Εθνικό και Καποδιστριακό Πανεπιστήμιο Αθηνών
Τμήμα Πληροφορικής & Τηλεπικοινωνιών

Προηγμένες Μέθοδοι Προγραμματισμού

ΠΜΣ 2022-23 (M135.CS1E, M135.CS23B, M135.IC1E, παλαιό: M117)

Πρότυπα σχεδίασης (3)

Δρ. Κώστας Σαΐδης (saiko@di.uoa.gr)

Σχεδιαστικά πρότυπα λογισμικού στον Παγκόσμιο Ιστό

AJAX

- Asynchronous Javascript and ~~XML~~ (JSON)
- Μια απλή τεχνική που κάνει τις εφαρμογές πιο αποκρίσιμες
- Μηχανισμός που επιτρέπει στον προγραμματιστή να "πει" στον web-browser να εκτελέσει την αποστολή του HTTP αιτήματος ασύγχρονα (σε ξεχωριστό thread)
- Παρέχεται callback hook για να λάβουμε το αποτέλεσμα

Παράδειγμα AJAX

http://jsfiddle.net/ddtxra/60wme3pf/

The callback function pattern

Σκοπός: Να εκτελέσουμε ένα κομμάτι κώδικα αφού έχει ολοκληρωθεί ένας υπολογισμός (σύγχρονα ή ασύγχρονα)

```
function foo(args, callbackFunction)
// Hey, foo, run my callbackFunction when you're done
```

Θυμηθείτε!

```
interface Datastore {
  void load(String id, Consumer<Item> itemConsumer)
  void save(Item item, Consumer<Result> resultConsumer)
}
```

Callback nesting

```
$(document).ready(function(e) { //1 (DOM event)
    $("form[ajax=true]").submit(function(e) { //2 (DOM event)
        e.preventDefault();
        ...
        $.ajax({
            url: form_url,
            type: form_method,
            data: form_data,
            cache: false,
            success: function(returnhtml) { //3 (AJAX "event")
                $("#result").html(returnhtml);
                $("#loadingimg").hide();
            }
        });

    });

});
```

Callback Hell

```
getData = function(param, callback){
  $.ajax('http://example.com/get/'+param,
    function(responseText){
      callback(responseText);
    });
}

getData(0, function(a) {
  getOtherData(a, function(b) {
    getOtherOtherData(b, function(c) {
      getOtherOtherOtherData(c, function(d) {
        ...
      });
    });
  });
});
```

Future / Promise

Σκοπός: Να διαχειριστούμε το αποτέλεσμα ενός υπολογισμού (=μια τιμή που κάποια στιγμή θα γίνει διαθέσιμη) με ομοιόμορφο τρόπο, ανεξάρτητα του αν ο υπολογισμός γίνεται σύγχρονα ή ασύγχρονα

Τρεις καταστάστεις

- Pending (not yet available)
- Fulfilled (with an optional value)
- Rejected (due to an error or a timeout, with an optional value)

Διαφορές ανά γλώσσα

Java: Future, ComletableFuture

Javascript: Promise

Future

Μια read-only αναφορά σε μια τιμή που δεν έχει ακόμα υπολογιστεί (ο χρήστης του Future δεν έχει έλεγχο στην τιμή που θα προκύψει).

Promise / CompletableFuture

Μια single-assignment μεταβλητή για την τιμή του Future (ο χρήστης του Promise μπορεί να θέσει άπαξ την τιμή που θα προκύψει, a promise = a future with a setter)

Future στη Java

Package java.util.concurrent

```
interface ExecutorService {
  Future<V> submit(Callable<V> callable)
}
```

Το `ExecutorService` είναι συνήθως κάποιο ThreadPool

```
interface Future<V> {
  //Attempts to cancel execution of this computation.
  boolean cancel(boolean mayInterruptIfRunning);

  //Waits if necessary for the computation to complete,
  //and then retrieves its result.
  V get();

  //Waits if necessary for at most the given time for
  //the computation to complete, and then retrieves
  //its result, if available.
  V get(long timeout, TimeUnit unit)

  //Returns true if this computation was cancelled before it
  //completed normally.
  boolean isCancelled()

  //Returns true if this computation completed.
  boolean isDone()
}
```

```
public class CompletableFuture<T> implements Future<T>, ... {
  /**
   * Forcibly sets or resets the value subsequently returned
   * by method {@link #get()} and related methods, whether or
   * not already completed. This method is designed for use
   * only in error recovery actions...
   */
  public void obtrudeValue(T value) {
    ...
  }

  /**
   * Forcibly causes subsequent invocations of method
   * {@link #get()} and related methods to throw the given
   * exception, whether or not already completed. This
   * method is designed for use only in error recovery
   * actions...
   */
  public void obtrudeException(Throwable ex) {
    ...
  }

}
```

Promise στην Javascript (ES6)

```
var handlerFunction = function(resolve, reject) {
  //resolve is the function to call in case
  //of successful completion

  //reject is the function to call in case of
  //failure
};
var promise = new Promise(handlerFunction);
promise.
       then(someFunction).   //gets the resolved value
       catch(errorFunction); //gets the rejected value
```

Με το νέο συντακτικό (arrow functions)

```
var promise = new Promise((resolve, reject) => {
  try {
    //perform a task (usually asynchronous)
    resolve(task.result);
  }
  catch(e) {
    reject(e);
  }
});
```

Escaping from the callback hell

```
getData = function(param){
  return $.get('http://example.com/get/'+param);
}

getData(0).
  then((a) => getOtherData(a)).
  then((b) => getOtherOtherData(b)).
  then((c) => {
    ...
  });
```

Παράδειγμα

`JQuery.ajax(...)` as well as the new stanard `fetch` facility return a promise.

```
function fetchJsonData(){
  return fetch('http://example.com/data.json').
            then(response => response.json());
}

function processJsonData() {
  fetchJsonData().
    then((json) => {
     // process json data
    }).
    catch(error => console.error(error));
}
```

Async / await

- Συντακτική ευκολία της ES6
- Κάνει τις ασύγχρονες κλήσεις να μοιάζουν σαν σύγχρονες
- `async` marks a function that performs an asynchronous call and returns a promise
- `await` waits for the completion of the underlying `Promise` (resolved or rejected) and is used inside async functions

Παράδειγμα

Μια async συνάρτηση "φαίνεται" σαν να κάνει await ένα promise

```
async function processJsonData() {
  try {
    const json = await fetchJsonData();
    // process json data
  }
  catch(error) {
    alert(error);
  }
}
```

Pull vs Push mechanisms

Iterator / Iterable

Σκοπός: Να μοντελοποιήσουμε την επανάληψη ή τη δυνατότητα αυτής

Iterator

java.util.Iterator

```
interface Iterator<E> {
  boolean hasNext();
  E next();
  void remove(); //we don't care about this in the class
}
```

Χρήση

```
while(iterator.hasNext()) {
  Element e = iterator.next();
  //do something with e
}
```

Iterable

java.lang.Iterable

```
interface Iterable<E> {
  Iterator<E> iterator();
}
```

Pull paradigm

- Ο χρήστης/client του Iterable κάνει pull για την επόμενη τιμή (καλεί τις hasNext/next)

Observer / Observable

- Σκοπός: Να ενημερωνόμαστε για τις αλλαγές στην κατάσταση ενός αντικειμένου
- Σχέση 1-N (1 observable, πολλοί observers)
- Παρόμοια λογική με events & event handling

Observable (Subject)

java.util.Observable

```
class Observable {
  void addObserver(Observer o);
  void deleteObserver(Observer o);
  boolean hasChanged();
  void notifyObservers();
  void notifyObservers(Object arg);
}
```

Observer

java.util.Observer

```
interface Observer {
  void update(Observable o, Object arg);
}
```

Observable σε Javascript

Με το knockout.js

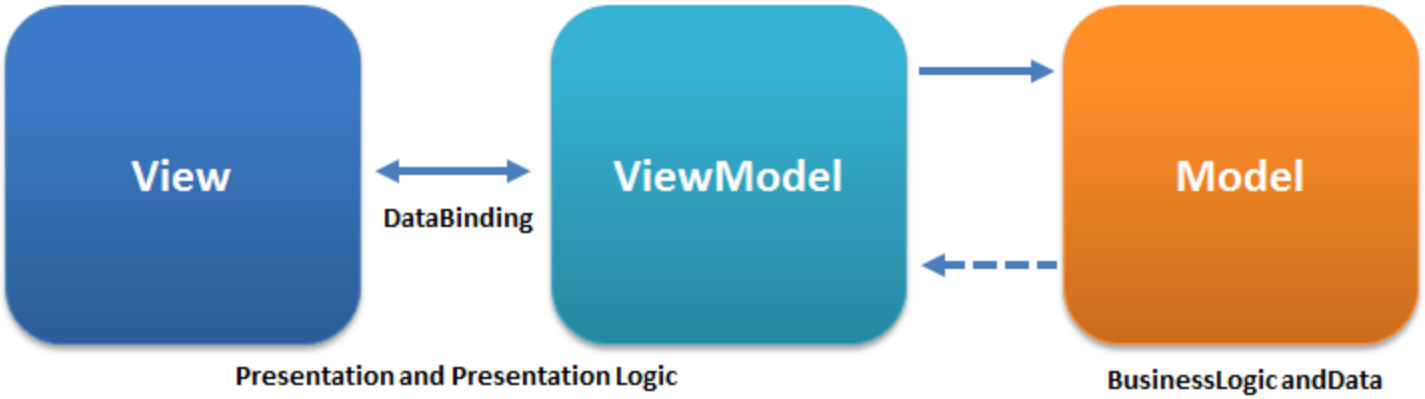
```
var name = ko.observable("Costas")
name.subscribe(function(newName) {
        // the observer is the function
        console.log("The new name is " + newName);
})

name("Kostas");
// output: The new name is Kostas
```

Push paradigm

- Ο χρήστης/client του Observable (δηλ. ο Observer) ενημερώνεται "αυτόματα" για την επόμενη τιμή (το Observable κάνει push)
- Publish/Subscribe

Το MVVM πρότυπο

Παραλλαγή του Model-View-Controller



Ας θυμηθούμε το MVC (βλ. σχετική διάλεξη)

Restful Web APIs

- URL Addressable endpoints που παρέχουν μια "καθαρή" κωδικοποίηση των δεδομένων (χωρίς HTML), συνήθως σε μορφή JSON.
- Υλοποίηση μέσω παραλλαγής του MVC, όπου το View component αποτελεί έναν "κωδικοποιητή" (serializer, encoder) των δεδομένων του Model.
- Η "αρμοδιότητα" της παραγωγής του HTML κώδικα μεταφέρεται στο client-side (client-side rendering).

Παράδειγμα (Restful Controller)

```
@Controller(url='/items')
class ItemController {

  void get(Request req, Response res) {

        // process the input request
        Filter filter = readFilter(req)

    // load the data (model_
    List<Item> items = store.loadItems(filter)

    //encode the items
    Encoder.generateResponse(res, "json", items)
  }

}
```

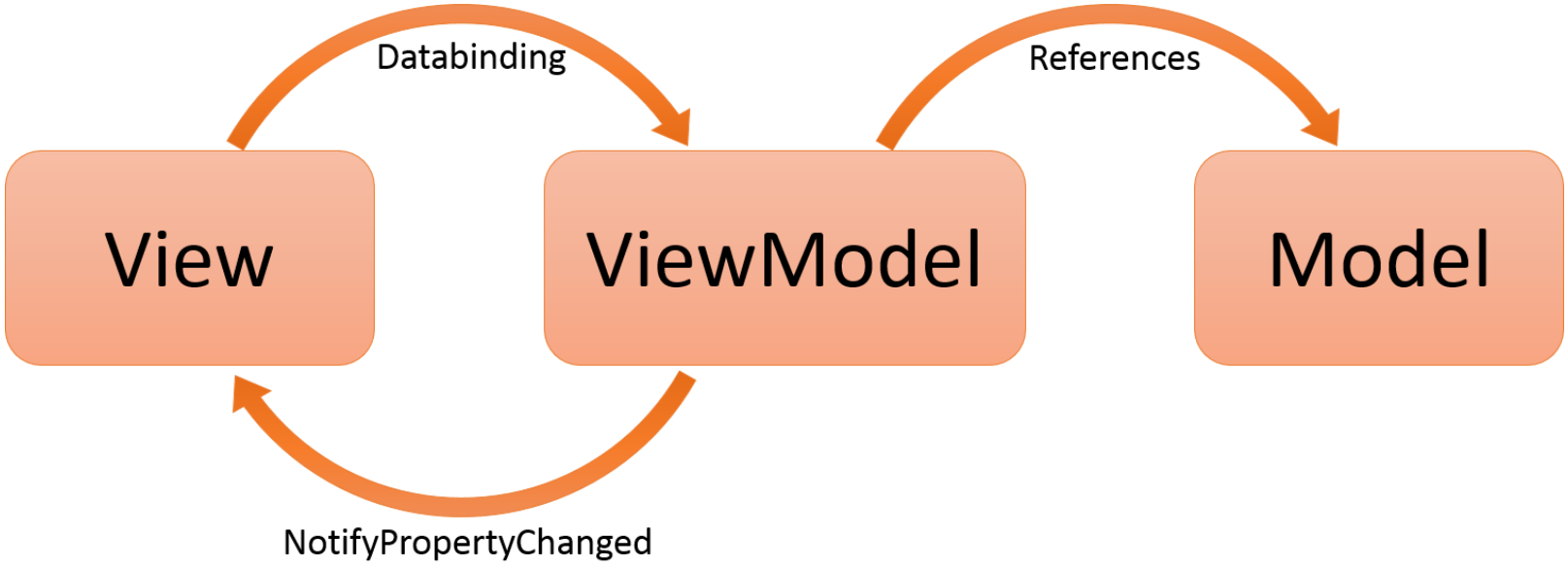
Δεν υπάρχει "View"!

Συστατικά

- View: τα στοιχεία του UI
- Model: τα δεδομένα
- ViewModel (Presenter ή ViewController):
  - Data bindings (UI elements as observables of the data model elements)
  - Change notifications (two-way data bindings)

Εφαρμογή του Observable στο MVVM

- ViewModel: Ενθυλακώνει τα δεδομένα (model elements) σε observables
- Τα UI elements γίνονται bind ως observers στα model elements


Databinding
References
View
ViewModel
Model
NotifyPropertyChanged

Παράδειγμα

http://knockoutjs.com/examples/

Συζήτηση

- Δηλωτικός κώδικας
- Υψηλό επίπεδο αφαίρεσης
- Αυτόματη -για τον προγραμματιστή- ενημέρωση του UI με τις αλλαγές στα δεδομένα
- Λίγο σύνθετο για απλές διεπαφές χρήσης (αλλά αξίζει τον κόπο)
- Ενδεχόμενα θέματα απόδοσης σε πολύ μεγάλες εφαρμογές ή σύνολα δεδομένων

Front-end frameworks

- Angular (two-way data bindings)
- Vue (two-way data bindings)
- React (one-way data flows: model -> UI)