PAID Object Design

24 November 1998

Georgios Markakis - Architecture

Reynald Ong - User Interface

Adam Phelps - Network

Pooja Saksena - Authentication

Georgios Markakis - Database

Jonathan Wildstrom - Learning/Event Service

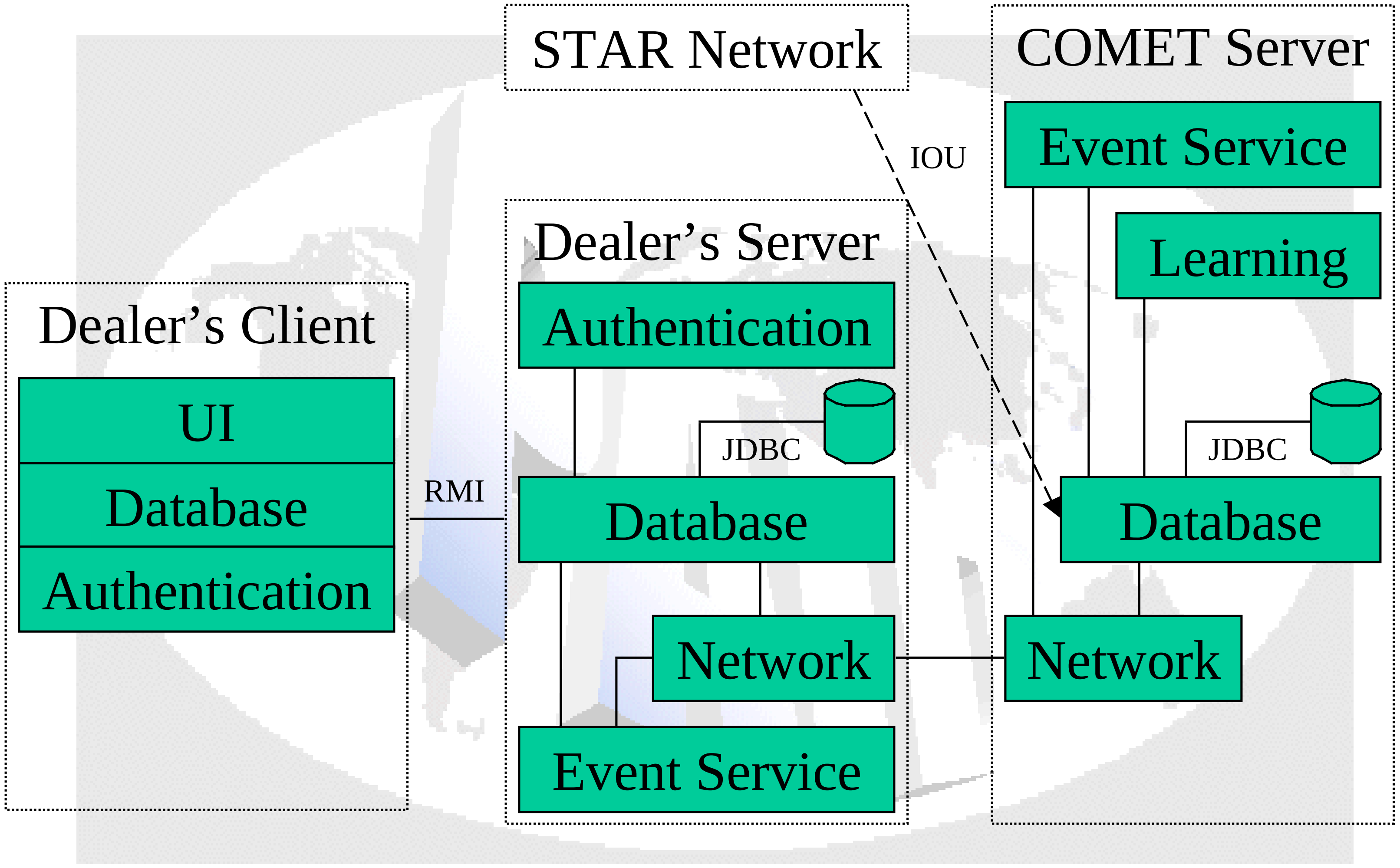
Architectural Overview

Presenter: Georgios Markakis

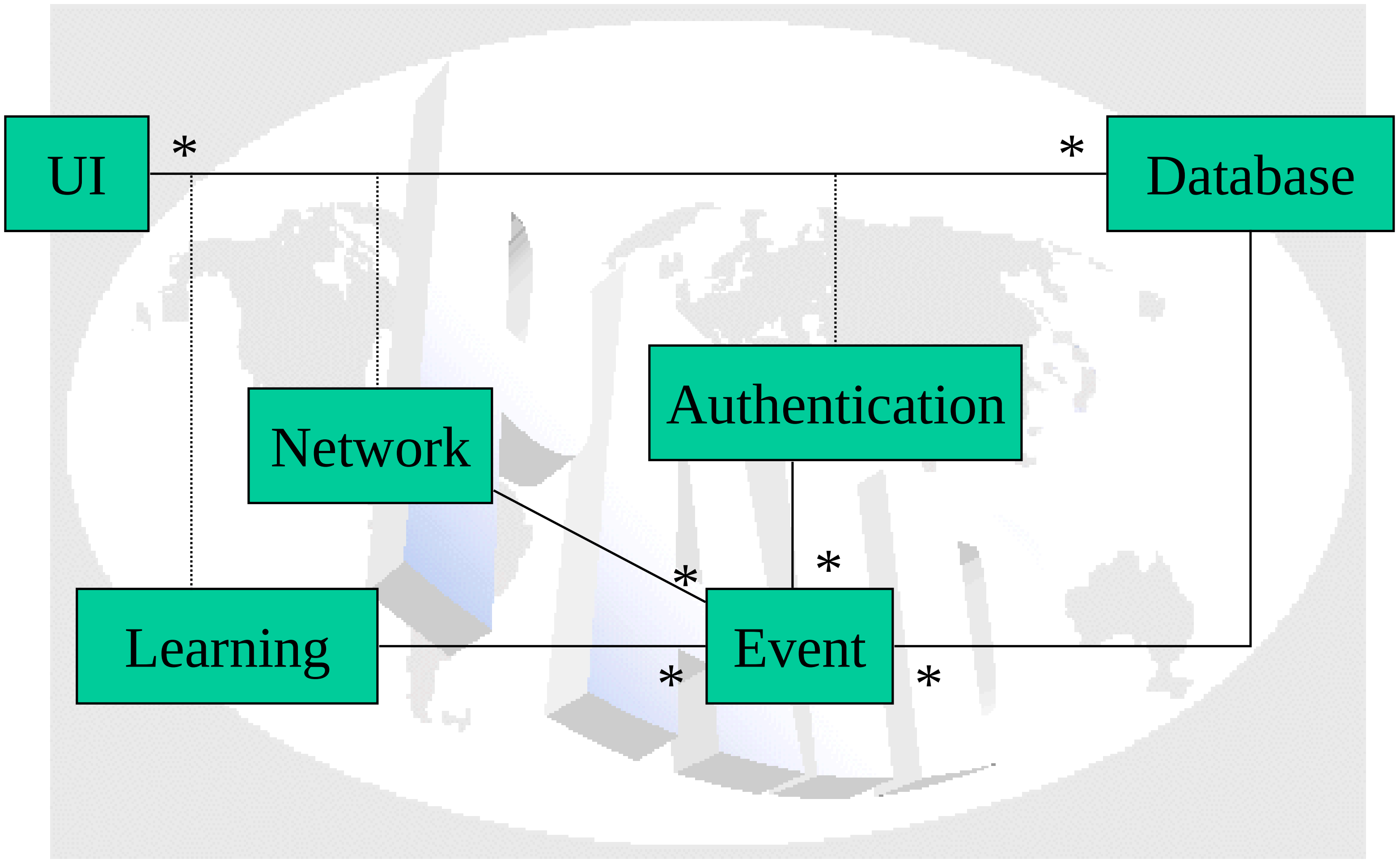
Architecture Team: Luis Alonso
Kent Ma
Michael Smith
Anthony Watkins

Coach: Elizabeth Bigelow

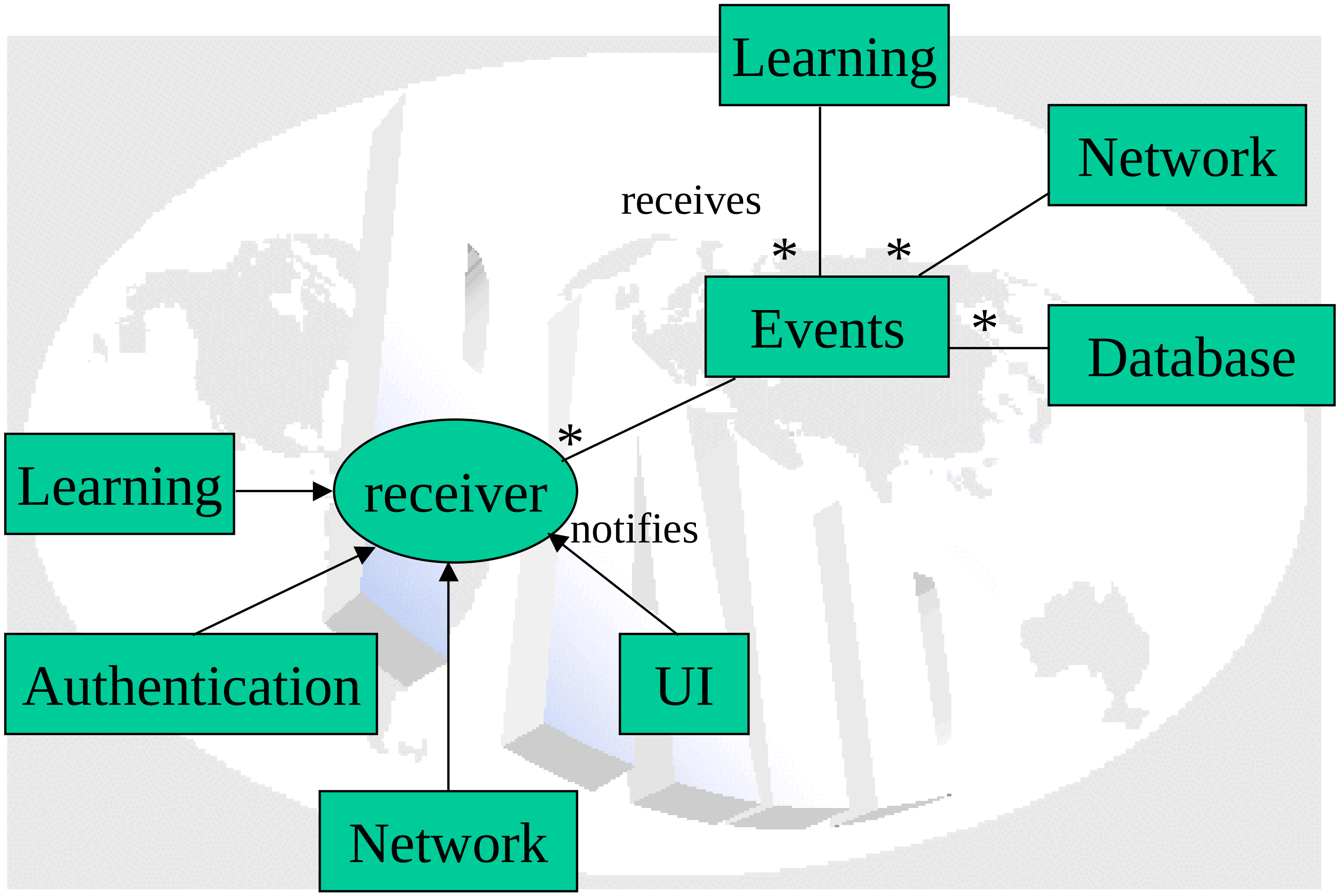
Hardware Deployment



System Object Model



More on Events



Overall Supported Scenarios

- All scenarios except Billing are supported
  - UI does not support Authentication and Security. Prefers to demonstrate Mobile Garage scenario
  - Methods to implement Mobile Garage are still looked into (disconnected user, wireless modems)
  - UI will not demonstrate functionality of Adding/Removing Users (Administration)

Authentication Object Design

Presenter: Pooja Saksena

Architecture Team: Luis Alonso
David Garmire
Arnaldo Piccinelli
Qiang Rao

Coach: Rus Heywood

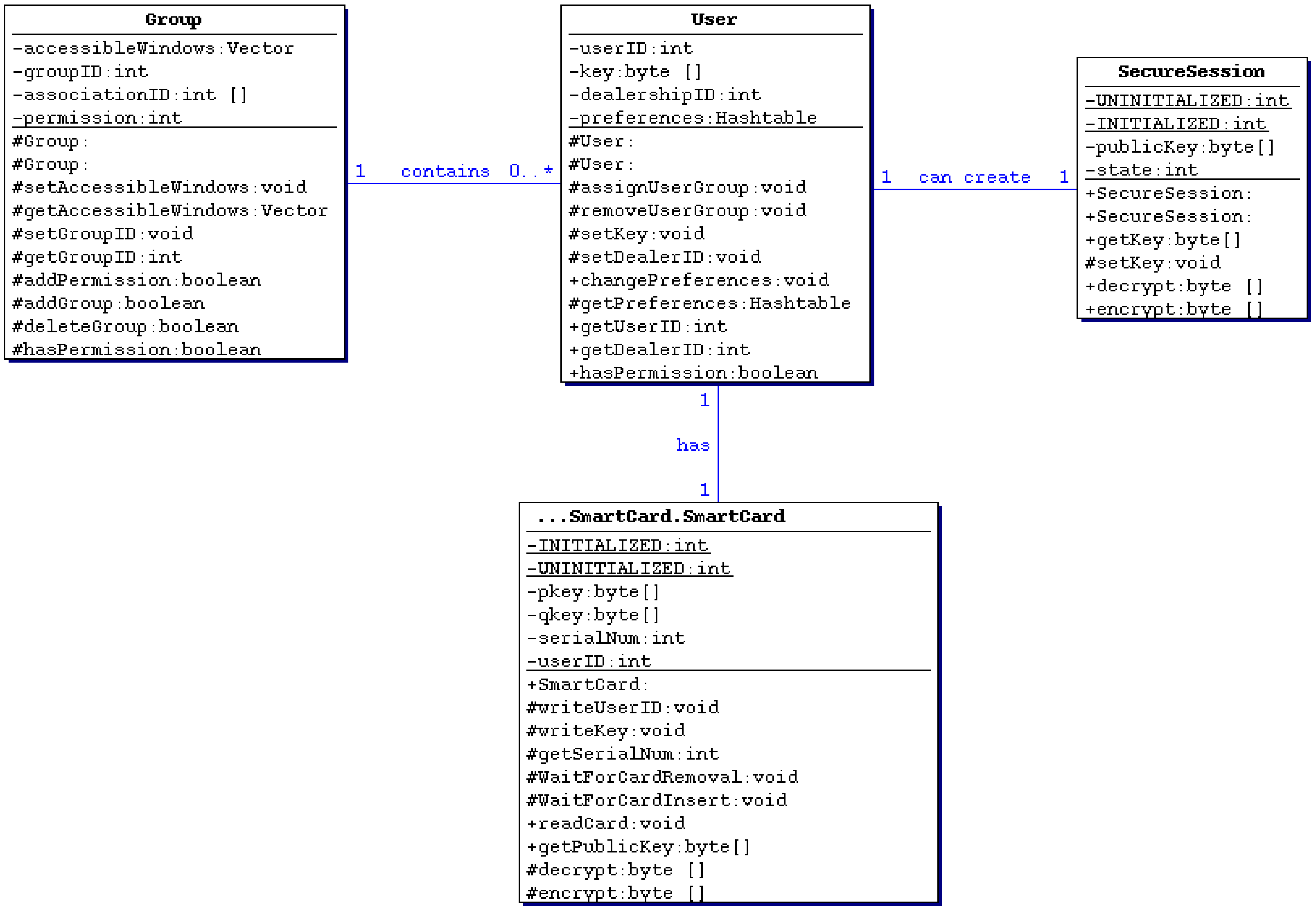
Supported Scenarios

- The three best supported scenarios :
  - Scenario 1: Adding a dealer.
    - Authentication is designed to implement this scenario.
  - Scenario 2: Poor network performance.
    - Application level security allows access to locally stored information, even if we PAID is unable to establish network connection.

Supported Scenarios(cont.)

– Scenario 8: Security
  - Expanding on the idea of Application security, users will only have access to information for their User Group.
  - Assuming that Extranet in insecure, we establish a first session by means of a challenge/response and encrypt all transmission with the session key.

Authentication Package



Required From UI:

- void login(User user)
  - Initiates the user interface based on User’s Group.
  - Using the notion of application security access is restricted by the functionality of the interface.
- void logout()
  - Signals that user has removed Smartcard.
  - Assuming each instance of the PAID application can have a single current user.

Provided for UI:

- void user.changePreference(Hashtable preference)
  - Public method, part of the User object.
  - Raises an exception if it fails.
- boolean user.hasPermission(int appID)
  - Each Application ID refers to a specific interface of UI.
  - This function queries the User object to see if the user has access to the specified application.

Required From Database (Users):

- Dealer Administration
  - Enumeration returnUsers()
    - Returns list of local users.
  - User getUserProfile(byte [] key)
    - Look up one user. (Needed to instantiate User object.)
- Daimler-Benz Administrators
  - void addUserProfile(byte[] key, User user)
    - Adds a user.
  - void changeUserProfile(byte [] key, User user)
    - Change user preferences.
  - void removeUserProfile(byte [] key)
    - Remove local user.

Required From Database (Groups):

- Diamler-Benz Administrators:
  - Enumeration getGroups()
    - Returns list of groups (should be updated regularly).
  - User getGroupProfile(int groupid)
    - Look up one group. (Needed to instantiate User object.)
  - void addGroupProfile(int groupid, Group group)
    - Adds a group. (For administrative purposes.)
  - void changeGroupProfile(int groupid, Group group)
    - Change group. (For UI's changeprefs, and for admins.)
  - void removeGroupProfile(int groupid)
    - Remove local groups.

Provided for Database:

- void infoChanged(int changeType, Object changeObject)
  - Either a User or Group object has changed.
  - ChangeType equals either GROUP or USER.
  - changeObject is a deserialized object of changed data.

Required From Network

- Object sendPublicKey(Object key)
  - An RMI call that allows the server and the client to receive each other's public keys.
- Boolean sendIdentity(Object Identity)
  - Sends the client's identity (from the key object) and the Comet server lookup to recognize the user.
- Serves to secure Extranet communications.

Provided for Network:

- boolean handshake()
  - Returns weather the handshake was successful.
- SecureSession.getKey()
  - get the other guys public key.
  - Returns exception NoKeyException.
- Object SecureSession.encrypt(Object msg)
- Object SecureSession.decrypt(Object msg)

Class:User

- The User object has information about the user's identity, preferences, and permissions
- The actual User object is stored in the database and the currentUser is instantiated at login.
- UI will use hasPermission(int appID) to query our user object to see if they can open the interface represented by the application ID.

Class: Group

- Attempted composite pattern, but hierarchy too hard to support for UI.
- Contains a vector of permissions to applications.
  - Format to be determined by UI.
- The call hasPermission(int appID) by UI is actually resolved by the Group permission functions.

Class: SmartCard

- Not visible to other subsystems.
- Exposes two main methods:
  - User waitForCardInsertion(void)
  - void waitForCardRemoval(void)
- For administrative purposes only:
  - void writeUserID(int uid)
  - void writeGroupID(int gid)
  - void writeKey(byte [] key)

Class: SecureSession

- After handshake() procedure provides successful session key, this object is instantiated.
- The decrypt and encrypt procedures are used with the session key for secure transfer of data on open network.

Stage of Development

- Smartcard.
  - Still in implementation.
- Security of Extranet through proxy/firewall.
  - Still in implementation.
- Encryption of transmissions through Extranet.
  - Still in implementation (have initial Caesar encryption).
- Interfacing to other groups.
  - Please contact Luis or Arnaldo if you have any questions concerning usage or side-effects of our API.

Remaining Issues

- How will we handle those w/o Smartcard
  - Lease a key for a fixed amount of time.
- To maintain the latest Group and User info
  - We will subscribe to the channels that have events related to these updates. Work out the details with events subgroup.
- Work out the weaknesses of out model and draw an outline of limitation our authentication system.

Summary

- Interactions with other groups:
  - UI:
    - Notify them of Smartcard status.
    - Check User/Group permissions.
  - DB
    - Need to retrieve User/Group preferences/permissions.
  - Network
    - Establish secure session and session key.
    - Encryption/decryption of data on Extranet.
- Some unresolved issues have to be closed by communication with other groups.

Database Object Design

Presenter: Georgios Markakis

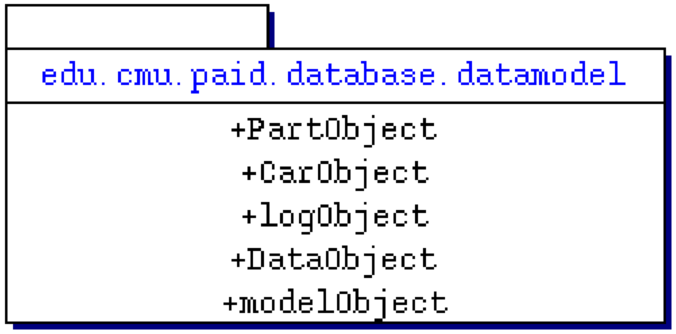
Architecture Team: Richard Markwart
Tim Shirley
Ivan Tumanov

Coach: Keith Arner

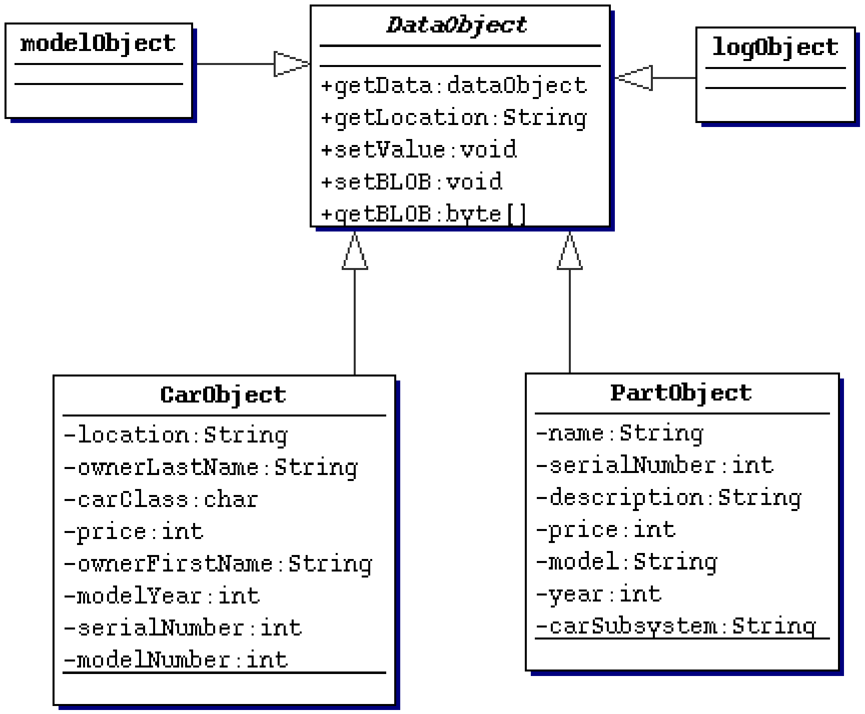
Scenarios to be Supported

- All but billing scenario
- Mobile garage still uncertain
    - Implementing local store on mobile devices is an open issue:
        - Wireless modems considered
        - Actual database server one option
        - Local store without database server an alternative

Database API - Datamodel



DataObject is the primary object that database implements and supports

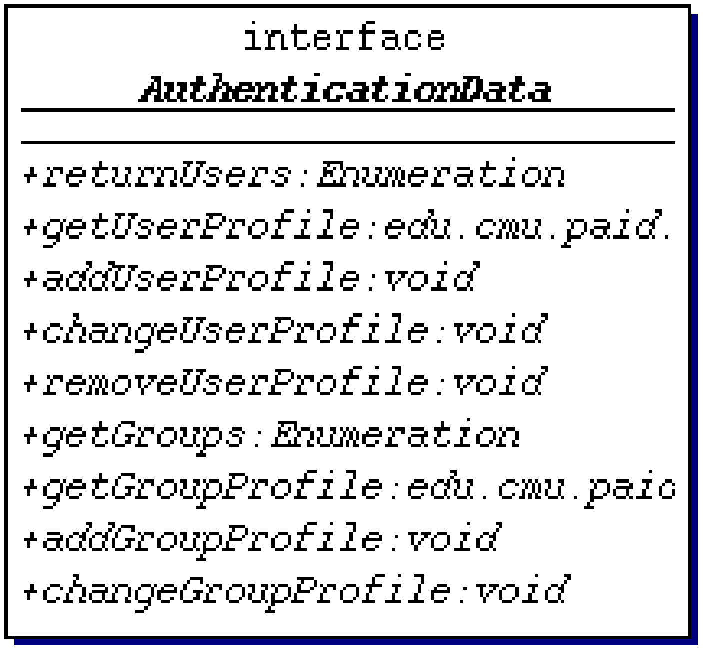


- modelObject, logObject are used by Learning
- CarObject, PartObject are used by UI

They all implement DataObject

Interface for Authentication

Methods provided to Authentication



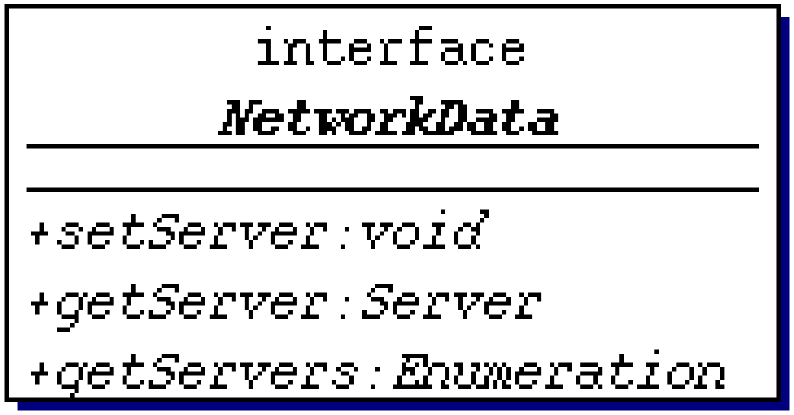
```
interface
AuthenticationData

+returnUsers:Enumeration
+getUserProfile:edu.cmu.paid.
+addUserProfile:void
+changeUserProfile:void
+removeUserProfile:void
+getGroups:Enumeration
+getGroupProfile:edu.cmu.paic
+addGroupProfile:void
+changeGroupProfile:void
```

- Return list of Users
- Add/Edit/Delete single Users
- Return list of Groups
- Add/Edit/Delete single Groups

Interface for Network

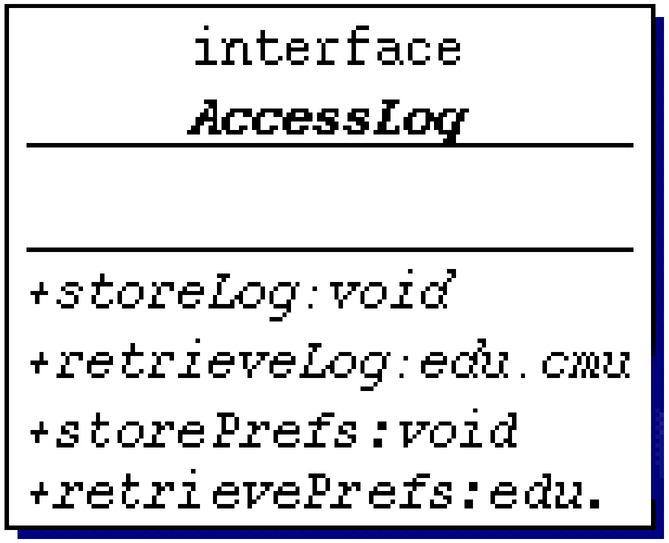
Methods provided to Network

- Managing COMET server information
- Retrieving server information (determining location of data)
- Listing all COMET servers



Interface for Learning

Methods provided to Learning



```
interface
AccessLog

+storeLog:void
+retrieveLog:edu.cmu
+storePrefs:void
+retrievePrefs:edu.
```

- Store log information
- Retrieve log information
- Store User preferences
- Retrieve User preferences

Database API (cont.)

edu.cmu.paid.database.comet

- Description of classes that will be running on COMET servers:
  - Retrieving data
  - Storing data (storage for other subsystems)
  - Update notification

NOTE …

Object model has not been finalized yet

Database API (cont.)

edu.cmu.paid.database.dealer

- Description of classes that will be running on dealer server:
  - Check local store
  - Store data
  - Determine location of data

edu.cmu.paid.database.iou

- Will simulate IOUs since those are not currently available

NOTE …

Object model has not been finalized yet

Services needed by Database

From Network:

- When data is not found on dealer server (local), invoke getRemoteData method

From Learning:

- Method to be invoked when an update occurs

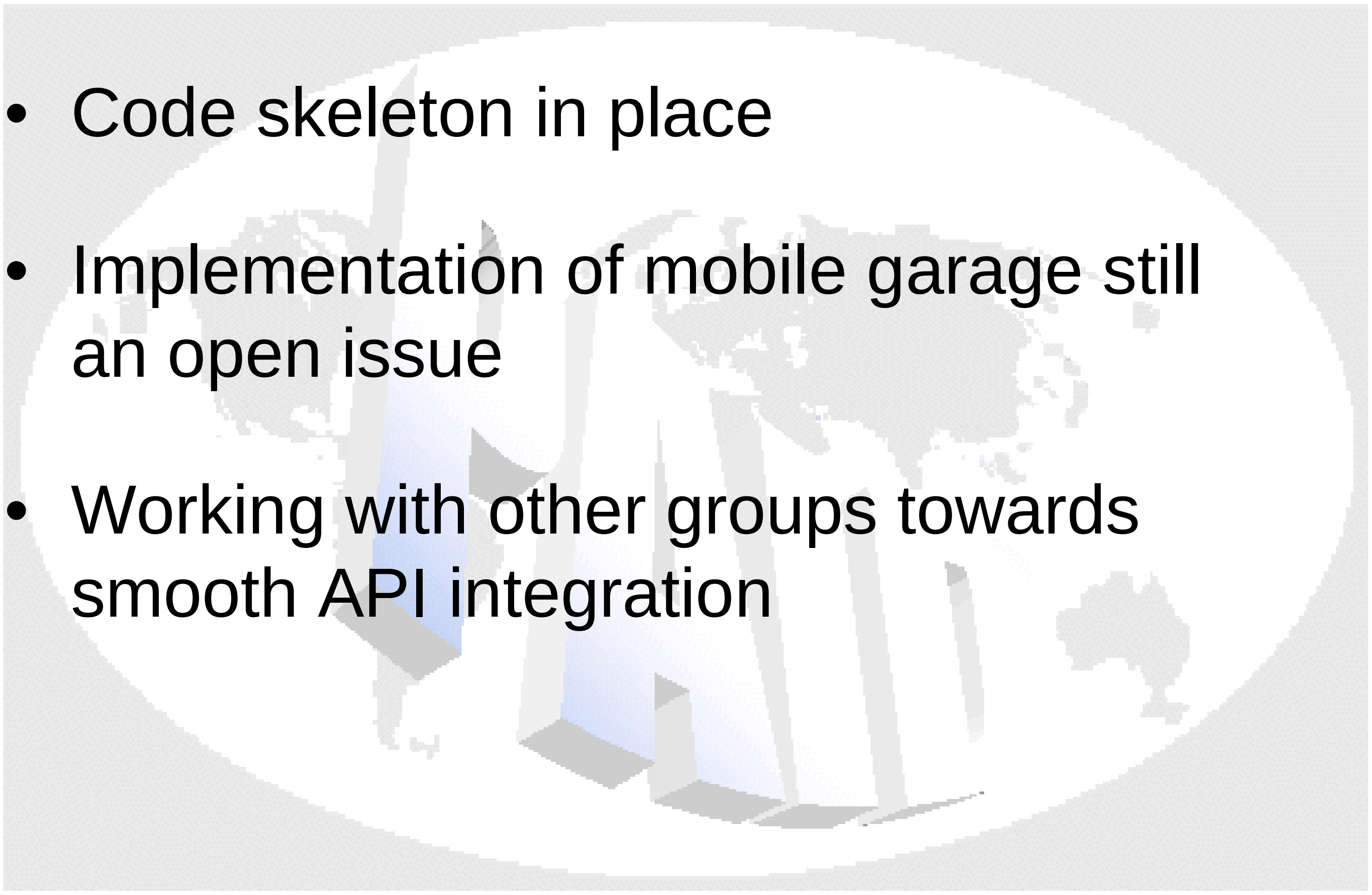
From Events:

- Event channels through which we can broadcast changes/updates

Note for Authentication:

- Database will assume it will receive only calls that the user is allowed to execute

Status of system development



- Code skeleton in place
- Implementation of mobile garage still an open issue
- Working with other groups towards smooth API integration

Learning and Event Services Object Design

Presenter: Jonathan Wildstrom

Learning & Event Services Team: Jonathan Hsieh
James Lampe
Yun-Ching Lee
Wing Ling Leung
Rudy Setiawan
Andrew Zimdars

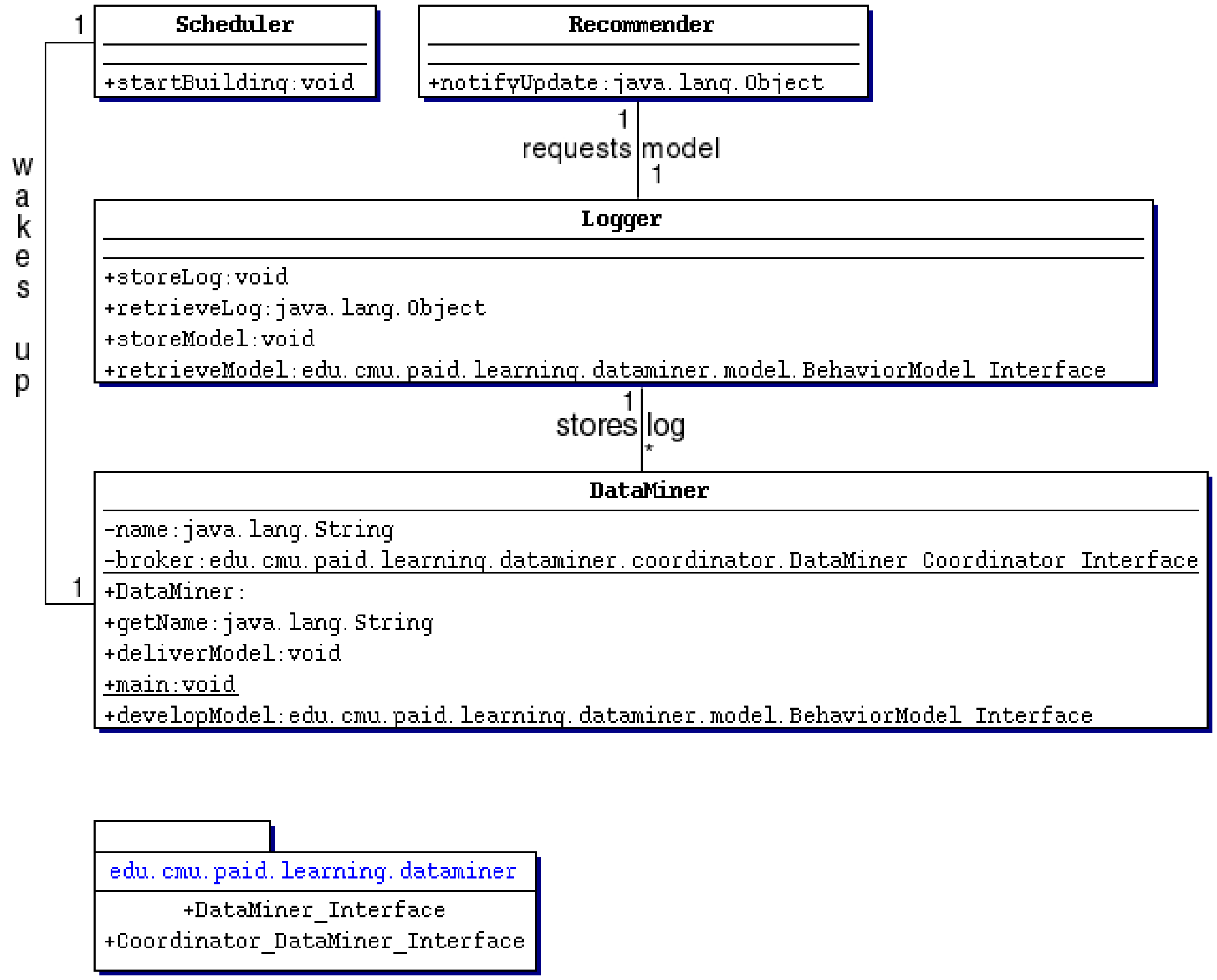
Coach: Eric Stein

Scenarios

- Dealer’s Workshop at 8 AM
  Minimization of connection costs
  - Ability to learn best time for updates

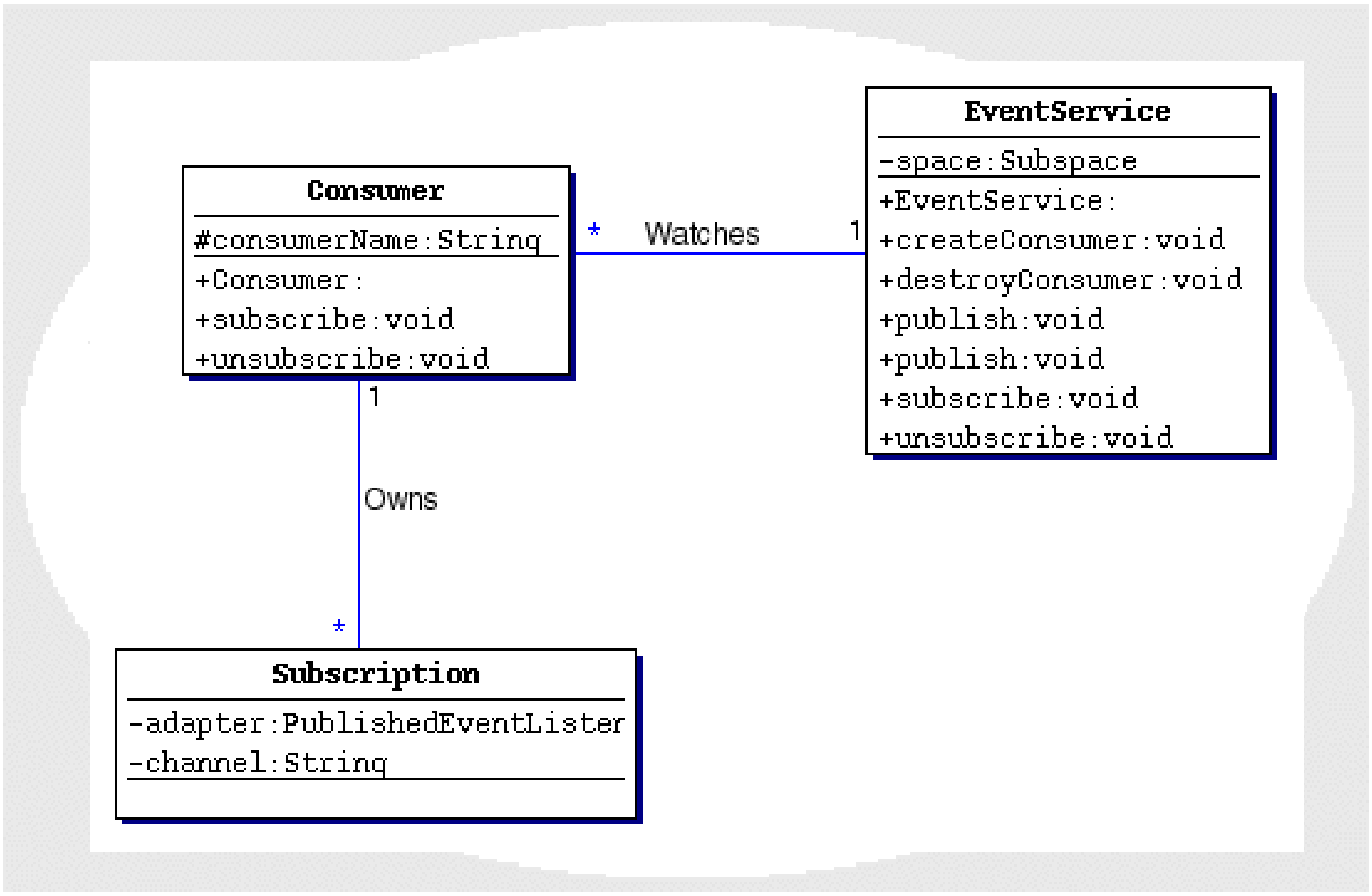
- Introduction of M-class
  - Monitor document requests
  - Learn about repeated accesses
  - Recommendations for local DB

Learning Object Model



Scheduler
+startBuilding:void
Recommender
+notifyUpdate:java.lang.Object
1
requests model
1
Logger
+storeLog:void
+retrieveLog:java.lang.Object
+storeModel:void
+retrieveModel:edu.cmu.paid.learning.dataminer.model.BehaviorModel_Interface
1
stores log
*
DataMiner
-name:java.lang.String
-broker:edu.cmu.paid.learning.dataminer.coordinator.DataMiner_Coordinator_Interface
+DataMiner:
+getName:java.lang.String
+deliverModel:void
+main:void
+developModel:edu.cmu.paid.learning.dataminer.model.BehaviorModel_Interface
1
wakes up
1
edu.cmu.paid.learning.dataminer
+DataMiner_Interface
+Coordinator_DataMiner_Interface

Event Service Object Model



API

Event Service
public void subscribe(Object Adapter, String channel);
public void unsubscribe(String channel);
public void publish(String channel, String eventString);
public void publish(String channel, Object eventObject);

Learning
public void logTrigger(String dealerID);

Required Services

<u>Database</u>

User Preferences Database:

- Dealer Server ID (unique identifier)
- Locale of Dealer Server
- Connection cost (by hour or flat rate)

Log Database:

- Document information

<u>Network</u>

- Elapsed Times on Download
- Net Load statistics
- Time and Date of Download

Required Services cont.

<u>UI</u>

- Preference Panel

<u>Event Services</u>

- Channel to UI

Provided Services

Learning

-Recommendations for update times
-Recommendations for additions to local DB

Event Services

-Channels for communication between different subsystems

Development Status

-Skeletal code in place

-Wrappers being written for off-the-shelf solutions

- Voyager: Event service
- JaNet: Neural Network
- ID3: Decision Tree
- BayesNet: Naïve Bayesian Method
- Data Miner: Ibiza

- Scheduler pseudocoded
- Logger not started
- Recommender not started

Network Object Design

Presenter: Adam Phelps

Network Team: Anthony Watkins
Barret Trask
Orly Canlas
William Ross

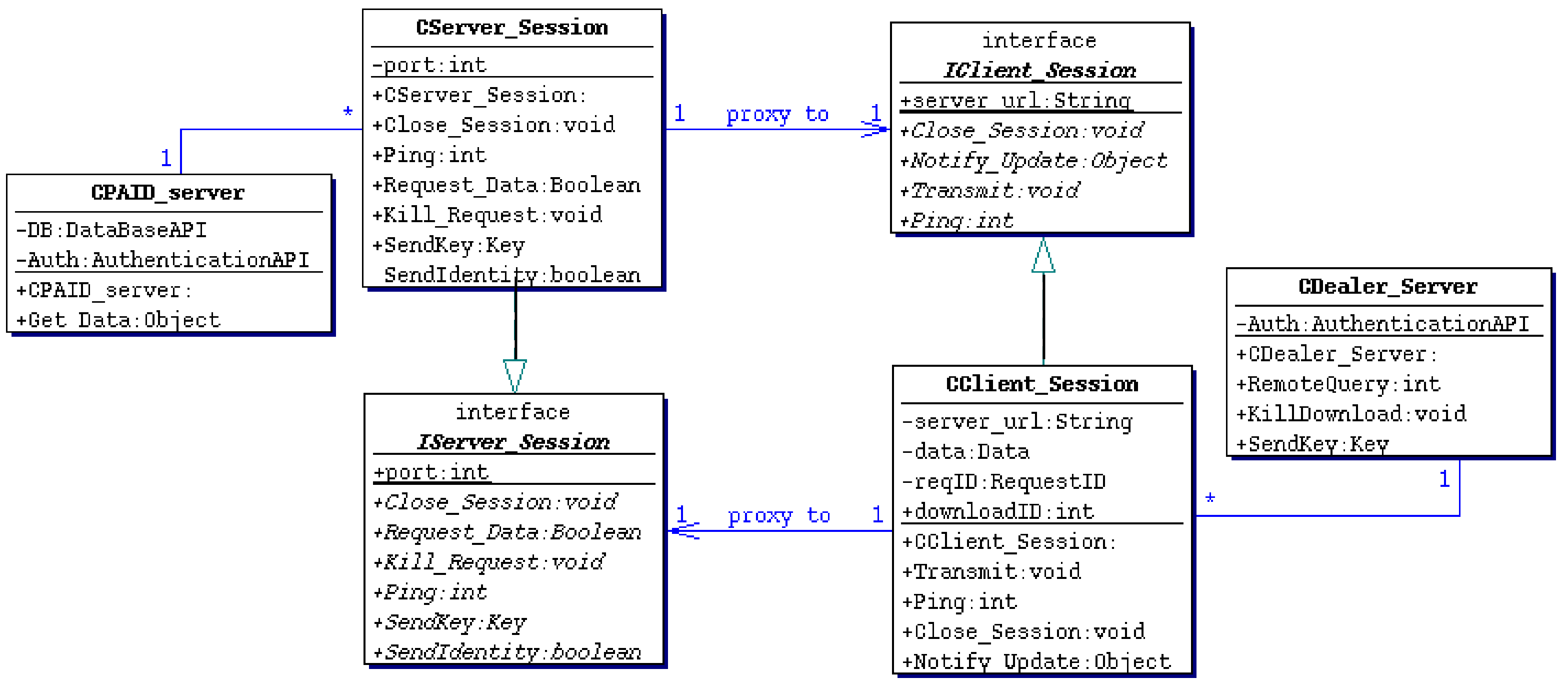
Coach: Robin Loh

Scenarios

Scenario 3 : Download Termination

In this scenario, a dealer is having poor network response time and doesn't want his customers to have to wait on an ongoing download. Instead of waiting for the download to complete, the dealer can choose to terminate the download and handle his customers. Also, when the dealer receives notification of a database update, he may choose to wait until later to download the update.

Object Model



API

Event Notification

This function will be used by the Learning/Event Team to transmit events.

Format:

```
String notify(Event event, String[] targetlist);
```

This function multicasts `event` to all the machines included in the group in `targetlist`. The returned string is the list of machines that successfully received the event.

API

Remote Database Query

This functions allows Database to request data from a remote database.

Format:
```
int RemoteQuery(Data data, RequestID reqID, String[]
serverList);
```

This function attempts to get the data specified by `reqID` from one of the servers included in `serverList`. Upon successful completion of transmission, the `data` object will contain the requested data. However, this function returns as soon as the request is made, so `data` does not contain valid data upon termination of this function.
The value returned is the identification number used to reference the request.

API

Terminate Download

This function allows an ongoing transmission to be terminated.

Format:
```
void KillDownload(int DownloadID);
```

This function terminates the download referenced by `DownloadID` (which was returned by a `RemoteQuery` call).

Events

The Network subsystem will generate a number of events.

- `Network.NetworkDown`

  This event is generated if the network connection is lost.

- `Network.ConnectionLimitExceeded.(downloadID)`

  This event is generated if a server has reached its maximum connection capacity. `downloadID` is the reference ID of the request that produced this event.

Events

The Network subsystem will generate a number of events.

- `Network.DownloadBegun.(downloadID).(x)`

  This event is generated when a download has been successfully initiated. `downloadID` is the reference ID of the initiated download, and `x` is the size of the download

- `Network.DownloadFailed.(downloadID)`

  This event is generated if the download referenced by `downloadID` cannot be completed.

- `Network.DownloadPercent.(downloadID).(x)%`

  This event indicates that the download referenced by `downloadID` is `x`% complete.

- `Network.DownloadComplete.(downloadID).(x)`

  This event is generated upon completion of the download referenced by `downloadID`. The total time (in seconds) of the download is `x.`

Required Services

- The Database subsystem must provide a method via which Network will access data.
- The Event subsystem must provide a method for Network to initiate events.
- Authentication needs to provide methods to get the public key and dealer identity, as well as to encrypt and decrypt data.

Provided Services

- The Network subsystem shall provide a method via which Event Services may transmit events.
- The Network subsystem will provide a method for transferring database objects between a server and dealer machine.
- The Network subsystem will provide a method via which Authentication can authenticate a connection between a PAID server and dealer.

Development Status

Tasks Complete:

- Voyager operation on the Linux machines in the SE Lab.
- Skeleton classes defined.

Tasks Not Yet Complete:

- Full implementation of classes

User Interface Object Design

Presenter: Reynald Ong

User Interface Team: Euijung Ra
Brian Woo
Stephane Zermatten

Coaches: Elaine Hyder
Jack Moffett

Supported Scenarios

- The scenarios that can not be supported are:
  - Scenario 7: Billing. (Deferred)
  - Scenario 1: Adding a Dealer. (UI has no role)
  - Scenario 8: Security. (UI has no role)
- The scenario that best demonstrates UI:
  - Scenario 6: Mobile Garage.

Required from other subsystems:

- Learning:
  - List of updates to display to the users.
    - Comes in as an event.
  - Ability to get the dealer level preferences.
    - DealerObject getDealerPreferences (void)
  - Ability to set the dealer level preferences.
    - void setDealerPreferences (DealerObject Dealer)
  - Ability to start updating.
    - void startUpdate (void)

Required from other subsystems:

- Authentication:
  - User permission given an application ID.
    - Boolean hasPermission (int appID)
  - Ability to set the different user preferences.
    - void changePreference (Hashtable preference)
- Database:
  - Query results and data.
    - DataObject getData (Request request)
- Network:
  - Ability to kill a download given the download ID.
    - void killDownload (int downloadID)
  - All other events will go through Event Service.

Required from other subsystems:

- Event Service:
  - Create and subscribe to channels to receive events.
    - void createConsumer (string Consumer)
    - void destroyConsumer (string Consumer)
    - void subscribe (string Consumer, PublishedEventListener eventListener, string channel)
    - void unsubscribe (string Consumer, string channel)

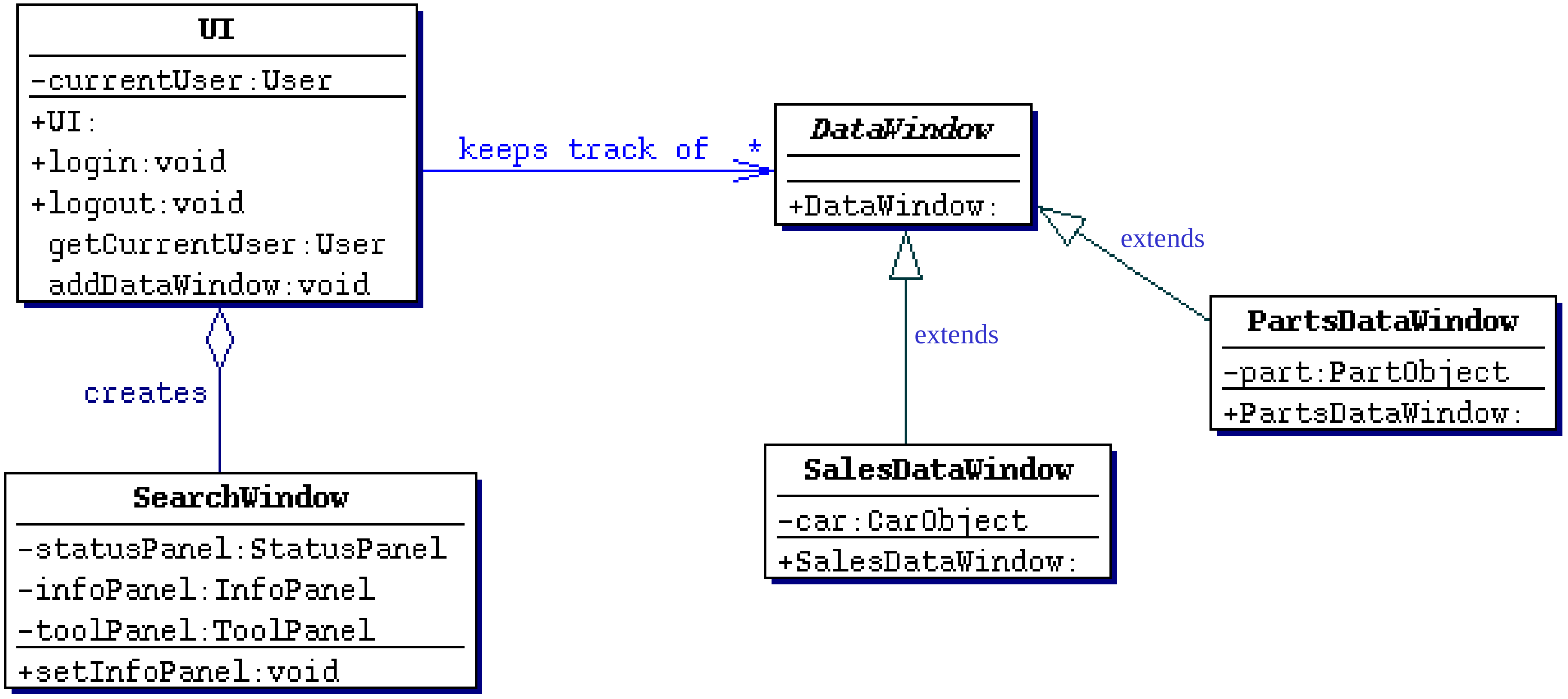
Provided for other subsystems:

- All teams: UI object
  - Class UI: UI (void)
  - public void login (User user)
  - public void logout (void)

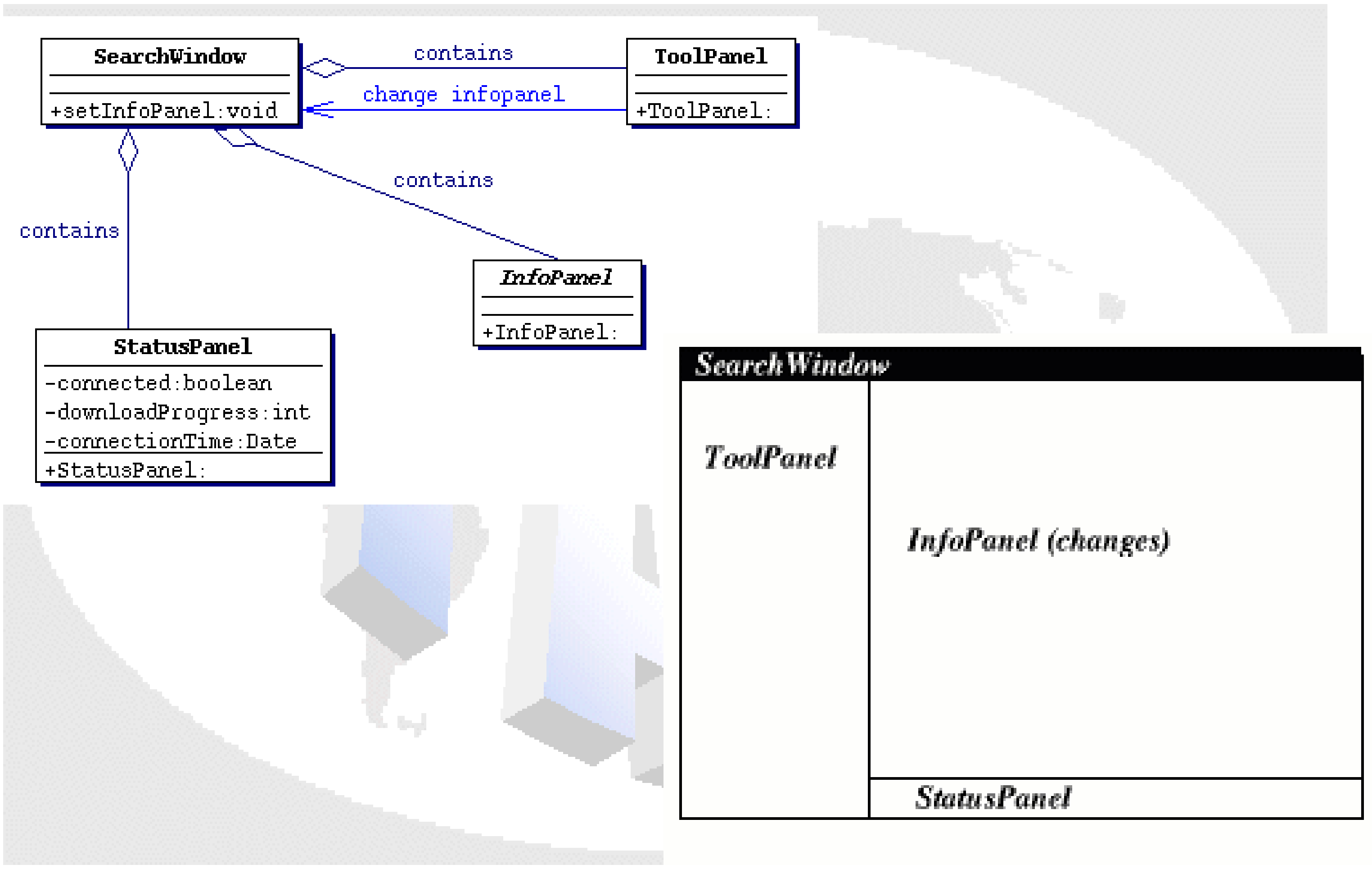
  - Local methods not intended for other subsystems:
    - User getCurrentUser (void)
    - void addDataWindow (DataWindow dataWindow)

Note: Class User is an object from the Authentication Team.

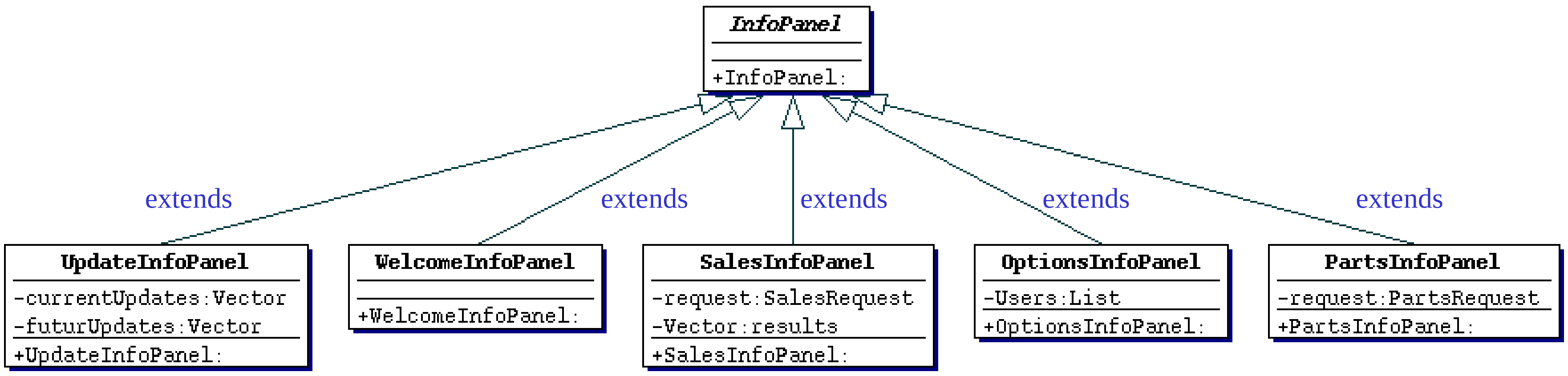
UI Objects: Top level (Class UI)



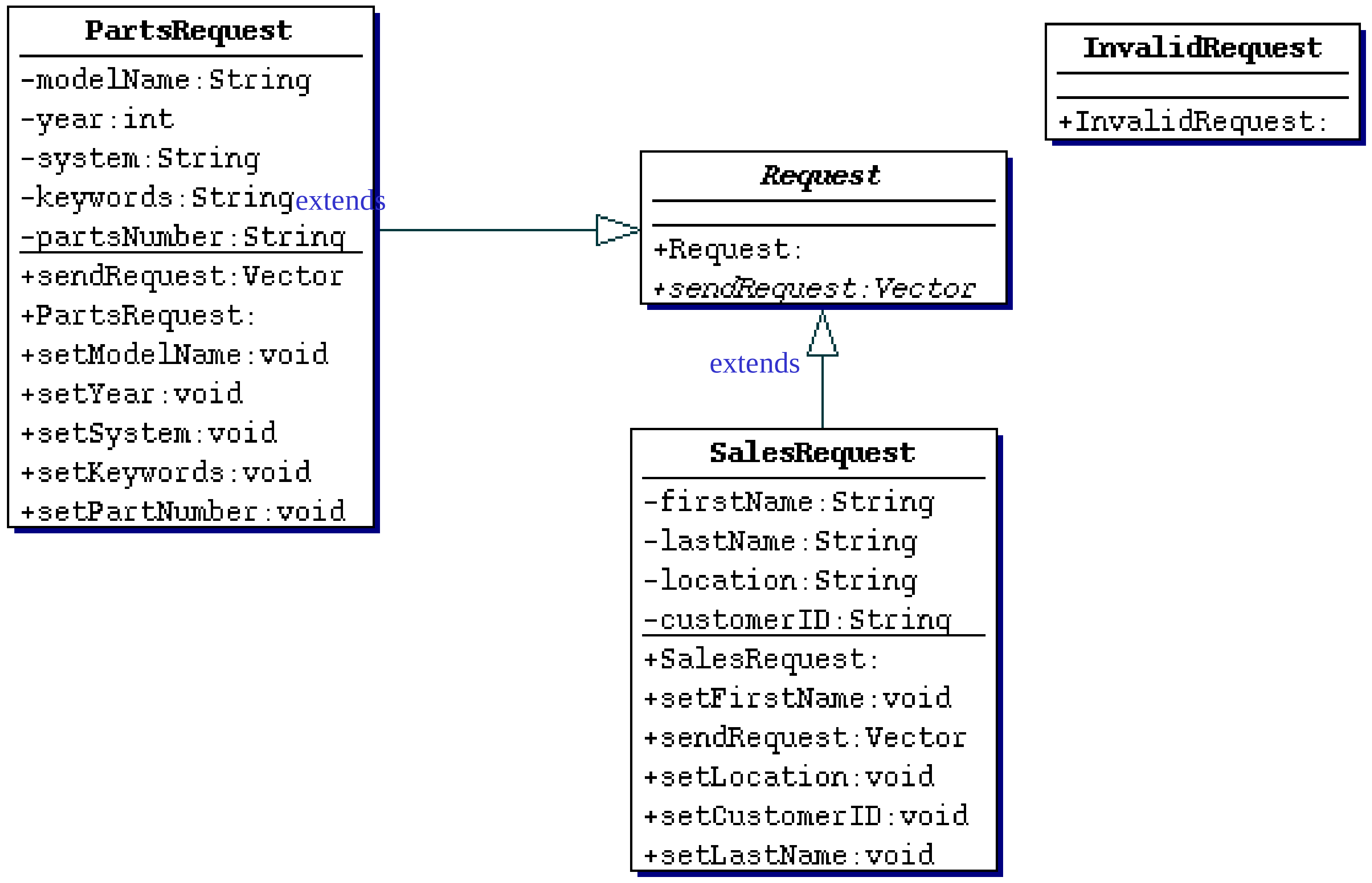
UI Objects: Internal components (Search Window)



UI Objects: Internal Components (InfoPanel)



UI Objects: Internal Classes (Request)



Public Class: UI

- Only visible class to other subsystems.
- 2 public methods:
  - public void login (User user)
  - public void logout (void)
- 2 local methods:
  - User getCurrentUser (void)
  - void addDataWindow (DataWindow dataWindow)
- Private variable:
  - User currentUser: represents the current user.

Private Classes:

- Class SearchWindow (void)
  - Extends Java Swing JFrame.
  - The main container for all other panels including tools, status, and info panels.
- Class DataWindow (void)
  - Extends Java Swing Jframe.
  - Abstract class implemented by other classes.
- Class SalesDataWindow (CarObject data)
  - Extends class DataWindow and displays sales info.
- Class PartsDataWindow (PartObject data)
  - Extends class DataWindow and displays parts info.

Private Classes:

- Class InfoPanel (JFrame Parent)
  - Extends Java Swing JPanel.
  - Container for other panels displaying queries and information.
  - Abstract class implemented by other classes.
- Class ToolPanel (JFrame Parent)
  - Extends Java Swing JPanel.
  - Container of buttons for control.
- Class StatusPanel (JFrame Parent)
  - Extends Java Swing JPanel.
  - Container for displaying status information.

Private Classes:

- Class WelcomeInfoPanel (JFrame Parent)
- Class OptionsInfoPanel (JFrame Parent)
  - Panel for setting user preferences.
- Class UpdateInfoPanel (JFrame Parent)
  - Panel for update information and download.
- Class SalesInfoPanel (JFrame Parent)
  - Panel for searching sales information.
- Class PartsInfoPanel (JFrame Parent)
  - Panel for searching parts.
- All above classes extends class InfoPanel.

Private Classes:

- Class Request (void)
  - Contains a request to construct a query to the database.
  - Abstract class implemented by other classes.
- Class SalesRequest (void)
  - Contains a request to find certain sales info.
  - Has internal methods to set the request.
- Class PartsRequest (void)
  - Contains a request to find certain parts info.
  - Has internal methods to set the request.

Stage of Development

- Working on prototype and API.
  - Prototype design is at its final stage of modeling.
  - API still under discussion. More requirements are starting to surface from other teams.
- Working on coding.
  - Implementation of user interface has just begun.
- Researching into mobile devices.

Summary

- Scenarios not supported:
  - Scenario 1 and 8: UI has no role
  - Scenario 7: Deferred

- Scenarios demonstrating UI:
  - Scenario 6: Mobile Garage

- Services provided to other subsystems:
  - All teams: UI object
    - Class UI: UI (void)
    - public void login (User user)
    - public void logout (void)

Summary Cont.

- Services required from other subsystems:
- Learning:
  - Event for updates.
  - DealerObject getDealerPreferences(void)
  - void setDealerPreferences (DealerObject Dealer)
  - void startUpdate (void)
- Authentication:
  - Boolean User.hasPermission (int appID)
  - void changePreference (Hashtable preference)

Summary Cont.

- Database:
  - DataObject getData (Request request)
- Network:
  - void killDownload (int downloadID)
- Event Service:
  - void createConsumer (string Consumer)
  - void destroyConsumer (string Consumer)
  - void subscribe (string Consumer, PublishedEventListener eventListener, string channel)
  - void unsubscribe (string Consumer, string channel)