Universal Reference Design Review

John Ujhazy

jujhazy@eos.hitc.com

17 April 1996

Overview

- UR Requirements
- Design Overview
- OO Design Model
- Dynamic Models
- Pseudocode
- UR Structure

UR Driving Requirements

- Provide a mechanism to access ECS object instances through logical references
- Access object instances which can exist anywhere in the system
- Object instances can be any ECS object
- Encapsulation of identifier specific to particular object classes
- Accommodate nesting of identifiers
  - For example Advertising Service providing references to data/services
- Support multiple formats
  - C++ objects and ASCII string
- Infinite persistence not required
  - If object no longer exists, UR is invalid
- Unique within application types across DAACs

UR Solution Overview

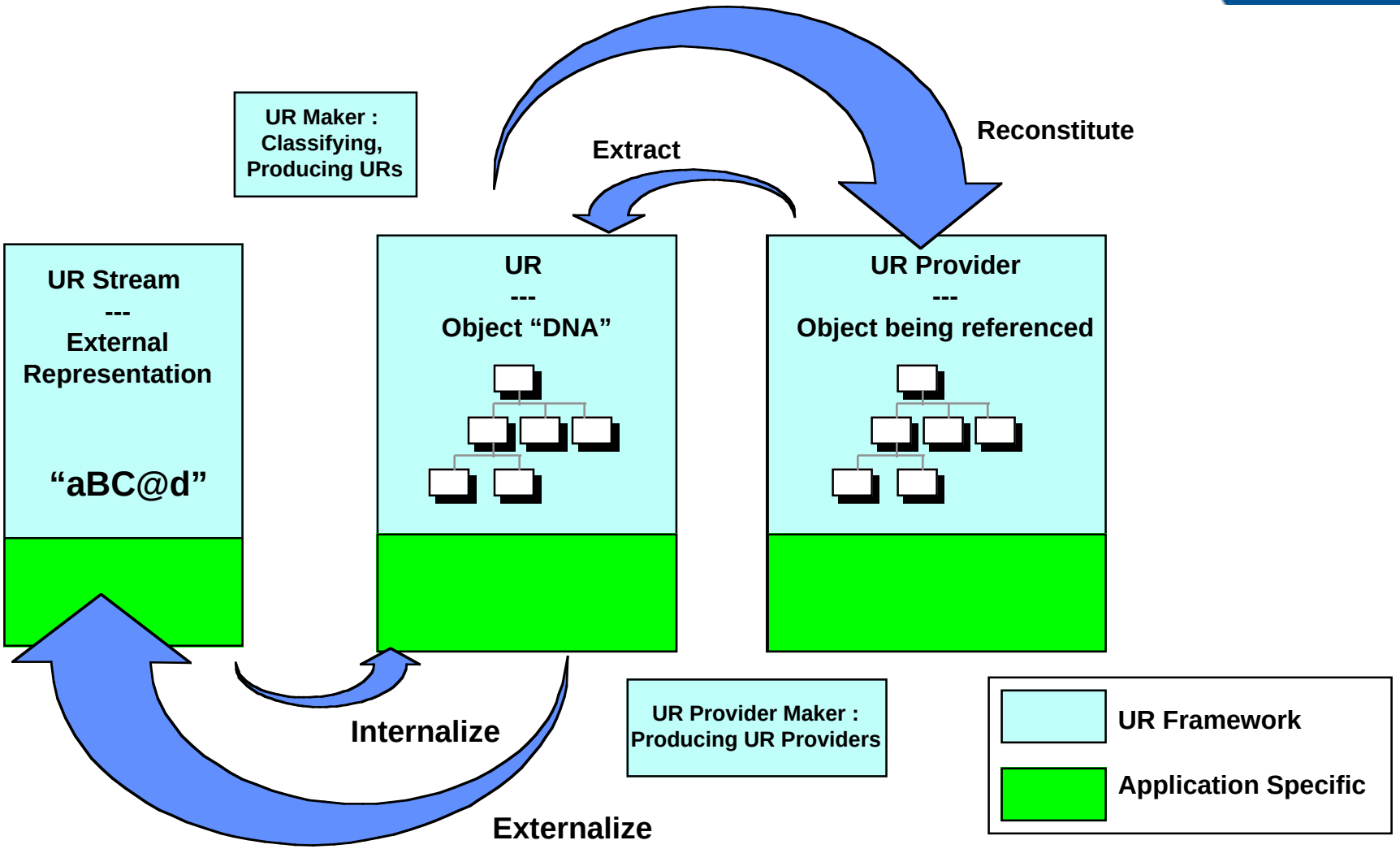
- **Release B Design**
  - **Reuse of Release A UR Framework**
  - **Current and developing standards (ISO DOR, URL, IETF URC) insufficient/too late for ECS use**
  - **Principles are based on established design patterns (Memento and Object Factory)**

- **Framework Features**
  - **Objects create URs for themselves**
  - **URs used to reconstitute/access original object**
  - **Content of URs is specific to the class of object**
  - **Supports evolution**

    Allows abstraction and inheritance to accommodate new objects

- **Design Detailed in Release B System Design Specification, 305-CD-028-002**

UR Solution Overview



UR Framework Context

Who Uses the UR Framework?

- **Earth Science Data/Services e.g., EDSTs**
- **Clients connecting to servers e.g., Sessions**
- **Advertisements**
- **Subscriptions**
- **Subscription Events**
- **Server/Request Framework (SRF) e.g., Clients and Servers**

UR Framework Design Overview

UR Framework supports three categories of classes

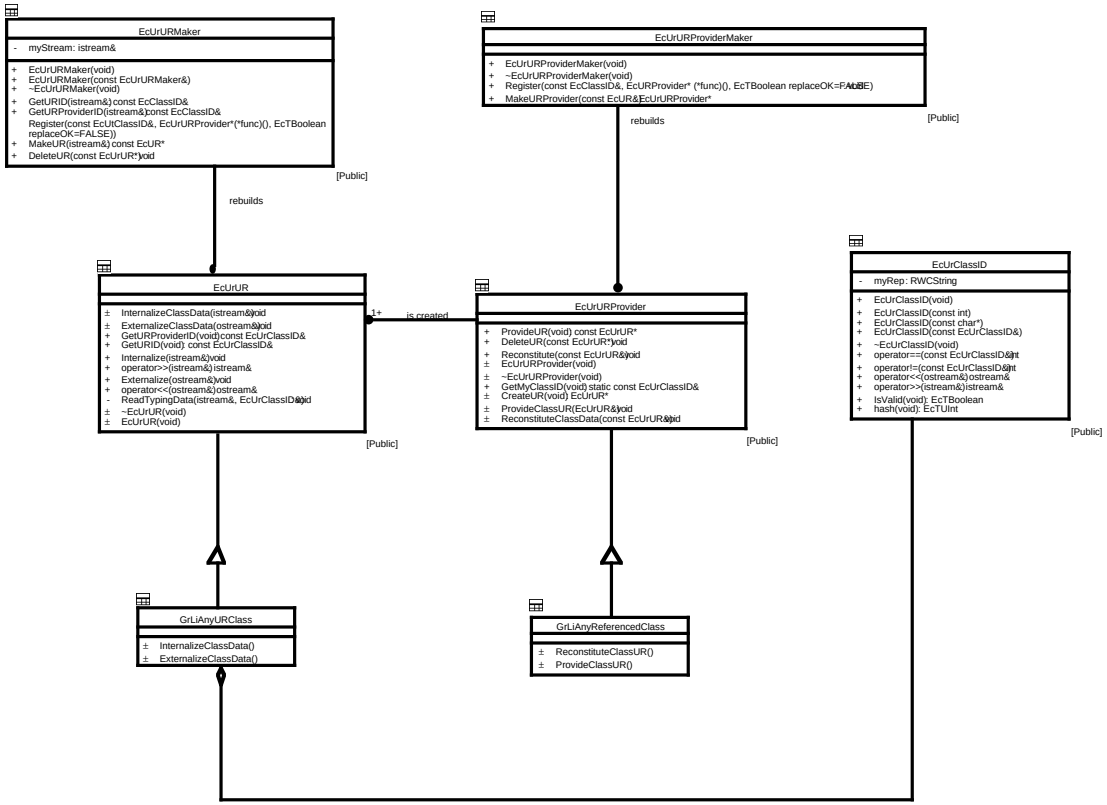
- UR Classes
  - Used to manage the URs themselves
  - Parent Class specialized by developers for their UR classes
- UR Provider Classes
  - Used by Classes providing URs (UR referents)
  - Parent Class specialized by developers for their UR Provider (UR referents) Classes
- UR Utility Classes
  - Used by UR and UR Provider Classes

Object Model

The following object model will be reviewed:

Model Name	Document Reference
EcUrMain	305-CD-028-002, Section 4, Figure 4.5.4.3.-1

UR Object Model



UR Framework Classes

UR Classes

- EcUrUR The abstract base class for all Universal Reference (UR)s. Must be specialized by developers to provide UR classes which are application specific.
- GrLiAnyUR Example of a specialization of the EcUrUR class. Any class referenced by a UR must define its own UR, which is a specialization of the EcUrUR class.

UR Provider Classes

- EcUrURProvider The abstract base class for all data and service objects referred to by Universal Reference (UR)s. Must be specialized by developers to provide specific provider classes.
- GrLiAnyReferencedClass Example of a specialization of the EcUrURProvider class. Any class referenced by a UR must have its own specialization of the EcUrURProviders class.

-

UR Framework Classes

UR Utility Classes

- <u>EcUrURMaker</u> An object factory for URs. Provides several capabilities:
  - Registering of UR object classes,
  - Creation of URs based on registered object class type,
  - Read streams with URs and rebuild the UR objects.

- <u>EcUrURProviderMaker</u> An object factory for UR providers. Provides capability to register data and services object classes, and rebuilds UR provider objects using URs.

- <u>EcUrClassID</u> This class supports a mechanism for comparing two instances of this class, and the ability to read/write itself from streams.

Dynamic Model

The following event traces will be reviewed:

Event Trace Name	Reference
Creating and Externalizing a UR	305-CD-028-002, Section 4, Figure 4.5.4.5-1
Internalizing and Reconstituting a UR	305-CD-028-002, Section 4, Figure 4.5.4.5-2

Event Traces

Name: Creating and Externalizing a UR

Scenario

- **A UR provider provides an ASCII version of a UR to itself**

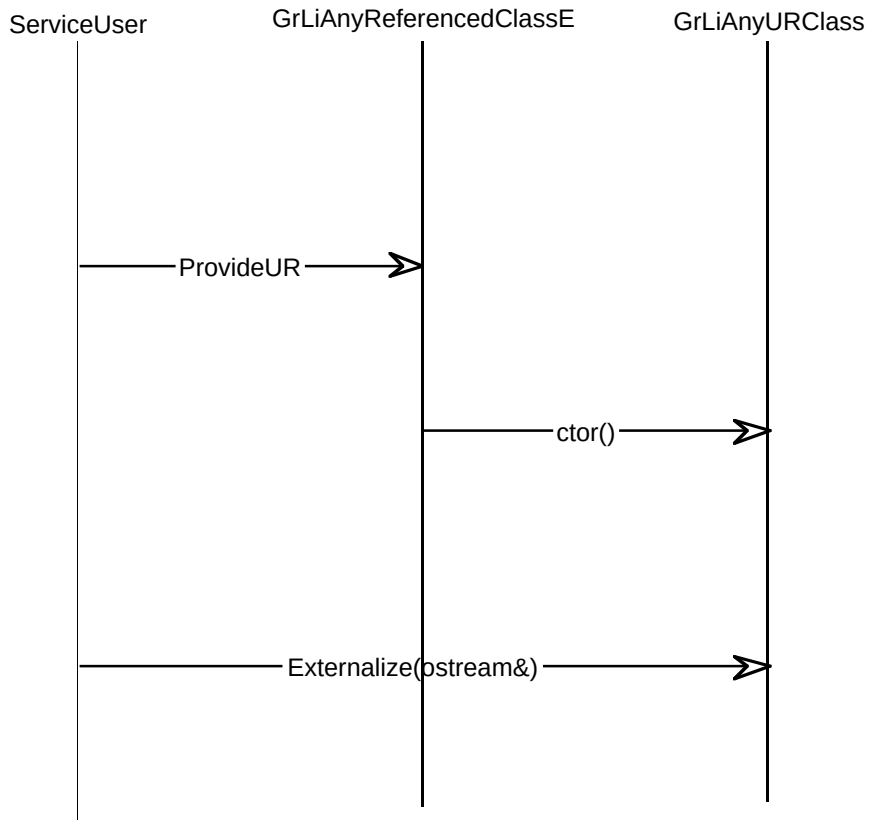
Functional Description

- **An object (GrLiAnyReferencedClass) is asked (by ServiceUser) to provide a UR**
- **GrLiAnyReferenceClass Creates a UR (GrLiAnyURClass) for itself**
- **ServiceUser then tells the UR to Externalize**

Assumptions/ Preconditions

- **GrLiAnyReferencedClass is a child class of the ECUrURProvider Class**
- **GrLiAnyURClass is a child class of the ECUrUR Class**

Event Trace: Creating and Externalizing a UR



Selected Public Interfaces

```
// This is the public method to request a UR from an object
const EcUrUR* EcUrURProvider::ProvideUR()
{
  EcUrUR* ur = CreateUR();   // call to virtual method below
  ProvideClassUR(*ur);        // call to virtual method below
  return ur;
}

// This is a protected overide of the virtual method
const EcUrUR* GrLiAnyReferencedUR::CreateUR()
{
   EcUrUR* ur = GetMyEcUrClassID(); // identifies my registered type

  EcUtGuardedFactory<EcUrUR>* factory = // Call to create a  object factory
      EcUtGuardedFactory<EcUrUR>::Instance();
```

Selected Public Interfaces (cont.)

```
  // use ur for factory to construct the right class of ur object
  EcURUR* ur  =  factory->Make(ur.GetURProviderID());

  return ur;
}

// This is a protected override of a pure virtual method
const EcUrUR* GrLiAnyReferencedUR::ProvideClassUR(EcURUR& ur)
{
  ur.myFirstParameter = parameterA;              // set my state in the UR
  ur.mySecondParameter = parameterB
  ...
  ur.myLastParameter = parameterZ
  return ur;
}
```

Event Traces

Name: Internalizing and Reconstituting a UR

Scenario

- An ASCII version of a UR is used to recreate the referenced object

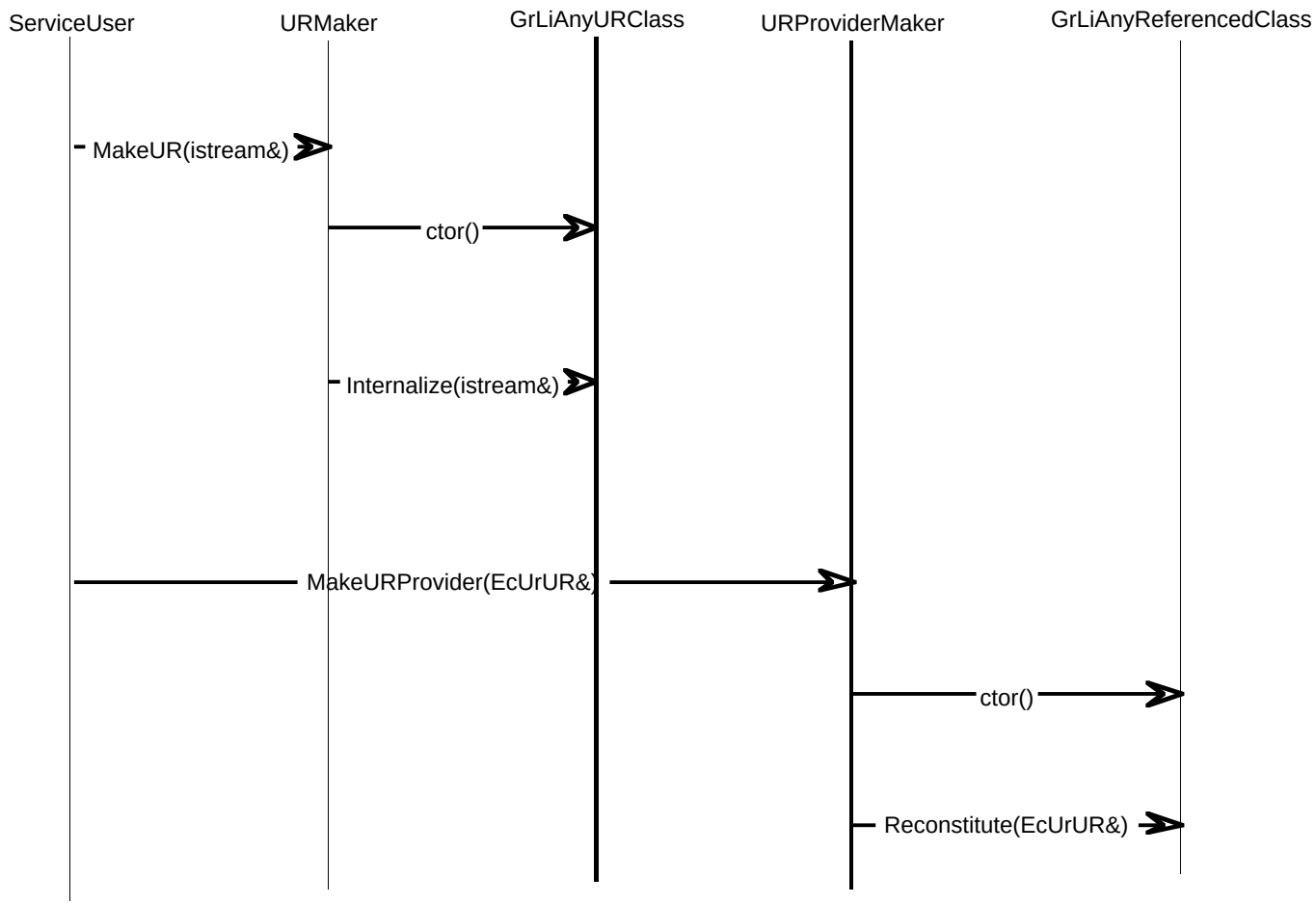
Functional Description

- First a UR object is made
  - ServiceUser Tells URMaker to Make a UR using the ASCII UR
  - URMaker creates a generic UR and Passes the UR the ASCII UR
- Second the Referenced Object is made
  - ServiceUser Tells UrProviderMaker to make it the object
  - UrProviderMaker creates the reference class object and restores the object's state by passing it the UR

Assumptions/ Preconditions

- GrLiAnyReferencedClass is a child class of the ECUrURProvider Class
- GrLiAnyURClass is a child class of the ECUrUR Class
- URMaker and URPRoviderMaker are already instantiated

Event Trace: Internalize and Reconstitute a UR



Selected Public Interfaces

```
// This is the public method to create a UR Provider (i.e., the referent of a
// ur) in the state determined by the UR
EcUrURProvider*  EcUrURProviderMaker::MakeURProvider
  (const EcUrUR& ur)
{
  // First call is to create an  object factory
  EcUtGuardedFactory<EcUrURProvider>* factory =
       EcUtGuardedFactory<EcUrURProvider>::Instance();

  // use ur for factory to construct the right class of object
  EcURURProvider* provider  =  factory->Make(ur.GetURProviderID());

  if (provider != NULL)          // if we got a good object
  {
       provider->Reconstitute(ur);  // call to public method
  }
  return provider;
}
```

Selected Public Interfaces (cont.)

```
//  Public Interface for the Referenced Class (i.e., the UR provider)
EcUrURProvider ::Reconstitute( const EcUrUR&  ur)
{
        ///   Implicit 'this' in call to reconstitute
        ReconstituteClassData(ur);          // call to my specialized method
}

// This is a protected override of a pure virtual method
GrLiAnyReferencedClass::ReconstituteClassData(const EcUrUR& ur)
{
        parameterA = ur.myFirstParameter;           // restore my state
        parameterB= ur.mySecondParameter;
        ...
        parameterZ = ur.myLastParameter;
}
```

UR Structure: System Defined

```
UR:var  {
      <URTypeID:14 bytes>  {          // Example for ECS Domain
            <URID:4 bytes> {
                  <ECRegistryDomains:4 bits>      //Domains are ECS, DAAC, SCF,
                                                          //UNIV, USER
                  <ECSubsystemDomains:8 bits>  //Client, PDPS, Common, etc.
                  <ECCSCIDomains: 8 bits>          //LIM, DIM, SDSRV, etc.
                  <ECClassID:12 bits>                //Session, ESDT, Advertisement, etc.
                  }
            <checksum:10 bytes>
            }
      }
      <implementation specific data:var>
}
```

Externalized:

UR:<UR ID string><UR checksum string><implementation specific string>

UR Structure: Data Granule Example

```
UR {
    <URTypeID:14 bytes> {
        // Structure defined in previous slide
    }
    <implementation specific data:var> {
        <serverUR> {
            <URTypeID:14 bytes> {
                // Structure defined in previous slide
            }
            <CDSname:var>
        }
        <granuleID:40 bytes> {
            <type:16 bytes>
            <subtype:16 bytes>
            <uniqueID:8 bytes>      //assigned by database
        }
    }
}
```

Externalized: UR:<UR ID string><UR checksum string>UR:
<UR ID string><UR checksum string><CDSname string>
<granule type string><subtype string><uniqueID string>

Infrastructure Implementation Review Issues Resolution

- **String Delimiters within URs**
  - **ECS using string routines from Rogue Wave Libraries**
  - **Rogue Wave String Classes use Carriage Returns**

- **Available Namespace for Data Server granules to guarantee UR uniqueness**
  - **Data Server UR structure is such that a particular unique ID must be unique only to that subtype of UR**
  - **A particular unique ID can be reused in a different subtype**
  - **Data Server allows up to 8 bytes for unique ID**

Event Trace: Building an Advertisement

Scenario

- An application that provides a service will build and submit an advertisement for itself. Examples of services providers are new interfaces, Data Subsetting services, etc.

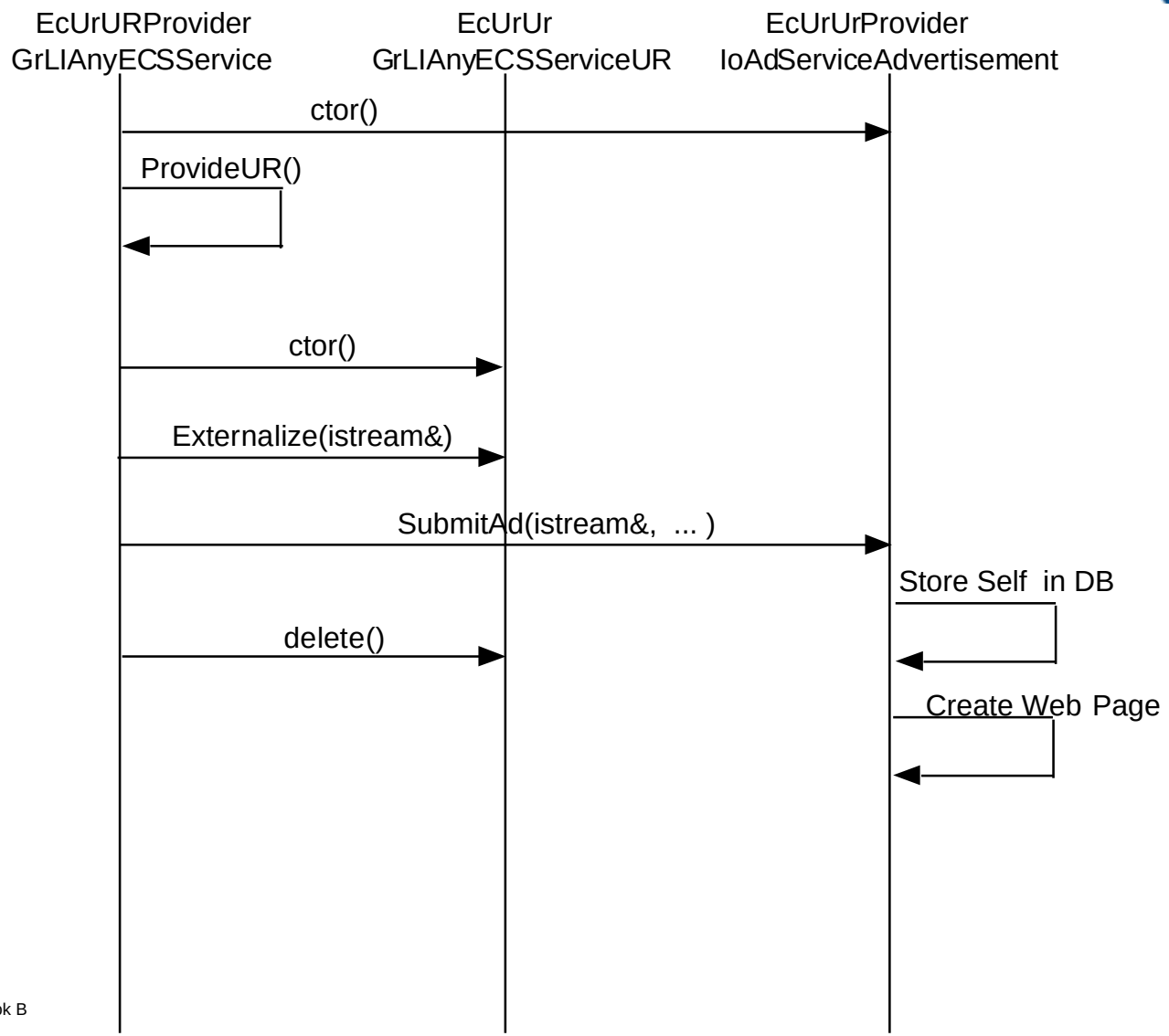
Functional Description

- Service provider creates an advertisement to describe itself
- Create a Web page to advertise the service
- Place an externalized UR in the HTML file
  - Service Provider extracts a UR object from itself
  - UR object is asked to externalize
  - This externalized UR placed in the advertisement
- Post Web Page

Assumptions/ Preconditions

- Service Provider is derived class from EcUrURProvider
- Service Provider UR is derived class from EcUrUR

Event Trace: Building an Advertisement



Event Trace: Using an Advertisement

Scenario

- A UR is used to install an Advertised Service on the desktop

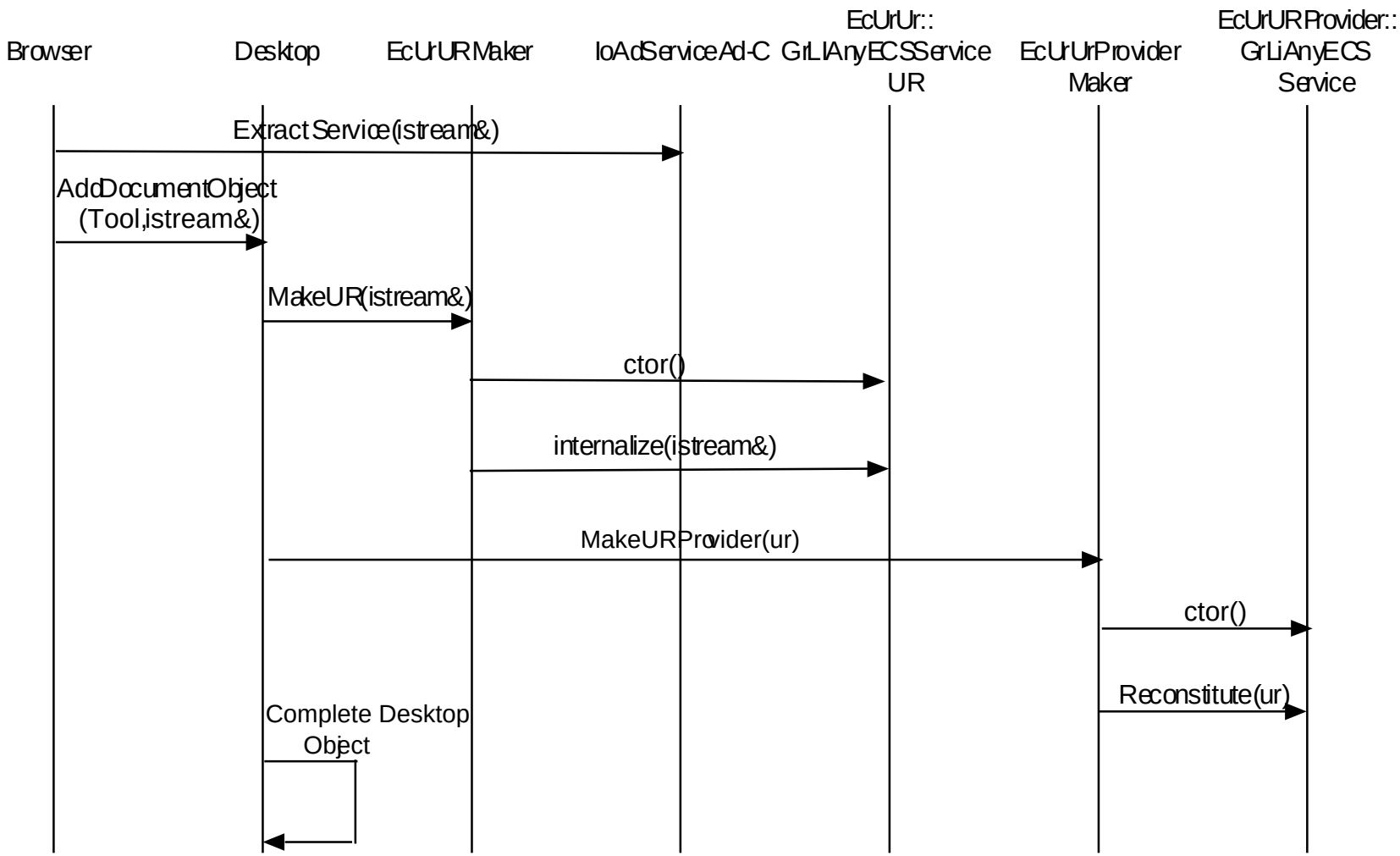
Functional Description

- The Installer uses the URMaker object to creates a Service Advertisement UR object and requests the UR object to internalize the externalized UR
- The Installer uses URProviderMaker object to create a new Service Advertisement, and requests the service advertisement reconstitute itself using the UR object
- The Advertised object is now available on the Client. Installer can now use the object's operations (to install its GUI, for example).

Assumptions/ Preconditions

- A user has used a Web browser to install an advertisement on the desktop
- The UR from the HTML page is the one created in the previous Building an Advertisement scenario

Event Trace: Using an Advertisement



Advertising UR Example

```
UR:var  {
      <URTypeID:14 bytes>  {          // Example for Advertising Subsystem
            <URID:4 bytes> {
                  ECRegistryDomains:4 bits = 0000              //ECS Domain
                  ECSubsystemDomain:8 bits = 00000110          //Interoperability
                  ECCSCIDomain: 8 bits = 00000000              //Advertising CSCI
                  ECClassID:12 bits = 000000000000             //Assume Class 0
                  }
            <checksum:10 bytes>
            }
      }
      <implementation specific data{
            CDS Name Entry:var
            CD Group Entry:var
      }
}
```

Externalized:

UR: 600000 12345678901234567890 /jpl/advertise /groups/advertise