# T1B3 – The Demo

## 1. Demo Description

The demo presents a prototype implementation of a payment system for public Internet access on airports. It is designed so that clients of several air-carriers from the same group (e.g. SkyTeam) can have access to the Internet. Clients access the system via a wireless network (e.g. WiFi); before being able to establish communication with the Internet, they have to authenticate themselves and/or pay for the service. There are currently three different ways a client can gain access to the Internet:

a) All clients that have a valid fly ticket for first class or business class have full Internet access during the ticket validity period free of charge. For this access method the fly ticket identification number is used during the client login as authentication credentials in the system.
b) Any client that has a valid Frequent Flyer Card and has any valid fly ticket has also full Internet access during the ticket validity period free of charge. The Frequent Flyer Card identification number is used as credentials. The system checks that a valid fly ticket exists for the card.
c) Any client of any air-carrier can prepay Internet access by a credit card. The clients using this method will get a user name and a password that are valid for a certain amount of time (e.g. 1 week or 1 month) and in this time period the prepaid time for Internet access needs to be used up (else the remaining prepaid time will expire).

The client session (the client's ability to communicate with Internet servers) starts when the client authenticates using one of the previous methods and terminates when one of the following events occurs:

a) The client disconnects from the wireless network – any prepaid time not used up during the session being terminated can be used up in future sessions assuming the client's user name will not expire until then.
b) Client's fly ticket becomes invalid or all of the client's prepaid time is used up – the session terminates immediately and the client can start a new prepaid session.

The key part of the system behavior will be implemented in Fractal components. However, the clients will communicate with the system via JSP web pages that will then call the Fractal components with appropriate requests. The demo environment is divided into three areas/networks:

a) Airport WiFi – The public wireless network that clients connect to.
b) Airport LAN – All demo Fractal components run on computers in this network. The communication between this network and the Airport WiFi is separated by the Firewall that is controlled by the Firewall Fractal component.
c) Internet – The part representing the outer world. It hosts central servers and web services (e.g. credit card web services, air-carriers database servers) and also the client communication goes there (if not blocked by the Firewall)

On the lowest level the client connections are managed by the DhcpServer Fractal component. This component assigns IP addresses to new clients and notifies other demo Fractal components when clients disconnect, so that their sessions can be terminated. The DhcpServer component might also communicate with some sort of wireless network access point, in order to get more accurate information about client connection and disconnection events.

### 1.1. Demo Behavior

See the diagram of the whole demo in the "T1B3 - Demo - Whole Picture.pdf" file.
Note: The numbered symbols like ②, [3a], ⓷ⓑ or ⟨3c⟩ reference the appropriate numbered steps in the picture.

**Initial state:**

- No clients connected
- Any DHCP communication is allowed through the Firewall
- All HTTP/HTTPS requests from clients are redirected to the airport's WebServer residing in the Airport LAN. All such requests are redirected to the Login page, that all clients use to enter login credentials to open connection to the Internet.
- All other outgoing or incoming communication from/to clients is initially blocked ⓪ by the Firewall service.

**Event: A new client connects to the Airport WiFi network**

- The Client sends a DHCP request for an IP address ① and the DhcpServer replies ① assigning a new IP address to the Client (see section 2.1 for a detailed description of DhpcServer composite component behavior). The Client then uses this IP address for all communication with the system and during any access to the Internet. The assigned IP address also serves as a unique client identifier.
- The Client enters the Login page (by typing any URL in the web browser) ②
- The Client then continues in one of the three possible ways depending on the access method selected:
  - ***a) Free access using first class or business class fly ticket ID:***
    - The Client fills the fly ticket ID in the login form
    - The Login page calls the Arbitrator:ILogin.LoginWithFlyTicketId(Id) method with the fly ticket Id supplied by the Client [3a]
    - The Arbitrator calls the FlyTicketDatabase:IFlyTicketAuth.CreateToken(FlyTicketId) method [4a] to create the Token component representing the "logged in" state of the Client
    - The FlyTicketDatabase:FlyTicketClassifier component then selects the appropriate fly ticket database component depending on the supplied fly ticket ID and calls the IFlyTicketDb.GetTicketValidity(FlyTicketId) method on it [5a] (CsaDbConnection component is selected in the example).
    - The database component then uses any proprietary protocol to connect to the air-carrier's database server to get the requested information [6a]
    - If the supplied FlyTicketId is valid then the associated fly ticket validity time is passed back to the FlyTicketClassifier component [5a]
    - The FlyTicketClassifier creates a new instance of the Token component [7a] with the validity set to the time returned in the previous step. The Token component instance will be created without the Token:CustomToken inner component as it is not needed in this type of authentication.
    - Further steps are common for all three authentication methods (to continue skip the specific steps for methods b) and c) ).
  - ***b) Free access using the Frequent Flyer Card ID:***
    - The Client fills the Frequent Flyer Card ID in the login form
    - The Login page calls the Arbitrator:ILogin.LoginWithFrequentFlyerId(Id) method with the Frequent Flyer Card Id supplied by the Client ⓷ⓑ
    - The Arbitrator calls the CreateToken(FrequentFlyerId) method ⓸ⓑ on the FrequentFlyerDatabase:IFreqFlyerAuth interface to create the Token component representing the "logged in" state of the Client

- The FrequentFlyerDatabase component then connects to the central database of issued Frequent Flyer Cards (5b) ("SkyTeam Frequent Flyer Database") and checks the FrequentFlyerId validity
- If the supplied Frequent Flyer Card ID is valid then the FlyTicketDatabase:IFlyTicketDb.GetTicketByFreqFlyerId( FrequentFlyerId) method is called (6b) to check if there exists any valid fly ticket bought with the Frequent Flyer Card at any air-carrier's office of the group
- The FlyTicketDatabase:FlyTicketClassifier requests the same information from all of the database connection providers by calling the IFlyTicketDb:GetTicketByFreqFlyerId method on them (7b)
- Each of the database connection components then connects to its own air-carrier's database server (8b) to retrieve list of all valid fly tickets bought using the supplied FrequentFlyerId.
- The FlyTicketClassifier gathers responses from all fly ticket databases (7b) and the resulting list of valid fly tickets (possibly empty) is retured back from the FlyTicketDatabase:IFlyTicketDb. GetTicketByFreqFlyerId (FrequentFlyerId) call (6b)
- If the resulting list of valid fly tickets is not empty the FrequentFlyerDatabase selects one of the tickets that is currently valid and calls the FlyTicketDatabase:IFlyTicketAuth.CreateToken(FlyTicketId) method (9b) to create a new instance of the Token component. The following steps to create the Token component are similar to the most of the steps of the a) authentication method:
  - The FlyTicketDatabase:FlyTicketClassifier component selects the appropriate fly ticket database component depending on the supplied fly ticket ID and calls the IFlyTicketDb.GetTicketValidity(FlyTicketId) method on it (10b) (AfDbConnection component is selected in the example).
  - The database component then uses any proprietary protocol to connect to the air-carrier's database server to get the requested information (11b)
  - If the supplied FlyTicketId is valid then the associated fly ticket validity time is passed back to the FlyTicketClassifier component (10b)
  - The FlyTicketClassifier creates a new instance of the Token component (12b) with the validity set to the time returned in the previous step. The Token component instance will be created without the Token:CustomToken inner component as it is not needed in this type of authentication.
- The instance of the Token component (12b) returned from the FlyTicketDatabase:IFlyTicketAuth.CreateToken(FlyTicketId) method call (9b) is then returned back (4b) to the Arbitrator component
- Further steps are common for all three authentication methods (to continue skip the specific steps for method c) ).

***c) Prepaid access***

- If the Client does not possess a user name and a passport to a prepaid access account then such an account needs to be created first:
  - The Client fills his or her credit card number and the card expiration date into the account creation form on the Login page. The Client also chooses the amount of time that will be prepaid to his or her new account.
  - The Client selects an AccountId or a random AccoundId can be generated by calling the AccountDatabase:IAccount. GenerateRandomAccoundId() method.
  - The Client selects a password or a random password is generated by the Login page
  - The Login page calls the AccountDatabase:IAccount.CreateAccount(AccountId, Password) method (3c) to create a new prepaid account for the Client
  - The AccountDatabase creates a new account in the central database (4c). The account will have default validity or timeout period associated with it (e.g. 1 week or 1 month as mentioned earlier) and will have no Internet access time prepaid.
  - The Login page uses the AccoundId and other information supplied by the Client to call the AccountDatabase:IAccount.RechargeAccount(AccountId, CardId, CardExpirationDate, PrepaidTime) method (5c) used to pay for more time to access the Internet.
  - The AccountDatabase calls the CardCenter:ICardCenter.Withdraw(CardId, CardExpirationTime, Amount) method (6c) to withdraw the correct amount (depending on the PrepaidTime) from the Client's account.
  - The CardCenter component then selects the right credit card authorization center ("VISA Card Center" in the example), checks the validity of the requests and communicates with the card center (7c) to process the request.
  - If the requested amount was successfully withdrawn from the Client's account the AccountDatabase enters the new prepaid time into the Client's account record in the central Account Database (8c) and success is returned to the Login page (5c).
- The Client fills his or her account used id and password into the login form on the Login page (these credentials are generated/selected during the account creation)
- The Login page calls the Arbitrator:ILogin.LoginWithAccountId(AccountId, Password) method with the credentials supplied by the Client (9c)
- The Arbitrator calls the CreateToken(AccountId, Password) method (10c) on the AccountDatabase:IAccountAuth interface to create the Token component representing the "loged in" state of the Client
- The AccountDatabase component gets the prepaid time of the account from the central Account Database (11c)
- The AccountDatabase creates a new instance of the Token component (12c) with its validity time set to the prepaid time from the previous step. The Token component instance will contain the CustomToken subcomponent. If the Client's session terminates the CustomToken component is used to communicate the amount of prepaid time already used up back to the AccountDatabase via the IAccount:AjustAccountPrepaidTime(AccountId, SecurityCookie, TimeLeft) method (21). The SecurityCookie is a random string generated during creation of the Token component and it is passed by the AccountDatabase component to the CustomToken component during its construction. The CustomToken saves the SecurityCookie, so that it can use it later to prove its connection to the AccountId specified in the IAccount:AjustAccountPrepaidTime call.
- The following steps are common for all three authentication methods:

- Now the Arbitrator component already has a reference to a new instance of the Token component returned either from the FlyTicketDatabase, the FrequentFlyerDatabase or the AccountDatabase component, depending on the authentication method selected. The Arbitrator adds this reference into its internal table that maintains the bijection between Token component instances and connected Clients (Clients' IP addresses)
- The Arbitrator calls the Firewall:IFirewall.DisablePortBlock(IpAddress) method (13) ([8a] or (13b) or (13c))
- The Firewall component uses a proprietary communication mechanism to forward the request to the Firewall system service (14). The Firewall service opens all communication ports to/from the Client and stops redirecting all HTTP/HTTPS communication from the Client to the airport's WebServer (15 – Cancel port block). Until the session terminates the Client can access the Login page only using a special URL (server address that was displayed on the Login page). If the Client forgets the URL, he or she can simply disconnect from the wireless

network (this action will automatically terminate the current session) and reconnect again, so that a new session is started and all Client requests are redirected back to the Login page.

- The Client has now full access to the Internet ⑯ until its session terminates (its access Token becomes invalid or he or she disconnects from the Airport WiFi network).

**Event: A client disconnects from the Airport WiFi network (17 – Client disconnects):**

- If, after certain amount of time, the Client does not send the "Renew IP address" DHCP request the DhcpServer:IpAddressManager component will deduce that the Client has disconnected and will call the Abitrator:IDhcpCallback:IpAddressInvalidated(IpAddress) method ⑱ (see section 2.1 for a more detailed description of this process). This will start the process of terminating current Client's session
- The Arbitrator calls the IToken.InvalidateAndSave() method ⑲α on the right Token component instance (based on the IP address passed in the IpAddressInvalidated call and the IP address/Token instance mapping stored in the internal table)
- If the Token component instance contains the CustomToken subcomponent (i.e. if the instance was created by the AccountDatabase:IAccountAuth.CreateToken calls) the following steps will occur too:
  - The Token:ValidityChecker component calls the Token:CustomToken.InvalidatingToken( TimeLeft) method ⑳ passing it the amount of time until the Token should have become invalid.
  - The CustomToken calls the AccountDatabase:IAccount.AjustAccountPrepaidTime(AccountId, SecurityCookie, TimeLeft) method ㉑ to update the amount of prepaid time left on the Client's account.
  - The AccountDatabase updates the prepaid time in the central database ㉒.
- The Token:ValidityChecker component calls the Arbitrator:ITokenCallback.TokenInvalidated( TokenId) method ㉓. This signals the Arbitrator component that the Token have become invalid and that the associated Client's session should be terminated.
- The Arbitrator calls the Firewall:IFirewall.EnablePortBlock(IpAddress) ㉔
- The Firewall component then communicates with the Firewall system service ㉕ so that all existing connections to or from the Client are closed and no new ones are allowed (except for the DHCP communication) and that all HTTP/HTTPS requests from the Client are again redirected to the Login page on the WebServer ㉖.
- The Client's session is terminated and, from the point of view of the Client, the system is in the same state as it was in the initial state.

**Event: Client's Token becomes invalid – i.e. Client's fly ticket becomes invalid or all of the prepaid time is used up:**

- The steps that will follow are similar to the steps following the previous event. The only difference is that the session termination is not initiated by the DhcpServer component, but by one of the Token components itself.
- The Token:Timer component times out – i.e. Token validity has ended and it calls the ValidityChecker:ITimerCallback.Timeout() method ⑲β.
- *If the Token component instance contains the CustomToken subcomponent (i.e. if the instance was created by the AccountDatabase:IAccountAuth.CreateToken calls) the following steps will occur too:*
  - *The Token:ValidityChecker component calls the Token:CustomToken.InvalidatingToken( TimeLeft) method ⑳ passing the amount of time until the Token should have become invalid.*
  - *The CustomToken calls the AccountDatabase:IAccount.AjustAccountPrepaidTime(AccountId, SecurityCookie, TimeLeft) method ㉑ to update the amount of prepaid time left on Client's account.*
  - *The AccountDatabase updates the prepaid time in the central database ㉒.*
- *The Token:ValidityChecker component calls the Arbitrator:ITokenCallback.TokenInvalidated( TokenId) method ㉓. This signals the Arbitrator component that the Token have become invalid and that the associated Client's session should be terminated.*
- *The Arbitrator calls the Firewall:IFirewall.EnablePortBlock(IpAddress)* ㉔
- *The Firewall component then communicates with the Firewall system service ㉕ so that all existing connections to or from the Client are closed and no new ones are allowed (except for the DHCP communication) and that all HTTP/HTTPS requests from the Client are again redirected to the Login page on the WebServer ㉖.*

*The Client's session is terminated and, from the point of view of the Client, the system is in the same state as it was in the initial state.*

# 2. DhcpServer Component Description

See DhcpServer component diagram in the "T1B3 – DhcpServer with Synchronization.pdf" file.

## 2.1. DhcpServer Behavior

The DhcpServer component can behave in two different ways:

1. IP addresses for new clients are automatically generated (by a preconfigured pattern – e.g. valid IP address range) by the DhcpServer component. In this scenario the IP/MAC address mappings are stored in the TransientIpDb database component only and IpAddressManager's IIpMacPermanentDb interface is not used at all (this implies that the DhpcServer:IIpMacPermanentDb interface does not need to be bound in this scenario). This scenario is used by the demo.
2. The second option is that the IP/MAC address mappings can be permanently stored in an external database component (providing an IIpMacDb interface) and the DhcpServer assigns IP addresses to new clients according to their MAC address and the mapping stored in the database. As in the first scenario the IP address/MAC address mapping is also temporarily stored in the TransientIpDb component for the time the corresponding IP address is actually assigned to the client. Calling the DhcpServer:IManagement.UsePermanentIdDatabase method activates this behavior.

**Event: A new client connects to the Airport WiFi network**

- The Client sends a DHCP request ① for an IP address. This request is accepted by the DhcpServer:DhcpListener component.
- The DhcpListener component calls the IpAddressManager.RequestNewIpAddress method ①.
- The IpAddressManager component determines a IP address for the new client – this action is different for each of the DhcpServer usage scenarios:
  1. IpAddressManager tries to generate a new IP address and checks whether it has been already assigned – by calling the IIpMacTransientDb.GetMacAddress ①.② method. If the generated IP address is already used it will generate another one and repeat the check.
  2. IpAddressManager calls the IIpAddressPermanentDb.GetIpAddress method to check if a mapping for client's MAC address exists. If the mapping is not found the IpAddressManager can try to generate an automatic IP address as in the previous scenario.

- The IP address with client's MAC address is then added by the IpAddressManager to the TransientIpDb database by calling it's IIpMacDb.Add (1.3) method.
- The assigned IP address is returned to the client via returning the IDhcpListenerCallback.RequestNewIpAddress method call (1.1) from IpAddressManager component back to the DhcpListener. The assigned IP address is then used as a unique client identifier by the rest of the system.
- The Client can enter the Login page ② (by typing any URL in the web browser)

**Event: A client disconnects from the Airport WiFi network (17 – Client disconnects):**

- The client disconnection event can occur, or be detected in one of the two ways:
  - d) If, after certain amount of time, the Client does not send the "Renew IP address" DHCP request to the IpAddressManager component (via DhcpListerner:IDhcpListernerCallback.RenewIpAddress method call) the present timer will expire calling the IpAddressManager:TimerCallback.Timeout method.
  - e) The WiFi Access Point detects client's disconnection from the WiFi network and notifies the DhcpServer:DhcpListener component of that event. The DhcpListerner will call the IpAddressManager:IDhcpListenerCallback.ReleaseIpAddress method (17.1).
- The IpAddressManager will the remove the IP address/MAC address mapping from the TransientIpDb database component by calling it's IIpMacTransientDb.Remove method (17.2).
- The IpAddressManager will notify the rest of the system of client disconnection by calling the IDhcpCallback.IpAddressInvalidated (17.3) method.
- That will lead into calling Abitrator:IDhcpCallback.IpAddressInvalidated method (18) that will start the process of terminating current Client's session.

# 3. Demo Components Specification

## 3.1. Interfaces Types

```
interface IFlyTicketDb {
        java.util.Date GetFlyTicketValidity(String FlyTicketId) throws InvalidFlyTicketIdException;
        FlyTicket[] GetFlyTicketsByFrequentFlyerId(String FrequentFlyerId) throws InvalidFrequentFlyerIdException;
}

interface IFlyTicketAuth {
        Token CreateToken(String FlyTicketId) throws InvalidFlyTicketIdException;
}

interface IFrequentFlyerAuth {
        Token CreateToken(String FrequentFlyerId) throws InvalidFrequentFlyerIdException;
}

interface IToken {
        boolean InvalidateAndSave();
}

interface ICustomCallback {
        boolean InvalidatingToken(TimeSpan TimeLeft);
}

interface ITokenCallback {
        void TokenInvalidated(Token InvalidatedToken);
}

inteface IAccount {
        String GenerateRandomAccountId();
        boolean CreateAccount(String AccountId, String Password);
        boolean RechargeAccount(String AccountId, String CreditCardId, java.util.Date CreditCardExpirationDate, TimeSpan PrepaidTime);
        void AjustAccountPrepaidTime(String AccondId, String SecurityCookie, TimeSpan PrepaidTime)
                                        throws InvalidAccountCredentialsException;
}

interface IAccountAuth {
        Token CreateToken(String AccountId, String Password);
}

interface ICardCenter {
        boolean Withdraw(String CreditCardId, java.util.Date CreditCardExpirationDate, java.math.BigDecimal Amount)
                        throws InvalidCreditCardIdException, CreditCardExpiredException;
}

interface ILogin {
        String GetTokenIdFromIpAddress(java.net.InetAddress IpAddress);
        String LoginWithFlyTicketId(String FlyTicketId);
        String LoginWithFrequentFlyerId(String FrequentFlyerId);
        String LoginWithAccountId(String AccountId, String Password);
        void Logout(java.net.InetAddress IpAddress);
}

interface IFirewall {
        boolean DisablePortBlock(java.net.InetAddress IpAddress);
```

```
        boolean EnablePortBlock(java.net.InetAddress IpAddress);
}

interface ITimer {
        void SetTimeout(java.util.Date Timeout);
        void CancelTimeout();
}

interface ITimerCallback {
        void Timeout()
}

interface IDhcpCallback {
        void IpAddressInvalidated(java.net.InetAddress IpAddress);
}

interface IDhcpListenerCallback {
        java.net.InetAddress RequestNewIpAddress(byte[] MacAddress);
        boolean RenewIpAddress(byte[] MacAddress, java.net.InetAddress IpAddress);
        boolean ReleaseIpAddress(byte[] MacAddress, java.net.InetAddress IpAddress);
}

interface IIpMacDb {
        void Add(byte[] MacAddress, java.net.InetAddress IpAddress, java.util.Date ExpirationTime);
        void Remove(java.net.InetAddress IpAddress);
        byte[] GetMacAddress(java.net.InetAddress IpAddress) throws AddressNotRegisteredException;
        java.net.InetAddress GetIpAddress(byte[] MacAddress) throws AddressNotRegisteredException;
        java.util.Date GetExpirationTime(java.net.InetAddress IpAddress) throws AddressNotRegisteredException;
        void SetExpirationTime(java.net.InetAddress IpAddress, java.util.Date ExpirationTime)
                                throws AddressNotRegisteredException;
}

interface IManagement {
        void UsePermanentIpDatabase();
        void StopUsingPermanentIpDatabase();
        void StopRenewingPermanentIpAdrresses();
}

interface ILock {
        void Lock();
        void Unlock();
}
```

## 3.2. Fractal Components

**FlyTicketDatabase composite component**

- provides IFlyTicketDb (IFlyTicketDb interface type)
- provides IFlyTicketAuth (IFlyTicketAuth interface type)

behavior protocol:

```
(
    ?IFlyTicketAuth.CreateToken + ?IFlyTicketDb.GetFlyTicketsByFrequentFlyerId
)*
```

→ **AfDbConnection component**

- provides IFlyTicketDb (IFlyTicketDb interface type)

behavior protocol:

```
(
    ?IFlyTicketDb.GetFlyTicketValidity + ?IFlyTicketDb.GetFlyTicketsByFrequentFlyerId
)*
```

→ **CsaDbConnection component**

- provides IFlyTicketDb (IFlyTicketDb interface type)

behavior protocol:

```
(
    ?IFlyTicketDb.GetFlyTicketValidity + ?IFlyTicketDb.GetFlyTicketsByFrequentFlyerId
)*
```

→ **FlyTicketClassifier component**

- provides IFlyTicketDb (IFlyTicketDb interface type)
- provides IFlyTicketAuth (IFlyTicketAuth interface type)
- requires IOutFlyTicketDb (IFlyTicketDb interface type)
- requires ICsaFlyTicketDb (IFlyTicketDb interface type)

behavior protocol:

```
(
    ?IFlyTicketDb.GetFlyTicketsByFrequentFlyerId {
        !IAfFlyTicketDb.GetFlyTicketsByFrequentFlyerId
        ||
        !ICsaFlyTicketDb.GetFlyTicketsByFrequentFlyerId
    }
    +
    ?IFlyTicketAuth.CreateToken {
        !IAfFlyTicketDb.GetFlyTicketValidity
        +
        !ICsaFlyTicketDb.GetFlyTicketValidity
    }
)*
```

**FrequentFlyerDatabase**

- provides IFrequentFlyerAuth (IFrequentFlyerAuth interface type)
- requires IFlyTicketDb (IFlyTicketDb interface type)
- requires IFlyTicketAuth (IFlyTicketAuth interface type)

behavior protocol:

```
(
    ?IFrequentFlyerAuth.CreateToken {
        (
            !IFlyTicketDb.GetFlyTicketsByFrequentFlyerId ;
            (!IFlyTicketAuth.CreateToken + NULL)
        )
        +
        NULL
    }
)*
```

**Token composite component**

- provides IToken (IToken interface type)
- requires ITokenCallback (ITokenCallback interface type)
- *optional* requires IAccount (IAccount interface type)

behavior protocol:

```
?ILifetimeController.Start
;
(
    ?IToken.InvalidateAndSave {
        (!IAccount.AjustAccountPrepaidTime_1 + NULL);
        !ITokenCallback.TokenInvalidated_1
    }*
    |
    (
        (!IAccount.AjustAccountPrepaidTime_2 + NULL);
        !ITokenCallback.TokenInvalidated_2
    )*
)
```

→ **AccountCustomToken *optional* component**

- attribute SecurityCookie
- provides ICustomCallback (ICustomCallback interface type)
- requires IAccount (IAccount interface type)

behavior protocol:

```
(
    ?ICustomCallback.InvalidatingToken_1 {
        !IAccount.AjustAccountPrepaidTime_1
    }*
    |
    ?ICustomCallback.InvalidatingToken_2 {
        !IAccount.AjustAccountPrepaidTime_2
    }*
)
```

→ **ValidityChecker component**

- provides ITimerCallback (ITimerCallback interface type)
- provides IToken (IToken interface type)
- requires ITimer (ITimer interface type)
- requires ITokenCallback (ITokenCallback interface type)
- *optional* requires ICustomCallback (ICustomCallback interface type)

behavior protocol:

```
(
    ?ILifetimeController.Start^ ;
    !ITimer.SetTimeout_1^ ;
    [?ITimer.SetTimeout_1$, !ILifetimeController.Start$]
)
;
(
    ?IToken.InvalidateAndSave {
        (!ICustomCallback.InvalidatingToken_1 + NULL);
        !ITimer.CancelTimeout;
        !ITokenCallback.TokenInvalidated_1
    }*
    |
    ?ITimerCallback.Timeout {
        (!ICustomCallback.InvalidatingToken_2 + NULL);
        !ITokenCallback.TokenInvalidated_2
    }*
)
```

**Arbitrator component**

- provides ILogin (ILogin interface type)
- provides ITokenCallback (ITokenCallback interface type)
- provides IDhcpCallback (IDhcpCallback interface type)
- requires IFrequentFlyerAuth (IFrequentFlyerAuth interface type)
- requires IFlyTicketAuth (IFlyTicketAuth interface type)
- requires IAccountAuth (IAccountAuth interface type)
- requires IFirewall (IFirewall interface type)
- requires *collection* of IToken (IToken interface type)

behavior protocol:

```
(
    ?IArbitratorLifetimeController.Start^ ;
    !ITokenLifetimeController.Start^ ;
    [?ITokenLifetimeController.Start$, !IArbitratorLifetimeController.Start$]
)
;
(
    (
        ?ILogin.GetTokenIdFromIpAddress
        +
        ?ILogin.LoginWithFlyTicketId {
            !IFlyTicketAuth.CreateToken ;
            (!IFirewall.DisablePortBlock + NULL)
        }
        +
        ?ILogin.LoginWithFrequentFlyerId {
            !IFreqFlyerAuth.CreateToken ;
            (!IFirewall.DisablePortBlock + NULL)
        }
        +
        ?ILogin.LoginWithAccountId {
            !IAccountAuth.CreateToken ;
            (!IFirewall.DisablePortBlock + NULL)
        }
        +
        ?ILogin.Logout {
            !IToken.InvalidateAndSave_1
        }
    )*
    |
    ?ITokenCallback.TokenInvalidated_1 {
        !IFirewall.EnablePortBlock_1
    }*
    |
    ?ITokenCallback.TokenInvalidated_2 {
        !IFirewall.EnablePortBlock_2
    }*
    |
    ?ITokenCallback.TokenInvalidated_3 {
        !IFirewall.EnablePortBlock_3
    }*
    |
    ?IDhcpCallback.IpAddressInvalidated_1 {
        !IToken.InvalidateAndSave_2
    }*
)
```

**AccountDatabase component**

- provides IAccount (IAccount interface type)
- provides IAccountAuth (IAccountAuth interface type)
- requires ICardCenter (ICardCenter interface type)

behavior protocol:

```
(
    (
        ?IAccount.GenerateRandomAccountId
        +
        ?IAccount.CreateAccount
        +
        ?IAccount.RechargeAccount {
            !ICardCenter.Withdraw
        }
    )*
    |
    ?IAccount.AjustAccountPrepaidTime_1*
    |
    ?IAccount.AjustAccountPrepaidTime_2*
    |
    ?IAccount.AjustAccountPrepaidTime_3*
)
```

**CardCenter component**

- provides ICardCenter (ICardCenter interface type)

behavior protocol:

```
(
    ?ICardCenter.Withdraw*
)
```

**Firewall component**

- provides IFirewall (IFirewall interface type)

behavior protocol:

```
(
    ?IFirewall.EnablePortBlock_1*
    |
    ?IFirewall.EnablePortBlock_2*
    |
    ?IFirewall.EnablePortBlock_3*
    |
    ?IFirewall.DisablePortBlock*
)
```

**DhcpServer composite component**

- provides IManagementIn (IManagement interface type)
- *optional* requires IDhcpCallback (IDhcpCallback interface type)
- *optional* requires IIpMacPermanentDb (IIpMacDb interface type)

behavior protocol:

```
(
    !IDhcpCallback.IpAddressInvalidated_1*
    |
    !IDhcpCallback.IpAddressInvalidated_2*
    |
    (
        ?IManagementIn.UsePermanentIpDatabase^ ; (
            !IIpMacPermanentDb.GetIpAddress*
            |
            (!IManagementIn.UsePermanentIpDatabase$ ; ?IManagementIn.StopUsingPermanentIpDatabase^)
        ) ; !IManagementIn.StopUsingPermanentIpDatabase$
    )*
)
```

- note: The component is too “smart” for the demo, i.e. the demo does not use all the functionality the DhcpServer component provides. The IManagement interface is not bound in the demo environment and also the optional IIpMacPermanentDb interface does not need to be bound.
- subcomponents:

→ **TransientIpDb component**

- provides IIpMacTransientDb (IIpMacDb interface type)

behavior protocol:

```
(
    (
        ?IIpMacTransientDb.Add_1 +
        ?IIpMacTransientDb.Remove_1 +
        ?IIpMacTransientDb.GetMacAddress_1 +
        ?IIpMacTransientDb.GetIpAddress_1 +
        ?IIpMacTransientDb.GetExpirationTime_1 +
        ?IIpMacTransientDb.SetExpirationTime_1
    )*
    |
    (
        ?IIpMacTransientDb.Add_2 +
        ?IIpMacTransientDb.Remove_2 +
        ?IIpMacTransientDb.GetMacAddress_2 +
        ?IIpMacTransientDb.GetIpAddress_2 +
        ?IIpMacTransientDb.GetExpirationTime_2 +
        ?IIpMacTransientDb.SetExpirationTime_2
    )*
)
```

→ **DhcpListerner component**

- requires IDhcpListenerCallback (IDhcpListnerCallback interface type)

behavior protocol:

```
(
    !IDhcpListenerCallbackIn.RequestNewIpAddress +
    !IDhcpListenerCallbackIn.RenewIpAddress +
    !IDhcpListenerCallbackIn.ReleaseIpAddress
)*
```

→ **IpAddressManager component**

- provides IManagementOut (IManagement interface type)
- provides IDhcpListenerCallbackOut (IDhcpListnerCallback interface type)
- provides ITimerCallback (ITimerCallback interface type)
- requires ITimer (ITimer interface type)
- requires IIpMacTransientDb (IIpMacDb interface type)
- *optional* requires IIpMacPermanentDb (IIpMacDb interface type)
- *optional* requires IDhcpCallback (IDhcpCallback interface type)

behavior protocol:

```
(
    (
        (
            ?IDhcpListenerCallbackOut.RequestNewIpAddress {
                (!IIpMacTransientDb.GetIpAddress_1 + NULL) ;
                !IIpMacTransientDb.GetMacAddress_1* ;
                ((!IIpMacTransientDb.Add_1 ; (!ITimer.SetTimeout_1 + NULL)) + NULL)
            }
            +
            ?IDhcpListenerCallbackOut.RenewIpAddress {
                !IIpMacTransientDb.GetMacAddress_1 ; (
                    (
                        !IIpMacTransientDb.SetExpirationTime_1 ;
                        (!ITimer.SetTimeout_1 + NULL)
                    )
                    +
                    NULL
                )
            }
            +
            ?IDhcpListenerCallbackOut.ReleaseIpAddress {
                !IIpMacTransientDb.Remove_1 ; !IIpMacTransientDb.GetExpirationTime_1* ;
                (!ITimer.CancelTimeout + !ITimer.SetTimeout_1 + NULL) ;
                (!IDhcpCallback.IpAddressInvalidated_1 + NULL)
            }
        )*
        |
        (
            ?ITimerCallback.Timeout {
                !IIpMacTransientDb.Remove_2 ; !IIpMacTransientDb.GetExpirationTime_2* ;
                (!ITimer.SetTimeout_2 + NULL) ;
                (!IDhcpCallback.IpAddressInvalidated_2 + NULL)
            }
        )*
    )
    +
    (
        (
```

```
        (
            (
                ?IDhcpListenerCallbackOut.RequestNewIpAddress {
                    (!IIpMacTransientDb.GetIpAddress_1 + NULL) ;
                    !IIpMacTransientDb.GetMacAddress_1* ;
                    ((!IIpMacTransientDb.Add_1 ; (!ITimer.SetTimeout_1 + NULL)) + NULL)
                }
                +
                ?IDhcpListenerCallbackOut.RenewIpAddress {
                    !IIpMacTransientDb.GetMacAddress_1 ; (
                        (
                            !IIpMacTransientDb.SetExpirationTime_1 ;
                            (!ITimer.SetTimeout_1 + NULL)
                        )
                        +
                        NULL
                    )
                }
                +
                ?IDhcpListenerCallbackOut.ReleaseIpAddress {
                    !IIpMacTransientDb.Remove_1 ; !IIpMacTransientDb.GetExpirationTime_1* ;
                    (!ITimer.CancelTimeout + !ITimer.SetTimeout_1 + NULL) ;
                    (!IDhcpCallback.IpAddressInvalidated_1 + NULL)
                }
            )*
            |
            (
                ?ITimerCallback.Timeout {
                    !IIpMacTransientDb.Remove_2 ; !IIpMacTransientDb.GetExpirationTime_2* ;
                    (!ITimer.SetTimeout_2 + NULL) ;
                    (!IDhcpCallback.IpAddressInvalidated_2 + NULL)
                }
            )*
        )
        |
        ?IManagementOut.UsePermanentIpDatabase^
    ) ; !IManagementOut.UsePermanentIpDatabase$ ; (
        (
            (
                ?IDhcpListenerCallbackOut.RequestNewIpAddress  {
                    !IIpMacPermanentDb.GetIpAddress ; (
                        (!IIpMacTransientDb.Add_1 ; (!ITimer.SetTimeout_1 + NULL))
                        +
                        (
                            !IIpMacTransientDb.GetMacAddress_1* ; (
                                (
                                    !IIpMacTransientDb.Add_1 ;
                                    (!ITimer.SetTimeout_1 + NULL)
                                )
                                +
                                NULL
                            )
                        )
                    )
                }
                +
                ?IDhcpListenerCallbackOut.RenewIpAddress {
                    !IIpMacTransientDb.GetMacAddress_1 ; (
                        (
                            !IIpMacTransientDb.SetExpirationTime_1 ;
                            (!ITimer.SetTimeout_1 + NULL)
                        )
                        +
                        NULL
                    )
                }
                +
                ?IDhcpListenerCallbackOut.ReleaseIpAddress {
                    !IIpMacTransientDb.Remove_1 ; !IIpMacTransientDb.GetExpirationTime_1* ;
                    (!ITimer.CancelTimeout + !ITimer.SetTimeout_1 + NULL) ;
                    (!IDhcpCallback.IpAddressInvalidated_1 + NULL)
                }
            )*
            |
            (
                ?ITimerCallback.Timeout {
                    !IIpMacTransientDb.Remove_2 ; !IIpMacTransientDb.GetExpirationTime_2* ;
                    (!ITimer.SetTimeout_2 + NULL) ;
                    (!IDhcpCallback.IpAddressInvalidated_2 + NULL)
                }
```

```
                    )*
                )
                |
                ?IManagementOut.StopUsingPermanentIpDatabase^
            ) ; !IManagementOut.StopUsingPermanentIpDatabase$
        )
    )*
```

→ **Timer component**

- provides ITimer (ITimer inteface type)
- requires ITimerCallback (ITimerCallback interface type)

behavior protocol:

```
(
    (
        ?ITimer.SetTimeout_1 ;
        ?ITimer.SetTimeout_1* ;
        (
            (
                (
                    !ITimerCallback.Timeout
                    |
                    (
                        ?ITimer.SetTimeout_1 ;
                        ?ITimer.SetTimeout_1*
                    )
                )
                +
                (
                    !ITimerCallback.Timeout {
                        ?ITimer.SetTimeout_2 ;
                        ?ITimer.SetTimeout_2*
                    }
                    |
                    (
                        ?ITimer.SetTimeout_1*
                    )
                )
            )*
            ;
            !ITimerCallback.Timeout
        )
    )
    |
    ?ITimer.CancelTimeout*
)
```

## 3.3. Synchronization Components

The *virtual* synchronization components S1 to S5 won't be real Fractal components implemented as part of the demo. Their purpose is only to easily describe and easily autogenerate method call synchronization description in behavior protocols. These autogenerated components, more precisely only their behavior protocols, will be added in between real Fractal components during protocol checking.

**S1 – first synchronization component**

- provides IDhcpListenerCallbackIn (IDhcpListenerCallback interface type)
- requires IDhcpListenerCallbackOut (IDhcpListenerCallback interface type)
- requires IS2 (ILock interface type)

- The first synchronization component forwards the call on "In" interface to "Out" interface after it successfully locks the next synchronization component – i.e. after the INextLock.Lock call returns. When the call on the "Out" interface is finished the component unlocks the next synchronization component via INextLock.Unlock call.

```
(
    ?IDhcpListenerCallbackIn.RequestNewIpAddress {!IS2.Lock; !IDhcpListenerCallbackOut.RequestNewIpAddress; !IS2.Unlock}
)*
```

**Si – intermediate synchronization component (synchronization components S2 to S5)**

- provides IDhcpListenerCallbackIn (IDhcpListenerCallback interface type) (IManagement for components S4 and S5)
- provides ISi (ILock interface type)
- requires IDhcpListenerCallbackOut (IDhcpListenerCallback interface type) (IManagement for components S4 and S5)
- requires ISi+1 (ILock interface type)

- Each of the intermediate synchronization component processes one call from IDhcpListernerCallback (S2, S3) or IManagement (S4, S5) interfaces. If the previous component tries to lock an intermediate component (via IPrevLock.Lock call), it will try to lock the next component via INextLock.Lock method call. If the component accepts call from the "In" interface either it locks the next synchronization component and processes the call by calling the appropriate method on the "Out" interface and then unlocks the next component; or, if it has been already locked by the previous component, it will postpone the "Out" method call until it accepts the IPrevLock.Unlock call. Then it calls the

appropriate method on the “Out” interface (note that the next synchronization component is still locked) and only after this call returns it will unlock both the next component and itself (by returning the IPrevLock.Unlock method call).

- The following behavior protocol describes the S2 synchronization component. Behavior protocols for S3 to S5 components are similar, only the IDhcpListenerCallbackIn.RenewIpAddress and IDhcpListenerCallbackOut.RenewIpAddress method calls are replaced by a method name, that the synchronization component processes.

**S2 – intermediate synchronization component**

```
(
    (
        (
            ( ( ?IDhcpListenerCallbackIn.RenewIpAddress^; !S3.Lock; !IDhcpListenerCallbackOut.RenewIpAddress ) | ?S2.Lock^ )
            +
            ( ( ( ?S2.Lock^; !S3.Lock) | ?IDhcpListenerCallbackIn.RenewIpAddress^ ) ; !IDhcpListenerCallbackOut.RenewIpAddress )
        ) ;
        !S2.Lock$; ?S2.Unlock^; !S3.Unlock; [!S2.Unlock$, !IDhcpListenerCallbackIn.RenewIpAddress$]
    )
    +
    (
        ?S2.Lock {!S3.Lock}; ( ?IDhcpListenerCallbackIn.RenewIpAddress^ | ?S2.Unlock^ );
        !IDhcpListenerCallbackOut.RenewIpAddress; !S3.Unlock; [!S2.Unlock$, !IDhcpListenerCallbackIn.RenewIpAddress$]
    )
    +
    (
        ?S2.Lock {!S3.Lock}; ?S2.Unlock^; !S3.Unlock^; ( ?IDhcpListenerCallbackIn.RenewIpAddress^ | ?S3.Unlock$ );
        !S3.Lock; !IDhcpListenerCallbackOut.RenewIpAddress; !S3.Unlock; [!S2.Unlock$, !IDhcpListenerCallbackIn.RenewIpAddress$]
    )
    +
    (
        ?S2.Lock {!S3.Lock}; ?S2.Unlock {!S3.Unlock}
    )
    +
    (
        ?IDhcpListenerCallbackIn.RenewIpAddress^; !S3.Lock; !IDhcpListenerCallbackOut.RenewIpAddress;
        !S3.Unlock^; ( ?S2.Lock^ | ?S3.Unlock$ );
        !S3.Lock; !S2.Lock$; ?S2.Unlock^; !S3.Unlock; [!S2.Unlock$, !IDhcpListenerCallbackIn.RenewIpAddress$]
    )
    +
    (
        ?IDhcpListenerCallbackIn.RenewIpAddress {!S3.Lock; !IDhcpListenerCallbackOut.RenewIpAddress; !S3.Unlock}
    )
)*
```

**S3 – intermediate synchronization component**

```
(
    (
        (
            ( ( ?IDhcpListenerCallbackIn.ReleaseIpAddress^; !S4.Lock; !IDhcpListenerCallbackOut.ReleaseIpAddress ) | ?S3.Lock^ )
            +
            ( ( ( ?S3.Lock^; !S4.Lock) | ?IDhcpListenerCallbackIn.ReleaseIpAddress^ ) ; !IDhcpListenerCallbackOut.ReleaseIpAddress )
        ) ;
        !S3.Lock$; ?S3.Unlock^; !S4.Unlock; [!S3.Unlock$, !IDhcpListenerCallbackIn.ReleaseIpAddress$]
    )
    +
    (
        ?S3.Lock {!S4.Lock}; ( ?IDhcpListenerCallbackIn.ReleaseIpAddress^ | ?S3.Unlock^ );
        !IDhcpListenerCallbackOut.ReleaseIpAddress; !S4.Unlock; [!S3.Unlock$, !IDhcpListenerCallbackIn.ReleaseIpAddress$]
    )
    +
    (
        ?S3.Lock {!S4.Lock}; ?S3.Unlock^; !S4.Unlock^; ( ?IDhcpListenerCallbackIn.ReleaseIpAddress^ | ?S4.Unlock$ );
        !S4.Lock; !IDhcpListenerCallbackOut.ReleaseIpAddress; !S4.Unlock; [!S3.Unlock$, !IDhcpListenerCallbackIn.ReleaseIpAddress$]
    )
    +
    (
        ?S3.Lock {!S4.Lock}; ?S3.Unlock {!S4.Unlock}
    )
    +
    (
        ?IDhcpListenerCallbackIn.ReleaseIpAddress^; !S4.Lock; !IDhcpListenerCallbackOut.ReleaseIpAddress;
        !S4.Unlock^; ( ?S3.Lock^ | ?S4.Unlock$ );
        !S4.Lock; !S3.Lock$; ?S3.Unlock^; !S4.Unlock; [!S3.Unlock$, !IDhcpListenerCallbackIn.ReleaseIpAddress$]
    )
    +
    (
        ?IDhcpListenerCallbackIn.ReleaseIpAddress {!S4.Lock; !IDhcpListenerCallbackOut.ReleaseIpAddress; !S4.Unlock}
    )
)*
```

**S4 – intermediate synchronization component**

```
(
    (
        (
            ( ( ?IManagementIn.UsePermanentIpDatabase^; !S5.Lock; !IManagementOut.UsePermanentIpDatabase ) | ?S4.Lock^ )
            +
            ( ( ( ?S4.Lock^; !S5.Lock) | ?IManagementIn.UsePermanentIpDatabase^ ) ; !IManagementOut.UsePermanentIpDatabase )
        ) ;
        !S4.Lock$; ?S4.Unlock^; !S5.Unlock; [!S4.Unlock$, !IManagementIn.UsePermanentIpDatabase$]
    )
    +
    (
        ?S4.Lock {!S5.Lock}; ( ?IManagementIn.UsePermanentIpDatabase^ | ?S4.Unlock^ );
        !IManagementOut.UsePermanentIpDatabase; !S5.Unlock; [!S4.Unlock$, !IManagementIn.UsePermanentIpDatabase$]
    )
    +
    (
        ?S4.Lock {!S5.Lock}; ?S4.Unlock^; !S5.Unlock^; ( ?IManagementIn.UsePermanentIpDatabase^ | ?S5.Unlock$ );
        !S5.Lock; !IManagementOut.UsePermanentIpDatabase; !S5.Unlock; [!S4.Unlock$, !IManagementIn.UsePermanentIpDatabase$]
    )
    +
    (
        ?S4.Lock {!S5.Lock}; ?S4.Unlock {!S5.Unlock}
    )
    +
    (
        ?IManagementIn.UsePermanentIpDatabase^; !S5.Lock; !IManagementOut.UsePermanentIpDatabase;
        !S5.Unlock^; ( ?S4.Lock^ | ?S5.Unlock$ );
        !S5.Lock; !S4.Lock$; ?S4.Unlock^; !S5.Unlock; [!S4.Unlock$, !IManagementIn.UsePermanentIpDatabase$]
    )
    +
    (
        ?IManagementIn.UsePermanentIpDatabase {!S5.Lock; !IManagementOut.UsePermanentIpDatabase; !S5.Unlock}
    )
)*
```

**S5 – last synchronization component**

- provides IManagementIn (IManagement interface type)
- provides IS5 (ILock interface type)
- requires IManagementOut (IManagement interface type)

- If the last synchronization component accepts the IManagementIn.StopUsingPermanentIpAddresses method call it can immediately forward the call to the “Out” interface (if it was not locked in the past). If it accepts the IPrevLock.Lock call it will only postpone the “In” interface call until the IPrevLock.Unlock call arrives.

```
(
    (
        ( ?S5.Lock^ | ( ?IManagementIn.StopUsingPermanentIpDatabase^; !IManagementOut.StopUsingPermanentIpDatabase ) );
        !S5.Lock$; ?S5.Unlock^; [!S5.Unlock$, !IManagementIn.StopUsingPermanentIpDatabase$]
    )
    +
    (
        ?S5.Lock; ( ?IManagementIn.StopUsingPermanentIpDatabase^ | ?S5.Unlock^ );
        !IManagementOut.StopUsingPermanentIpDatabase; [!S5.Unlock$, !IManagementIn.StopUsingPermanentIpDatabase$] )
    +
    (
        ?S5.Lock; ?S5.Unlock
    )
    +
    (
        ?IManagementIn.StopUsingPermanentIpDatabase {!IManagementOut.StopUsingPermanentIpDatabase}
    )
)*
```