be identified before using the requested service. All services that modify user profiles can only be started after a named access session has been set-up.

An anonymous access session shall be used if the service user has not established any contractual relationship with the service provider for more than one call or if the service user only needs a contractual relationship for this specific call. After this call is ended, the contractual relationship does not exist any more.

4.3. Call Control interfaces

This chapter give the list of the Parlay's call control interfaces [17] that are retained in our answer to the TINA RFP "IN access to TINA services & connection management". It is understood that the specifications for the CCI might evolve in a future version according to upcoming specifications from standardisation bodies such as for instance ETSI SPAN3.
Since the Parlay interfaces are already defined in Parlay specification 1.1 the purpose of this chapter is only to highlight the mapping of the Parlay call control interfaces onto INAP operation. The proposed mapping is supposed to be independent of any service logic: it should work for any service. The INAP CS-1 specifications from ITU-T [15, 16] are used as a base to define the mapping. It is expected that future version of this document will provide a more detailed mapping at the parameter level. However we have chosen to retain some Parlay operations related to CS-2 capabilities (mainly i_Call.attachCallLeg() and i_Call.detachCallLeg()). An adequate comment highlight this point where it is relevant.
The remaining of this chapter is divided following Parlay interfaces.

4.3.1. i_IntegrityManagement

activityTest()

Direction: Application ➔ Call Control ➔ SSP
Mapping: In this document we choose to define a mapping not related to INAP.
Remarks: This method is called by the application to test the framework. Possible actions at the framework level are: to check for DPE load, CPU load and any other relevant indicators.

4.3.2. i_AppIntegrityManagement

infrastructureFaultDetected()

Direction: Call Control ➔ Application
Mapping: This method is not relevant to the INAP mapping.
Remarks: This method may be invoked upon detection of an internal failure of the Parlay gateway (internal communication failure, software fault)

infrastructureRecovered()

Direction: Call Control ➔ Application
Mapping: This method is not relevant to the INAP mapping.
Remarks: This method may be invoked upon detection of the recovery of a previous failure

apiUnavailable()

Direction: Call Control ➔ Application
Mapping: This method is not relevant to the INAP mapping.
Remarks: This method may be invoked upon an OAM operation, a software fault, …

apiAvailable()

Direction: Call Control ➔ Application
Mapping: This method is not relevant to the INAP mapping.

Remarks: This method may be invoked upon an OAM operation, a software fault recovery, …

serviceUnavailable()

Direction: SSP ➔ Call Control ➔ Application
Mapping: This method is not relevant to the INAP mapping (but is relevant to the SS7 mapping).
Remarks: This method may be invoked upon detection of a communication fault between the Call Control implementation and the IN (this may be due to an SS7 communication layer fault).

serviceAvailable()

Direction: SSP ➔ Call Control ➔ Application
Mapping: This method is not relevant to the INAP mapping (but is relevant to the SS7 mapping).
Remarks: This method may be invoked upon detection of a communication recovery between the Call Control implementation and the IN (this may be due to an SS7 communication layer fault).

4.3.3. i_Service

setCallback()

Direction: Application ➔ Call Control
Mapping: This method is not relevant to the INAP mapping.
Remarks:

4.3.4. i_CallControlManager

setCallLoadControl()

Direction: Application ➔ Call Control ➔ SSP
Mapping: Two level are possible to manage the load.

1) If the call load is managed at the call control implementation level (this may be necessary for network resources that are not able to manage the load of the service layer) then it is not necessary to send an INAP operation. If this solution is retained it means that upon reception on an `InitialDP` operation, the call control implementation check if the call match the call load control criteria and decide if a notification should be sent to the application.
2) If the call load is managed at the network resource level (in this case the SSP level) then the INAP operation `CallGap` should be sent to the SSP.

Remarks: Only a subset of the `CallGap` parameters can be mapped onto the `i_CallControlManager.setCallLoadControl()` parameters. For instance with INAP it is possible to filter calls that are related to a given `serviceKey` (which is not necessarily linked to a called address range). Therefore the Parlay specifications only give access to a subset of the functionality of the `CallGap` operation.

createCall()

Direction: Application ➔ Call Control
Mapping: The invocation of this method is related to the `InitiateCallAttempt` INAP operation. Since, the `i_CallControlManager.createCall()` invocation does not provide enough information to the call control implementation it is not possible to translate this invocation directly onto an `InitiateCallAttempt`. According to that, the call control implementation should simply record the fact that a call is about to be created by the application and that upon reception of a future `i_Call.routeCallToOrigination_Req()` which provide the missing parameters an invocation of `InitiateCallAttempt` will have to be sent.
Remarks:

4.3.5. i_AppCallControlManager

callAborted()

Direction: SSP ➔ CallControl ➔ Application

Mapping: This method is not relevant to the INAP mapping but it is relevant to the TCAP mapping. When the call control implementation receive a TC_ABORT request from an IN switch then the call control implementation has to invoke `i_AppCallControlManager.callAborted()`.
Remarks:

callOverloadEncountered()

Direction: SSP ➔ Call Control ➔ Application
Mapping: The `i_AppCallControlManager.callOverloadEncountered()` method should be invoked upon reception of an `InitialDP` operation with the `cGEncountered` parameter indicating that call gapping was encountered. However the InitialDP operation only provide information on the destination number and the `i_AppCallControlManager.callOverloadEncountered()` method needs to provide the destination address range matching the one given when the application has invoked `i_CallControlManager.setCallLoadControl()`. Therefore the call control implementation has to keep track of the address ranges provided by the `i_CallControlManager.setCallLoadControl()` message in order to associate the destination number provided by `InitialDP` with an address range.
Remarks: The reception of an InitialDP operation should also trigger a NewCallRequest notification to the application (supposing that the application provider has a pre-arranged agreement). Then it might be useful to agree on the order of the invocation of `i_AppCallControlManager.callOverloadEncountered()` and i_AppEventNotification.eventNotify(). In this document we propose that `i_AppCallCOntrolManager.callOverloadEncountered()` is called before `i_AppEventNotification.eventNotify()`.

callOverloadCeased()

Direction: SSP ➔ Call Control ➔ Application
Mapping: See `i_AppCallControlManager.callOverloadEncountered()`. This method is invoked when an InitialDP operation is received indicating that call gapping is not encountered if a previous InitialDP operation has indicated that call gapping was encountered.
Remarks: The addressRange parameter should correspond to the addressRange parameter of a previous invocation of `i_AppCallControlManager.callOverloadEncountered()`.

callFaultDetected()

Direction: SSP ➔ Call Control ➔ Application
Mapping: If the difference between this method and the `i_AppCallControlManager.callAborted()` is that further communication between the call and the application is still possible then there is no INAP mapping (but this method may be used to signal a software fault related to a call object to the application).
Remarks:

4.3.6. i_Call

routeCallToDestination_Req()

Direction: Application ➔ Call Control ➔ SSP
Mapping: If the `responseRequested` parameter is empty then the reception of this invocation should trigger a call to the `Connect` INAP operation. Otherwise, the reception of this invocation should be translated into a call to the `RequestReportBCSMEvent` and the `Connect` INAP operations. In the last case, the call control implementation should record that upon reception of the corresponding `EventReportBCSM` a `i_AppCall.routeCallToDestination_Res()` should be invoked (the recording of this information is necessary to distinguish the invocations of `i_AppCallLeg.callLegEventReport_Res()` and `i_AppCall.routeCallToDestination_Res()`. The mapping between call events defined in Parlay and O_BCSM DP + T_BCSM DP are given by table 3 and table 4.
Remarks:

routeCallToOrigination_Req()

Direction: Application ➔ Call Control ➔ SSP
Mapping: If the call control implementation has already received an invocation of i_CallControlManager.createCall() then the call control can send an InitiateCallAttempt operation.
Remarks: See `i_CallControlManager.createCall()`.

releaseCall()

Direction: Application ➔ Call Control ➔ SSP
Mapping: The invocation of the operation should be translated into the INAP operation `ReleaseCall`.
Remarks:

deassignCall()

Direction: Application ➔ Call Control
Mapping: There is no mapping to INAP, however the call control implementation can de-associate the relationship between a TCAP dialog and a `CallSessionID` so that possible further INAP operations coming from the switch are untreated.
Remarks: This operation should not be mapped onto a TC_END or TC_ABORT since this may abort the call (the treatment of an unexpected TC_END operation by an IN switch depend on its configuration).

getCallInfo_Req()

Direction: Application ➔ Call Control ➔ SSP
Mapping: This message is translated into a CallInformationRequest operation.
Remarks: The mapping of the parameters is left for further studies.

setCallChargePlan()

Direction: Application ➔ Call Control ➔ SSP
Mapping: This message is translated into a FurnishChargingInformation operation.
Remarks: Since the specification of the FurnishChargingInformation report is operator dependent it will be nearly impossible to specify a generic mapping at the parameter level.

getCallLegs()

Direction: Application ➔ Call Control
Mapping: This message is not relevant to the INAP mapping.
Remarks: The call control implementation should be able to retrieve the call legs interface references by using the callSessionID.

createCallLeg()

Direction: Application ➔ Call Control
Mapping: This message is not relevant to the INAP mapping. This method might be seen as a placeholder for future INAP CS-2 extensions of the Parlay specifications.
Remarks:

attachCallLeg()

Direction: Application ➔ Call Control
Mapping: See i_Call.createCallLeg().
Remarks:

detachCallLeg()

Direction: Application ➔ Call Control
Mapping: See i_Call.createCallLeg()
Remarks:

getControlLeg()

Direction: Application ➔ Call Control
Mapping: This method is not relevant to the INAP mapping
Remarks: The call control implementation should be able to retrieve the controlling call leg interface reference by using the callSessionID.

4.3.7. i_AppCall

routeCallToDestination_Res()

Direction: SSP ➔ Call Control ➔ Application
Mapping: The call control implementation should invoke this method upon reception of an `EventReportBCSM` operation corresponding to a previous `RequestReportBCSMEvent` operation that was triggered after that the call control implementation has received a `i_Call.routeCallToDestination_Req(). The mapping between O_BCSM + T_BCSM DP and Parlay call events are given by table 1 and table 2.`
Remarks:

routeCallToDestination_Err()

Direction: SSP ➔ Call Control ➔ Application
Mapping: See i_AppCall.routeCallToDestination_Res().
Remarks:

routeCallToOrigination_Res()

Direction: SSP ➔ Call Control ➔ Application
Mapping: See i_AppCall.routeCallToDestination_Res().
Remarks:

routeCallToOrigination_Err()

Direction: SSP ➔ Call Control ➔ Application
Mapping: See i_AppCall.routeCallToDestination_Res().
Remarks:

getCallInfo_Res()

Direction: SSP ➔ Call Control ➔ Application
Mapping: This message is invoked upon reception of a CallInformationReport operation.
Remarks: The mapping of the parameters should be worked-out.

getCallInfo_Err()

Direction: Call Control ➔ Application
Mapping: This message is not related to INAP mapping.
Remarks: This message should be invoked when the call control detect an erroneous `i_Call.getCallInfo_Req()` request.

4.3.8. i_CallLeg

routeCallLegToAddress()

Direction: Application ➔ Call Control ➔ SSP
Mapping: The invocation of this message should result in a behaviour similar to the one triggered by the invocation of i_Call.routeCallToDestination_Req() or i_Call.routeCallToOrigination_Req() depending on the leg to which the message apply with the exception that it is not possible to request a report of the outcome of the routing (therefore i_CallLeg.callLegEventReport_Req() should be used).
Remarks:

callLegEventReport_Req()

Direction: Application ➔ Call Control ➔ SSP
Mapping: This message should be translated into a RequestReportBCSMEvent operation. The mapping of the CallLeg events onto O_BCSM DP + T_BCSM DP are given by table 3 and table 4.
Remarks:

releaseCallLeg()

Direction: Application ➔ Call Control ➔ SSP

Mapping: It is not possible to map this message onto an INAP operation because this correspond to an INAP-CS2 capability.
Remarks: In INAP-CS2 this message would be translated onto something like `DisconnectLeg`.

getAddresses()

Direction: Application ➔ Call Control
Mapping: This operation is not relevant to the INAP mapping.
Remarks: The call control implementation should record the address associated to each CallLeg interface reference. Here there is an issue when the call leg address correspond to the address of a PSTN to VoIP vocal gateway. In such a case it is also interesting to record the destination IP address. This point needs to be worked out and is outside the scope of pure Parlay-INAP mapping.

getCallLegInfo_Req()

Direction: Application ➔ Call Control ➔ SSP
Mapping: This message is translated onto the `CallInformationRequest` operation.
Remarks: The mapping of the parameters should be worked-out.

getCallLegType()

Direction: Application ➔ Call Control
Mapping: This message is not relevant to INAP mapping
Remarks:

getCall()

Direction: Application ➔ Call Control
Mapping: This message is not relevant to INAP mapping
Remarks:

4.3.9. i_AppCallLeg

callLegEventReport_Res()

Direction: SSP ➔ Call Control ➔ Application
Mapping: See `i_AppCall.routeCallToDestination_Res()`.
Remarks:

callLegEventReport_Err()

Direction: SSP ➔ Call Control ➔ Application
Mapping: See `i_AppCall.routeCallToDestination_Res()`.
Remarks:

getCallLegInfo_Res()

Direction: SSP ➔ Call Control ➔ Application
Mapping: This message should be invoked upon reception of a `CallInformationReport` operation.
Remarks:

getCallLegInfo_Err()

Direction: Call Control ➔ Application
Mapping: This message is not relevant to INAP mapping.
Remarks: The call control implementation has to send this message if it receive an erroneous `i_CallLeg.getCallLegInfo_Req()`.

4.3.10. i_CallINAP1

getMoreDialledDigits_Req()

Direction: Application ➔ Call Control ➔ SSP

Mapping: This message should be called by the application after reception of a `NewCallRequest` notification if there was a pre-arranged agreement with the network provider that the application should be notified when some sequence of digits are dialled. In INAP this notification correspond to TDP `Analyse_Information`. Then `i_CallINAP1.getMoreDialledDigits()` has to be translated onto a `RequestReportBCSMEvent` for EDP `Collect_Information` with the length parameter mapped to the `dpSpecificCriteria.numberOfDigits` and onto a CollectInformation operation to continue the collection of digits. See also table 3.
Remarks:

superviseCallINAP1_Req()

Direction: Application ➔ Call Control ➔ SSP
Mapping: This message could not be mapped onto an INAP-CS1 operation. However, it is possible to map this message onto an operation of some INAP flavour. Otherwise there is also the possibility to manage a time inside the Call Control implementation (instead of managing a timer inside the SSP).
Remarks:

4.3.11. i_AppCallINAP1

getMoreDialledDigitsINAP1_Res()

Direction: SSP ➔ Call Control ➔ Application
Mapping: This message should be invoked upon reception of an `EventReportBCSM` operation that correspond to a previous `RequestReportBCSMEvent` that was send after the reception of a `i_CallINAP1.getMoreDialledDigits_Req()`. See also table 1.
Remarks:

getMoreDialledDigitsINAP1_Err()

Direction: SSP ➔ Call Control ➔ Application
Mapping: This message should be invoked upon reception of an `EventReportBCSM` indicating the `O_Abandon` DP if a `i_CallINAP1.getMoreDialledDigits_Req()` was previously invoked. This message should also be invoked upon reception of an unexpected `TC_END` message if a `i_CallINAP1.getMoreDialledDigits_Req()` was previously invoked.
Remarks:

superviseCallINAP1_Res()

Direction: SSP ➔ Call Control ➔ Application
Mapping: When the timer set upon the previous invocation of `i_CallINAP1.superviseCallINAP1_Req()` times out this message should be invoked. Otherwise the INAP mapping may also be network operator dependent (depending of the INAP flavour).

Remarks:

superviseCallINAP1_Err()

Direction: Call Control ➔ Application
Mapping: This message is not relevant to INAP mapping
Remarks:

4.3.12. i_UIManager

createUI()

Direction: Application ➔ Call Control
Mapping: Since there is no parameter that would enable the call control implementation to choose the SRP that should be used, it is not possible to map the message onto an INAP operation.
Remarks:

4.3.13. i_AppUIManager

userInteractionTerminated()

Direction: Call Control ➔ Application
Mapping: This message is not relevant to the INAP mapping. However this message could be sent if an unrecoverable communication failure is detected between the Call Control and the SRP.
Remarks:

userInteractionFaultDetected()

Direction: Call Control ➔ Application
Mapping: This message is not relevant to the INAP mapping. However this message could be sent if a temporary communication failure is detected between the Call Control and the SRP.
Remarks:

4.3.14. i_UICall

abortLegAction_Req()

Direction: Application ➔ Call Control ➔ SRP
Mapping: This message should be translated onto the Cancel operation.
Remarks:

sendInfoAndCollectCall_Req()

Direction: Application ➔ Call Control ➔ SRP
Mapping: This message should be translated onto a `PromptAndCollectUserInformation` operation. However this is not sufficient since in Parlay there is no message to connect a party to an intelligent peripheral. According to that this message should also trigger the sending of the `ConnectToRessource` operation (this also means that the Call Control implementation has to decide alone to which SRP the party should be connected).
Remarks:

sendInfoCall_Req()

Direction: Application ➔ Call Control ➔ SRP
Mapping: This message should be translated onto a `PlayAnnouncement` operation. However, with regards to the connection to the intelligent peripheral there is the same problem as for the `i_UICall.sendInfoAndCollectCall_Req()` message. Therefore this message should also trigger the sending of a `ConnectToRessource` operation.
Remarks:

releaseUICall()

Direction: Application ➔ Call Control ➔ SRP
Mapping: This message should be translated into the sending of a Cancel operation for each `PlayAnnouncement` and `PromptAndCollectUserInformation` that was sent previously. Then the call control implementation can send a `DisconnetForwardConnection` in order to end the connection(s) between the SRP and involved parties.
Remarks:

4.3.15. i_AppUICall

abortLegAction_Res()

Direction: Application ➔ Call Control ➔ SRP
Mapping: This message is not directly related to INAP mapping. However this message could be sent by the call control implementation after having sent the `Cancel` operation.
Remarks:

abortLegAction_Err()

Direction: Application ➔ Call Control ➔ SRP

Mapping: This message is not directly related to INAP mapping. However this message could be sent by the call control implementation if it detects an error after having sent the Cancel operation (but how this kind of error can be detected ? Answer: by means of CancelFailed).
Remarks:

sendInfoAndCollectCall_Res()

Direction: SRP ➔ Call Control ➔ Application
Mapping: This message is invoked upon reception of the result of the `PromptAndCollectInformation` operation.
Remarks:

sendInfoAndCollectCall_Err()

Direction: SRP ➔ Call Control ➔ Application
Mapping: This message is invoked upon reception of an error after having invoked the `PromptAndCollectInformation` operation.
Remarks:

sendInfoCall_Res()

Direction: SRP ➔ Call Control ➔ Application
Mapping: This message is sent if no error are detected after having invoked the `PlayAnnoucement` operation but since the `PlayAnnouncement` operation does not return any result it may be difficult for the call control implementation to find out that the operation was successful.
Remarks:

sendInfoCall_Err()

Direction: SRP ➔ Call Control ➔ Application

Mapping: This message is invoked upon the reception of an error related to a `PlayAnnouncement` operation.
Remarks:

4.3.16. Mapping between Parlay CallLeg events and INAP BCSM detection points

The following tables provide a mapping between Parlay CallLeg events and INAP O_BCSM, T_BCSM.
This mapping is needed in order to translate the following Parlay methods:

- i_Call.routeCallToDestination_Req()
- i_AppCall.routeCallToDestionation_Res()
- i_AppCall.routeCallToDestination_Err()
- i_Call.routeCallToOrigination_Req()
- i_AppCall.routeCallToOrigination_Res()
- i_AppCall.routeCallToOrigination_Err()
- i_CallLeg.callLegEventReportReq()
- i_AppCallLeg.callLegEventReport_Res()
- i_AppCallLeg.callLegEventReport_Err()
- i_CallINAP1.getMoreDialledDigitsINAP1_Req()
- i_AppCallINAP1.getMoreDialledDigitsINAP1_Res()

The following table defines the mapping of incoming INAP messages concerning O_BCSM call events onto Parlay methods invocations.

INAP operation	O_BCSM DP	Additional criteria	Parlay method invocation	CallLeg O Event triggered	Trigger CallLeg O object creation?	CallLeg T Event triggered	Trigger CallLeg T object creation?
EventReportBCSM	Origination_Attempt_Authorized	X	NO CORRESPONDING METHOD	X	YES	X	NO
EventReportBCSM	Collected_Information	X	i_AppCallINAP1.getMoreDialledDigitsINAP1_Res()	X	YES	X	NO
EventReportBCSM	Analysed_Information	X	NO CORRESPONDING METHOD	X	YES	X	NO
EventReportBCSM	Route_Select_Failure	Cause	i_AppCall.routeCallToOrigination_Res()	PARLAY_CALL_LEG_ROUTING_FAILURE	YES	X	NO
EventReportBCSM	Route_Select_Failure	Cause	i_AppCall.routeCallToDestination_Res()	PARLAY_CALL_LEG_ROUTING_FAILURE	YES	X	NO
EventReportBCSM	Route_Select_Failure	Cause	i_AppCallLeg.callLegEventReport_Res()	PARLAY_CALL_LEG_ROUTING_FAILURE	YES	X	NO
EventReportBCSM	Route_Select_Failure	Cause	i_AppCall.routeCallToOrigination_Err()	PARLAY_CALL_ERROR_ROUTING_ABORTED	YES	X	NO
EventReportBCSM	Route_Select_Failure	Cause	i_AppCall.routeCallToDestination_Err()	PARLAY_CALL_ERROR_ROUTING_ABORTED	YES	X	NO
EventReportBCSM	Route_Select_Failure	Cause	i_AppCallLeg.callLegEventReport_Err()	PARLAY_CALL_ERROR_ROUTING_ABORTED	YES	X	NO
EventReportBCSM	Route_Select_Failure	Cause	i_AppCall.routeCallToOrigination_Err()	PARLAY_CALL_ERROR_INVALID_ADDRESS	YES	X	NO
EventReportBCSM	Route_Select_Failure	Cause	i_AppCall.routeCallToDestination_Err()	PARLAY_CALL_ERROR_INVALID_ADDRESS	YES	X	NO
EventReportBCSM	Route_Select_Failure	Cause	i_AppCallLeg.callLegEventReport_Err()	PARLAY_CALL_ERROR_INVALID_ADDRESS	YES	X	NO
EventReportBCSM	O_Called_Party_Busy	X	i_AppCall.routeCallToOrigination_Res()	PARLAY_CALL_LEG_REPORT_REFUSED_BUSY	YES	X	NO
EventReportBCSM	O_Called_Party_Busy	X	i_AppCall.routeCallToDestination_Res()	PARLAY_CALL_LEG_REPORT_REFUSED_BUSY	YES	X	NO
EventReportBCSM	O_Called_Party_Busy	X	i_AppCallLeg.callLegEventReport_Res()	PARLAY_CALL_LEG_REPORT_REFUSED_BUSY	YES	X	NO
EventReportBCSM	O_No_Answer	X	i_AppCall.routeCallToOrigination_Res()	PARLAY_CALL_LEG_REPORT_NO_ANSWER	YES	X	NO
EventReportBCSM	O_No_Answer	X	i_AppCall.routeCallToDestination_Res()	PARLAY_CALL_LEG_REPORT_NO_ANSWER	YES	X	NO
EventReportBCSM	O_No_Answer	X	i_AppCallLeg.callLegEventReport_Res()	PARLAY_CALL_LEG_REPORT_NO_ANSWER	YES	X	NO
EventReportBCSM	O_Answer	X	i_AppCall.routeCallToOrigination_Res()	PARLAY_CALL_LEG_REPORT_ANSWER	YES	X	YES
EventReportBCSM	O_Answer	X	i_AppCall.routeCallToDestination_Res()	PARLAY_CALL_LEG_REPORT_ANSWER	YES	X	YES
EventReportBCSM	O_Answer	X	i_AppCallLeg.callLegEventReport_Res()	PARLAY_CALL_LEG_REPORT_ANSWER	YES	X	YES
EventReportBCSM	O_Mid_Call	LegID (originating)	i_AppCallLeg.callLegEventReport_Res()	PARLAY_CALL_LEG_REPORT_SERVICE_CODE	YES	X	YES
EventReportBCSM	O_Mid_Call	LegID (terminating)	i_AppCallLeg.callLegEventReport_Res()	X	YES	PARLAY_CALL_LEG_REPORT_SERVICE_CODE	YES
EventReportBCSM	O_Disconnect	LegID (originating)	i_AppCallLeg.callLegEventReport_Res()	PARLAY_CALL_LEG_REPORT_DISCONNECT	YES	X	YES
EventReportBCSM	O_Disconnect	LegID (terminating)	i_AppCallLeg.callLegEventReport_Res()	X	YES	PARLAY_CALL_LEG_REPORT_DISCONNECT	YES
EventReportBCSM	O_Abandon	X	i_AppCall.routeCallToDestination_Err()	PARLAY_CALL_ERROR_CALL_ABANDONED	YES	X	NO
EventReportBCSM	O_Abandon	X	i_AppCallLeg.callLegEventReport_Err()	PARLAY_CALL_ERROR_CALL_ABANDONED	YES	X	NO

Table 1: Mapping of O_BCSM call events onto Parlay methods invocations

The following table defines the mapping of incoming INAP messages concerning T_BCSM call events onto Parlay methods invocations.

INAP operation	T_BCSM DP	Additional criteria	Parlay method invocation	CallLeg O Event triggered	Trigger CallLeg O object creation?	CallLeg T Event triggered	Trigger CallLeg T object creation?
EventReportBCSM	Term._Attempt_Authorized	X	NO CORRESPONDING METHOD	X	YES	X	NO
EventReportBCSM	T_Busy	X	i_AppCallLeg.callLegEventReport_Res()	PARLAY_CALL_LEG_REPORT_REFUSED_BUSY	YES	X	NO
EventReportBCSM	T_No_Answer	X	i_AppCallLeg.callLegEventReport_Res()	PARLAY_CALL_LEG_REPORT_NO_ANSWER	YES	X	NO
EventReportBCSM	T_Answer	X	i_AppCallLeg.callLegEventReport_Res()	PARLAY_CALL_LEG_REPORT_ANSWER	YES	X	YES
EventReportBCSM	T_Mid_Call	LegID (originating)	i_AppCallLeg.callLegEventReport_Res()	PARLAY_CALL_LEG_REPORT_SERVICE_CODE	YES	X	YES
EventReportBCSM	T_Mid_Call	LegID (terminating)	i_AppCallLeg.callLegEventReport_Res()	X	YES	PARLAY_CALL_LEG_REPORT_SERVICE_CODE	YES
EventReportBCSM	T_Disconnect	LegID (originating)	i_AppCallLeg.callLegEventReport_Res()	PARLAY_CALL_LEG_REPORT_DISCONNECT	YES	X	YES
EventReportBCSM	T_Disconnect	LegID (terminating)	i_AppCallLeg.callLegEventReport_Res()	X	YES	PARLAY_CALL_LEG_REPORT_DISCONNECT	YES
EventReportBCSM	T_Abandon		i_AppCallLeg.callLegEventReport_Err()	PARLAY_CALL_ERROR_CALL_ABANDONED	YES	X	NO

Table 2: Mapping of T_BCSM call events onto Parlay methods invocations

The following table defines the mapping between Parlay method invocations onto O_BCSM detection point arming.

Parlay method invocation	Object invoked	Event requested	Invocation allowed ?	INAP message	O_BCSM DP armed	Additional parameters
i_CallINAP1.getMoreDialledDigits_Req( )	C	X	YES	CollectInformation + RequestReportBCSMEvent	Collect_Information	
i_Call.routeCallToDestination_Req()	C	PARLAY_CALL_RESPONSE_ADDRESSED_PARTY_RESPONSE	YES	RequestReportBCSMEvent	O_Called_Party_Busy + O_No_Answer + O_Answer + Route_Select_Failure	Cause (for O_No_Answer)
i_Call.routeCallToDestination_Req()	C	PARLAY_CALL_REPONSE_ADDRESSED_PARTY_RELEASE	YES	RequestReportBCSMEvent	O_No_Answer + Route_Select_Failure	Cause (for O_No_Answer)
i_Call.routeCallToDestination_Req()	C	PARLAY_CALL_RESPONSE_CALL_END	YES	RequestReportBCSMEvent	O_Abandon + Route_Select_Failure	X
i_Call.routeCallToOrigination_Req()	C	PARLAY_CALL_RESPONSE_ADDRESSED_PARTY_RESPONSE	YES	RequestReportBCSMEvent	?	?
i_Call.routeCallToOrigination_Req()	C	PARLAY_CALL_REPONSE_ADDRESSED_PARTY_RELEASE	YES	RequestReportBCSMEvent	?	?
i_Call.routeCallToOrigination_Req()	C	PARLAY_CALL_RESPONSE_CALL_END	YES	RequestReportBCSMEvent	?	?
i_CallLeg.callLegEventReport_Req()	O_CL	PARLAY_CALL_LEG_REPORT_PROGRESS	YES	NO CORRESPONDING INAP MESSAGE	X	X
i_CallLeg.callLegEventReport_Req()	O_CL	PARLAY_CALL_LEG_REPORT_ROUTING_SUCCESS	YES	RequestReportBCSMEvent	O_Called_Party_Busy + O_No_Answer + O_Answer + O_Abandon	X
i_CallLeg.callLegEventReport_Req()	O_CL	PARLAY_CALL_LEG_REPORT_ANSWER	YES	RequestReportBCSMEvent	O_Answer	X
i_CallLeg.callLegEventReport_Req()	O_CL	PARLAY_CALL_LEG_REPORT_REFUSED_BUSY	YES	RequestReportBCSMEvent	O_Called_Party_Busy	X
i_CallLeg.callLegEventReport_Req()	O_CL	PARLAY_CALL_LEG_REPORT_NO_ANSWER	YES	RequestReportBCSMEvent	O_No_Answer	X
i_CallLeg.callLegEventReport_Req()	O_CL	PARLAY_CALL_LEG_REPORT_DISCONNECT	YES	RequestReportBCSMEvent	O_Disconnect	LegID (originating)
i_CallLeg.callLegEventReport_Req()	O_CL	PARLAY_CALL_LEG_REPORT_REDIRECTED	YES	NO CORRESPONDING INAP MESSAGE	X	X
i_CallLeg.callLegEventReport_Req()	O_CL	PARLAY_CALL_LEG_REPORT_SERVICE_CODE	YES	RequestReportBCSMEvent	O_Mid_Call	LegID (originating)
i_CallLeg.callLegEventReport_Req()	O_CL	PARLAY_CALL_LEG_REPORT_ROUTING_FAILURE	YES	RequestReportBCSMEvent	Route_Select_Failure	X
i_CallLeg.callLegEventReport_Req()	O_CL	PARLAY_CALL_LEG_REPORT_CALL_ENDED	YES	RequestReportBCSMEvent	O_Abandon	X
i_CallLeg.callLegEventReport_Req()	T_CL	PARLAY_CALL_LEG_REPORT_PROGRESS	NO	X	X	X
i_CallLeg.callLegEventReport_Req()	T_CL	PARLAY_CALL_LEG_REPORT_ANSWER	NO	X	X	X
i_CallLeg.callLegEventReport_Req()	T_CL	PARLAY_CALL_LEG_REPORT_REFUSED_BUSY	NO	X	X	X
i_CallLeg.callLegEventReport_Req()	T_CL	PARLAY_CALL_LEG_REPORT_NO_ANSWER	NO	X	X	X
i_CallLeg.callLegEventReport_Req()	T_CL	PARLAY_CALL_LEG_REPORT_DISCONNECT	YES	RequestReportBCSMEvent	O_Disconnect	LegID (terminating)
i_CallLeg.callLegEventReport_Req()	T_CL	PARLAY_CALL_LEG_REPORT_REDIRECTED	NO	X	X	X
i_CallLeg.callLegEventReport_Req()	T_CL	PARLAY_CALL_LEG_REPORT_SERVICE_CODE	YES	RequestReportBCSMEvent	O_Mid_Call	X
i_CallLeg.callLegEventReport_Req()	T_CL	PARLAY_CALL_LEG_REPORT_ROUTING_FAILURE	NO	X	X	X
i_CallLeg.callLegEventReport_Req()	T_CL	PARLAY_CALL_LEG_REPORT_CALL_ENDED	NO	X	X	X

Table 3: Mapping of Parlay method invocations onto O_BCSM DP arming

The following table defines the mapping between Parlay method invocations onto T_BCSM detection point arming.

Parlay method invocation	Object invoked	Event requested	Invocation allowed ?	INAP message	T_BCSM DP armed	Additional parameters
i_CallLeg.callLegEventReport_Req()	O_CL	PARLAY_CALL_LEG_REPORT_PROGRESS	YES	RequestReportBCSMEvent	Term.Attempt_Authorized	X
i_CallLeg.callLegEventReport_Req()	O_CL	PARLAY_CALL_LEG_REPORT_ROUTING_SUCCESS	YES	NO CORRESPONDING INAP MESSAGE	X	X
i_CallLeg.callLegEventReport_Req()	O_CL	PARLAY_CALL_LEG_REPORT_ANSWER	YES	RequestReportBCSMEvent	T_Answer	X
i_CallLeg.callLegEventReport_Req()	O_CL	PARLAY_CALL_LEG_REPORT_REFUSED_BUSY	YES	RequestReportBCSMEvent	T_Busy	X
i_CallLeg.callLegEventReport_Req()	O_CL	PARLAY_CALL_LEG_REPORT_NO_ANSWER	YES	RequestReportBCSMEvent	T_No_Answer	X
i_CallLeg.callLegEventReport_Req()	O_CL	PARLAY_CALL_LEG_REPORT_DISCONNECT	YES	RequestReportBCSMEvent	T_Disconnect	LegID (originating)
i_CallLeg.callLegEventReport_Req()	O_CL	PARLAY_CALL_LEG_REPORT_REDIRECTED	YES	NO CORRESPONDING INAP MESSAGE	X	X
i_CallLeg.callLegEventReport_Req()	O_CL	PARLAY_CALL_LEG_REPORT_SERVICE_CODE	YES	RequestReportBCSMEvent	T_Mid_Call	LegID (originating)
i_CallLeg.callLegEventReport_Req()	O_CL	PARLAY_CALL_LEG_REPORT_ROUTING_FAILURE	YES	NO CORRESPONDING INAP MESSAGE	X	X
i_CallLeg.callLegEventReport_Req()	O_CL	PARLAY_CALL_LEG_REPORT_CALL_ENDED	YES	RequestReportBCSMEvent	T_Abandon	X
i_CallLeg.callLegEventReport_Req()	T_CL	PARLAY_CALL_LEG_REPORT_PROGRESS	NO	X	X	X
i_CallLeg.callLegEventReport_Req()	T_CL	PARLAY_CALL_LEG_REPORT_ANSWER	NO	X	X	X
i_CallLeg.callLegEventReport_Req()	T_CL	PARLAY_CALL_LEG_REPORT_REFUSED_BUSY	NO	X	X	X
i_CallLeg.callLegEventReport_Req()	T_CL	PARLAY_CALL_LEG_REPORT_NO_ANSWER	NO	X	X	X
i_CallLeg.callLegEventReport_Req()	T_CL	PARLAY_CALL_LEG_REPORT_DISCONNECT	YES	RequestReportBCSMEvent	T_Disconnect	LegID (terminating)
i_CallLeg.callLegEventReport_Req()	T_CL	PARLAY_CALL_LEG_REPORT_REDIRECTED	NO	X	X	X
i_CallLeg.callLegEventReport_Req()	T_CL	PARLAY_CALL_LEG_REPORT_SERVICE_CODE	YES	RequestReportBCSMEvent	T_Mid_Call	X
i_CallLeg.callLegEventReport_Req()	T_CL	PARLAY_CALL_LEG_REPORT_ROUTING_FAILURE	NO	X	X	X
i_CallLeg.callLegEventReport_Req()	T_CL	PARLAY_CALL_LEG_REPORT_CALL_ENDED	NO	X	X	X

Table 4: Mapping of Parlay method invocation onto T_BCSM DP arming

5. Management interfaces

If an enterprise wants to offer services to the intelligent network via the ITAU, a business relationship between the owner of the IN platform at one hand and the enterprise at the other is required. Besides the business relationship, for each individual service offered via the ITAU the enterprise and the IN domain need to agree upon a service contract that describes the service level agreements. Ingredients of this service level agreement include the maximum number of instances allowed, the time of day these services can be used, the billing scheme agreed upon, the time and day a service started and ended, the contact person, etc. These aspects are already covered by the TINA subscription management specifications. What need to be added are the specific subscription ingredients that are needed to support an ITAU. Examples of these specific ingredients include the use of specific network resources, the messages that need to be played, the criteria needed for a service to start, etc. This section focuses on these latter specific subscription services, referring to TINA subscription management specifications for general subscription services.

All services offered via the ITAU consist of a mandatory part and an optional part. The mandatory part describes the requirements and specifies the conditions that must hold before the service can be used. The optional part indicates what the ITAU should do on behalf of the enterprise before the service provider is consulted. This optional part lowers the communication between the ITAU and the enterprise that offers the service, avoiding the need to exchange information real time that can be configured in advance.

An example of the mandatory part is whether the service requires a named or an anonymous access session. The service provider offering its services via the ITAU needs to specify this information in order to realise a correct settlement of the service. Recall that the user of the service is an end-user, and not the ITAU. The ITAU only represents the end-user. If the service requires a named access session, the adaptation unit needs to provide the password. If the password is not part of the `requestNamedAccess` operation executed by the adaptation unit, the retailer will throw an exception and the password needs to be resolved by querying the end-user. The retailer should indicate how to resolve the exception, what voice response system should be used, which message should be played, etc. Another solution is that a service optional part includes the possibility to resolve the consumer's password by the ITAU before the actual service needs to be started. To realise this, the service provider who offers its services via the ITAU needs to indicate which resources in the network must be used to resolve this information, which messages must be played, how the information must be gathered, what kind of information can be expected, etc. By making use of the optional part, the configuration needs to be done once, and can be applied each time the service is invoked.
Remark that the use of resources in the network prior to the establishment of the access session is not limited to resolving password information. Another option is to play specific messages, depending on e.g. time of day, day of month, etc. Knowing this, it is also clear that the service optional part is not restricted to services that require a Named access session, but can also be applied to services that demand an anonymous access session.

The management interface allows the enterprise, among other things, to configure the mandatory and optional settings that apply to a service that she desires to offer via the ITAU. The configuration describes the way the service can be triggered, but also captures the service criteria, the voice respond system that needs to be used when the service is triggered, whether an anonymous or a named access is required, the information that is supposed to be exchanged, etc. As starting point for the ITAU management interface, the TINA subscription management specification is chosen. This specification is described in detail in [9, version 1.0b]. As a matter of fact, the management interface is constructed such that the TINA specifications for subscription management are further resolved, and require no changes.

5.1. Common types

Type definitions needed for the management interface are as follows:

```
typedef long t_Duration;
typedef long t_RangeLength;
typedef short t_DigitPressed;
enum t_OtherKeyPressed {PoundSign, Star};

union t_NetworkAddress switch(short) {
```

```
      case 1: string IPAddress;
      case 2: string SS7Address;
      case 3: long E164;
      case 5: long GSM;
      case 6: string URL;
      case 7: string addressNotKnown;
};

enum t_AddressType{
      CallFromOrigination,
      CallFromDestination,
      CallOfOriginalDestinationAddress,
      RedirectingCallAddress
};


struct t_Address {
      string addressName;
      t_AddressType adressType;
      t_NetworkAddress networkAdress;
      boolean fixedRange;
      //may be convenient to know; also in INAP
      t_RangeLength rangeLength;
      //the range value, if appropriate
};

struct t_Resource {
      string ResourceName;
      boolean disConnectFromResourceForbidden; //default = true
      t_Address networkAddress;
};

enum t_templateErrorType {unKnownServiceTemplate, InvalidServiceTemplate};
enum t_templateState {active, passive};
```

The duration can e.g. be used to indicate the maximum length of a message that can be played and should be measured in seconds. The union `t_AddressType` allows addressing resources by different address types; the network address of a resource that needs to be used is subsequently specified by `t_Address`. The range length of, for instance, the digits that need to be provided can be restricted by `t_RangeLength`. The TINA subscription management is extended by two template related issues, namely `t_templateState` and `t_templateErrorType`. The latter allows a flexible use of exceptions, whereas the first is needed as soon as you allow a service template to be in more positions than only the active state. The resource that can be used in the IN domain, for services outsourced via the ITAU, is indicated by `t_Resource`.

Service offered via the ITAU can be described by using the `t_ServiceProfile` as introduced in [9, version 1.0b]. To realise this with a minimum of changes to TINA's subscription management specifications, the type definition of `t_ServiceDescription` (the primary part of `t_ServiceProfile`) needs to be specified. The specification `t_ServiceDescription` consists of 4 elements: a serviceId, a user service name, a list of service common parameters and a list of service specific parameters. The management interface of the ITAU is constructed by specifying a set of common service parameters. These service parameters are specific for the ITAU and therefore need to be identified as such. The use of an ITAU service is outlined by using a service id that starts with the number 4828. Why choose 4828? Well, we need to identify an ITAU service some how, and the numbers 4828 spell – among other words – ITAU on the keypad of a telephone set. Summarising the above, all service ids provided by the ITAU **must** start with 4828. If not, the ITAU is non-compliant to the specification answered in this RFP.
Recall that the parameter type as specified in [9] already has a field that describes the extent to which the parameter can be configured.

5.1.1. Mandatory parameters

This subsection describes the parameters that are compulsory when offering telephony services via the ITAU.

- The type of access session that is requested for the ITAU service is obligatory. This type is described by `t_AccessType`, defined as follows:
```
enum t_AccessType {NamedAccess, AnonymousAccess};
```

- The criteria for the service offered via the ITAU. These criteria are specified as follows:
```
union t_ServiceCriteria switch(unsigned short) {
      case 1:long dialedNumber;
      case 2:long callingLineID;
      case 3: string serviceKey;
      //e.g. detection point; maybe typed as a short?
      case 4: boolean serviceNotAvailable; //default = false
      default: t_AddressAndService addressAndService;
};
```
In this context, t_AddressAndService is typed as follows:
```
struct t_AddressAndService {
      long calledAddress;
      string serviceKey;
      long callingAddress;
      long locationNumber;
};
```
- The information that needs to be sent to the party that initiated the service is defined in the structure called t_InformationToSend. It is typed as follows.
```
Struct t_InformationToSend {
      t_MessagePlayed message;
      string displayInformation;
      t_Tone tone;
};
```
The newly introduced information fields to realise this are typed as follows:
```
struct t_MessagePlayed {
      string messageName;
      t_Resource resourceToBeUsed;
      long messageValue;
      long numberOfRepetitions;
      t_Duration duration;
};
```
where,
```
struct t_Tone {
      long toneID;
      t_Duration duration;
      long frequency;              // measured in [Hz]
};
```

- The access network used by the party involved in the service. Depending on the access network used, the service offered via the ITAU may differ. As an example, if the calling consumer uses an ISDN access network, the service provider may send information to be displayed at his telephone set, whereas this is not possible for POTS consumers. Other information that can be useful for the service provider to know is the minimum transmission capacity of the access network used. It is typed as follows
```
struct t_AccessNetworkUsed{
   t_CalledAccessNetworkType accessNetworkUsed;
   t_CalledAccessNetworkBearerCapacity capacityAccessNetwork;
};
```
- Call event. If the party who is involved in the service presses during this service one of the keys on the telephone pad, e.g., a number, the flash key or another key, the service may react correspondingly on this

action. For instance, pressing ‘9’ may mean to end the service. The enterprise that offers a service needs to indicate whether possibilities in the area of mid call events are sustained during the service, such that the ITAU knows whether it should take action or ignore it. The corresponding action is performed at the call control level. It is typed as follows:

```
struct t_MidCallEvent {
   string eventName;
   boolean midCallEventSustained;
   t_EventType eventType;
};
```

where,

```
union t_EventType switch (short) {
   case 1: t_OtherKeyPressed otherKeyPressed;
   case 2: t_DigitPressed digitPressed;
};
```

5.1.2. Optional parameters

Just as the mandatory parameters, the optional parameters for telephony services offered via the ITAU must have a service id that starts with 4828. As the title already gives away, the enterprise that offers a service via the ITAU is not obligatory to specify the service description parameters described in this section. This means that given a service offered via the ITAU, the parameters explained in this section may be used, but not have to.

- Services offered via the ITAU may require that the IN domain gathers information prior to the actual start of the service. This lowers the communication between the ITAU and the enterprise that offers the service, not needing to exchange information real time that is known in advance. If the enterprise that offers the telephone service via the ITAU requests that information needs to be gathered, it should be typed as follows.

```
struct t_CollectedInfoNeeded {
   t_Resource resourceToBeUsed;
   unsigned long minumumNumberOfDigits;
   unsigned long maximumNumberOfDigits;
   short numberOfDigits;
   boolean fixedLength;
   string endOfReplyDigit;
   t_RecordedMessage speech;
};
```

where,

```
struct t_RecordedMessage {
      t_Duration maxDuration;
      t_Duration timeOut;
      string recordedMessage;
};
```

5.1.3. Exceptions

In what follows, we describe the exceptions that are needed for the control of service templates.

- The parameter given is invalid.

```
exception e_invalidParameter {
SubscriptionCommonTypes::t_Parameter parameter;
};
```

- The service id given is not known to the ITAU

```
exception e_unknownServiceId {
AccessCommonTypes::t_ServiceId serviceId;
```

```
};
```

- The parameter given is not expected in e.g. the access type used for this service.

```
exception e_unExpectedParameter {
string unExpectedParameter;
};
```

- The template given does not contain one or more parameters that are classified as mandatory.

```
exception e_missingParameter {
string mandatoryParameter;
```

5.2. Interfaces

A single interface covers all the operations needed to define and modify service templates, and not yet defined in TINA's subscription management specifications. Exceptions introduced for this interface are `e_invalidTemplate` and `e_invalidTemplateState`. The interface that enables service template control is called `i_ServiceTemplateControl` and covers the following operations.

- **listServiceTemplates** () – It returns the list of service ids that are related to the templates associated with the service contract. Recall that a TINA service contract is uniquely identified by the combination of a service id and the account number.
- **getServiceTemplate** () – It returns a service template, associated by the service id provided.
- **deActivateService** () – It deactivates a particular service, associated by the service id provided.
- **reActivateService** () – It re-activates a particular service, associated by the service id provided.
- **addServiceTemplate** () – It adds a particular service template to the service contract and, as such, adds a new service that becomes available via the ITAU. If the template does not confirm to the service contract, the exception `e_invalidTemplate` is raised. As mentioned before, the template requires some specific parameters. If a mandatory parameter is missing, a given parameter is not correct or if a parameter is specified that is not confirming to the service contract, the exceptions `e_missingParameter, e_unExpectedParameter, e_invalidParameter` are raised accordingly.
- **modifyServiceTemplate** () – It modifies an already existing service template. If the service template does not exist yet, the exception `e_invalidTemplate` is raised. If a mandatory parameter is missing, a given parameter is not correct or if a parameter is specified that is not confirming to the service contract, the exceptions `e_missingParameter, e_unExpectedParameter, e_invalidParameter` are raised accordingly.
- **removeServiceTemplate** () It removes a service template from the associated service contract. If the service ID provided is not correct, or if no service template is associated with the service id given, the exception `e_unknownServiceId` is raised.

6. Answer to specific issues

6.1. MSCs for Terminating Screening and Credit Card Calling Service

In this section, we use some Message Sequence Charts (MSC) to describe how the two RfP services, the terminating call screening service and the credit card calling service, can be realised with the proposed ITAU.

6.1.1. Terminating Screening Service

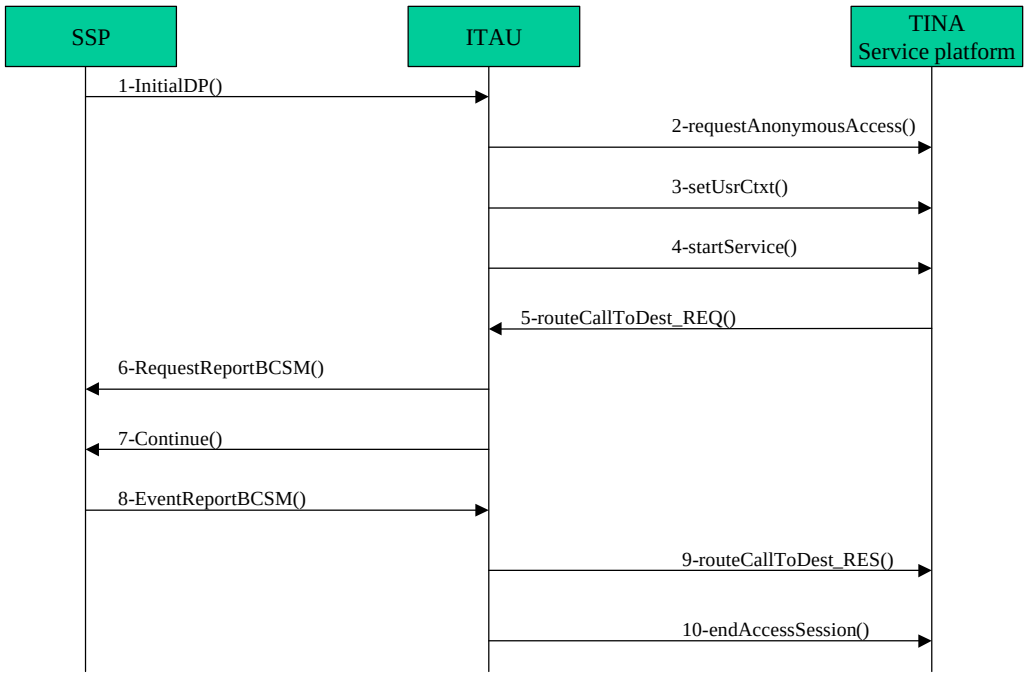

Figure 7. Call Screening PARLAY style , and TINA access; number already complete

Description:
2. requestAnonymousAccess(): dummy access that executes common parts of services
3. setUsrCtxt(): informs about consumer domain configuration
4. startService(): the service execution starts
5. routeCallToDest_Req(): the will be tried to establish
9. routeCallToDest_Res(): The application will be informed about the status of the call (busy, answer...)
10. endAccessSession(): now we have a TINA compliant end of the session. We think there is no problem in terminating the access session initiated from the ITAU

6.1.2. CallingCardService

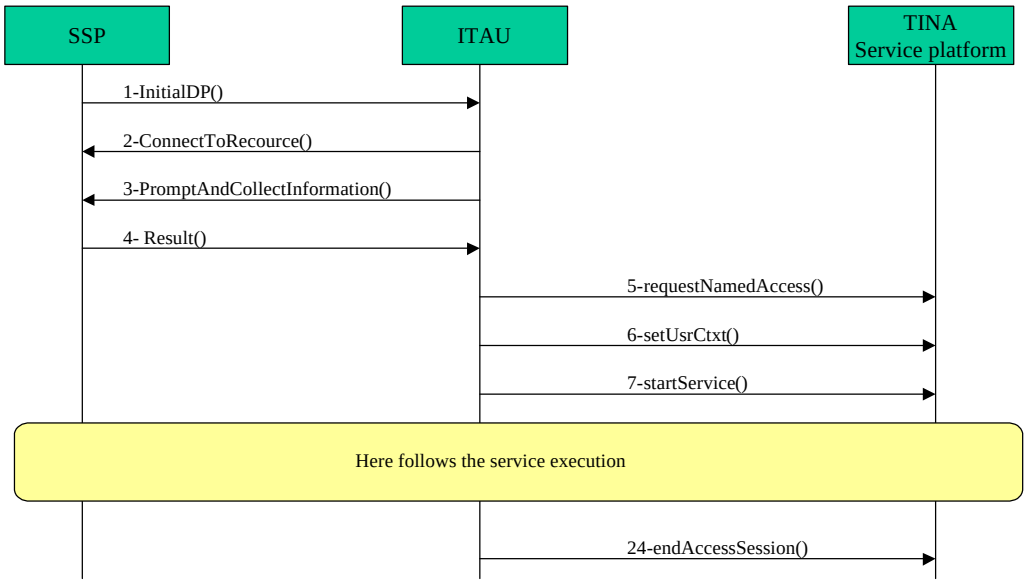

Figure 8. Access of CallingCardService

Description:
5. requestNamedAccess(): establishing access session for calling card user
6. setUsrCtxt(): informs about consumer domain configuration
7. startService(): the service execution starts

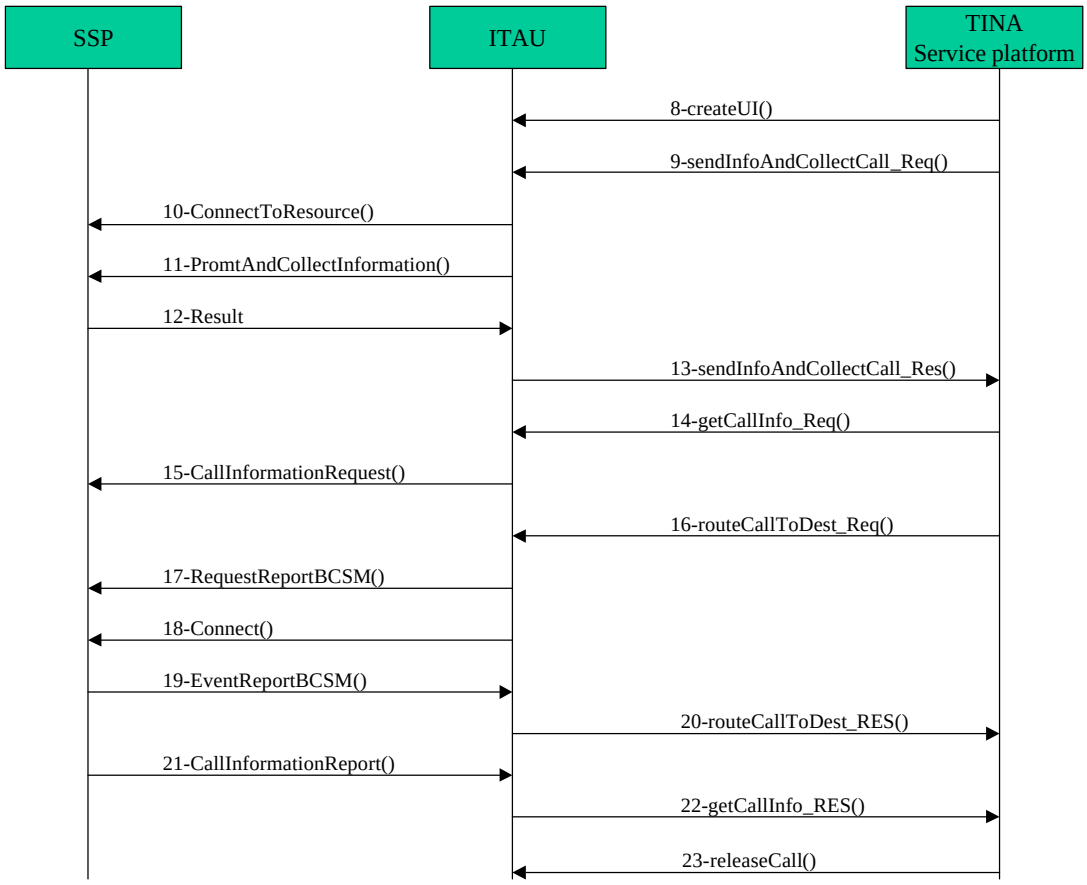

Figure 9. Service part of Calling Card Service

8. createUI(): retrieve an user interaction interface (special resource), this means the physical equipment will be automatically chosen
9. plays an announcement and requests the user input
13. retrieves the information given by the user
14. request call related information (duration of the call time and date)
16. starting the establishment of the call to the desired destination
20. retrieves status of the call
22. answer to 14. received
23. release the call

6.2. How to use TINA Ret Usage

This section explains how to translate INAP into the usage part of the Retailer Reference Point. The main motivation for this mapping is to define, based on TINA previous works, a service architecture for IN, i. e. to (re)define service components which can be reused in telecommunication services, or only in IN-like services. The definition of this components is expected to ease reusability, service composition and federation.

6.2.1. Overview

The usage part of Ret-RP can be split into the TINA Service Session Model and TINA Communication Session Model. The TINA Service session model can be itself divided in Feature Sets, which correspond to functionalities an end user can adopt or not once the service is launched. As it is explained in the following section, a subset of the TINA Service Session Model, i. e. some feature sets, is enough to express IN capabilities in this TINA formalism. Concerning the Communication Session model, an altered use of the interfaces defined in the Retailer Reference Point [6] could be proposed for IN [1]. In this document, a simple telephony-dedicated communication session model is proposed in 6.2.3. This addition is due to performance reason, rather than to TINA limitations, and takes into account the two telephony specificities: first, the simple hop-by-hop bi-directional connection, which does not require a TINA connection management, second the possibility of a called party to be busy or non answering. This telephony communication session model can of course be used for

a IN to TINA call, and the CSM in this case can be compliant with this telephony communication session model on the IN side, and still compliant with the classical TINA communication session on the other side.
As defined in [8], TINA services include at least the following services: telecommunication services, management services, and information services. The usage defined in [6] proposes interfaces for telecommunication services, and could be extended easily for information services such as video on demand. They are designed to be independent of service and network infrastructure, and to be accessed universally, independently of the terminal location and type. Even if the TINA usage is optional, the motivation for reusing the classical usage interface is to offer:

- the IN access to the services which are accessible by classical TINA terminals (DPE-enabled). Then the service developed following the TINA principles is really accessible by different networks.
- a unified and long term solution for telecommunication services, since a real architecture is here proposed in the service side. Rapid new services development, service composition require a modular reusable and extendable service architecture.

Eventually, information services such as vocal mail services could be defined with extensions of the service session model, for resource control.

6.2.2. Service Session interfaces

Several feature sets, composing the service session model, as defined in [6], seem irrelevant in the IN environment. The vote, the Stream Binding information and the resource control feature sets do not have a strong meaning for a simple IN call: the IN services are simple, compared to the many possibilities a TINA service can offer. Anyway, at the service beginning, the user communicates the interfaces available in its domain, thus choosing the feature sets which will be uses in this particular service session. It is then expected that an End User proxy will declare only the relevant feature set it needs for an IN service.
Therefore, it is proposed to avoid the use of the Vote, the Stream Binding information and the Resource Control Feature Sets since they offer capacities which are never used in IN CS-1 services. Eventually, they could be implemented on each side of Ret RP (in the SSM and in the ssUap, located in the adaptation unit), but since the ssUap would not need to translate them in INAP, it is more efficient to decline their use the soonest, i.e. at the service initialisation (setPartyCtxt()), in order to limit the unnecessary communication between the ITAU and the TINA service. Unlike those three feature sets, two others have a strong meaning in IN and must be used. First, the invitation feature set will cover the permission to route a call, and the stream binding will be used to describe the connection between the caller and other parties. The Communication session as defined in [6] will be used to perform the IN connection.

The Basic Feature Set
Obviously, the Basic Feature set is mandatory, and is used to start the service session interactions and to end it. However, according to [24] is proposed to use **`setPartyCtxt()`** instead of the **`registerInterfaces().`** The interface of the Basic Feature Set **`i_ProviderBasicReq`** is defined in annex.

The Multiparty Feature Set
The Multiparty feature set must be supported in the case the adaptation unit and the service are compliant to the TINA usage. The mandatory interfaces are **`i_PartyMultipartyExe`** and **`i_ProviderMultipartyReq.`** The **`i_ProviderMultipartyInfo`** interface is not mandatory in the ITAU case, since the invitation acknowledgement is quite untranslatable in INAP, but it can be implemented, if the adaptation unit developer thinks it can be useful (for example a message is played informing the caller a new party is about to join the session). It is known that the choice of adopting it or not has no consequence on the service implementation.

When the service session is handled by the EUP, its ssUap connects to the USM and if it needs to be connected to a party, or resource, i.e. to establish a connection in the IN, it should follow the TINA procedure, which begins by sending an invitation. The invitation does not appear in INAP, but it formalises a very common IN mechanism. For instance, an 800 service may check conditions, e.g. the caller identity and the destination of the call, and if it is authorised, the call can be routed. Similarly, a black list service is all about checking the caller identity, typically in the called party UA. In these cases, the IN service, or at least a part of it, can be seen as an invitation filtering service. The Multiparty Feature Set invitation provides fully this capability. Once the service

session has been launched, the EUP must invoke the **i_PartyMultiPartyReq::inviteUserReq()**. This invitation must be accepted if the service logic allows the connection. For an 800 call, for instance, if the calling party location is in the good area, and if the date is in the right time frame, an IN service logic accepts the call and then redirect it with the INAP **connect()** operation. Here, the service logic should first answer positively the invitation, allowing the EUP to request a stream binding establishment with the called party.

The Participant Oriented Stream Binding Feature Set

The Participant Oriented Stream Binding Feature Set must be supported in the case the adaptation unit and the service are compliant to the TINA usage. The needed interfaces are**: i_PartyPaSBExe, i_PartyPaSBInfo** and **i_ProviderPaSBReq.**

If the invitation is positively answered by the SSM, the following TINA Ret-RP operation is the establishment of a stream binding. Here, The EUP would invoke an **i_ProviderPaSBReq::addProviderPaSBReq()** operation. The parameters would be :

```
i_ProviderPaSBReq::addProviderPaSBReq(
  in TINACommonTypes::t_ParticipantSecretId myId,
  in TINAStreamCommonTypes::t_SBType sbType,
  in TINASBCommSCommonTypes::t_MediaDescList media (QoS phone or mobile
type: audio)
  in TINAPaSBTypes::t_ParticipantDescList reqMembers,
  in TINAStreamCommonTypes::t_SFEPServDescList requesterSIs,
  in TINAStreamCommonTypes::t_SBSuccessCriteria criteria,
  in TINAStreamCommonTypes::t_SBRecover recActions,
  in boolean wait,
  out TINAStreamCommonTypes::t_SBBindState status );
```

The **sbtype** is a string.

media is a list of **t_TypeDesc**, which is a complex data type to describe and set requirements for a media type within a stream binding. It is composed of a pair of **t_typeId** and a **t_attribList**. The value of **t_typeId** should be "**phoneCall**", to define the QoS of the connection.

The **reqMembers** type contains the description of the parties to be connected with. In most cases, there is only two parties in an IN call, so there should be one party described in this data.

The **requesterSIs** is a list of the following structure::

```
struct t_SFEPServDesc
{
    t_SFEPId id;
    TINASBCommSCommonTypes::t_SFEPComDesc desc;
    t_SFEPBindName tag;
    t_SIName si;
    Object siIfRef;
};
```

As the IN call is a by directional connection, there will be two SFEPs to communicate to the service, thus two SFEP descriptions. The description of the SFEP is in the type: **t_SFEPComDesc.**

```
struct t_SFEPComDesc
{
    t_SFEPName name;
    t_SFEPDirection dir;
    t_AdministrativeState adState;
    t_MediaDesc media;
    Object ifRef;
};
```

One SFEP will have the direction source, and the other the sink.
The **ifRef** is the TCSM interface reference.

It is expected that the wait value is false, as this method should return is this case after the actual IN connection. Because this method could block quite a long time, until the time the call is re-routed, and the called party unhooks.

At this step, the service has requested the establishment of a connection for a phone call, i.e. s stream binding with two SFEPs managed by the EUP, to be connected with the invited party. The communication session is then launched to resolved the SFEP and set-up the Stream flow connections.

6.2.3. Communication Session interfaces

The Communication Session is used in TINA to determine the network flow end points, i.e. network addresses, and to establish the connection between them. Thus, the communication session can be compared in the IN world to the routable number retrieval. In this section, a communication session dedicated to IN is proposed, in order to cope with the IN specificities, and to limit the overhead due to the whole TINA communication procedures, which are quite complex for a simple POTS connection. This alternative communication prevents the use of the ConS and TCon Reference Points for a classical call from and to the IN, but does not preclude its use in case the call is re-routed to a layer network controlled by a TINA connection management. This chapter explains this communication session model, its impact on the CSM and the way the TCSM translates the communication session into INAP. The new TCSM interface explained in the following does not preclude the use of the classical TCSM interface with the other parties, in the case of an IN to TINA call (e.g. IN to Internet).

Overview

The TCSM interface is similar to the classical one defined in [6], but the processing of the CSM has been changed, in order limit the number of CORBA interactions. This adaptation of the communication session is proposed to make it efficient, and to avoid a connectivity session through ConS and the use of TCon Reference Point. However, the CSM should integrate this customisation and adopt the IN dedicated behaviour when needed, but it can also keep its classical TINA behaviour, in case of a real generic TINA platform.

Justification of the simplified communication session model

Since IN relies on the classical hop-by-hop routing mechanism of the Switched Circuit technology, the connection part of the IN service logic does not address the connection management, as it is defined in the Network Resource Architecture. IN SCPs does not control the network switches in a top-down mode, they only give routing and charging information that the classical call control functions inside the switches can process.
Moreover, the INAP connection related mechanisms are very simple, compared to the Communication and Connectivity Session, which offer advanced capacities, not needed in IN. What is needed in IN is the communication of an E.164 number, or even nothing, when the call just need to be continued. The powerful interfaces defined in the Communication and Connectivity Session allows advanced stream bindings, based on unidirectional stream flow connection between, possibility to choose different QoS, technologies and codings, the possibility to choose a connectivity provider, etc… These capacities are not offered in IN: the connection will always be a two way voice connection, already initiated in the network operator domain. These capacities introduce an overhead which must be avoided in the IN case, where performances remain an important issue, and where the benefit of these TINA mechanism is not straightforward.
Because of these considerations, an adapted Communication Session Model is proposed, in order to cope efficiently with the IN limitations. This change should be implemented in the generic CSM as an alternative behaviour when the caller meet the IN conditions, but a service platform only accessible through ITAU could only implement this IN dedicated communication session.

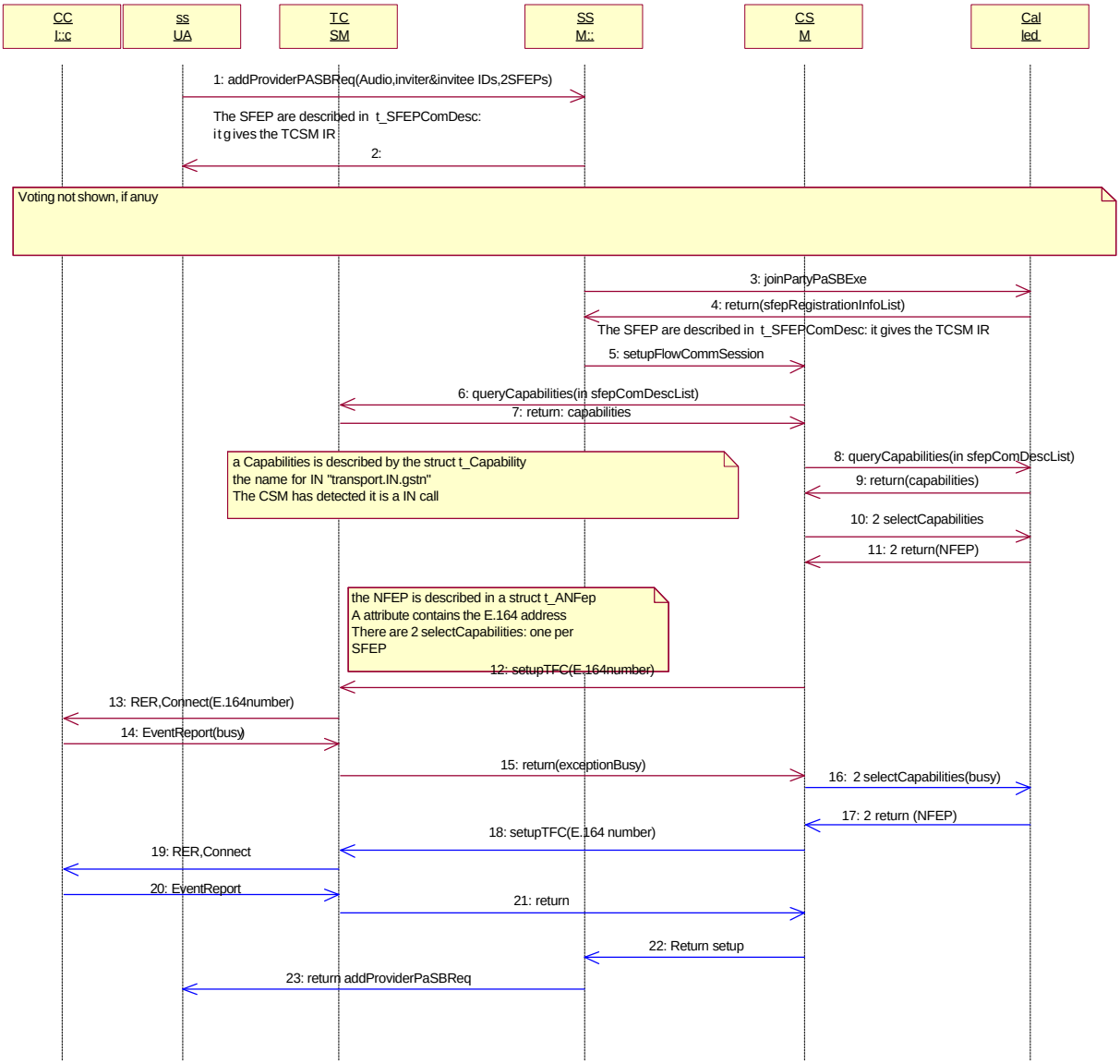

Figure 10. The adapted communication session

Description of the simplified communication session model

The idea is to bypass the use of ConS between the CSM and a CC , and then the TCon use between the LNC and the TLA, and finally to adapt the communication session to the IN needs.

Because the SSP only requires a simple phone number, without QoS or coding requirements, the Terminal Flow Connection, binding the SFEPs (sink and source) to a bi-directional NFEP is enough. Here, the operation connects the SFEPs, that is to say a TINA representation of the initiating half call, to the NFEP which will be the encapsulation of the call destination address. This idea is a systematisation of the existing situation where the TINA terminal is already distributed in the telephone network: the user interface is performed by the phone set and the network node implementing the SRF, and that the end user's service components are in the ITAU.

This trick requires a technology dependant CSM, or at least an additional behaviour in the classical CSM. When the CSM detects the NFEP from the calling party is in the IN, the result of the **`selectCapabilities()`** invoked on the called party, the destination NFEP, must be send through the **`setupTFC()`**. In Fig. 10: The

adapted communication session, the mechanism is shown and commented. The interaction between CSM and the called party are out of the scope of the ITAU, but explains the E.164 number retrieval. The **queryCapabilities** invocations are performed normally by the CSM, and the CSM detects with the capabilities name that the call is initiated by an IN user: the **lnType** of the returned Nfep is equal to "**IN_halfcall**". At this step, the CSM knows the initiating party is in the IN world, thus will not invoke a **selectCapabilities()**on the EUP's TCSM, but asks possibly through a **selectCapabilities()** the routable number to the called party. In the return value, the called party gives its routable number, and this NFEP which must be an E.164 address can be returned to the ITAU as the NFEP in the **setupTFC**. If the connection can not be performed, for example because the called party is busy or non answering, the CSM would ask with another **selectCapabilities** a new NFEP/routable number to the called party, until the called party decide the call to be aborted.. In the case, the **selectCapabilities** will return a abandon value, that will be transmitted to the SSM. Then, the SSM can send back to the EUP' s ssUap the **t_PaSBSetupErrors** exception, with possible value: **PaSBSetup_CommsNotAvailable** or **PaSBSetup_NoPathFound**. On the called party TCSM, the **selectCapabilities** parameters must be enriched by a parameter giving the value of the previous attempt: busy or no_answering.

In the case the called party is not in the GSTN, the CSM could eventually retrieve an address of a media gateway, in order to redirect the call to another technologic network. Then the CSM can of course use, through ConS-RP, a connection management, and an adapter [10, section 4.4.2.3] which encapsulates the control of the media gateway. However, the TINA connection management which could control the connection from the media gateway to the called party terminal through a TINA transport network must been implemented outside the ITAU and is therefore out of the scope of the RFP.

6.3. Intelligent Peripheral Interface

6.3.1. The interface description

In this chapter the optional requirement concerning the IN special resource is answered. The external Intelligent Peripheral (IP) is currently INAP interfaced, to be controlled by the IN service logic. However, its INAP interface could be replaced by a CORBA based interface for non functional reasons. A CORBA based service could then access the IP directly through CORBA instead of SS7.
Another possible work could be to define the whole integration of the IP in a TINA architecture with an associated complex IP control API such as the Call Control Interface. This could allow to specify the use of vocal resources for TINA information services, e.g. a unified messaging service. This work is not addressed in this document and is for further studies.

In the IN-TINA configuration defined in this document, the ITAU isolates the network technology adaptation, here, the narrow-band voice network. Since the End User Proxy represents the end user in the IN service layer, it should handle one TCAP dialogue with the SSP and should translate it in the TINA side. If the IP is not integrated in an SSP and is INAP interfaced, a new TCAP dialogue is launched with an **AssistRequestInstruction** invocation. The SCP must then detect the correlation of this new TCAP dialogue with the corresponding SLI. Of course, in the TINA configuration, the IP still needs to begin a communication with the IN TINA service. Because the IP use is an IN technology mechanism, this should be done by the ITAU. That is particularly obvious if the IP is TCAP interfaced. However, as the same SLI controls two TCAP dialogues, the same EUP must control the IP for the corresponding call. Then the EUP has a complete control of the network and the service. The adopted design for the CORBA IP interface is then the EUP/specialised resource object (SRO) which represents a call in the IP.
However, the whole service could be designed in another way, where the SSM itself, or the called party components access the IP to control it, bypassing the ITAU. In this configuration, the same CORBA interfaces are still valid, although the **i_EUP_srControlCS1** interface is not the interface of one of the ITAU EUPs, but the interface of the SSM or of another component involved in the service session.

The CORBA interfaces defined for the IP IN-TINA service communication are the following:
i_EUPdirectory, which allows the brokering of EUP references
i_EUP_srControlCSl, which allows the EUP to receive messages from the IP
i_SRF_callCS1, which allows the specialized resource object to receive messages from the IN-TINA service

6.3.2. Initiating the dialogue

When the IP receives an assist indication from an initiating SSP, the IP instantiates a SRO, and needs to communicate its i_SRF_callCS1 reference to the EUP, as well it needs to get the EUP's **i_EUP_srControlCSl** reference. The **i_EUPdirectory** object implements the brokerage of EUP reference. This object is chosen by the SRF, according to the scfID given in the **EstablishTemparyConnection** operation invoked to reconnect the call to the IP. As the mapping from the scfID to an SCP SS7 address with the INAP **AssistRequestIntruction** operation, the mapping of scfID on the CORBA reference is decided by the network operator. A straightforward solution is to instantiate an EUP directory object in every ITAU, in order to have a simple relationship between scfID and **i_EUPdirectory** reference.
Executing the **retrieveEUP** method, the **i_EUPdirectory** implementation object transmits the **i_SRF_callCS1** reference to the corresponding EUP according to the correlationID, and gets the EUP reference to send as a return value.

6.3.3. Interaction with the call party

The i_SRF_callCS1 object implements **playAnnouncement, cancel** and **promptAndCollectUserInformation** which are the counterpart of the same name CS-1 INAP operation.
To cope with the INAP transaction mechanism, the following INAP to CORBA mapping is proposed:

- The INAP primitive is translated by a synchronous operation in CORBA, with a void return value.
- The CORBA arguments are the same than in the INAP operation, plus an ID determining the operation for the possible exception, and an additional boolean **lastComp** parameter determining if the counterpart of the TCAP transaction is ended. If the boolean is set to false, there is another operation to come and to be processed as it should be done if these operation were in the same TCAP message. If it is set to true, the operation should be considered as the last of the TCAP transaction: the previous operations and this one should be executed. If **lastComp** is set to true for every operation, the CORBA dialogue is almost classical, with operation exception defined as a CORBA operation, and if **lastComp** is used to emulate a TCAP transaction, i.e. set to false for at least one operation, the TCAP execution behaviour must be adopted.
- If the processing of the invocation triggers an error, the corresponding CORBA operation of the **i_EUP_srControlCSl** must be invoked. The ID allows the SRO to determine which invocation has failed.

6.3.4. Disconnection

When the connection from the initiating SSF is released, the specialised resource object can be deleted or reinitialised. In the ITAU, the EUP which sent the **DisconnectForwardConnection** operation knows that the i_SRF_callCS1 reference is no more valid.

6.3.5. Interface description

The IDL interfaces of the EUP and the specialised resource object implementing i_SRF_callCS1, are closely inspired from the TCAP primitives. To deal with the transaction capabilities and the cancellation of operation, the primitives are mapped onto CORBA operation with an operation ID, which is the CORBA counterpart of the component ID. The return value of the INAP primitive which has a returned value, **PromptAndCollectUserInformation**, is sent to the EUP with the CORBA operation: **PCUIReceivedInformation**. The EUP choose by itself the **opID**, which must be new in this dialogue. If there is a conflict, which of course should not happen, the SRO must launch the **e_BadParameter** exception, and the corresponding operation is ignored.
The interface of the EUP directory is:

```
interface i_EUPdirectory {

exception e_retrieveError { string reason ;};

 i_EUP_srControlCSl retrieve(in i_SRF_callCS1 sro, in CorIDType corId)
    raises ( e_retrieveError );
  } ;
```

The interface of the specialised resource object is :

```
interface i_SRF_callCS1 {
  void playAnnouncement (
              in t_opID id,
              in t_playAnnouncementArg paarg,
              in boolean lastComp)
       raises (e_BadArguments, e_Error ) ;

  void promptAndCollectUserInformation (
              in t_opID id,
              in t_promptAndCollectUserInformationArg pcuiarg,
              in boolean lastComp)
       raises ( e_BadArguments, e_Error );


  void cancel (in t_opID id,
              in t_opID opToCancelID,
              in boolean lastComp)
       raises ( e_BadArguments, e_Error );
  };
```

The interface for the EUP is :

```
interface i_EUP_srControlCSl {
  void specializedResourceReport(in t_opID id)
         raises ( e_BadArguments, e_Error);
  void PCUIreceivedInformation (in t_opID id,
                                in string info);
```

The INAP digits have been translated into a CORBA string.
The INAP error are mapped into CORBA operations, since the asynchronism and the TCAP transaction mechanism prevents a real use of CORBA exceptions:

```
  void cancelled(in t_opID id);
  void cancelFailed(in t_opID id,
                    in t_CancelFailedArg cfarg);
  void missingParameter(in t_opID id);
  void parameterOutofRange (in t_opID id);
  void systemFailure(in t_opID id);
  void taskRefused(in t_opID id,
                   in t_taskRefusedArg trarg);
  void unavailableResource(in t_opID id);
  void unexpectedComponentSequence(in t_opID id);
  void unexpectedDataValue(in t_opID id);
  void unexpectedParameter(in t_opID id);
};
```

The following MSC shows a new mechanism replacing the classical INAP scenario: **EstablishTemporaryConnection/ assistRequestInstruction/ playAnnouncement/ PromptAndCollectUserInformation/ DisconnectForwardConnection**.

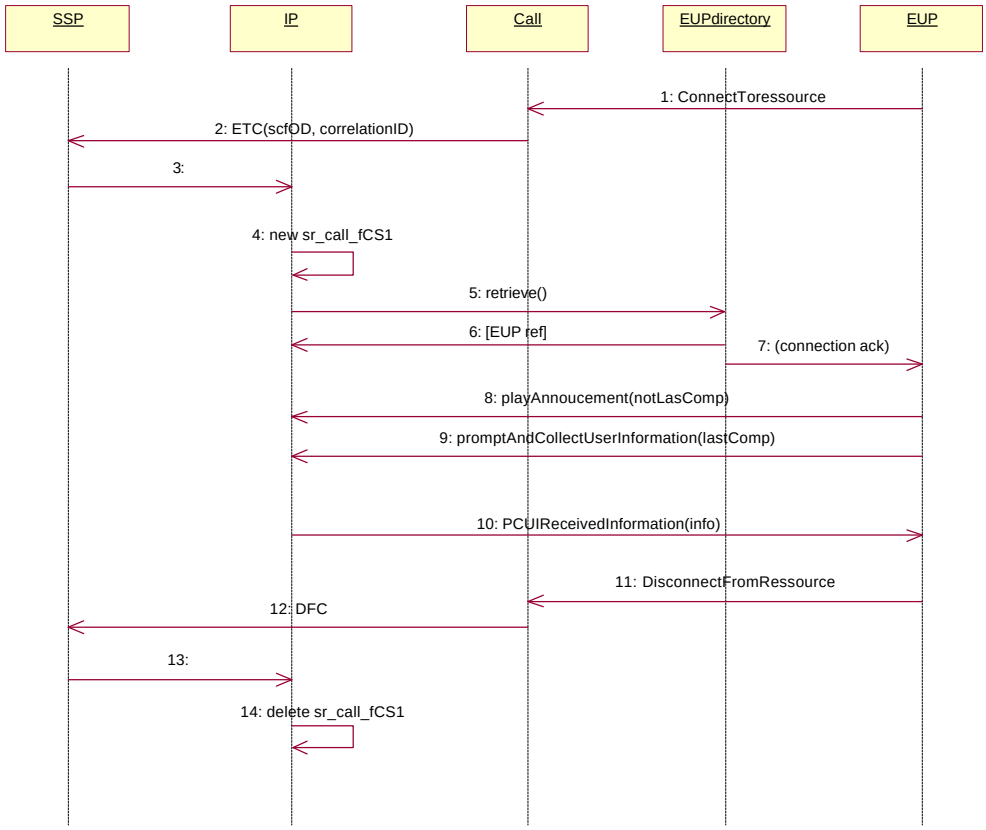

Figure 11. Use case of the CORBA based IP

7. Related standards and documents

[1] Alcatel response to RfP “IN access to TINA services & Connection management (IN-TINA Adaptation Unit)”, March 1999

[2] Deutsche Telekom & France Télécom response to RfP “IN access to TINA services & Connection management (IN-TINA Adaptation Unit)”, March 1999

[3] Lucent Technology’s response to RfP “IN access to TINA services & Connection management (IN-TINA Adaptation Unit)”, March 1999

[4] TINA-IN Work Group Request for Proposals: IN access to TINA services & Connection management (IN-TINA Adaptation Unit) http://www.tinac.com/wg_sig/in/RFPs/IN-TINA-AU/IN-TINA-AU.html

[5] TINA-C: Business Model and Reference Points, TINA-C Document Version 4.0, 20.5.97 http://www.tinac.com

[6] TINA-C: Ret Reference Point Specification, TINA-C Document Version: 1.0, 27.1.98

[7] Clarification of the Retailer role, TINA-SARP-Work Group, Frans Panken, Lucent Technologies, NL (making use of input provided by Mike Fisher, BT, reviewed by Marcel Mampaey, Alcatel), Version 1.1, 28 January 1999

[8] TINA-C: TINA Service Architecture, TINA-C Document Version: 5.0, 16.06.97

[9] TINA-C: Service Component Specification

[10] TINA-C: Network Resource Architecture

[11] TINA-C RFS “The ConS Reference Point V1.0”draft document, February 1997

[12] International Telecommunication Union (ITU-T): “Q.121x, Intelligent Networks”, Geneva, October 1995

[13] International Telecommunication Union (ITU-T): "Q.1201, Principles of intelligent network architecture", 10/92
[14] International Telecommunication Union (ITU-T): "Q.1211, Introduction to Intelligent Network Capability Set 1"
[15] International Telecommunication Union (ITU-T): "Q.1214, Distributed Functional Plane for Intelligent Network CS-1"
[16] International Telecommunication Union (ITU-T): "Q.1218, Interface Recommendation for Intelligent Network CS-1", ITU-T, 1995
[17] Parlay API specification version 1.0, http://www.parlay.org.
[18] Interworking between CORBA and Intelligent Networks Systems, OMG Document : telecom/97-12-06
[19] Interworking between CORBA and TC Systems, OMG Document : telecom/98-10-03
[20] EURESCOM Project P508: „CORBA as an Enabling Factor for Migration from IN to TINA: A P508 Perspective", Final Version, January 1997.
[21] C. Capellmann, J.-M. Pageot: "A TINA Service Platform integrated with current Intelligent Network systems", In the Proceedings of TINA'99
[22] T. Eckhardt et al.: " A TINA Trial – Interworking Experience with the Legacy Telephone System", TINA'97 Conference, November 17-20, 1997, Santiago, Chile
[23] Vital Project , D13, ODTA Validation Third Trial, October 1998
[24] ODTA Validation 3rd Trial, Access and service session subsystem

Abbreviations

API	Application Programming Interface
ASN.1	Abstract Syntax Notation One
CORBA	Common Object Request Broker Architecture
CPE	Customer Premises Equipment
CTI	Computer Telephony Interface
DPE	Distributed Processing Environment
EUP	End User Proxy
IDL	Interface Definition Language
IN	Intelligent Network
IN CS1	Intelligent Network Capabilities Set 1
INAP	Intelligent Network Application Part
ITAU	IN TINA Adaptation Unit
MAP	Mobile Application Part
MGCP	Media Gateway Controller Protocol
MSC	Messages Sequence Charts
OMG	Object Management Group
OO	Object Oriented
RET	Retailer in TINA terminology
Ret RP	Retailer Reference Point
RFP	Request For Proposal
SCF	Service Control Function
SDF	Service Data Function
SRF	Service Resource Function
SSF	Service Switching Function
SSM	Service Session Manager
SSP	Service Switching Point
SS7	Signalling System N° 7
TCAP	Transaction Capabilities Application Part
TINA	Telecommunication Information Network Architecture
VoIP	Voice over Internet Protocol

ANNEX

IDLs

Common Types

Access Interface

Call Control Interface

Management Interface

//file cfmgmntitaucommontypes.idl

```
#ifndef _cf_mgmnt_itau_common_types_idl_
#define _cf_mgmnt_itau_common_types_idl_


#include "CfCommonTypes.idl"
#include "CfAccessCommonTypes.idl"
#include "CfSubscriptionCommonTypes.idl"

module MgmntItauCommonTypes {

      typedef long t_Duration;           // [sec]
      typedef long t_RangeLength;
      typedef long t_CalledAccessNetworkBearerCapacity; //[bits/s]
      typedef short t_DigitPressed;
      enum t_OtherKeyPressed {poundSign, star};

      enum t_CalledAccessNetworkType {ISDN, GSM, IP, ATM, NotDefined};

      struct t_AccessNetworkUsed{
            t_CalledAccessNetworkType accessNetworkUsed;
            t_CalledAccessNetworkBearerCapacity capacityAccessNetwork;
      };

      union t_NetworkAddress switch(short) {
            case 1: string IPAddress;
            case 2: string SS7Address;
            case 3: long E164;
            case 5: long GSM;
            case 6: string URL;
            case 7: string addressNotKnown;
      };


      union t_EventType switch (short) {
            case 1: t_OtherKeyPressed otherKeyPressed;
            case 2: t_DigitPressed digitPressed;
      };

      struct t_MidCallEvent {
            string eventName;
            boolean midCallEventSustained;
            t_EventType eventType;
      };
```

```
      enum t_AddressType{
            CallFromOrigination,
            CallFromDestination,
            CallOfOriginalDestinationAddress,
            RedirectingCallAddress
      };

      struct t_Address {
            string addressName;
            t_AddressType adressType;
            t_NetworkAddress networkAdress;
            boolean fixedRange;          //may be convenient to know; also
present in INAP
            t_RangeLength rangeLength;   //the range value, if appropriate
      };


      // already in subscriptionCommonTypes
      // Configurability of the ITAU owner must be realized by choosing
      // FIXED_BY_SERVICE_PROVIDER

      // enum t_ParameterConfigurability {
      //    FIXED_BY_RETAILER,
      //    FIXED_BY_SERVICE_PROVIDER,
      //    CONFIGURABLE_BY_SUBSCRIBER,
      //    CUSTOMIZABLE_BY_END_USER
      // };

      // already in subscriptionCommonTypes; included for clarification
reasons
      //
      // typedef sequence<t_Parameter> t_ParameterList;
      //
      // struct t_Parameter {
      //    t_ParameterName name;
      //    t_ParameterConfigurability configurability;
      //    t_ParameterValue value;
      //};


      // already in subscriptionCommonTypes; included for clarification
reasons
      // struct t_ServiceDescription {
      //    AccessCommonTypes::t_ServiceId serviceId;
      //    AccessCommonTypes::t_UserServiceName serviceName;
      //    SubscriptionCommonTypes::t_ParameterList serviceCommonParams;
      //    SubscriptionCommonTypes::t_ParameterList serviceSpecificParams;
      // };
      // Typing of parameter values that makes outsourcing via ITAU
      possible:
      // Services for the ITAU are characterized by a service ID that
starts with 4828
      // (FYI: 4828 spells ITAU - as well as many other words- on the
keypad of a telephone set)
      // If the service ID starts with 4828,
      // one of the service common parameters MUST contain a value access
type.
      // It shows the kind of access that is required for this service.
      // If this parameter is missing, an exception is raised.
      // This parameter value is typed as follows:
```

```
//


enum t_AccessType{NamedAccess,AnonymousAccess};



struct t_Resource {
          string ResourceName;
      boolean disConnectFromResourceForbidden; //default = true
          t_Address networkAddress;
      // Since the t_resource is the parameter value, the
      // configuarability is already covered at t_parameter level.
};


struct t_RecordedMessage {
      t_Duration maxDuration;
      t_Duration timeOut;
      string recordedMessage;
};


struct t_CollectedInfoNeeded {
      t_Resource resourceToBeUsed;
      unsigned long minumumNumberOfDigits;
      unsigned long maximumNumberOfDigits;
      short numberOfDigits;
      boolean fixedLength;
      string endOfReplyDigit;
      t_RecordedMessage speech;
};


struct t_Tone {
      long toneID;
      t_Duration duration;
      long frequency;         // [Hz]
};


struct t_MessagePlayed {
      string messageName;
      t_Resource ResourceToBeUsed;
      long messageValue;
      long numberOfRepetitions;
      t_Duration duration;
};

struct t_InformationToSend {
      t_MessagePlayed message;
      string displayInformation;
      t_Tone tone;
};


struct t_AddressAndService {
      long calledAddress;
      string serviceKey;
      long callingAddress;
      long locationNumber;
```

```
      };


      union t_ServiceCriteria switch(unsigned short){
            case 1: long dialedNumber;
            case 2: long callingLineID;
            case 3: string serviceKey;    //e.g. detection point; maybe
typed as a long?
            case 4: boolean serviceNotAvailable;
            default: t_AddressAndService addressAndService;
      };


      enum t_templateErrorType {unKnownServiceTemplate,
InvalidServiceTemplate};
      enum t_templateState {active, passive};

      exception e_invalidParameter {
                SubscriptionCommonTypes::t_Parameter parameter;
      };


      exception e_unknownServiceId {
            AccessCommonTypes::t_ServiceId serviceId;
      };

      exception e_unExpectedParameter {
            string unExpectedParameter;
      };

        exception e_missingParameter {
            string mandatoryParameter;
        };

}; // MgmntItauCommonTypes

#endif // _cf_subscription_common_types_idl_

//File CfServiceTemplateControl.idl

#ifndef _cf_service_template_control_idl_
#define _cf_service_template_control_idl_

#include "CfAccessCommonTypes.idl"
#include "CfSubscriptionCommonTypes.idl"
#include "CfMgmntItauCommonTypes.idl"

module ServiceTemplateControl {

      exception e_invalidTemplate {
            MgmntItauCommonTypes::t_templateErrorType templateErrorType;
      };


      exception e_invalidTemplateState {
            MgmntItauCommonTypes::t_templateState currentTemplateState;
      };


      interface i_ServiceTemplateControl {
```

```
            void listServiceTemplates (
                  out SubscriptionCommonTypes::t_ServiceTemplateList
svcTemplateList
            ) raises (
                  SubscriptionCommonTypes::e_applicationError
            );

            void getServiceTemplate (
                  in AccessCommonTypes::t_ServiceId serviceId,
                  out SubscriptionCommonTypes::t_ServiceTemplate template
            ) raises (
                  SubscriptionCommonTypes::e_unknownServiceId,
                  SubscriptionCommonTypes::e_applicationError
            );


            void deActivateService (
                  in AccessCommonTypes::t_ServiceId serviceId
            ) raises (
                  SubscriptionCommonTypes::e_unknownServiceId,
                  e_invalidTemplateState,
                  SubscriptionCommonTypes::e_applicationError
            );


            void reActivateService (
                  in AccessCommonTypes::t_ServiceId serviceId
            ) raises (
                  SubscriptionCommonTypes::e_unknownServiceId,
                  e_invalidTemplateState,
                  SubscriptionCommonTypes::e_applicationError
            );


            void addServiceTemplate (
                  in SubscriptionCommonTypes::t_ServiceTemplate template,
                  out AccessCommonTypes::t_ServiceId serviceId
            ) raises (
                  SubscriptionCommonTypes::e_applicationError,
                  e_invalidTemplate,
                  MgmntItauCommonTypes::e_missingParameter,
                  MgmntItauCommonTypes::e_unExpectedParameter,
                  MgmntItauCommonTypes::e_invalidParameter
            );


            void modifyServiceTemplate (
                  in SubscriptionCommonTypes::t_ServiceTemplate template
            ) raises (
                  SubscriptionCommonTypes::e_applicationError,
                  e_invalidTemplate,
                  e_invalidTemplateState ,
                  MgmntItauCommonTypes::e_missingParameter,
                  MgmntItauCommonTypes::e_unExpectedParameter,
                  MgmntItauCommonTypes::e_invalidParameter
            );


            void removeServiceTemplate (
                  in SubscriptionCommonTypes::t_ServiceIdList serviceList
            ) raises (
```

```
                SubscriptionCommonTypes::e_unknownServiceId,
                SubscriptionCommonTypes::e_applicationError
            );


      }; // i_ServiceTemplateControl
}; // ServiceTemplateControl

#endif // _cf_service_template_control_idl
```

TCSM Interface

```
#ifndef _TCSM_IDL
#define _TCSM_IDL

#include <CnsConnCommon.idl>

module TP_TCSM
{


   ////////////////////////////////////////////////////////////
   // TCSM Interface i_FlowControl
   //
   // TINAClassical operations : queryCapabilities
   //                          selectCapabilities
   //                          initiateTFC
   //                          deleteTFC
   //                          activateTFC
   //                          deactivateTFC
   //                          updateTFC
   //
   //  used in ITAU : queryCapabilities
   //                 setupTFC
   //
   // Provided to: CSM
   //
   ////////////////////////////////////////////////////////////

interface i_ITAUFlowControl
  {
  // This operation allows the communication session to query a terminal
  // for the capabilities available to support the specified SFEPs.
  // One capabilitySet is returned per requested Sfep.
  void queryCapabilities (
                  in TINASBCommSCommonTypes::t_SFEPComDescList
sfepDescList,
                    out CNS_CMData::t_CapabilitySetList capabilities)
            raises(CNS_CMExp::e_Error);

  enum telephoneConExpceptionCause {
    BUSY,
    NO_ANSWER,
    ROUTE_FAILURE
  };

  exception e_telephoneConException {
    telephoneConExpceptionCause cause;
```

```
  };

  void setupTFC (in t_ANfep)
    raises (e_telephoneConnectionException);
  };

}
```