This report describes the installation and invocation environments for Users and Services in the service Domain.
Report List
Chapter 1
Introduction

This document describes the design and implementation of the only OOram Module in the Invocation Service Constituent Creation of our example system. The module provides functionality for the installation of users and their services, and for invoking any of these services. A limitation of this example design is that at most one service may be invoked for any given user at the time.

There are three exported models from this module:

1. **Installation (Inv2InstallEXP)**. Client-server which provides functionality for the creation of new Users and for updating the users' service data bases.

2. **Invocation (Inv2InvEXP)**. Mechanism that defines the invocation environment for all services.

3. **Invocation (Inv2InvFW)**. Framework that defines the invocation environment for all services. This model is identical to the above, but carries additional information about superclasses.

![IN-Lab2 support value chain.](image)

©Taskon 1998. Page 2
Chapter 2
Overview of the Installation and Invocation module

Installation is done by an Installer object which sets the attributes of all relevant objects, possibly after having created and installed them.

Invocation is a stage in the service life cycle as described in the article The intelligent network life cycle by Nilsen and Vestli in Telektronikk 2.92 pp 15-25.

The invocation stage is concerned with the initialization prior to the execution of a service at the request of a client. The term service means in this context all applications in the IN service domain that have passed the creation, installation and activation stages of the service life cycle. A client is an application acting as agent for a user (i.e. a person). Such an application can be any any application acting on behalf of an actor (person or organization) that has the right to execute services in the network. This does include applications running outside the network provider's domain as e.g. value added services or private network applications.

![Invocatio Model Diagram](image)

Figure 2. The invocation model.
The roles are as follows:

1. **Invocation Manager (InvMngr)**. There is one global instance of the Invocation Manager. In a distributed system, this object will be an amoeba-like thing with pseudopods representing them in each process. More prosaically: The objects representing the Invocation Manager in a distributed system are implemented in such a way that they behave like a single object.

2. **Invocation Analyzer (InvAnalzr)**. There is one Invocation Analyzer object for each User in the Service Domain. In our example, the responsibility of this object is to store a data base of service objects available to its User, and to select and start a particular service as requested from a client.

3. **Invocation Client (InvClient)** is any client of the Invocation Analyzer who wants to access one of its services. A particular InvClient will be a User object, this object represents a User in relation to its Analyzer, and there will be a one to one relationship between User objects and Analyzer objects.

4. **Invocation Service (InvServ)**. This role represents the invocation-specific aspects of any service object.

The invocation phase is the stage immediately before the execution stage in the life cycle of a service and is characterized by the following steps (see the Use Case in the Message Sequence Diagram in figure 3?? below):

1. An invocation request comes from a client, *InvClient*, to the Invocation Manager (InvMngr) which constitutes the main access point to the Service Domain for any client.

2. The InvocationAnalyzer, *InvAnalzr*, for the appropriate User is identified and control passed to it.

3. The invocation request is analyzed by *InvAnalzr* in order to identify the requested service.

4. The client is identified. Authentication of a client may be necessary. On the other hand, the client may wish to authenticate both the service and the network, if the authenticity of these are vital to him.

5. *InvAnalzr* keeps a data base containing a master copy of all service objects installed for its User. The relevant service is identified, a clone (copy) of the object(s) is created, initialized, installed and given control.

6. The invocation lasts until the client achieves contact with the requested service object (which per definition is the transition to the Service Execution stage for the requested service) or until the invocation is rejected.
The Analyzer selects a suitable service object on the following criteria:

1. The kind of required service (a Symbol, e.g. *Telephone service*)
2. The identity of the client being the Analyzer User or some other client.
3. The type of the client object. We permit several implementation of the same service kind, each implementation being adapted to a particular type of client object.

The export/import model structure of the Invocation Module is shown in figure 4. The module contains the following models:

1. *Imported models*
   - None.
2. **Role Models**

- *Installer* (*inv2InstRM*) describes the functionality for installing Users and their Services.
- *Invoker* (*Inv2InvRM*) describes the object environment for service invocation.
- Use Case *Inv2Inst1>>createDefaultUsers* is the report from a monitored execution of a system initialization and User installation test program.
- Use Case *Inv2Inst1>>example1: '12'* is the report from a monitored execution of a test program which installs and removes a service.
- Use Case *Inv2Inst1>>example2: '12'* is the report from a monitored execution of a test program which lets a User find a service.

3. **Object type specifications:**

- *InstallerAndInvoker* (*Inv2InstInvSPC*) specifies the types of a coordinated set of objects which play the roles needed for installation and service invocation.

4. **Export models**

- *Installation* (*Inv2InstallEXP*). Client-server which provides functionality for the creation of new Users and for updating the users' service data bases.
- *Invocation* (*Inv2InvEXP*). Mechanism that defines the invocation environment for all services.
- *Invocation* (*Inv2InvFW*). Framework that defines the invocation environment for all services. This model is identical to the above, but carries additional information about superclasses.

5. **Importing models**

- None

---

**Figure 4.** Export/import structure of the Installation and Invocation Module models.
Chapter 3
Invocator2 Monitored executions

We will be showing three monitored executions of the Invocation model running by itself without any specialized services:

1. Inv2Install1 >> createDefaultUsers. The creation of a number of default Users with their Analyzers.
2. Inv2Install1 >> example1: '67'. The addition and removal of a default service to one of the Users.
3. Inv2Install1 >> example2: '12'. The selection and invocation of a service from any Client (not necessarily the User).

3.1 Inv2Install1 >> createDefaultUsers {Role Model}

*Report including the following selections automatically generated 28 July 1993: Role Models Area of Concern - stimuli Role list - diagram Scenarios Message Sets* 

3.1.1 Area of Concern

Test execution example to create two default Users.

**Stimulus: Inv2Install1 createDefaultUsers**
3.1.2 The Roles

Graphical representation of the roles and their connections.
3.1.3 Interaction Scenarios

![Diagram of interaction scenarios]

Figure 5. Inv2Install1 createDefaultUsers {Scenario}

3.1.4 Message Sets

1. **Inv2Install#101 {Role}**
   - **1 {Port}**
     - executed {Contract}
       - userID:invocator:analyzer: {Message}
   - **2 {Port}**
     - executed {Contract}
       - userID:subscrID:invocator:user: {Message}
   - **7 {Port}**
     - executed {Contract}
       - addUser:analyzer: {Message}
   - **5 {Port}**
     - executed {Contract}
       - userID:invocator:analyzer: {Message}
3.2 Inv2Install1>>example1: '12' {Role Model}

Report including the following selections automatically generated 28 July 1993: Role Models
Area of Concern - stimuli Role list - diagram Scenarios Message Sets

3.2.1 Area of Concern

Test monitored execution showing the addition and removal of a default service to one of the Users.

Stimulus: Inv2Install1>>example1: '67'.
3.2.2 The Roles

3.2.3 Interaction Scenarios

Figure 6. Inv2Install1 example1: '12' {Scenario}
3.2.4 Message Sets

1. Inv2Install#109 {Role}
   - 1 {Port}
     - executed {Contract}
       - invocReq: {Message}
   - 4 {Port}
     - executed {Contract}
       - installService: {Message}
       - removeService: {Message}

3.3 Inv2Install1>>example2: '12' {Role Model}

Report including the following selections automatically generated 28 July 1993: Role Models
Area of Concern - stimuli Role list - diagram Scenarios Message Sets

3.3.1 Area of Concern

The selection and invocation of a service from any Client (not necessarily the User).

Stimulus: Inv2Install1>>example2: '12'.

3.3.2 The Roles

![Diagram of Service Installation and Invocation Service Constituent module]
3.3.3 Interaction Scenarios

![Interaction Scenarios Diagram]

Figure 7. Inv2Install1 example2: '12' {Scenario}

3.3.4 Message Sets

1. Inv2Client#102 {Role}
   - 3 {Port}
     - executed {Contract}
     - servicesFor: {Message}
     - getService:for: {Message}

2. Inv2Analzr#103 {Role}
   - 4 {Port}
     - executed {Contract}
     - kind {Message}
Chapter 4
Service Installation and Invocation export models summary

4.1 Inv2InvocationMCH (Inv2InvMCH) {Export Model}

Report including the following selections automatically generated 28 July 1993: Export Models
Area of Concern - stimuli Role list - diagram Scenarios - explanation

4.1.1 Area of Concern

This mechanism is for service invocation in an intelligent network. It conveys the principles for how to process a particular client's request for execution of a particular service.
4.1.2 The Roles

![Diagram showing roles Inv-Mngr, Inv2, Client, Inv2-Analzr, Inv-Serv]

4.1.3 Interaction Scenarios

![Diagram showing invocation rejection scenario]

**Figure 8. Invocation rejection 1 [Scenario]**

This scenario exemplifies the earliest possible rejection of an invocation request. The returned value is false.
4.2 Inv2InvocationFW (Inv2InvFW) {Export Model}

The role model is identical to Inv2InvocationMCH, but the roles define suitable classes to be subclassed by the importer.
4.3 Inv2InstallationEXP (Inv2InstallEXP) {Export Model}

Report including the following selections automatically generated 28 July 1993: Export Models
Area of Concern - stimuli Role list - diagram Scenarios - explanation

4.3.1 Area of Concern

The Service role represents the target application for the client. This is the software defining some client's telecommunications service.

4.3.2 The Roles

![Diagram of Service Installation and Invocation export models](image)
4.3.3 Interaction Scenarios

**Figure 11. Service installation (Scenario)**
Installing a service in an InvocationAnalyzer.

**Figure 12. Invocation rejection 1 (Scenario)**
This scenario exemplifies the earliest possible rejection of an invocation request. The returned value is false.
Figure 13. SelectAndOpenService {Scenario}

Figure 14. Service removal {Scenario}
InvocReq: clientIdRequest

Figure 15. Invocation success 1 (Scenario)

This scenario shows the basic standard expected message flow during perfect execution of the invocation phase.
Chapter 5
Invokerator2 design models

5.1 Inv2InvRM {Role Model}

Report including the following selections automatically generated 28 July 1993: Role Models
Area of Concern - stimuli Role list - diagram -role explanation - role import map Scenarios -
explanation Message Sets -port explanation - contract explanation -message explanation FSM

5.1.1 Area of Concern

This role model is for service invocation in an intelligent network. It conveys the principles for
how to process a particular client's request for execution of a particular service. The identities of
the client and the service are not known a priori and therefore must be established during this
process.

The client is always a piece of software (e.g. a service or an agent representing a user). The
model should be independent of terminal capabilities and all interfaces must be considered
application programming interfaces (APIs). The vital point is however that this model must be
service independent, i.e. it should work for all possible services.

Note that there is no communication between the Service and the Client during Service
Invocation. This is a crucial part of the definition of Service Invocation: it lasts until the
requested service is established and is able to take over control of the service execution,
alternatively until the invocation request is rejected.

All references to values in this model are pointers to instances of objects.

Stimulus messages

1. InvClient>>getService:for: Return a new instance of the specified service, the service is
   owned by the User and will be set up for the given Client.

2. InvClient>>invocReq: Return the InvocationAnalyzer for the User with the given user ID,
   or false if request is rejected.

3. InvClient>>servicesFor: Return a list of service kinds, one for each service available to this
   Client.
5.1.2 The Roles

1. **InvClient** (*Role*). This object represents any client of the Invocation Manager, an Analyzer and possibly one of the Analyzer's Services.
2. **InvManager (InvMngr) {Role}**. The Invocation Manager's role is to accept requests from Clients and start the Invocation Analyzer (IA) for further processing and to start specific services on request by the IA. The Invocation Analyzer is unique for a given User, and an Invocation Analyzer can be responsible for analyzing requests for many services. The Invocation Manager can be considered a part of the operating system of the network.

Setting the status of the Invocation Manager from roles outside this model may be necessary, for example when the network is going down. Then the Manager must be told not to accept any Clients or only a certain set of privileged Clients. Selective rejection of invocation requests must be handled by the invocator. There are many different schemas for rejection of clients and they may depend on the mode of the invocator. Normally all clients are accepted. During major maintenance all clients may be rejected for a short time period (this is not acceptable in a telecommunications network, but theoretically possible); Certain clients are refused; Most clients are refused. Screening functions may be introduced to decide who are acceptable or who rejectable and so on... This can be useful during crisis situations like war etc. Currently this sort of mechanisms are not contained in the invocator role because we assume all clients are accepted.

3. **InvAnalyzer (InvAnalzr) {Role}**. The Invocation Analyzer must perform a client-dependent analysis that will identify the particular service requested. This implies that the Invocation Analyzer must analyse the information received from the client, look up relevant information in its internal databases, load the service object and finally initiate it with the required information.

The Client can represent a User A. This Client requests the initiation of one of User A's services (via the Invocation Manager and Invocation Analyzer) that is to establish a connection to User B. That service, when executing, plays the role of Client recursively in order to request the initiation of one of User B's services, say User B's receive-call service. User B's IA is started and communicates with User A's service in order to analyze the situation and find out if it accepts the call from User A or not. User B can for example have a screening function excluding incoming calls from his mother-in-law. If User B's IA accepts the invocation request from User A's service, it initiates User B's receive-call service. Then this service and User A's service can communicate when executing, in order to set up the connection between User A and User B.

But of course User B's IA is in action also whenever User B himself wants to use his own services. So we have that User B's IA always has to analyse whether the Client is User B or only a user of User B's services. Note that the behavior of the IA can be very different in the two cases.

This object is the User's permanent representative in the IN system and exists as long as the User is a registered IN customer. This role is immutable as seen from users of this Service Constituent.

4. **InvService (InvServ) {Role}**. The Service role represents the target application for the client. This is the software defining some client's telecommunications service.
5.1.3 Interaction Scenarios

Figure 16. SelectAndOpenService {Scenario}

Figure 17. Invocation rejection 1 {Scenario}
This scenario exemplifies the earliest possible rejection of an invocation request. The returned value is false.
5.1.4 Message Sets

1. **InvClient (Role)**
   - **invocManager (im) (Port).** The one and only Invocation Manager.
   - **InvMngr<InvClient (Contract).** This scenario exemplifies the general process of selecting and opening any service. Specific services may add additional message interactions.
     - **invocReq: usrID (Message).** Return the InvocationAnalyzer for the User with the given user ID, or false if request is rejected.
   - **invocAnalyzer (ia) (Port).**
     - **InvAnalzr<InvClient (Contract).**
       - **servicesFor: aClient (Message).** Return a list of service kinds, one for each service available to this Client.
       - **getService: serviceKind for: aClient (Message).** Return a new instance of the specified service, the service is owned by the User and will be set up for the given Client.
2. **InvAnalyzer (InvAnalzr) {Role}**
   - client (c) {Port}.
     - InvClient<InvAnalzr {Contract}.
       - obeysProtocol: aType {Message}. Return true iff the Client obeys the given protocol.
       - clientIdRequest {Message}. Return the service ID of the client.
   - service (s) {Port}.
     - InvServ<InvAnalzr {Contract}.
       - setUp: parameters client: aClient analyzer: anAnalyzer invocator: anInvoc {Message}. Create and return an instance of a Service with the given parameters (one of them being owner UserID, another being the current Client).
       - kind {Message}. Return aSymbol, which represents the kind of service.
       - protocol {Message}. Answer the Client protocol permitted by this service.

3. **InvService (InvServ) {Role}**
   - invocationAnalyzer (ia) {Port}.
     - InvAnalzr<InvServ {Contract}.
       - userID {Message}. Return the user ID of the user.
       - user {Message}. Return the User object of the InvocationAnalyzer.
   - invocationManager (im) {Port}.
     - InvMngr<InvServ {Contract}.
       - invocReq: usrID {Message}. Return the InvocationAnalyzer for the User with the given user ID, or false if request is rejected.

5.2 **Service Installation (Inv2InstallRM) {Role Model}**

Report including the following selections automatically generated 28 July 1993: Role Models Area of Concern - stimuli Role list - diagram -role explanation - role import map Scenarios - explanation Message Sets -port explanation - contract explanation -message explanation FSM

5.2.1 **Area of Concern**

This role model shows the creation, initialization and installation of the Invocation Manager, the Users, the Invocation Analyzers and the Users' Services.
**Stimulus messages**

1. **InvInstaller>>installService**: Install the given service object instance in the service data base for this Analyzer’s User.

2. **InvInstaller>>removeService**: Remove the given service object instance from service data base for this Analyzer’s User, do nothing if it is not defined.

3. **InvInstaller>>addUser:analyzer**: Add the given User and Analyzer objects if a User with usr’s userID did not exist beforehand.

4. **InvInstaller>>userID:subscrID:manager:user**: The Invocator issues this initialization message to enter configuration information into the analyzer upon activation of the analyzer. This message contains the following parameters:
   - uID: the User identifier in the Service Domain.
   - sID: the Subscriber identifier in the Service Domain.
   - Pointer to the Invocation Manager.
   - Pointer to the User object corresponding to this Analyzer.

5. **InvInstaller>>userID:manager:analyzer**: Initialization message.

5.2.2 **The Roles**

![Diagram showing the roles of InvMngr, InvAnalzr, InvInstaller, InvUser, and InvService]
1. **InvocationAnalyzer (InvAnalzr) [Role]**. This role is responsible for managing a User's master service objects for later cloning and invocation.

2. **InvocationInstaller (InvInstaller) [Role]**. This role is responsible for the creation, initialization and installation of the Invocation Manager, the Users, the Invocation Analyzers and the Users' Services.

3. **InvMngr [Role]**. This globally available object is the first point of entry to the Service Domain.

4. **InvUser [Role]**. An object representing a User in the service Domain.

5. **InvService [Role]**. All services to be invocated by this model must have an object which plays this role.

### 5.2.3 Interaction Scenarios

**Figure 19. Service installation [Scenario]**

*Installing a service in an InvocationAnalyzer.*

**Figure 20. Service removal [Scenario]**

*Figure 19. Service installation [Scenario]*

*Figure 20. Service removal [Scenario]*
5.2.4 Message Sets

1. InvocationInstaller (InvInstaller) \{Role\}
   - analyzer (a) \{Port\}.
     - InvAnalzr<InvInstaller \{Contract\}.
       - installService: aService \{Message\}. Install the given service object instance in the service data base for this Analyzer's User.
       - removeService: aService \{Message\}. Remove the given service object instance from service data base for this Analyzer's User, do nothing if it is not defined.
       - userID: uID subscrID: sID manager: mngr user: aUser \{Message\}. The Invocator issues this initialization message to enter configuration information into the analyzer upon activation of the analyzer. This message contain sthe following parameters:
         - uID: the User identifier in the Service Domain.
         - sID: the Subscriber identifier in the Service Domain.
         - Pointer to the Invocation Manager.
         - Pointer to the User object corresponding to this Analyzer.
       - servicesOfKind: aKind \{Message\}. Return all services of the given kind for my User.
     - invocManager (im) \{Port\}.
       - InvMngr<InvInstaller \{Contract\}.
         - addUser: usr analyzer: anal \{Message\}. Add the given User and Analyzer objects if a User with usr's userID did not exist beforehand.
         - InvMngr<InvClient \{Contract\}.
           - invocReq: usrID \{Message\}. Return the InvocationAnalyzer for the User with the given userID, or \false\ if request is rejected.
     - user (u) \{Port\}.
       - InvUser<InvInstaller \{Contract\}.
         - userID: uID manager: aManager analyzer: anAnal \{Message\}. Initialization message.
     - service (s) \{Port\}.
       - InvService<InvInstaller \{Contract\}.
         - initializeWith: dataDict \{Message\}. Reset attributes according to the values given in dataDict.
Chapter 6
Invocator2 Object Type Specification models

6.1 Inv2InstallAndInvocationSPC (Inv2InstInvSPC) {Type Specification}

Report including the following selections automatically generated 28 July 1993: Type Specifications Area of Concern - stimuli Role list - diagram -role explanation - role import map Scenarios - explanation Message Sets -port explanation - contract explanation -message explanation FSM

6.1.1 Area of Concern

This role model is for service invocation in an intelligent network. It conveys the principles for how to process a particular client's request for execution of a particular service. The identities of the client and the service are not known a priori and therefore must be established during this process.

The client is always a piece of software (e.g. a service or an agent representing a user). The model should be independent of terminal capabilities and all interfaces must be considered application programming interfaces (APIs). The vital point is however that this model must be service independent, i.e. it should work for all possible services.

Note that there is no communication between the Service and the Client during Service Invocation. This is a crucial part of the definition of Service Invocation: it lasts until the requested service is established and is able to take over control of the service execution, alternatively until the invocation request is rejected.

All references to values in this model are pointers to instances of objects.

This role model shows the creation, initialization and installation of the Invocation Manager, the Users, the Invocation Analyzers and the Users' Services.
6.1.2 The Roles

1. **InvManager (InvMngr) {Type}**. The Invocation Manager's role is to accept requests from Clients and start the Invocation Analyzer (IA) for further processing and to start specific services on request by the IA. The Invocation Analyzer is unique for a given User, and an Invocation Analyzer can be responsible for analyzing requests for many services. The Invocation Manager can be considered a part of the operating system of the network.

Setting the status of the Invocation Manager from roles outside this model may be necessary, for example when the network is going down. Then the Manager must be told not to accept any Clients or only a certain set of privileged Clients. Selective rejection of invocation requests must be handled by the invocator. There are many different schemas for rejection of clients and they may depend on the mode of the invocator. Normally all clients are accepted. During major maintenance all clients may be rejected for a short time period (this is not acceptable in a telecommunications network, but theoretically possible); Certain clients are refused; Most clients are refused. Screening functions may be introduced to decide who are acceptable or who rejectable and so on... This can be useful during crisis situations like war etc. Currently this sort of mechanisms are not contained in the invocator role because we assume all clients are accepted.

This globally available object is the first point of entry to the Service Domain.
2. **Inv2Client (Type)**. This object represents any client of the Invocation Manager, an Analyzer and possibly one of the Analyzer's Services.

An object representing a User in the service Domain.

3. **Inv2Analzr (Type)**. The Invocation Analyzer must perform a client-dependent analysis that will identify the particular service requested. This implies that the Invocation Analyzer must analyze the information received from the client, look up relevant information in its internal databases, load the service object and finally initiate it with the required information.

The Client can represent a User A. This Client requests the initiation of one of User A's services (via the Invocation Manager and Invocation Analyzer) that is to establish a connection to User B. That service, when executing, plays the role of Client recursively in order to request the initiation of one of User B's services, say User B's receive-call service. User B's IA is started and communicates with User A's service in order to analyze the situation and find out if it accepts the call from User A or not. User B can for example have a screening function excluding incoming calls from his mother-in-law. If User B's IA accepts the invocation request from User A's service, it initiates User B's receive-call service. Then this service and User A's service can communicate when executing, in order to set up the connection between User A and User B.

But of course User B's IA is in action also whenever User B himself wants to use his own services. So we have that User B's IA always has to analyse whether the Client is User B or only a user of User B's services. Note that the behavior of the IA can be very different in the two cases.

This object is the User's permanent representative in the IN system and exists as long as the User is a registered IN customer. This role is immutable as seen from users of this Service Constituent.

This role is responsible for managing a User's master service objects for later cloning and invocation.

4. **Inv2Service (Inv2Serv) (Type)**. The Service role represents the target application for the client. This is the software defining some client's telecommunications service.

All services to be invocated by this model must have an object which plays this role.

5. **InvocationInstaller (InvInstaller) (Type)**. This role is responsible for the creation, initialization and installation of the Invocation Manager, the Users, the Invocation Analyzers and the Users' Services.
Table 21. Import relationships for Inv2InstallAndInvocationSPC (RMTTypeSpecification).

<table>
<thead>
<tr>
<th>Inv2InstInvSPC</th>
<th>Inv2InvRM</th>
<th>Inv2InstallRM</th>
</tr>
</thead>
<tbody>
<tr>
<td>InvMngr</td>
<td>InvMngr</td>
<td>InvMngr</td>
</tr>
<tr>
<td>Inv2Client</td>
<td>InvClient</td>
<td>InvUser</td>
</tr>
<tr>
<td>Inv2Analzr</td>
<td>InvAnalzr</td>
<td>InvAnalzr</td>
</tr>
<tr>
<td>Inv2Serv</td>
<td>InvServ</td>
<td>InvService</td>
</tr>
<tr>
<td>InvInstaller</td>
<td></td>
<td>InvInstaller</td>
</tr>
</tbody>
</table>

Table 21. Import relationships for Inv2InstallAndInvocationSPC (RMTTypeSpecification).
Chapter 7
Models exported from the Invocator2 module

7.1 Inv2InvocationMCH (Inv2InvMCH) {Export Model}

Report including the following selections automatically generated 28 July 1993: Export Models
Area of Concern - stimuli Role list - diagram -role explanation Scenarios - explanation
Message Sets - imports - port explanation - contract explanation - message explanation

7.1.1 Area of Concern

This mechanism is for service invocation in an intelligent network. It conveys the principles for
how to process a particular client's request for execution of a particular service.
7.1.2 The Roles

1. InvManager (InvMngr) (Type). The Invocation Manager's role is to accept requests from Clients and start the Invocation Analyzer (IA) for further processing and to start specific services on request by the IA. The Invocation Analyzer is unique for a given User, and an Invocation Analyzer can be responsible for analyzing requests for many services. The Invocation Manager can be considered a part of the operating system of the network.

Setting the status of the Invocation Manager from roles outside this model may be necessary, for example when the network is going down. Then the Manager must be told not to accept any Clients or only a certain set of privileged Clients. Selective rejection of invocation requests must be handled by the invocator. There are many different schemas for rejection of clients and they may depend on the mode of the invocator. Normally all clients are accepted. During major maintenance all clients may be rejected for a short time period (this is not acceptable in a telecommunications network, but theoretically possible); Certain clients are refused; Most clients are refused. Screening functions may be introduced to decide who are acceptable or who rejectable and so on... This can be useful during crisis situations like war etc. Currently this sort of mechanisms are not contained in the invocator role because we assume all clients are accepted.

This globally available object is the first point of entry to the Service Domain.
2. **Inv2Client** *(Type)*. This object represents any client of the Invocation Manager, an Analyzer and possibly one of the Analyzer's Services.

An object representing a User in the service Domain.

3. **Inv2Analzr** *(Type)*. The Invocation Analyzer must perform a client-dependent analysis that will identify the particular service requested. This implies that the Invocation Analyzer must analyse the information received from the client, look up relevant information in its internal databases, load the service object and finally initiate it with the required information.

The Client can represent a User A. This Client requests the initiation of one of User A's services (via the Invocation Manager and Invocation Analyzer) that is to establish a connection to User B. That service, when executing, plays the role of Client recursively in order to request the initiation of one of User B's services, say User B's receive-call service. User B's IA is started and communicates with User A's service in order to analyze the situation and find out if it accepts the call from User A or not. User B can for example have a screening function excluding incoming calls from his mother-in-law. If User B's IA accepts the invocation request from User A's service, it initiates User B's receive-call service. Then this service and User A's service can communicate when executing, in order to set up the connection between User A and User B.

But of course User B's IA is in action also whenever User B himself wants to use his own services. So we have that User B's IA always has to analyse whether the Client is User B or only a user of User B's services. Note that the behavior of the IA can be very different in the two cases.

This object is the User's permanent representative in the IN system and exists as long as the User is a registered IN customer. This role is immutable as seen from users of this Service Constituent.

This role is responsible for managing a User's master service objects for later cloning and invocation.

4. **Inv2Service** *(Inv2Serv)* *(Type)*. The Service role represents the target application for the client. This is the software defining some client's telecommunications service.

All services to be invoked by this model must have an object which plays this role.
7.1.3 Interaction Scenarios

*Figure 22. Invocation rejection 1 (Scenario)*
This scenario exemplifies the earliest possible rejection of an invocation request. The returned value is false.

*Figure 23. SelectAndOpenService (Scenario)*
7.1.4 Message Sets

1. Inv2Client {Type}
   - invocAnalyzer (ia) {Port}.
     - InvAnalzr<InvClient {Contract}.
       - servicesFor: aClient {Message}. Return a list of service kinds, one for each service available to this Client.
       - getService: serviceKind for: aClient {Message}. Return a new instance of the specified service, the service is owned by the User and will be set up for the given Client.
     - invocManager (im) {Port}. The one and only Invocation Manager.
     - InvMngr<InvClient {Contract}. This scenario exemplifies the general process of selecting and opening any service. Specific services may add additional message interactions.
     - invocReq: usrID {Message}. Return the InvocationAnalyzer for the User with the given user ID, or false if request is rejected.
2. **Inv2Analzr (Type)**
   - service(s) (Port).
     - InvServ<InvAnalzr (Contract).
         (Message). Create and return an instance of a Service with the given parameters (one of
         them being owner UserID, another being the current Client).
       - kind (Message). Return aSymbol, which represents the kind of service.
       - protocol (Message). Answer the Client protocol permitted by this service.
   - client (c) (Port).
     - InvClient<InvAnalzr (Contract).
       - obeysProtocol: aType (Message). Return true iff the Client obeys the given protocol.
       - clientIdRequest (Message). Return the service ID of the Client.

3. **Inv2Service (Inv2Serv) (Type)**
   - invocationAnalyzer (ia) (Port).
     - InvAnalzr<InvServ (Contract).
       - userID (Message). Return the user ID of the user.
       - user (Message). Return the User object of the InvocationAnalyzer.
   - invocationManager (im) (Port).
     - InvMngr<InvServ (Contract).
       - invocReq: usrID (Message). Return the InvocationAnalyzer for the User with the given
         user ID, or false if request is rejected.

7.2 **Inv2InvocationFW (Inv2InvFW) {Export Model}**

The role model is identical to Inv2InvocationMCH, but the roles define suitable classes to be
subclassed by the importer.

7.3 **Inv2InstallationEXP (Inv2InstallEXP) {Export Model}**

Report including the following selections automatically generated 28 July 1993: Export Models
Area of Concern - stimuli Role list - diagram -role explanation Scenarios - explanation
Message Sets -imports - port explanation - contract explanation -message explanation
7.3.1 Area of Concern

The Service role represents the target application for the client. This is the software defining some client's telecommunications service.

7.3.2 The Roles

1. **InvocationInstaller (InvInstaller)** (*Type*). This role is responsible for the creation, initialization and installation of the Invocation Manager, the Users, the Invocation Analyzers and the Users' Services.
2. **InvManager (InvMngr) {Type}**. The Invocation Manager's role is to accept requests from Clients and start the Invocation Analyzer (IA) for further processing and to start specific services on request by the IA. The Invocation Analyzer is unique for a given User, and an Invocation Analyzer can be responsible for analyzing requests for many services. The Invocation Manager can be considered a part of the operating system of the network.

Setting the status of the Invocation Manager from roles outside this model may be necessary, for example when the network is going down. Then the Manager must be told not to accept any Clients or only a certain set of privileged Clients. Selective rejection of invocation requests must be handled by the invocator. There are many different schemas for rejection of clients and they may depend on the mode of the invocator. Normally all clients are accepted. During major maintenance all clients may be rejected for a short time period (this is not acceptable in a telecommunications network, but theoretically possible); Certain clients are refused; Most clients are refused. Screening functions may be introduced to decide who are acceptable or who rejectable and so on... This can be useful during crisis situations like war etc. Currently this sort of mechanisms are not contained in the invocator role because we assume all clients are accepted.

This globally available object is the first point of entry to the Service Domain.

3. **Inv2User (InvUser) {Type}**. This object represents any client of the Invocation Manager, an Analyzer and possibly one of the Analyzer's Services.

An object representing a User in the service Domain.

4. **InvAnalzr {Type}**. The Invocation Analyzer must perform a client-dependent analysis that will identify the particular service requested. This implies that the Invocation Analyzer must analyse the information received from the client, look up relevant information in its internal databases, load the service object and finally initiate it with the required information.

The Client can represent a User A. This Client requests the initiation of one of User A's services (via the Invocation Manager and Invocation Analyzer) that is to establish a connection to User B. That service, when executing, plays the role of Client recursively in order to request the initiation of one of User B's services, say User B's receive-call service. User B's IA is started and communicates with User A's service in order to analyze the situation and find out if it accepts the call from User A or not. User B can for example have a screening function excluding incoming calls from his mother-in-law. If User B's IA accepts the invocation request from User A's service, it initiates User B's receive-call service. Then this service and User A's service can communicate when executing, in order to set up the connection between User A and User B.

But of course User B's IA is in action also whenever User B himself wants to use his own services. So we have that User B's IA always has to analyze whether the Client is User B or only a user of User B's services. Note that the behavior of the IA can be very different in the two cases.
This object is the User's permanent representative in the IN system and exists as long as the User is a registered IN customer. This role is immutable as seen from users of this Service Constituent.

This role is responsible for managing a User's master service objects for later cloning and invocation.

5. **InvServ (Type)**. The Service role represents the target application for the client. This is the software defining some client's telecommunications service.

   All services to be invocated by this model must have an object which plays this role.

### 7.3.3 Interaction Scenarios

**Figure 25. Service installation {Scenario}**

*Installing a service in an InvocationAnalyzer.*

**Figure 26. Invocation rejection 1 {Scenario}**

*This scenario exemplifies the earliest possible rejection of an invocation request. The returned value is false.*
Figure 27. SelectAndOpenService {Scenario}

Figure 28. Service removal {Scenario}
Figure 29. Invocation success 1 [Scenario]
This scenario shows the basic standard expected message flow during perfect execution of the invocation phase.

7.3.4 Message Sets

1. InvocationInstaller (InvInstaller) [Type]
   - user (u) [Port].
     - InvUser<InvInstaller [Contract].
   - service (s) [Port].
     - InvService<InvInstaller [Contract].
       - initializeWith: dataDict [Message]. Reset attributes according to the values given in dataDict.
- analyzer (a) {Port}.
  - InvAnalzr<InvInstaller {Contract}.
    - installService: aService {Message}. Install the given service object instance in the service data base for this Analyzer's User.
    - removeService: aService {Message}. Remove the given service object instance from service data base for this Analyzer's User, do nothing if it is not defined.
  - userID: uID subscrID: sId manager: mngr user: aUser {Message}. The Invocator issues this initialization message to enter configuration information into the analyzer upon activation of the analyzer. This message contains the following parameters:
    - uID: the User identifier in the Service Domain.
    - sID: the Subscriber identifier in the Service Domain.
    - Pointer to the Invocation Manager.
    - Pointer to the User object corresponding to this Analyzer.
    - servicesOfKind: aKind {Message}. Return all services of the given kind for my User.
- invocManager (im) {Port}.
  - InvMngr<InvInstaller {Contract}.
    - addUser: usr analyzer: anal {Message}. Add the given User and Analyzer objects if a User with usr's userID did not exist beforehand.
  - InvMngr<InvClient {Contract}.
    - invocReq: usrID {Message}. Return the InvocationAnalyzer for the User with the given userID, or false if request is rejected.

2. Inv2User (InvUser) {Type}
- invocManager (im) {Port}. The one and only Invocation Manager.
  - InvMngr<InvClient {Contract}. This scenario exemplifies the general process of selecting and opening any service. Specific services may add additional message interactions.
    - invocReq: usrID {Message}. Return the InvocationAnalyzer for the User with the given userID, or false if request is rejected.
- invocAnalyzer (ia) {Port}.
  - InvAnalzr<InvClient {Contract}.
    - servicesFor: aClient {Message}. Return a list of service kinds, one for each service available to this Client.
    - getService: serviceKind for: aClient {Message}. Return a new instance of the specified service, the service is owned by the User and will be set up for the given Client.
3. **InvAnalzr (Type)**
   - client (c) (Port).
     - InvClient<InvAnalzr (Contract).
       - obeysProtocol: aType (Message). Return true iff the Client obeys the given protocol.
       - clientIdRequest (Message). Return the service ID of the client.
   - service (s) (Port).
     - InvServ<InvAnalzr (Contract).
       - setUp: parameters client: aClient analyzer: anAnalyzer invocator: anInvoc (Message). Create and return an instance of a Service with the given parameters (one of them being owner UserID, another being the current Client).
       - kind (Message). Return aSymbol, which represents the kind of service.
       - protocol (Message). Answer the Client protocol permitted by this service.

4. **InvServ (Type)**
   - invocationAnalyzer (ia) (Port).
     - InvAnalzr<InvServ (Contract).
       - userID (Message). Return the user ID of the user.
       - user (Message). Return the User object of the InvocationAnalyzer.
   - invocationManager (im) (Port).
     - InvMngr<InvServ (Contract).
       - invocReq: usrID (Message). Return the InvocationAnalyzer for the User with the given user ID, or false if request is rejected.
Appendix 1
Implementation for monitored execution

App 1.1 Inv2InstallAndInvocationSPC (Inv2InstInvSPC) {Type Specification}

Report including the following selections automatically generated 23 July 1993: Type Specifications Implementation

App 1.1.1 Inv2InstallAndInvocationSPC {Implementation}

RMSObject1 subclass: #Inv2Analzr0
    instanceVariableNames: 'client invocManager service services subscriberServiceID userID'
    classVariableNames: ''
    poolDictionaries: ''
    category: 'ZZZInLab-Inv2+'

The Invocation Analyzer must perform a client-dependent analysis that will identify the particular service requested. This implies that the Invocation Analyzer must analyse the information received from the client, look up relevant information in its internal databases, load the service object and finally initiate it with the required information. The Client can represent a User A. This Client requests the initiation of one of User A's services (via the Invocation Manager and Invocation Analyzer) that is to establish a connection to User B. That service, when executing, plays the role of Client recursively in order to request the initiation of one of User B's services, say User B's receive-call service. User B's IA is started and communicates with User A's service in order to analyze the situation and find out if it accepts the call from User A or not. User B can for example have a screening function excluding incoming calls from his mother-in-law. If User B's IA accepts the invocation request from User A's service, it initiates User B's receive-call service. Then this service and User A's service can communicate when executing, in order to set up the connection between User A and User B. But of course User B's IA is in action also whenever User B himself wants to use his own services. So we have that User B's IA always has to analyse whether the Client is User B or only a user of User B's services. Note that the behavior of the IA can be very different in the two cases. This object is the User's permanent representative in the IN system and exists as long as the User is a registered IN customer. This role is immutable as seen from users of this Service Constituent. This role is responsible for managing a User's master service objects for later cloning and invocation.

Instance variables:
    client<Inv2Client0>
    invocManager <Inv2Mngr0>
    service <Inv2Serv0>
    services <Object> Unknown type
    subscriberServiceID <Object> Unknown type
    userID <Object> Unknown type
Subclasses should not implement methods for:

Subclasses must implement the following messages:

InvAnalzr<InvClient
  getService:for:
  servicesFor:
InvAnalzr<InvServ
  user
  userID
InvAnalzr<InvInstaller
  installService:
  removeService:
  servicesOfKind:
  userID:subscrID:manager:user:

RMSObject1 subclass: #Inv2Client0
  instanceVariableNames: 'invocAnalyzer invocManager service '
  classVariableNames: '
  poolDictionaries: '
  category: 'ZZZInLab-Inv2+'

This object represents any client of the Invocation Manager, an Analyzer and possibly one of the Analyzer's Services. An object representing a User in the service Domain.

Instance variables:
  invocAnalyzer  <Inv2Analzr0>
  invocManager  <Inv2Mngr0>The one and only Invocation Manager.
  service  <Inv2Serv0>

Subclasses should not implement methods for:

Subclasses must implement the following messages:

InvClient<InvAnalzr
  clientIdRequest
  obeysProtocol:
InvUser<InvInstaller
  userID:manager:analyzer:

RMSObject1 subclass: #Inv2Install0
  instanceVariableNames: 'analyzer invocManager service user '
  classVariableNames: '
  poolDictionaries: '
  category: 'ZZZInLab-Inv2+'

This role is responsible for the creation, initialization and installation of the Invocation Manager, the Users, the Invocation Analyzers and the Users' Services.

Instance variables:
  analyzer  <Inv2Analzr0>
  invocManager  <Inv2Mngr0>
  service  <Inv2Serv0>
  user  <Inv2Client0>

Subclasses should not implement methods for:

Subclasses must implement the following messages:
The Invocation Manager's role is to accept requests from Clients and start the Invocation Analyzer (IA) for further processing and to start specific services on request by the IA. The Invocation Analyzer is unique for a given User, and an Invocation Analyzer can be responsible for analyzing requests for many services. The Invocation Manager can be considered a part of the operating system of the network. Setting the status of the Invocation Manager from roles outside this model may be necessary, for example when the network is going down. Then the Manager must be told not to accept any Clients or only a certain set of privileged Clients. Selective rejection of invocation requests must be handled by the invocator. There are many different schemas for rejection of clients and they may depend on the mode of the invocator. Normally all clients are accepted. During major maintenance all clients may be rejected for a short time period (this is not acceptable in a telecommunications network, but theoretically possible); Certain clients are refused; Most clients are refused. Screening functions may be introduced to decide who are acceptable or who rejectable and so on... This can be useful during crisis situations like war etc. Currently this sort of mechanisms are not contained in the invocator role because we assume all clients are accepted. This globally available object is the first point of entry to the Service Domain.

Instance variables:

- userDatabase <Object> Unknown type

Subclasses should not implement methods for:

Subclasses must implement the following messages:

- InvMngr<InvInstaller
  - addUser:analyzer:
- InvMngr<InvServ
- InvMngr<InvClient
  - invocReq:

The Service role represents the target application for the client. This is the software defining some client's telecommunications service. All services to be invocated by this model must have an object which plays this role.

Instance variables:

- client<Inv2Client0>
- invocationAnalyzer<Inv2Analzr0>
- invocationManager<Inv2Mngr0>

Subclasses should not implement methods for:

Subclasses must implement the following messages:

- InvServ<InvAnalzr
  - kind
  - protocol
  - setUp:client:analyzer:invocator:
- InvService<InvInstaller
  - initializeWith:
Inv2Analzr0 subclass: #Inv2Analzr1
  instanceVariableNames: ''
  classVariableNames: ''
  poolDictionaries: ''
  category: 'ZZZInLab-Inv2+

Inv2Client0 subclass: #Inv2Client1
  instanceVariableNames: 'userID '
  classVariableNames: ''
  poolDictionaries: ''
  category: 'ZZZInLab-Inv2+

Inv2Install0 subclass: #Inv2Install1
  instanceVariableNames: ''
  classVariableNames: 'ServiceDict '
  poolDictionaries: ''
  category: 'ZZZInLab-Inv2+

Class variables:
  ServiceDict <Dictionary> of Symbol->Class (serviceProtocol -> serviceClass)

Inv2Mngr0 subclass: #Inv2Mngr1
  instanceVariableNames: ''
  classVariableNames: 'DefaultAnalyzer '
  poolDictionaries: ''
  category: 'ZZZInLab-Inv2+'

Class variables:
  DefaultAnalyzer <Inv2Mngr0 | nil>

Inv2Serv0 subclass: #Inv2Serv1
  instanceVariableNames: ''
  classVariableNames: ''
  poolDictionaries: ''
  category: 'ZZZInLab-Inv2+'

Inv2Analzr1 class>>subclass:instanceVariableNames:classVariableNames:poolDictionaries:category:
  (compilation)
  subclass: t instanceVariableNames: f classVariableNames: d poolDictionaries: s category: cat
  " Automatically generated. "
  TMWarningDialog
  warning: 'This class is immutable and cannot be subclassed.'.
  ^nil

Inv2Client1 class>>obeysProtocol: (InvClient<InvAnalzr)
  obeysProtocol: aType
  ^aType = #AbstrService

Inv2Install1 class>>addService:asClass: (class initialization)
  addService: aSymbol asClass: aClass
  | coll |
  coll := ServiceDict at: aSymbol asSymbol ifAbsent: [RMSOrderedCollection new].
  (coll includes: aClass) ifFalse: [coll add: aClass].
  ServiceDict at: aSymbol asSymbol put: coll.
Inv2Install1 class>>initialize (class initialization)
initialize
" Inv2Install1 initialize"
ServiceDict := RMSDictionary new.

Inv2Install1 class>>initializeAll (class initialization)
initializeAll
" Inv2Install1 initializeAll"
RMSObject1 withAllSubclassesDo: [:cl | (cl class includesSelector: #initialize) ifTrue: [cl initialize]].
RMSObject1 withAllSubclassesDo: [:cl | (cl class includesSelector: #tmInitialize) ifTrue: [cl tmInitialize]].

Inv2Install1 class>>createDefaultUsers (examples)
createDefaultUsers
RMSGuard1 startTrace: 'Inv2Install1 createDefaultUsers'.
self initializeAll.
self new createDefaultUsers.
RMSGuard1 endAndInspectTrace.

Inv2Install1 class>>createDefaultUsersAndServices (examples)
createDefaultUsersAndServices
RMSGuard1 startTrace: 'Inv2Install1 createDefaultUsersAndServices'.
self initializeAll.
self new createDefaultUsersAndServices.
RMSGuard1 endAndInspectTrace.

Inv2Install1 class>>example1: (examples)
example1: usrID
" Inv2Install1 example1: '12'
RMSGuard1 startTrace: 'Inv2Install1 example1: ''12'''.
self new example1: usrID.
RMSGuard1 endAndInspectTrace.

Inv2Install1 class>>example2: (examples)
example2: apID
" Inv2Install1 example2: '12'
| usr |
RMSGuard1 startTrace: 'Inv2Install1 example2: ''12'''.
usr := Inv2Mngr1 default invocReq: apID.
usr example2.
RMSGuard1 endAndInspectTrace.

Inv2Install1 class>>example4Connect:to: (examples)
example4Connect: callingUserID to: calledUserID
" Inv2Install1 example4Connect: '5601' to: '5602'
| usr |
RMSGuard1 startTrace: 'Inv2Install1 example4Connect: ', callingUserID printString, ' to: ', calledUserID.
usr := Inv2Mngr1 default invocReq: callingUserID.
usr telephoneConnectTo: calledUserID.
RMSGuard1 endAndInspectTrace.

Inv2Install1 class>>countProgramSizeInLab (utilities)
countProgramSizeInLab
" Inv2Install1 countProgramSizeInLab"
| clSet methCount linesCount |
clSet := IdentitySet new.
(SystemOrganization categories select: [:nam | (nam findString: 'ZZZInLab' startingAt: 1) = 1]) do:
 [:nam |
  (SystemOrganization listAtCategoryNamed: nam) do:
    [:clNam | clSet add: (Smalltalk at: clNam)]].

methCount := 0.
linesCount := 0.
clSet do:
  [:cl || strm2 |
   strm2 := ReadStream on: cl definition , cl comment , cl class instanceVariablesString.
   [strm2 atEnd] whileFalse:
     [(strm2 next = Character cr) | (strm2 next = Character lf)
      ifTrue: [linesCount := linesCount + 1]].
   methCount := methCount + cl selectors size + cl class selectors size.
  cl selectors do:
    [:sel || strm |
     strm := ReadStream on: (cl compiledMethodAt: sel) getSource.
     [strm atEnd] whileFalse:
       [(strm next = Character cr) | (strm next = Character lf)
        ifTrue: [linesCount := linesCount + 1]].
   cl class selectors do:
     [:sel || strm |
      strm := ReadStream on: (cl class compiledMethodAt: sel) getSource.
      [strm atEnd] whileFalse:
        [(strm next = Character cr) | (strm next = Character lf)
         ifTrue: [linesCount := linesCount + 1]].
  self dpsTrace:
    String cr , 'numCl: ' , clSet size printString
    , String cr , 'numMeth: ' , methCount printString
    , String cr , 'numLines: ' , linesCount printString.

Inv2Install1 class>>fileIn (utilities)
fileIn
  " Inv2Install1 fileIn."
  RMSGuard1 startTrace: 'Inv2Install1 fileIn'.
  self new fileIn: 'installData.asc'.
  RMSGuard1 endAndInspectTrace.

Inv2Install1 class>>fileOutINLabTo: (utilities)
fileOutINLabTo: fNam
  " 930721 trygve(4.1): IN-Lab Service Contract."
  " Inv2Install1 fileOutINLabTo: 'ZZInLab-930721.st'"
  | strm clSet |
  strm := SourceCodeStream on: fNam asFilename writeStream.
  clSet := IdentitySet new.
  (SystemOrganization categories select: [:nam | (nam findString: 'ZZZInLab' startingAt: 1) = 1]) do:
    [:nam |
     (SystemOrganization listAtCategoryNamed: nam) do:
       [:clNam | clSet add: (Smalltalk at: clNam)]].
  (ChangeSet superclassOrder: clSet) do: [:cl | cl fileOutSourceOn: strm]
  valueNowOrOnUnwindDo: [strm close].

Inv2Mngr1 class>>default (defaults)
default
  " Return the single, unique instance of this class."
  " Create new one if necessary."
  " Inv2Invoc1 default" 
  DefaultAnalyzer isNil ifTrue: [DefaultAnalyzer := self new].
  ^DefaultAnalyzer

Inv2Mngr1 class>>initialize (class initialization)
initialize
  DefaultAnalyzer := nil.
Inv2Mngr1 class>>subclass:instanceVariableNames:classVariableNames:poolDictionaries:category:
(compilation)
subclass: t instanceVariableNames: f classVariableNames: d poolDictionaries: s category: cat
" Automatically generated."
TMWarningDialog
   warning:'This class is immutable and cannot be subclassed.'.
   ^nil

Inv2Serv1 class>>protocol (constants)
protocol
self halt: 'illegal'.
^#AbstrService

Inv2Serv1 class>>protocolConst (constants)
protocolConst
^#AbstrService

Inv2Serv1 class>>tmInitialize (class initialization)
tmInitialize
  " Inv2Serv1 tmInitialize  
Inv2Install1 addService: self protocolConst asClass: self

Inv2Analzr1>>getService: (InvAnalzr<InvClient)
getService: serviceType for: aClient
  " Return a new instance of the specified service, the service is owned "
  " by the User and will be set up for the given Client."
  | servColl x |
  x := 0.
  servColl := services select: [:srv | x := x + 1.
    srv kind = serviceType
    and: [aClient obeysProtocol: srv protocol]].
  servColl isEmpty ifTrue: [:nil].
  servColl := servColl.
  servColl size > 1
  ifTrue:
    [self dpsCaution:
      'Multiple services: ', servColl asArray printString
      String cr, 'The first is used if you proceed.'].

Inv2Analzr1>>servicesFor: (InvAnalzr<InvClient)
servicesFor: aClient
  " Return a list of service kinds, one for each service available to "
  " this Client."
  | x |
  x := 0.
  ^services collect: [:srv | x := x + 1. srv kind]

Inv2Analzr1>>initialize (private)
initialize
  super initialize.
  client := nil.
  service := nil.
  userID := nil.
  services := RMSOrderedCollection new.

Inv2Analzr1>>user (InvAnalzr<Invoke)
user
  " Return the User object of the InvocationAnalyzer."
  ^Inv2Mngr1 default invocReq: userID
Inv2Analzr1>>userID: (InvAnalzr<InvServ)

userID
" Return the user ID of the user."
^userID

Inv2Analzr1>>installService: (InvAnalzr<InvInstaller)

installService: aService
" Install the given service object instance in the service data base "
" for this Analyzer's User."
/services includes: aService
  ifTrue: [self dpsCaution: aService printString, ' exists, duplicate installation ignored.']
  ifFalse: [services add: aService].

Inv2Analzr1>>removeService: (InvAnalzr<InvInstaller)

removeService: aService
" Remove the given service object instance from service data base for "
" this Analyzer's User, do nothing if it is not defined."
/services includes: aService
  ifTrue: [services remove: aService]
  ifFalse: [self dpsCaution: aService printString, ' unknown, removal ignored.'].

Inv2Analzr1>>servicesOfKind: (InvAnalzr<InvInstaller)

servicesOfKind: aKind
" Return all services of the given kind for my User."
" 930706 trygve(4.1): IN-Lab Service Contract."
|x|
x := 0.
^services select: [:srv | x := x + 1. srv kind = aKind]

Inv2Analzr1>>userID:subscrID:manager:user: (InvAnalzr<InvInstaller)

userID: uID subscrID: sID manager: mngr user: aUser
client := nil.
invocManager := mngr.
/service := nil.
services := RMSOrderedCollection new.
subsriberServiceID := sID.
userID := uID.

Inv2Analzr1>>postCopy (copying)

postCopy
super postCopy.
"client"
"invocManager"
service := nil.
services := services collect: [:serv | serv copy].
userID := nil.
"subscriberServiceID"

Inv2Client1>>clientIdRequest (InvClient<InvAnalzr)

clientIdRequest
" Return the service ID of the client."
^userID

Inv2Client1>>obeysProtocol: (InvClient<InvAnalzr)

obeysProtocol: aType
" Return true iff the Client obeys the given protocol."
self class obeysProtocol: aType
Inv2Client1>>example2 (private-examples)

example2
    self dpsTrace: (invocAnalyzer servicesFor: self guard).
    self dpsTrace: (invocAnalyzer getService:#AbstrService for: self guard).

Inv2Client1>>initialize (private)
initialize
    super initialize.
    invocAnalyzer := nil.
    invocManager := nil.
    service := nil.
    userID := nil.

Inv2Client1>>userID:manager:analyzer: (InvUser<InvInstaller)

userID: uID manager: aManager analyzer: anAnal
    " Initialization message."
    invocAnalyzer := anAnal.
    invocManager := aManager.
    userID := uID.

Inv2Client1>>userID (access)

^userID

Inv2Install1>>invocManager (private)

invocManager
    invocManager isNil ifTrue: [invocManager := Inv2Mngr1 default].
    ^invocManager

Inv2Install1>>createServiceFor:from: (private-fileIn)

createServiceFor: analz from: dataDict
    " 930706 trygve(4.1): IN-Lab Service Contract."
    | servKind xxx |
    servKind := (dataDict at: 'serviceKind') asSymbol.
    (analz servicesOfKind: servKind) isEmpty
    ifTrue:
        [(ServiceDict at: servKind ifAbsent: [self dpsCaution: 'No service class for ', servKind. #()]) do:
        [:servCl | analz installService: servCl new]].
    xxx := 0.
    (analz servicesOfKind: servKind) do:
    [:serv |
    xxx := xxx +1.
    serv initializeFrom: dataDict].

Inv2Install1>>createSubscriberFrom: (private-fileIn)

createSubscriberFrom: dataDict
    " 930706 trygve(4.1): IN-Lab Service Contract."
    ^dataDict at: 'subscrServiceId'

Inv2Install1>>createUserFor:from: (private-fileIn)

createUserFor: subscrId from: dataDict
    " 930706 trygve(4.1): IN-Lab Service Contract."
    | userDict userId analz xxx |
    userDict := RMSDictionary new.
    dataDict associationsDo:
    [:ass |
    (ass key findString: 'User-Data' startingAt: 1) = 1
    ifTrue:
    [userDict := ass value copy]].
userDict at: 'name' put: (dataDict at: 'title').
userld := userDict at: 'userID'.
analz := self invocManager invocReq: userld.
analz isNil
ifTrue:
    [self newUserID: userld subscrID: subscrId.
analz := self invocManager invocReq: userld].
xxx := 0.
dataDict associationsDo:
    [:ass |
        xxx := xxx + 1.
        ((ass value isKindOf: Dictionary)
            and: [ass value includesKey: 'serviceKind'])
            ifTrue:
            [self createServiceFor: analz from: ass value]].

Inv2Install1>>fileIn: (private-fileIn)
fileIn: fNam
" 930706 trygve(4.1): IN-Lab Service Contract. "
" Inv2Install1 new fileIn: 'installData.asc' "
| dataDict |
self class initializeAll.
dataDict := Object readStructureFromFile: fNam asFilename.
dataDict keysDo:
    [:dataKey || subscrId |
        (dataKey findString: 'subscriber' startingAt: 1) = 1
        ifTrue: [(dataDict at: dataKey) associationsDo:
            [:ass |
                (ass key findString: 'subscrData' startingAt: 1) = 1
                ifTrue: [self createSubscriberFrom: ass value]].
        (dataDict at: dataKey) associationsDo:
            [:ass |
                (ass key findString: 'User' startingAt: 1) = 1
                ifTrue: [self createUserFor: subscrId from: ass value]].
    ]].

Inv2Install1>>newUserID:subscrID: (private-fileIn)
newUserID: uID subscrID: sID
"  Establish a new User with 
"    serviceID = uID 
"    switchID = sID 
| usr anal |
usr := Term2User1 new.
anal := Inv2Analzr1 new.
usr userID: uID manager: self invocManager analyzer: anal.
anal userID: uID subscrID: sID manager: self invocManager user: usr.
self invocManager addUser: usr analyzer: anal.

Inv2Install1>>createDefaultUsers (private-examples)
createDefaultUsers
| accsns |
accsns := self defaultClients associations.
1 to: accsns size do:
    [:i || ass |
        ass := accsns at: i.
        self newUserID: ass key subscrID: 'subscr']

Inv2Install1>>createDefaultUsersAndServices (private-examples)
createDefaultUsersAndServices
| xxx |
self createDefaultUsers.

xxx := 0.
self defaultClients keys do:
[ :serId || anal |
anal := self invocManager invocReq: serId.
ServiceDict associations do:
[ :ass || clColl |
clColl := ass value.
clColl do:
[ :servCl |
xxx := xxx + 1.
anal installService: servCl new]]].

Inv2Install1>>defaultClients (private-examples)
defaultClients
^ (RMSDictionary new)
at: '12' put: $X;
at: '13' put: $X;
yourself

Inv2Install1>>example1: (private-examples)
exmple1: usrID
| anal serv |
anal := self invocManager invocReq: usrID.
serv := Inv2Serv1 new.
anal installService: serv; removeService: serv; installService: serv.

Inv2Mngr1>>initialize (private)
initialize
super initialize.
userDatabase := Dictionary new.

Inv2Mngr1>>scanFrom: (fileIn)
scanFrom: aStream
" 930706 trygve(4.1): IN-Lab Service Contract."
| string |
selself dpsTrace: 'Scan InvManager'.
[ string := aStream nextChunk.
string size > 0] "done when double terminators"
whileTrue:
[ self dpsTrace: string.
self class evaluatorClass new
evaluate: string
in: nil
to: self
notifying: nil
ifFail: [ ^nil ]]

Inv2Mngr1>>addUser:analyzer: (InvMngr<InvInstaller)
addUser: usr analyzer: anal
" Add the given User and Analyzer objects if a User with usr's userID "
" did not exist beforehand."

| uID |
uID := usr userID.
(userDatabase includesKey: uID)
ifTrue:
[ self dpsCaution: uID printString , ' already defined.'.
^self].
userDatabase at: uID put: (Array with: usr with: anal).
Inv2Mngr1>>invocReq: (InvMngr<InvClient)
invocReq: srvID
 " Return the InvocationAnalyzer for the User with the given userID, "
 " or false if request is rejected."
 | arr |
 arr := userDatabase at: srvID ifAbsent: [nil].
 ^arr isNil ifTrue: [nil] ifFalse: [arr last]

Inv2Serv1>>initialize (private)
initialize
 super initialize.
 client := nil.
 invocationAnalyzer := nil.
 invocationManager := nil.

Inv2Serv1>>postCopy (private)
postCopy
 super postCopy.
 client := nil.
 invocationAnalyzer := nil.
 invocationManager := nil.

Inv2Serv1>>kind (InvServ<InvAnalzr)
kind
 " Return aSymbol, which represents the kind of service. "
 ^#dummy

Inv2Serv1>>protocol (InvServ<InvAnalzr)
protocol
 " Answer the Client protocol permitted by this service. "
 ^self class protocolConst

Inv2Serv1>>setUp:parameters client:analyzer:invocator: (InvServ<InvAnalzr)
setUp: parameters client: aClient analyzer: anAnalyzer invocator: anInvoc
 " Create and return an instance of a Service with the given parameters "
 " (one of them being owner UserID, another being the current Client)."
 client := aClient.
 invocationAnalyzer := anAnalyzer.
 invocationManager := anInvoc.

Inv2Serv1>>initializeWith: (InvService<InvInstaller)
initializeWith: dataDict
 " Reset attributes according to the values given in dataDict. "
 self dpsError: 'Not implemented'.

Implementation for monitored execution 22 September 1993 at: 15:33

©Taskon 1998. Page 58
TABLE OF CONTENTS

Chapter 1  Introduction .......................................................... 2

Chapter 2  Overview of the Installation and Invocation module .......... 3

Chapter 3  Invocator2 Monitored executions ................................. 7

3.1  Inv2Install1>>createDefaultUsers {Role Model} ....................... 7
3.1.1  Area of Concern .................................................................. 7
3.1.2  The Roles ......................................................................... 8
3.1.3  Interaction Scenarios.......................................................... 9
3.1.4  Message Sets .................................................................. 9
3.2  Inv2Install1>>example1: '12' {Role Model} ......................... 10
3.2.1  Area of Concern .................................................................. 10
3.2.2  The Roles ......................................................................... 11
3.2.3  Interaction Scenarios.......................................................... 11
3.2.4  Message Sets .................................................................. 12
3.3  Inv2Install1>>example2: '12' {Role Model} ......................... 12
3.3.1  Area of Concern .................................................................. 12
3.3.2  The Roles ......................................................................... 12
3.3.3  Interaction Scenarios.......................................................... 13
3.3.4  Message Sets .................................................................. 13

Chapter 4  Service Installation and Invocation export models .......... 14

4.1  Inv2InvocationMCH (Inv2InvMCH) {Export Model} .................. 14
4.1.1  Area of Concern .................................................................. 14
4.1.2  The Roles ......................................................................... 15
4.1.3  Interaction Scenarios.......................................................... 15
4.2  Inv2InvocationFW (Inv2InvFW) {Export Model} .................... 16
4.3  Inv2InstallationEXP (Inv2InstallEXP) {Export Model} ........... 17
4.3.1  Area of Concern .................................................................. 17
4.3.2  The Roles ......................................................................... 17
4.3.3  Interaction Scenarios.......................................................... 18

Chapter 5  Invocator2 design models .......................................... 21

5.1  Inv2InvRM {Role Model} ..................................................... 21
5.1.1  Area of Concern .................................................................. 21
5.1.2  The Roles ......................................................................... 22
5.1.3  Interaction Scenarios.......................................................... 24
5.1.4  Message Sets .................................................................. 25
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.2</td>
<td>Service Installation (Inv2InstallRM) {Role Model}</td>
<td>26</td>
</tr>
<tr>
<td>5.2.1</td>
<td>Area of Concern</td>
<td>26</td>
</tr>
<tr>
<td>5.2.2</td>
<td>The Roles</td>
<td>27</td>
</tr>
<tr>
<td>5.2.3</td>
<td>Interaction Scenarios</td>
<td>28</td>
</tr>
<tr>
<td>5.2.4</td>
<td>Message Sets</td>
<td>29</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Chapter 6</th>
<th>Invocator2 Object Type Specification models</th>
<th>30</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.1</td>
<td>Inv2InstallAndInvocationSPC (Inv2InstInvSPC) {Type Specification}</td>
<td>30</td>
</tr>
<tr>
<td>6.1.1</td>
<td>Area of Concern</td>
<td>30</td>
</tr>
<tr>
<td>6.1.2</td>
<td>The Roles</td>
<td>31</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Chapter 7</th>
<th>Models exported from the Invocator2 module</th>
<th>34</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.1</td>
<td>Inv2InvocationMCH (Inv2InvMCH) {Export Model}</td>
<td>34</td>
</tr>
<tr>
<td>7.1.1</td>
<td>Area of Concern</td>
<td>34</td>
</tr>
<tr>
<td>7.1.2</td>
<td>The Roles</td>
<td>35</td>
</tr>
<tr>
<td>7.1.3</td>
<td>Interaction Scenarios</td>
<td>37</td>
</tr>
<tr>
<td>7.1.4</td>
<td>Message Sets</td>
<td>38</td>
</tr>
<tr>
<td>7.2</td>
<td>Inv2InvocationFW (Inv2InvFW) {Export Model}</td>
<td>39</td>
</tr>
<tr>
<td>7.3</td>
<td>Inv2InstallationEXP (Inv2InstallEXP) {Export Model}</td>
<td>39</td>
</tr>
<tr>
<td>7.3.1</td>
<td>Area of Concern</td>
<td>40</td>
</tr>
<tr>
<td>7.3.2</td>
<td>The Roles</td>
<td>40</td>
</tr>
<tr>
<td>7.3.3</td>
<td>Interaction Scenarios</td>
<td>42</td>
</tr>
<tr>
<td>7.3.4</td>
<td>Message Sets</td>
<td>44</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Appendix 1</th>
<th>Implementation for monitored execution</th>
<th>47</th>
</tr>
</thead>
<tbody>
<tr>
<td>App 1.1</td>
<td>Inv2InstallAndInvocationSPC (Inv2InstInvSPC) {Type Specification}</td>
<td>47</td>
</tr>
<tr>
<td>App 1.1.1</td>
<td>Inv2InstallAndInvocationSPC {Implementation}</td>
<td>47</td>
</tr>
</tbody>
</table>
INDEX

......Inv2Analzr1...1..........................50
class>>subclass:instanceVariableNames:classVariableNames:poolDictionaries:category:
(compilation)
Inv2Analzr1>>getService:for: (InvAnalzr<InvClient)...............................53
Inv2Analzr1>>initialize (private)..........................................................53
Inv2Analzr1>>installService: (InvAnalzr<InvInstaller)...............................54
Inv2Analzr1>>postCopy (copying)..............................................................54
Inv2Analzr1>>removeService: (InvAnalzr<InvInstaller)..................................54
Inv2Analzr1>>servicesFor: (InvAnalzr<InvClient)........................................53
Inv2Analzr1>>servicesOfKind: (InvAnalzr<InvInstaller)..............................54
Inv2Analzr1>>user (InvAnalzr<InvServ)....................................................53
Inv2Analzr1>>userID (InvAnalzr<InvServ)..................................................54
Inv2Analzr1>>userID:subscrID:manager:user: (InvAnalzr<InvInstaller).........54
Inv2Client1 class>>obeysProtocol: (InvClient<InvAnalzr)..............................50
Inv2Client1>>clientIdRequest (InvClient<InvAnalzr)....................................54
Inv2Client1>>example2 (private-examples)..............................................55
Inv2Client1>>initialize (private)............................................................55
Inv2Client1>>obeysProtocol: (InvClient<InvAnalzr)....................................54
Inv2Client1>>userID (access)........................................................................55
Inv2Client1>>userID:manager:analyzer: (InvUser<InvInstaller)......................55
Inv2Install1 class>>addService:asClass: (class initialization)......................50
Inv2Install1 class>>countProgramSizeInLab (utilities)..................................51
Inv2Install1 class>>createDefaultUsers (examples).......................................51
Inv2Install1 class>>createDefaultUsersAndServices (examples).......................51
Inv2Install1 class>>example1: (examples)......................................................51
Inv2Install1 class>>example2: (examples)......................................................51
Inv2Install1 class>>example4Connect:to: (examples)......................................51
Inv2Install1 class>>fileIn (utilities)............................................................52
Inv2Install1 class>>fileOutINLabTo: (utilities)...........................................52
Inv2Install1 class>>initialize (class initialization)........................................51
Inv2Install1 class>>initializeAll (class initialization)...................................51
Inv2Install1>>createDefaultUsers (private-examples)....................................56
Inv2Install1>>createDefaultUsersAndServices (private-examples)....................56
Inv2Install1>>createServiceFor:from: (private-fileIn)..................................55
Inv2Install1>>createSubscriberFrom: (private-fileIn)..................................55
Inv2Install1>>createUserFor:from: (private-fileIn)......................................55

Service Installation and Invocation Service Constituent module...
Service Installation and Invocation Service Constituent module...