Internet Applications Design and Implementation

(Lecture 9 - Modelling Interface & Interaction - React meets IFML)

MIEI - Integrated Master in Computer Science and Informatics
Specialization block

João Costa Seco (joao.seco@fct.unl.pt)
Jácome Cunha (jacome@fct.unl.pt)
João Leitão (jc.leitao@fct.unl.pt)

NOVA SCHOOL OF
SCIENCE & TECHNOLOGY

How do you specify interaction in a user interface?

How do you specify its views?

How do you specify navigation actions?

How do you specify the data needed?

From navigation? from server?

You don’t!

Most people use mocks only! or worse…

They just code ahead!

go to here

<form action=“jumptohere”…>…</form>

Outline

- Interaction Flow Modelling Language (IFML)
- IFML by example
- From IFML to React
- From User Stories to IFML to React

Internet Applications Design and Implementation

(Lecture 9 - Part 1 - IFML)

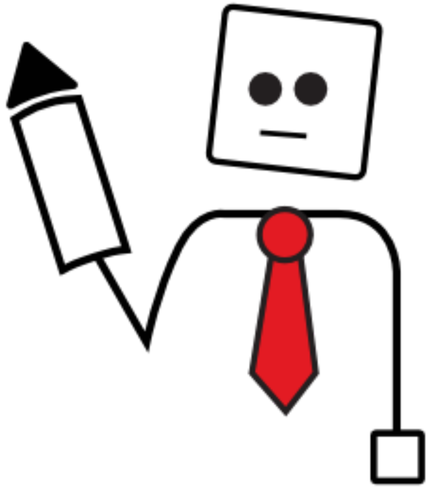
MIEI - Integrated Master in Computer Science and Informatics
Specialization block

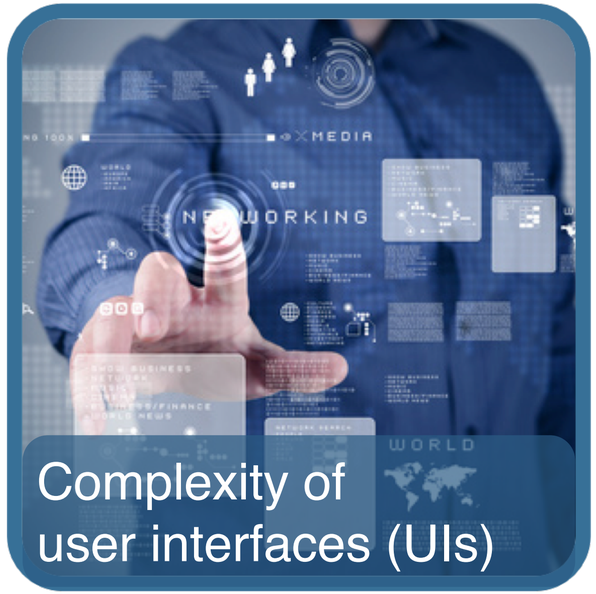
João Costa Seco (joao.seco@fct.unl.pt)
Jácome Cunha (jacome@fct.unl.pt)
João Leitão (jc.leitao@fct.unl.pt)

NOVA SCHOOL OF
SCIENCE & TECHNOLOGY

The UI Design Problem

Costly and
Inefficient process





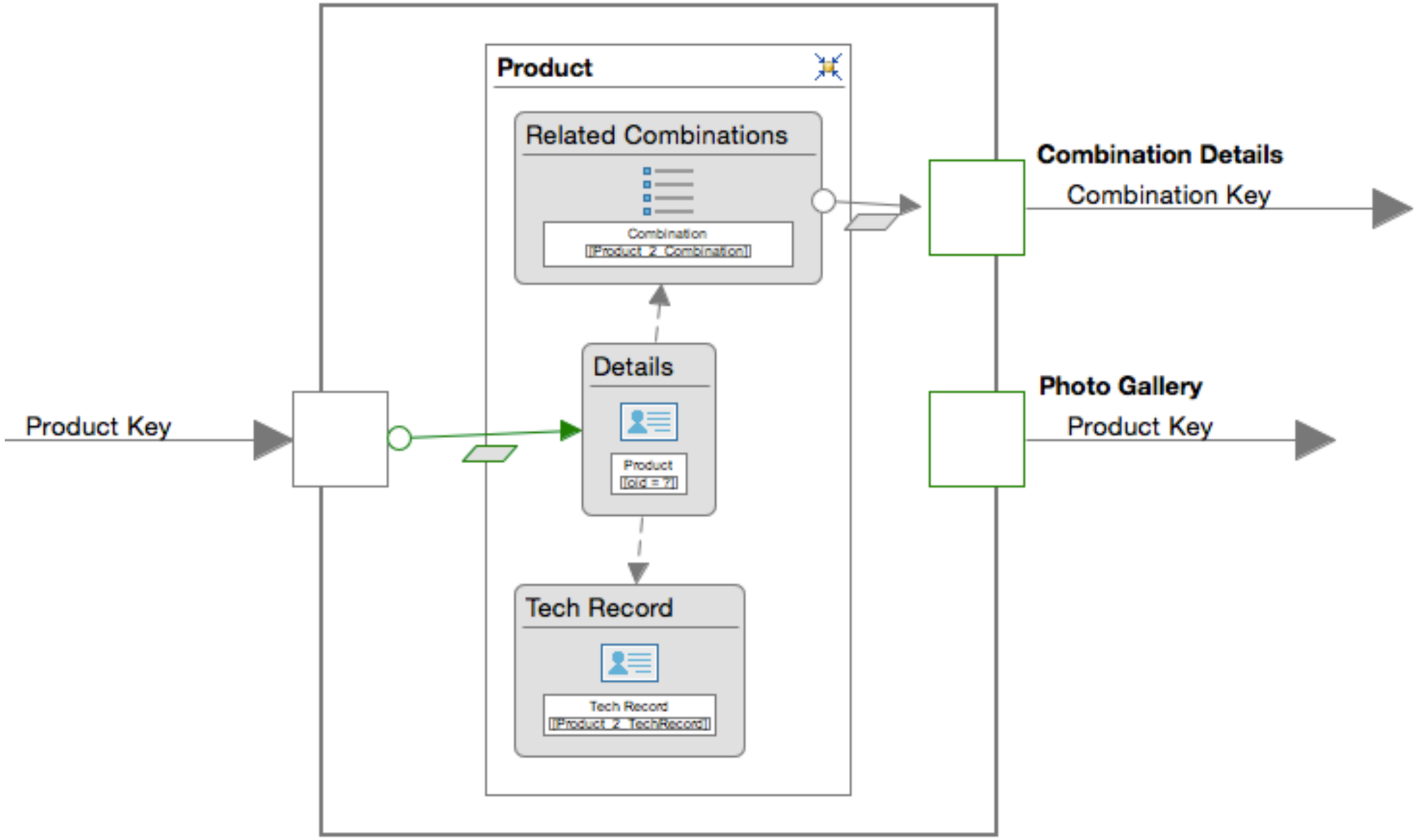
Complexity of
user interfaces (UIs)

Ineffective
tools

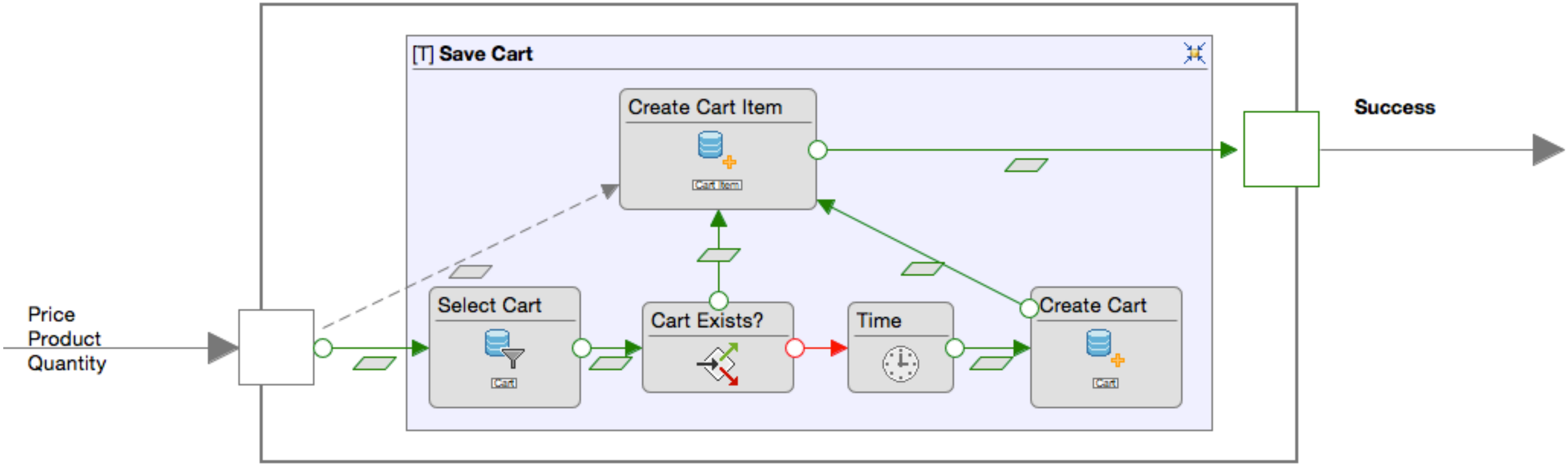
Manual
development

No MDE
technology

Solution: Abstraction of content and flow



Solution: Abstraction of content and flow



The Interaction Flow Modeling Language

The new OMG Standard for integrating the front-end design in your system and enterprise models

- OMG (Object Management Group) standard (since 03/2013)

- To express content, user interaction, and control behavior of front-end apps

Practical Results of Having a Standard

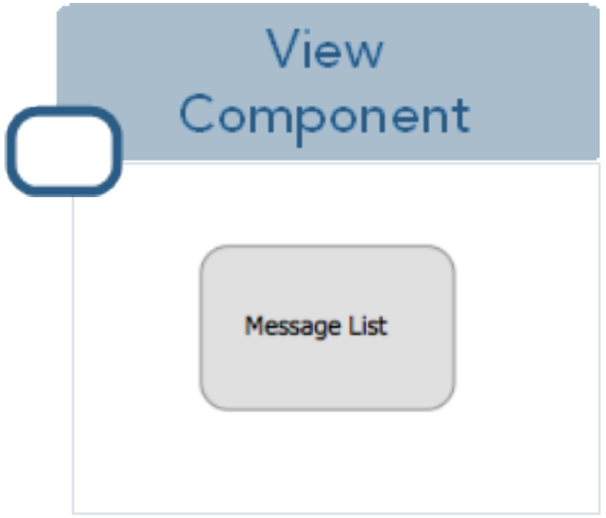
- An official metamodel/grammar of the language which describes the semantics of and relations between the modelling constructs
- A graphical concrete syntax for the interaction flow notation which provides an intuitive representation of the user interface composition, interaction and control logic for the front-end designer
- A UML Profile consistent to the metamodel
- An interchange format between tools using XMI
- All this, specified through standard notations themselves

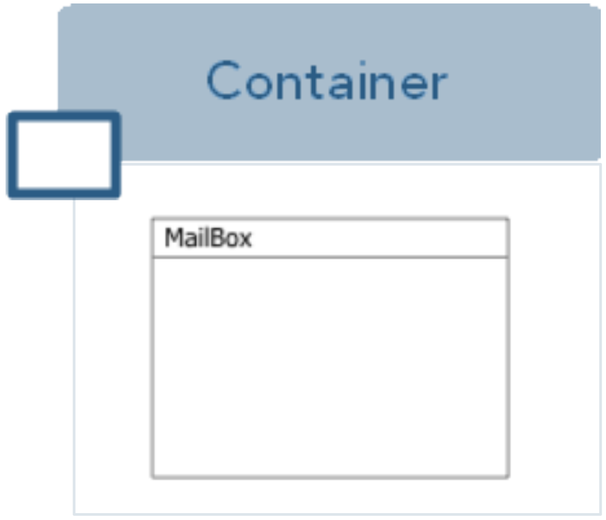
The UI Design solution: IFML

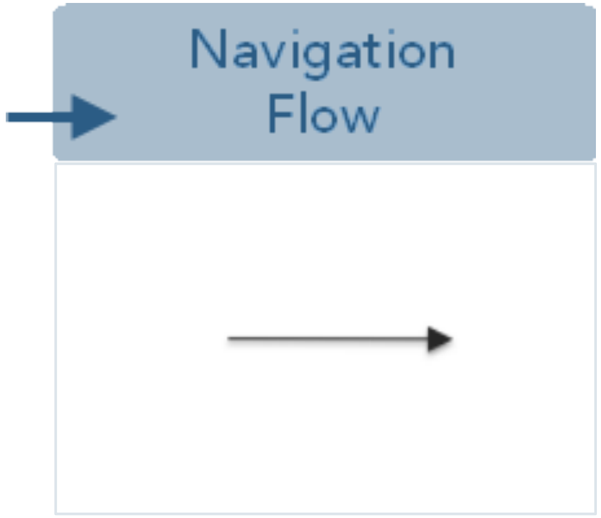
Covered aspects

- Multiple views for the same application
- Visualization and input of data, and production of events
- Components independent of concrete widgets and presentation
- Interaction flow, initiated by the user or by external events
- Modularization of the model (design-time containers for reuse purpose)
- User input validation and constraints, according to OCL or other existing constraint languages

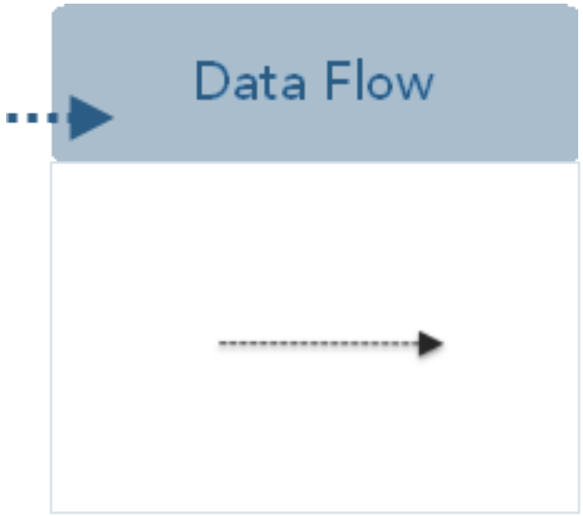
IFML Essentials

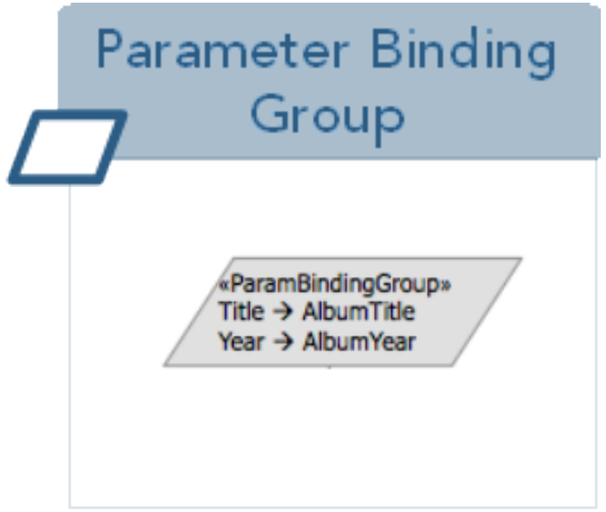


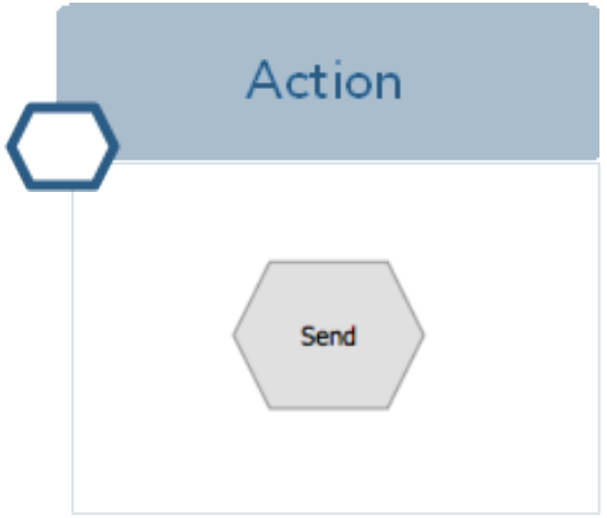




https://www.omg.org/spec/IFML/1.0/PDF







Internet Applications Design and Implementation

(Lecture 9 - Part 2 - IFML by Example)

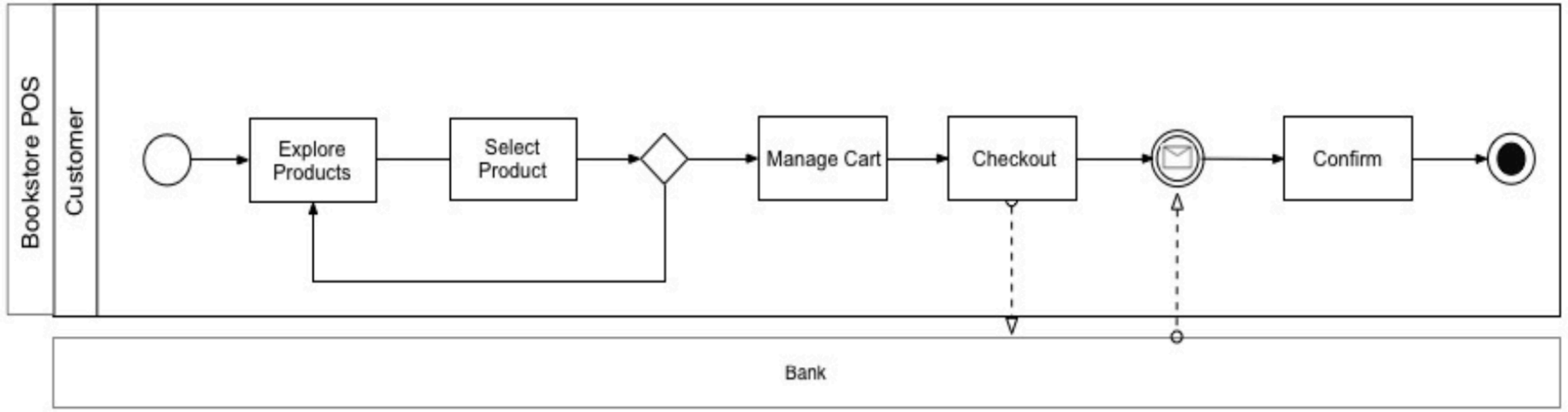
MIEI - Integrated Master in Computer Science and Informatics
Specialization block

João Costa Seco (joao.seco@fct.unl.pt)
Jácome Cunha (jacome@fct.unl.pt)
João Leitão (jc.leitao@fct.unl.pt)

NOVA SCHOOL OF
SCIENCE & TECHNOLOGY

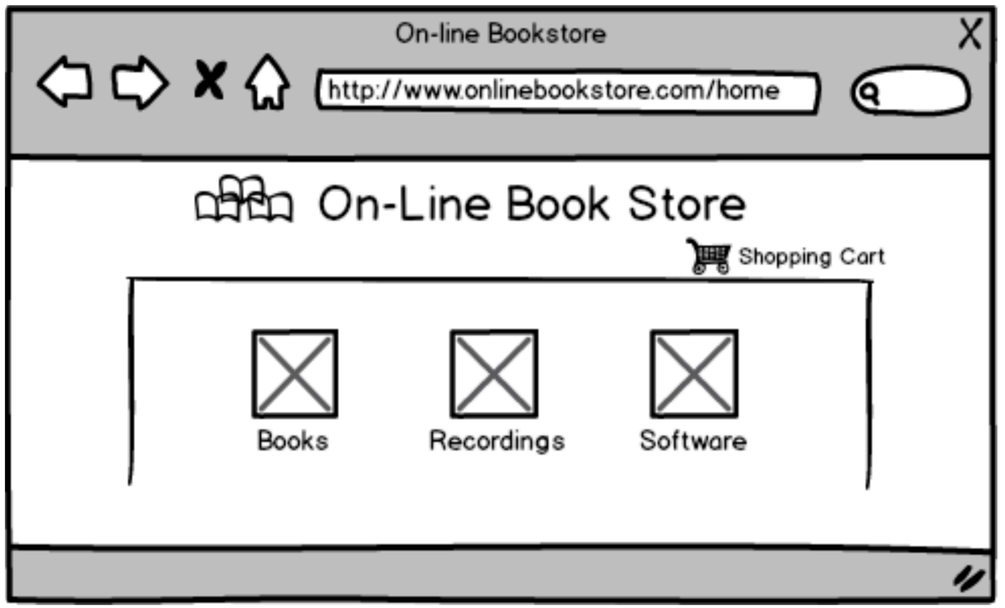
The design of UI usually starts with mockups (ifml.org)

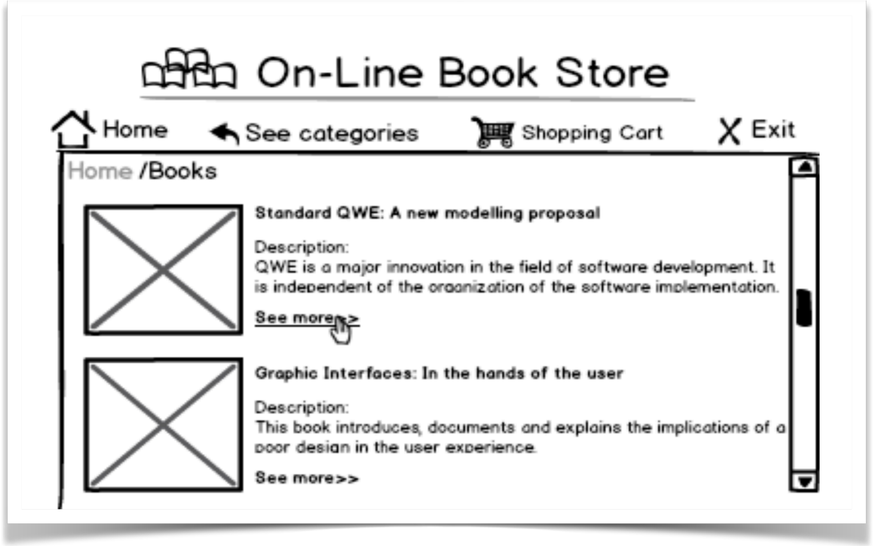
- IFML defines the behaviour and helps defining the components of UI parts.
- Start with a scenario of a book store and its shopping cart. The UI starts with a list of categories, products, and the corresponding details.
- The shopping process continues by asking user detailed information and connecting to the bank to process the payment.

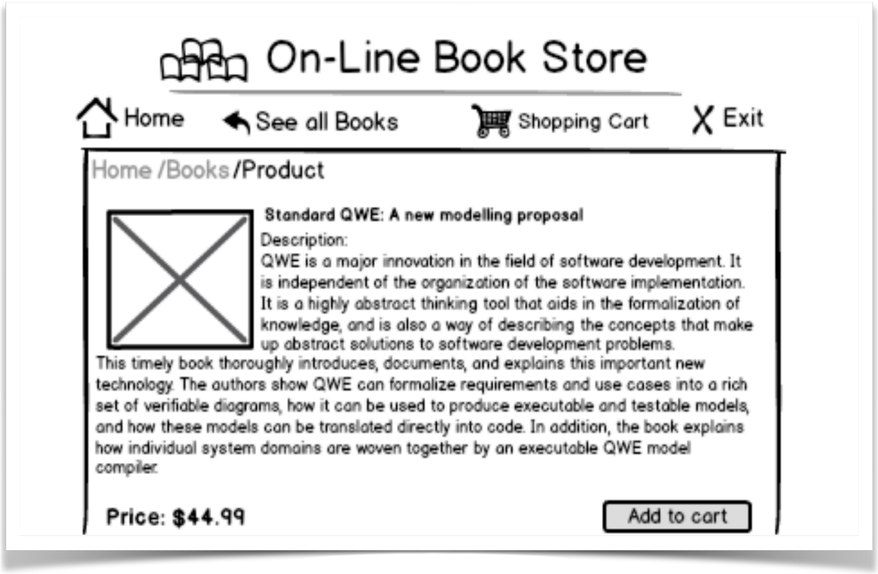


https://www.ifml.org/wp-content/uploads/IFML-Bookstore-Example.pdf

IFML by Example



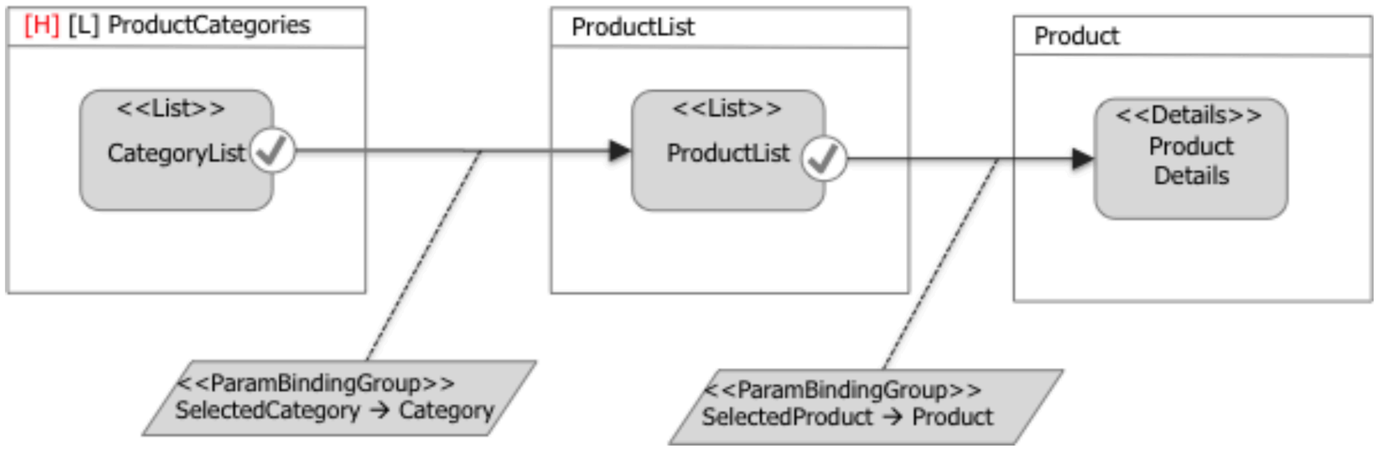




Consider the main screens of a sample application.
Only visual structure is defined.

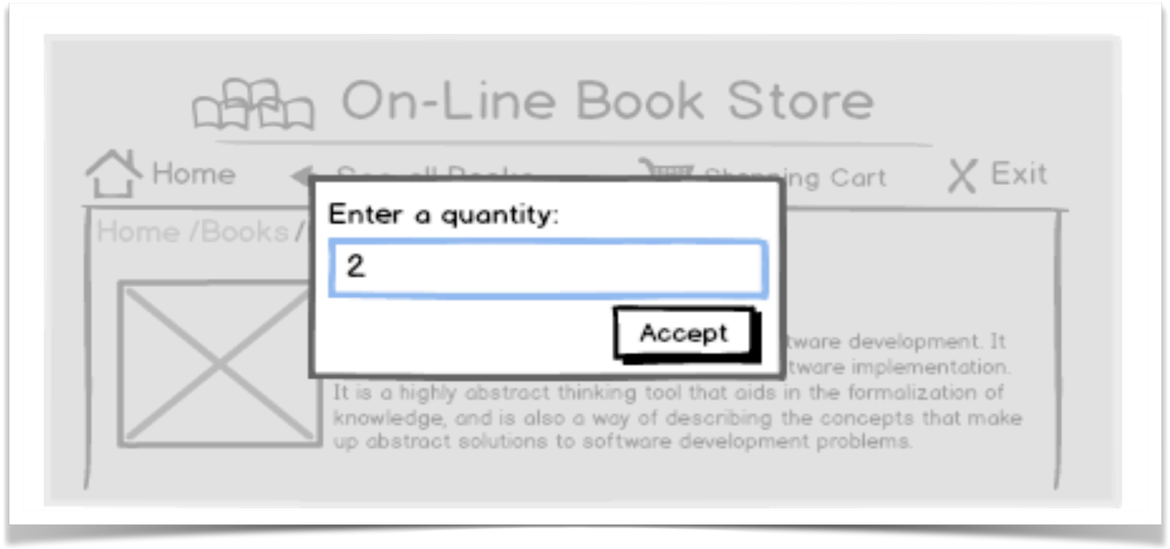
https://www.ifml.org/wp-content/uploads/IFML-Bookstore-Example.pdf

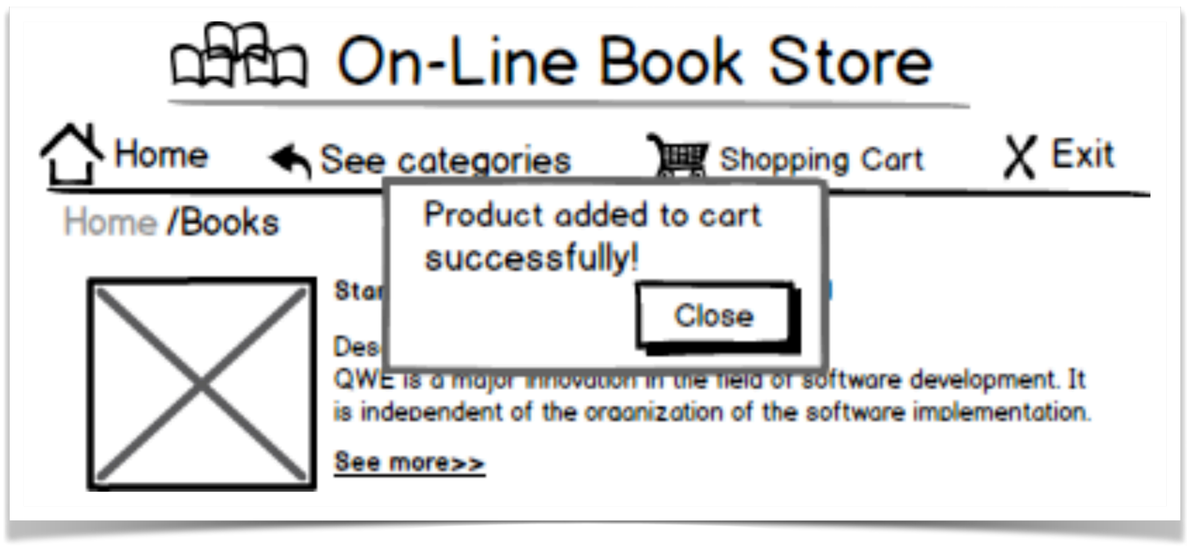
IFML navigation between pages



Flow between screens is defined using IFML. Data transmitted is also defined here.

Events, interactions, actions

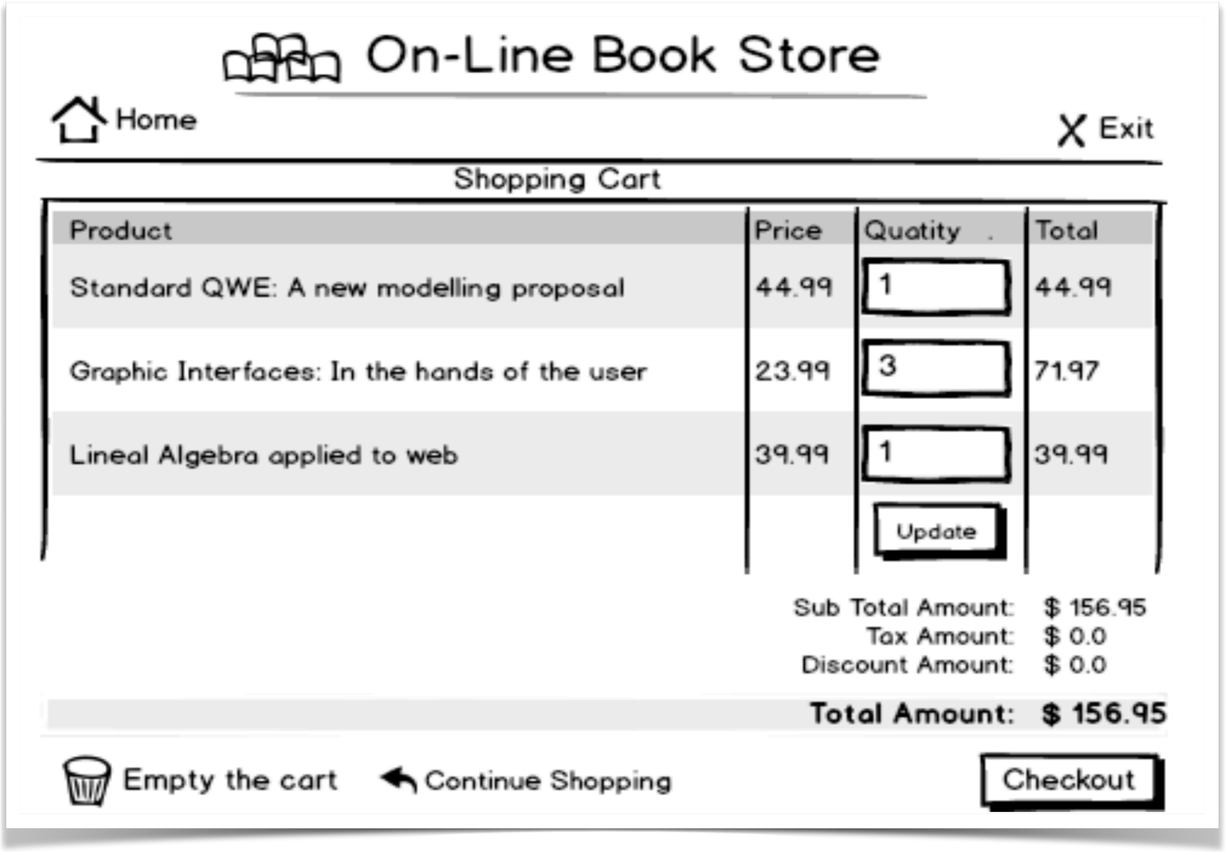




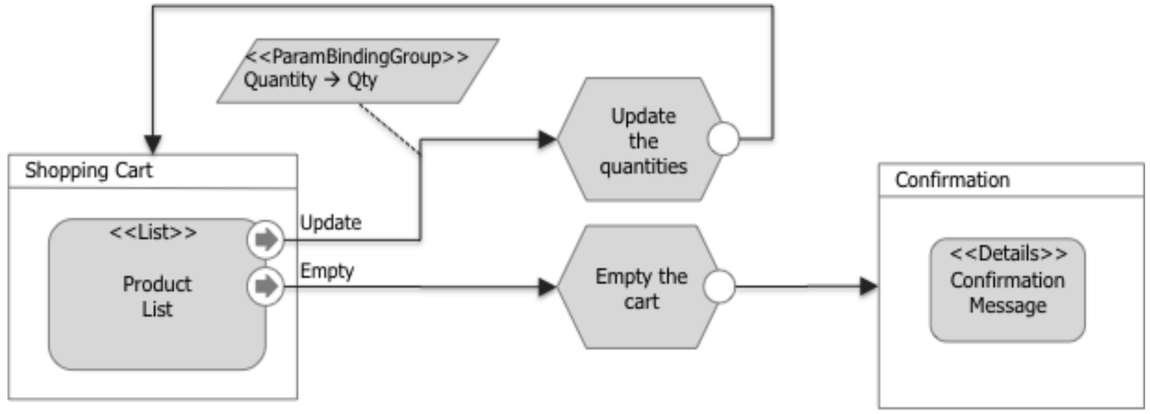
When a user “buys” a book

The event in the “Add to cart” button

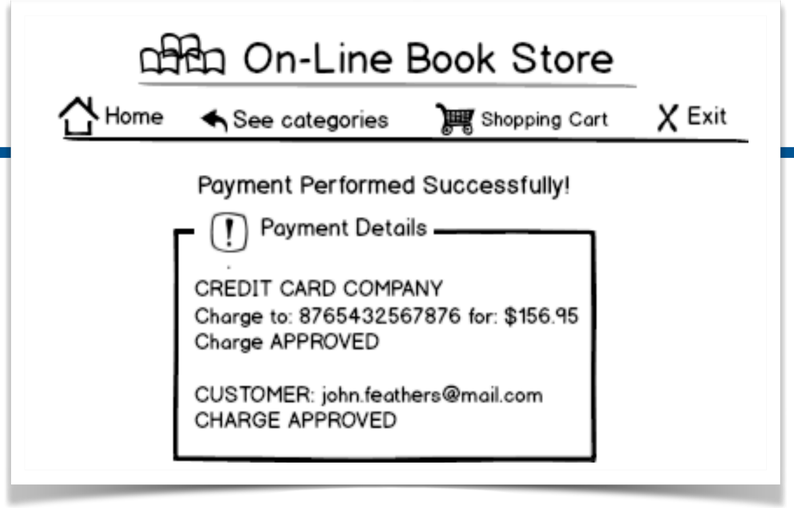
Events - Interactive interfaces

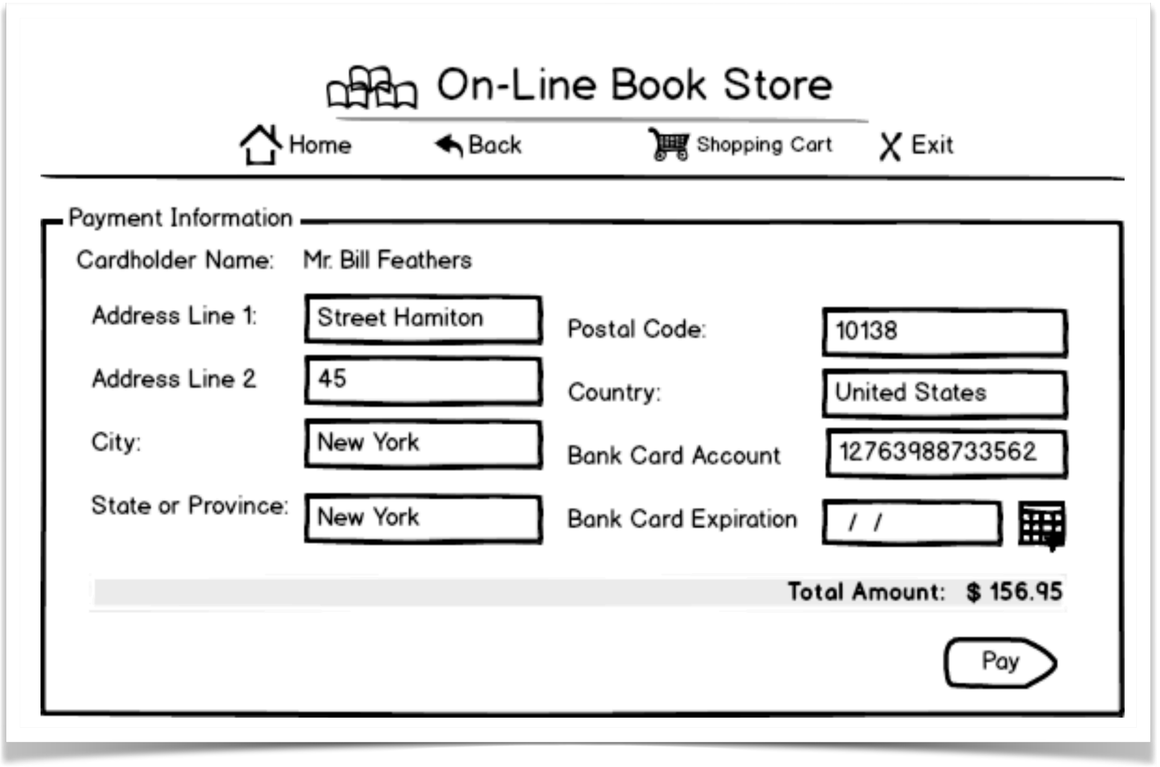


When landing on the shopping cart page

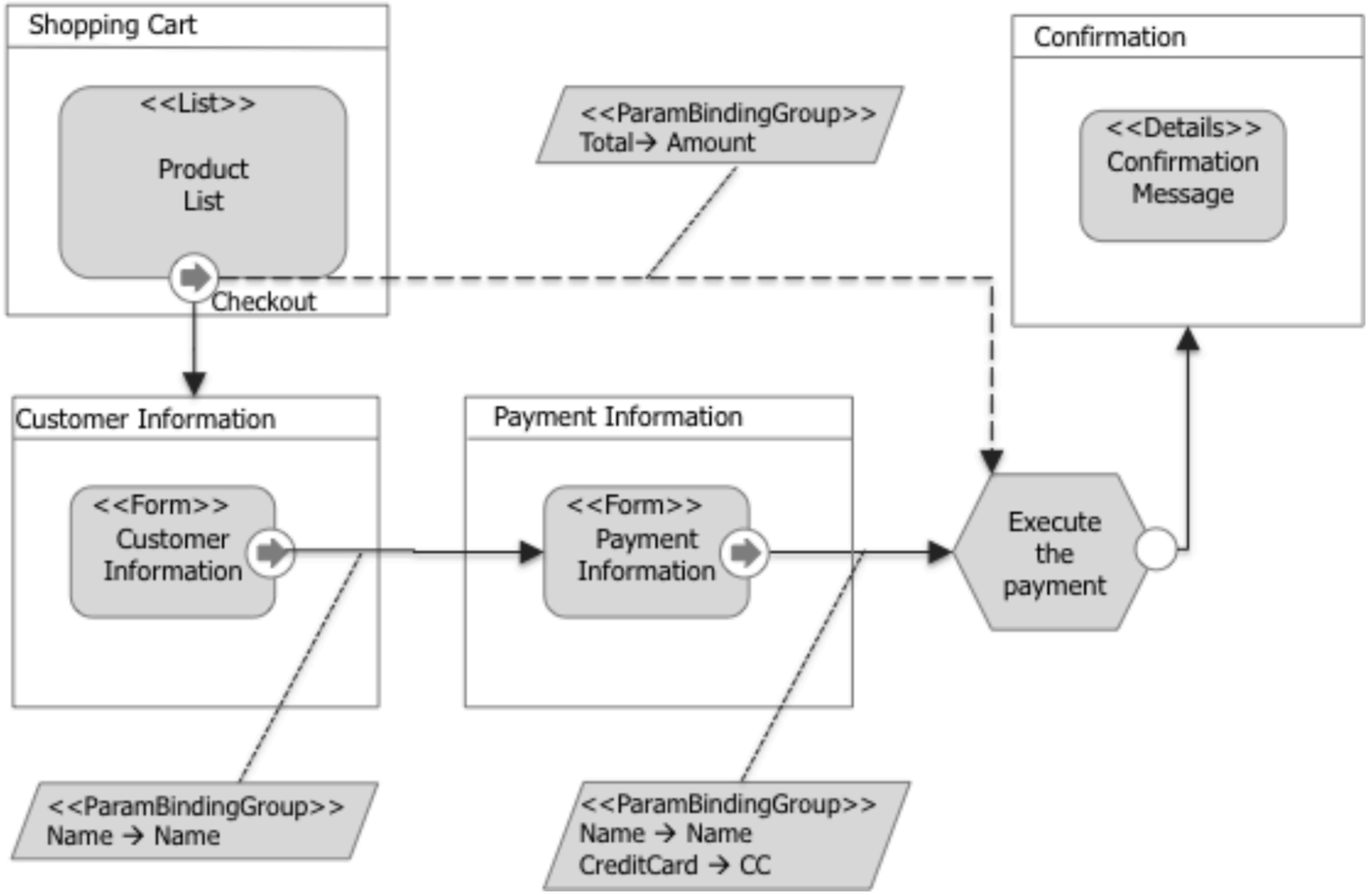


Events - Interactive interfaces

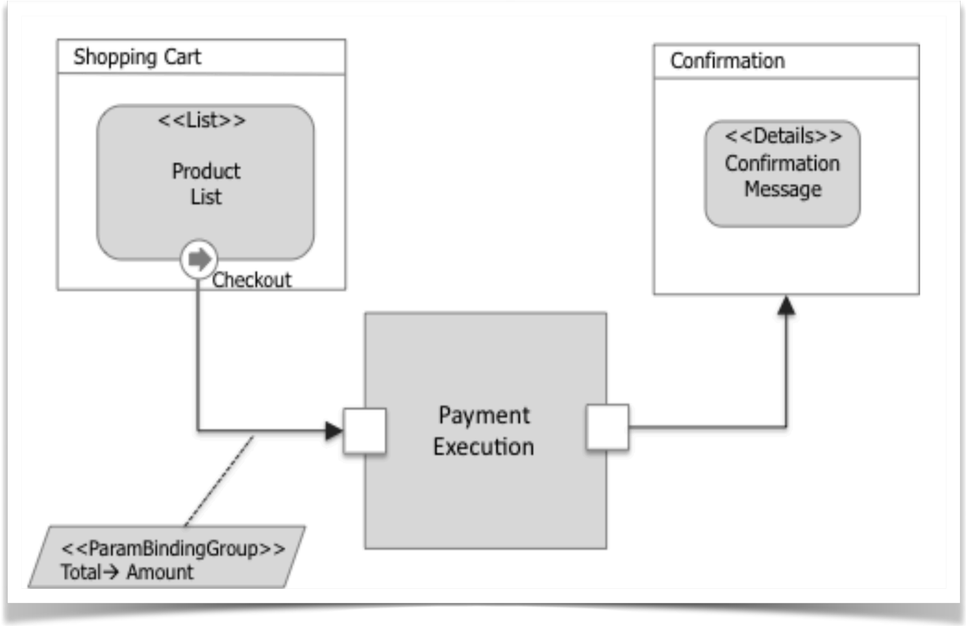


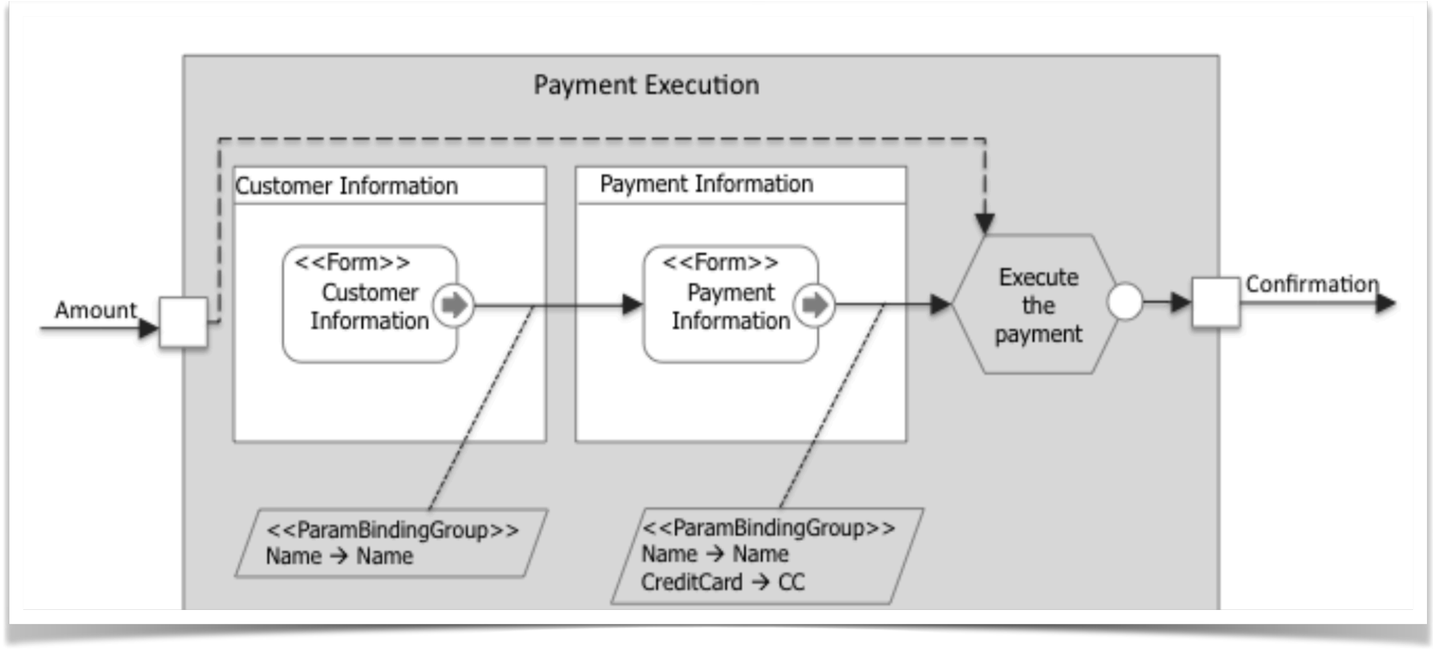


Events - Interactive interfaces



Composition and subcomponents





Summary

- Mock up screens and components are the main design tool for user interfaces
- IFML complements mock ups with behaviour: data passing, event handling, action triggering
- IFML is hierarchical and compositional

Internet Applications Design and Implementation

(Lecture 9 - Part 3 - From IFML to React)

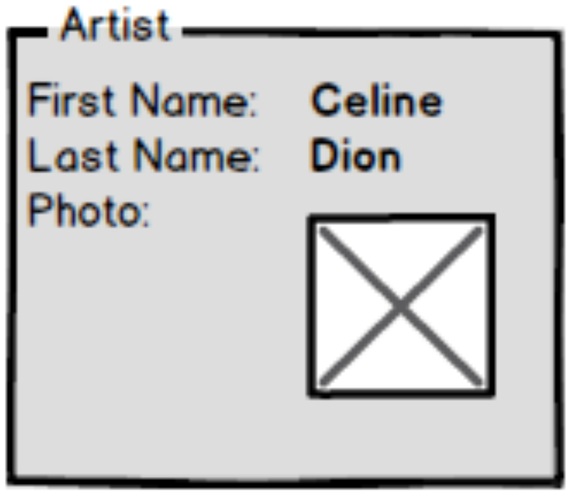
MIEI - Integrated Master in Computer Science and Informatics
Specialization block

João Costa Seco (joao.seco@fct.unl.pt)
Jácome Cunha (jacome@fct.unl.pt)
João Leitão (jc.leitao@fct.unl.pt)

NOVA SCHOOL OF
SCIENCE & TECHNOLOGY

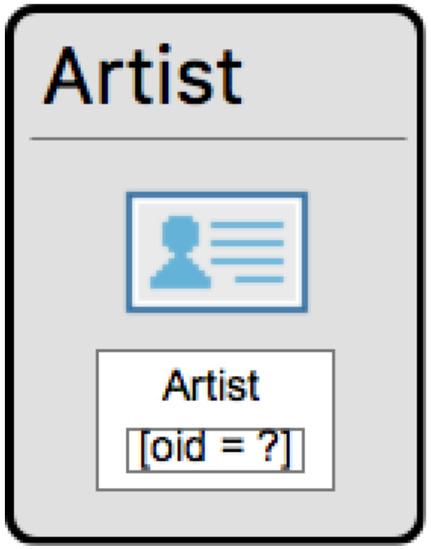
IFML Syntax – View Components: Details


```
const ArtistDetails = (props:{artist:Artist}) =>
    <div>
        <p>First Name: {props.artist.firstName}</p>
        <p>Last Name: {props.artist.lastName}</p>
        <img src={props.artist.photo} />
    </div>
```

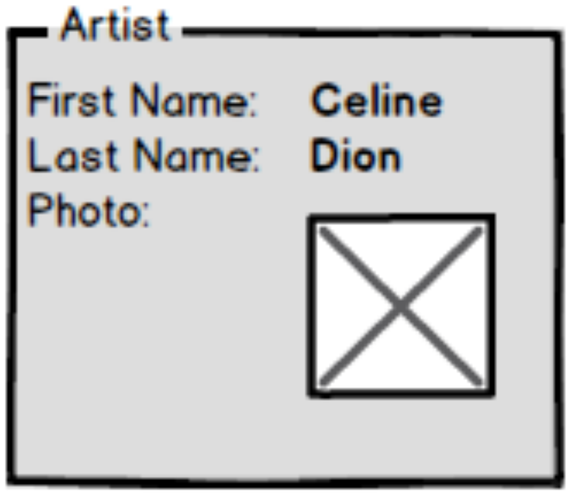


- User defines:
  - Type/Entity/Interface (Artist in this case)
  - Shown attributes (e.g. id, name, photo, etc.)
  - Value to be shown ("`artist = ?`")
- Also available *Multiple Details* and *Scroller*

IFML Syntax – View Components: Details

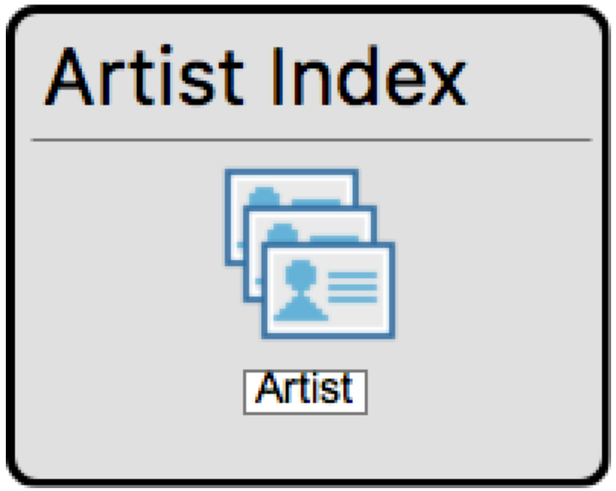



```
const ArtistDetails = (props:{artistId:number}) => {
    const [ artist, setArtist] = useState<Artist | null>(null);
    getData<Artist>(`/artists/${props.artistId}`)
        .then( data => data && setArtist(data));

    return <>
        {artist &&
            <div>
                <p>First Name: {artist.firstName}</p>
                <p>Last Name: {artist.lastName}</p>
                <img src={artist.photo}/>
            </div>}
        { !artist && <div>No artist</div>}
        </>;
};
```



- Value to be shown can be indirect
  if “oid = ?” is the parameter, then search the entry with the given oid

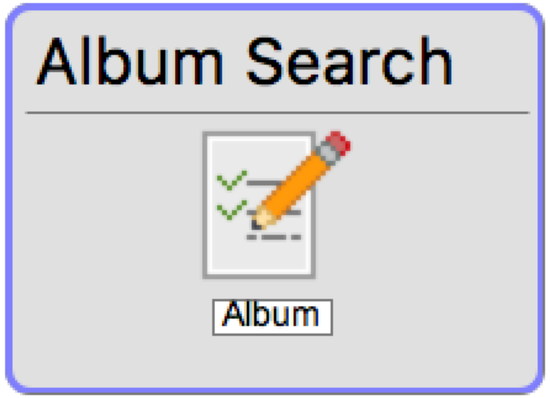
IFML Syntax – View Components: List




```
const ArtistListItem = (props:{artist:Artist}) =>
    <li key={props.artist.id}> {props.artist.firstName} {props.artist.lastName}</li>;

const ArtistIndex = (props:{artists:Artist[]}) =>
    <ul>
        { props.artists.map((artist:Artist) => <ArtistListItem artist={artist}/>)}
    </ul>;
```

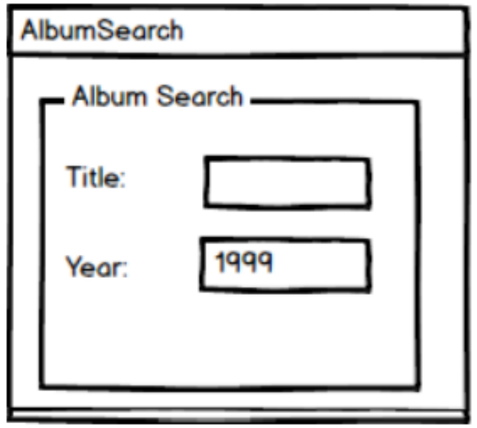
- User defines:
  - Type/Entity/Interface (Artist in this case)
  - Shown attributes (e.g. id, name, etc.)
  - Selection criteria (… where X = Y)
- Also available *Checkable List*, *Hierarchical List*, etc.

IFML Syntax – View Components: Form

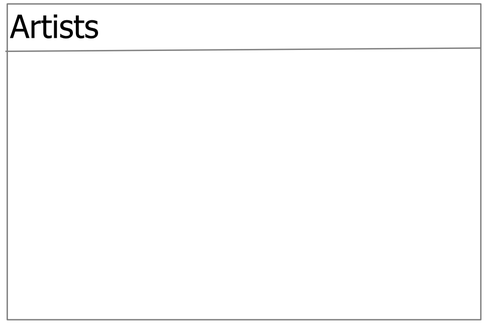


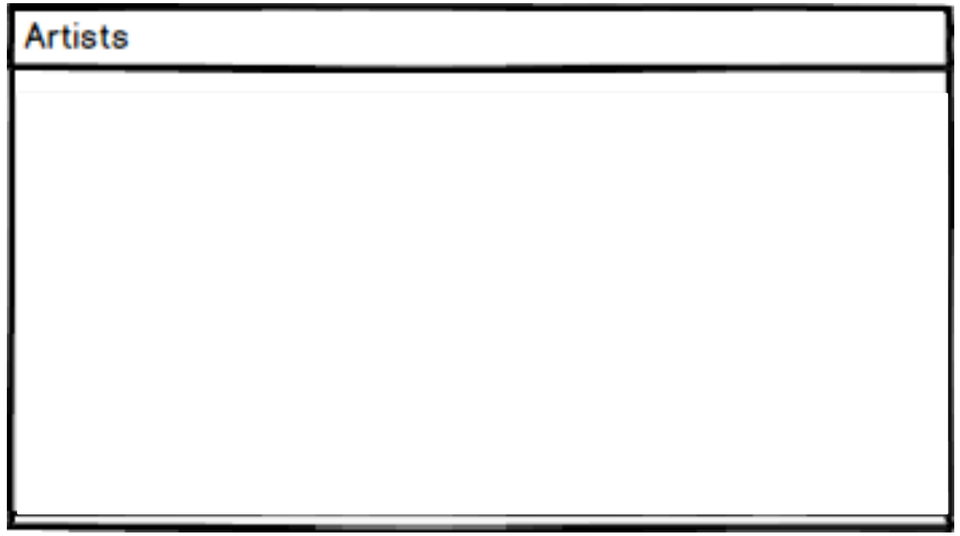

```
const AlbumSearch = (props:{loadAlbums:(title:string, year:string) => void}) => {
    const [ searchTitle, setSearchTitle ] = useState("");
    const [ searchYear, setSearchYear ] = useState("");

    useEffect(() => props.loadAlbums(searchTitle, searchYear), [searchTitle, searchYear]);

    let handleTitle = (e:ChangeEvent<HTMLInputElement>) => setSearchTitle(e.target.value);
    let handleYear = (e:ChangeEvent<HTMLInputElement>) => setSearchTitle(e.target.value);

    return <form>
            <label> Title: <input type="text" onChange={handleTitle}/></label>
            <label> Year: <input type="text" onChange={handleYear}/></label>
        </form>;
};
```

- User defines:
  - Entity, optionally (Album in this case)
  - Fields (e.g. title, year, etc.)
- Also available *Multiple Form*



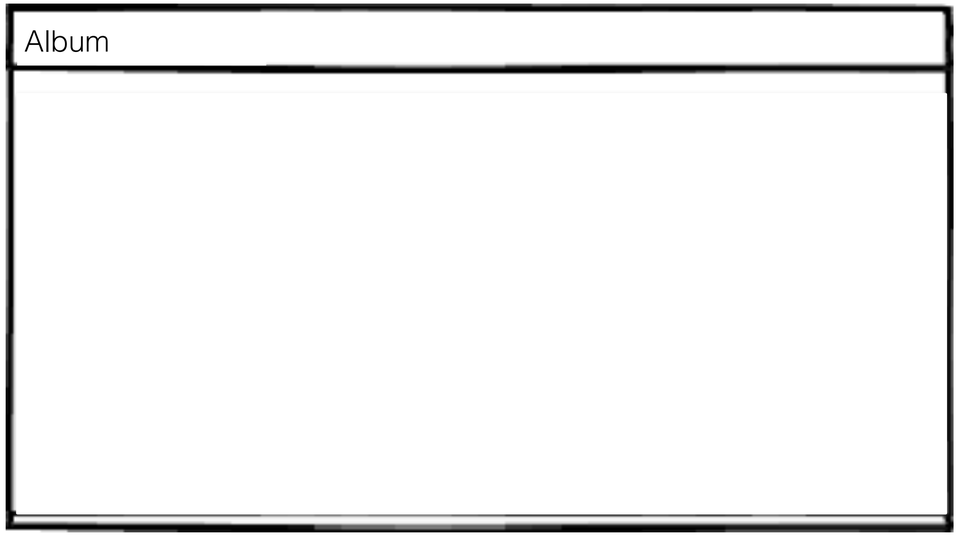
IFML Syntax – Containers: Page



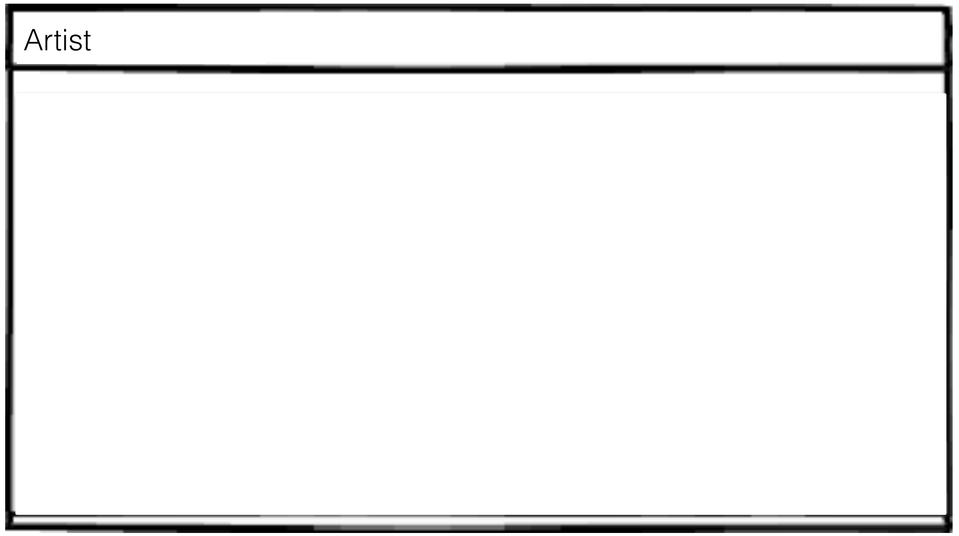


- Containers to hold other constructors (including other pages - nesting)
- Not necessarily mapped into web pages

IFML Syntax – Containers: Alternative Page



- Container to hold other pages in XOR mode
- It either contains Album or Artist, not both at the same time (depends on navigation)



IFML Syntax – (User) Navigation Flow

```
<Link to={`${this.props.id}`}>{"More details about "+this.props.firstName+" "+this.props.lastName}</Link>
```

- User defines:
  - Source (from where to navigate)
  - Target (to where to navigate)
  - Binding parameters/values
- Also available *OK Flow*, and *KO Flow*

IFML Syntax – Data Flow

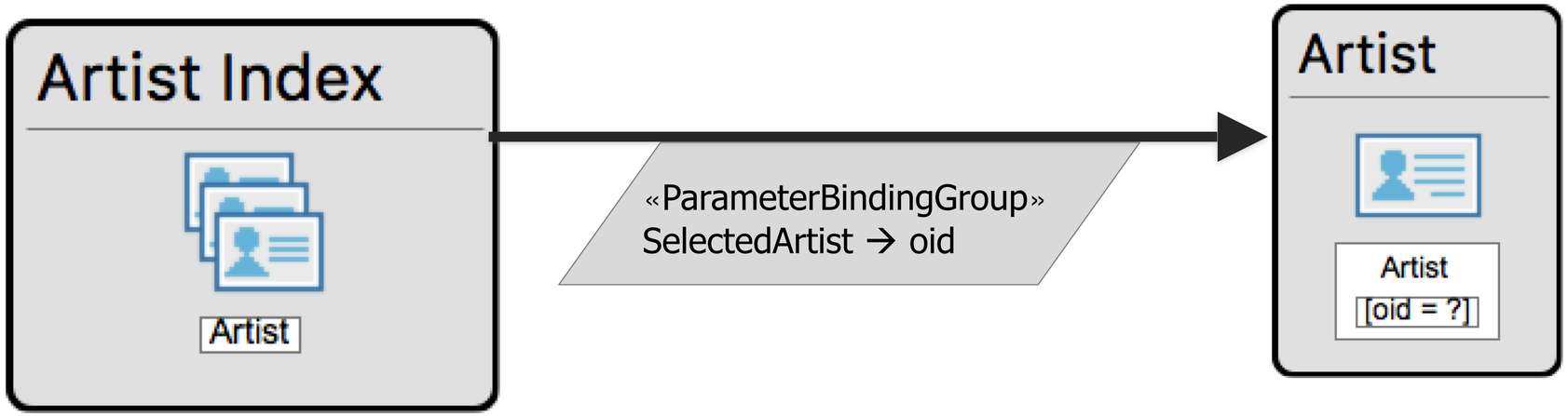
```
<ArtistDetails artist={artist} />
```

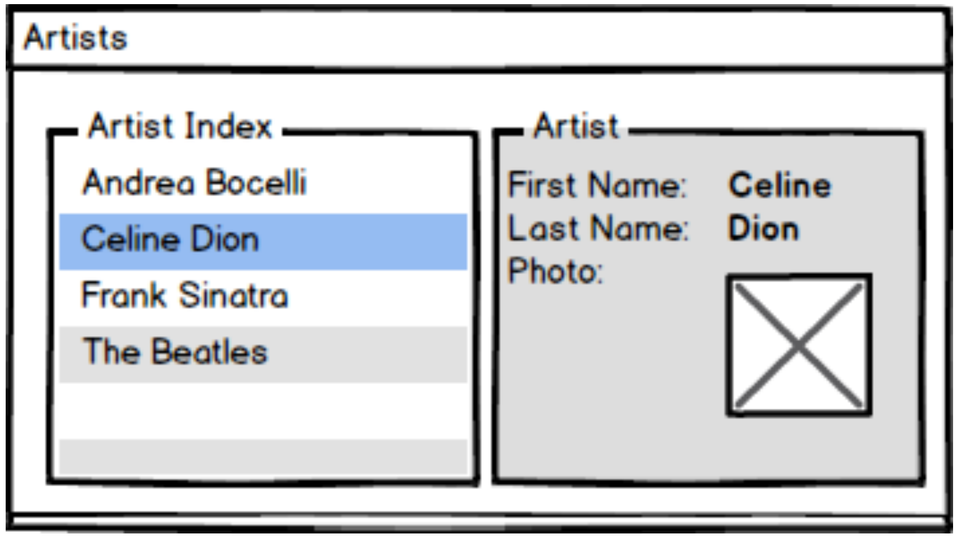
- User defines:
  - Source (from where to navigate)
  - Target (to where to navigate)
  - Binding parameters/values

IFML Syntax – Parameters Binding

- User defines:
  - Source parameter/value (SelectedArtist in this case)
  - Target parameter/value (AnArtist in this case)
  - Data flows from source to target
  - Usually associated with flows (user navigation flow or data flow)

IFML Syntax – Parameters Binding





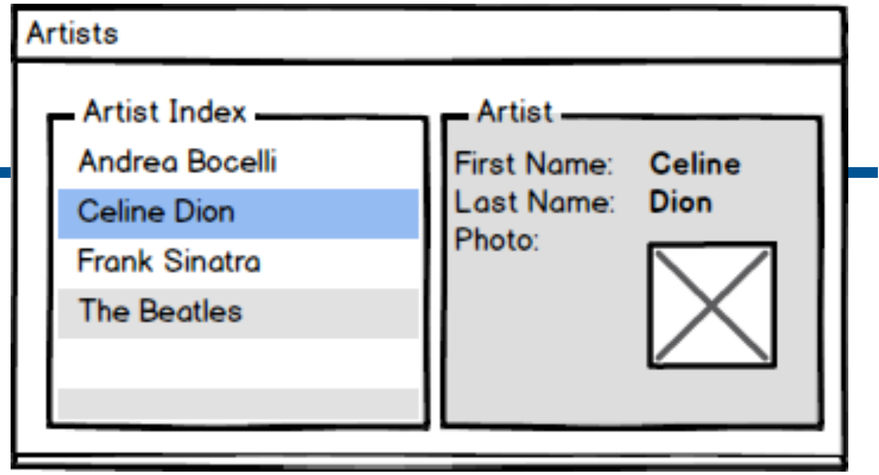
IFML Syntax – Parameters Binding

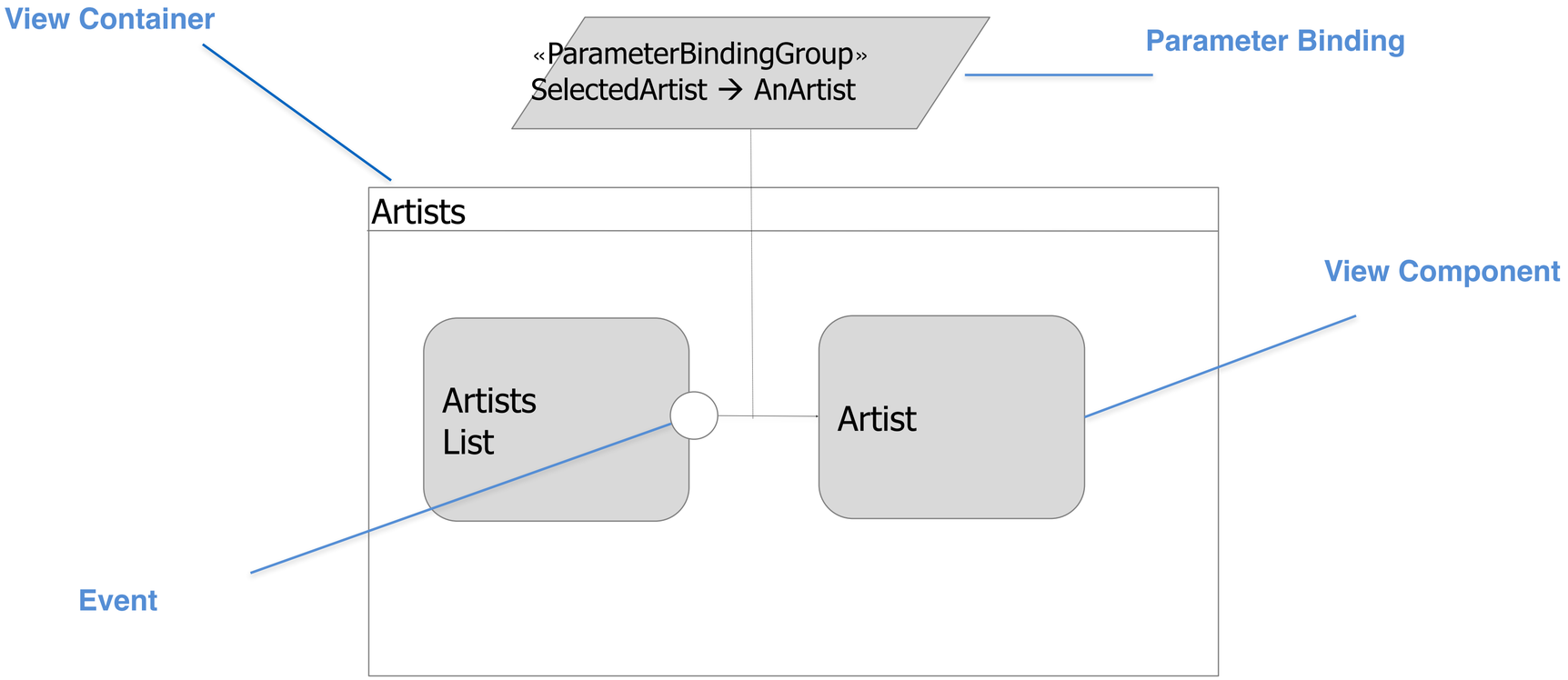
```
const ArtistIndex = (props:{artists:Artist[], selectArtist:(idx:number)=>void}) =>
    <ul>
        { props.artists.map((artist:Artist) => <ArtistListItem artist={artist}/>)}
    </ul>;

const ArtistIndexWithDetails = (props:{artists:Artist[]}) => {
    const [selectedArtist, setSelectedArtist] = useState<number | null>(null);
    return <>
            <ArtistIndex artists={props.artists} selectArtist={setSelectedArtist}/>
            { selectedArtist && <ArtistDetails artistId={artists[selectedArtist]}/> }
           </>;
};
```

IFML Syntax – Events

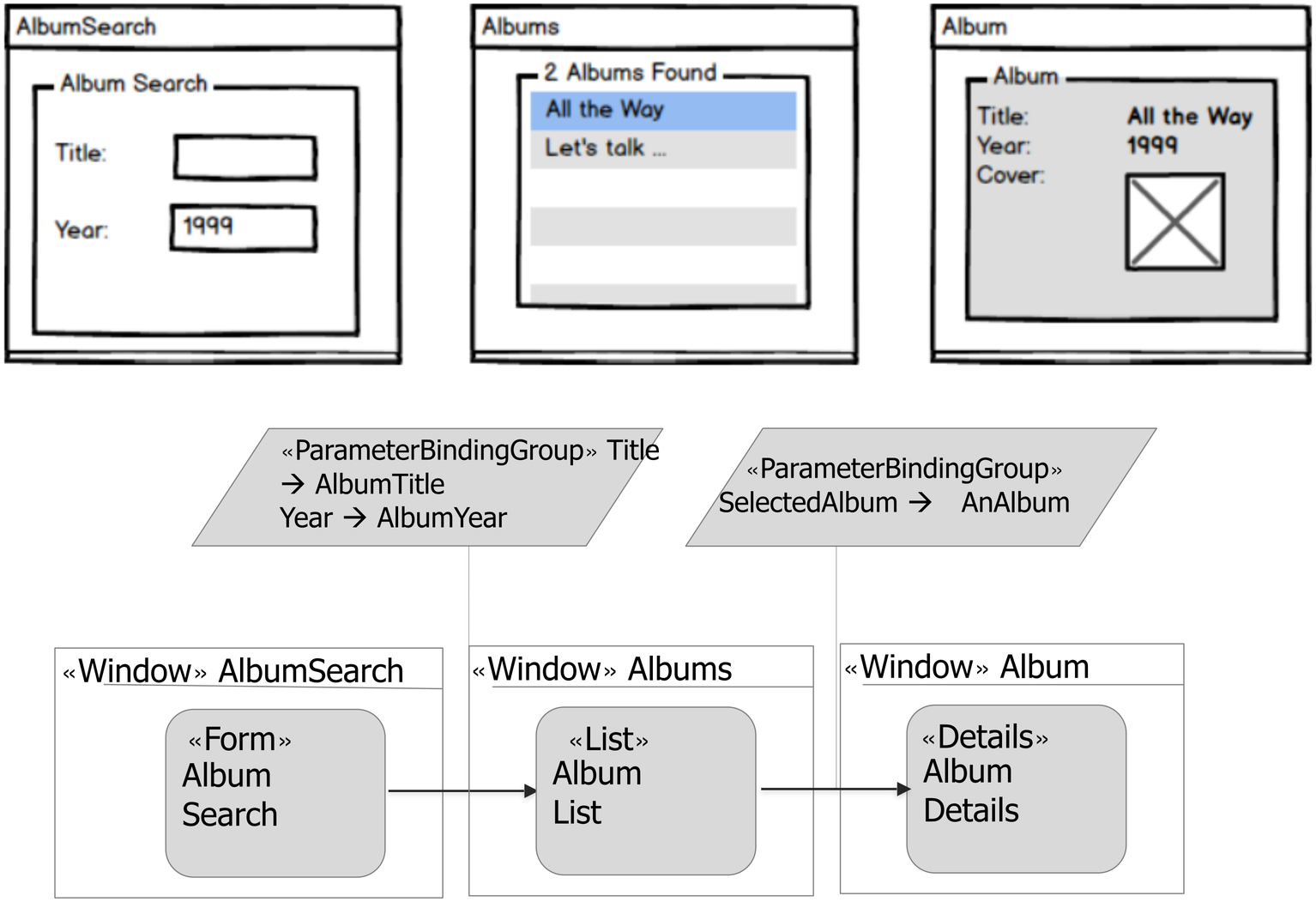
- Created when user clicks on something
  - To select an element from a list
  - To navigate
  - Etc.

♪ All Together Now, All Together Now ♪

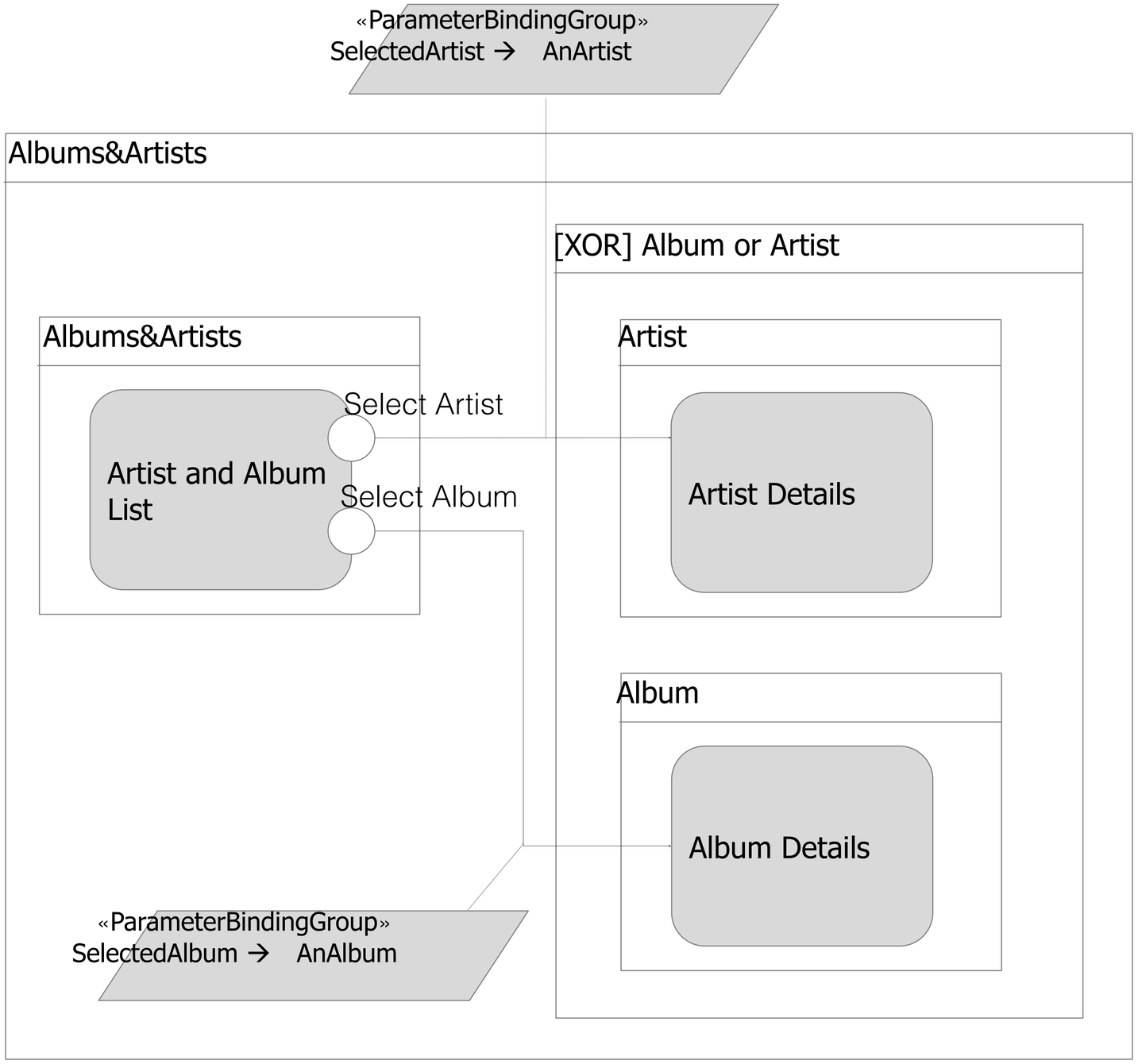




IFML by example



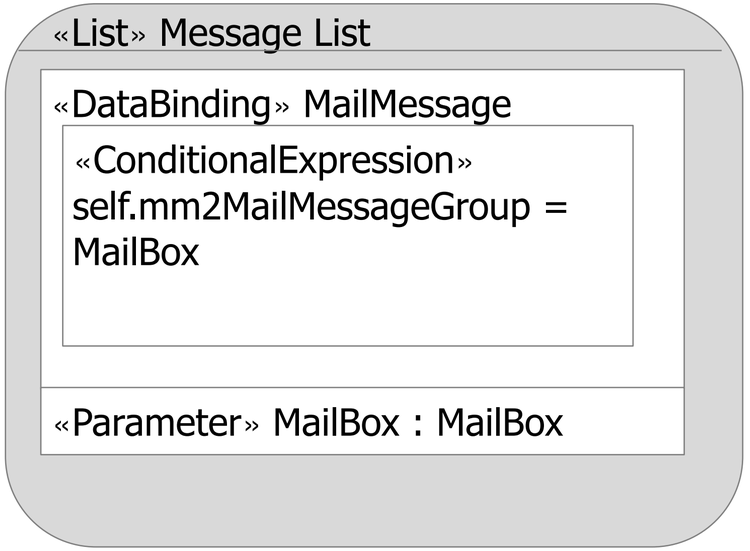
IFML by example



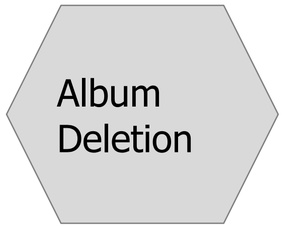
IFML – Adding Details to View Components

View Component Parts:

- Data binding
- Parameters

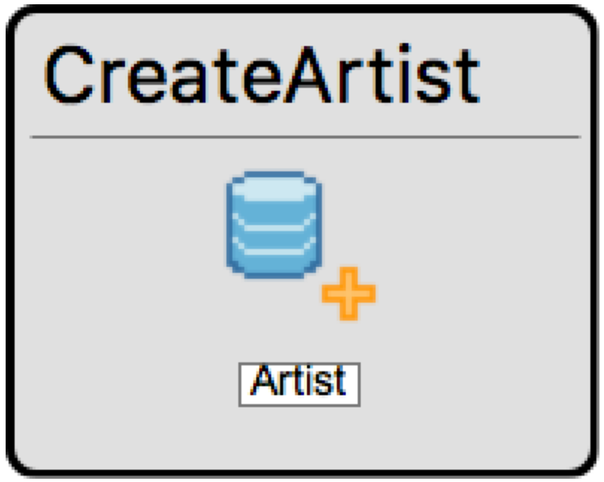


More IFML Syntax – Actions



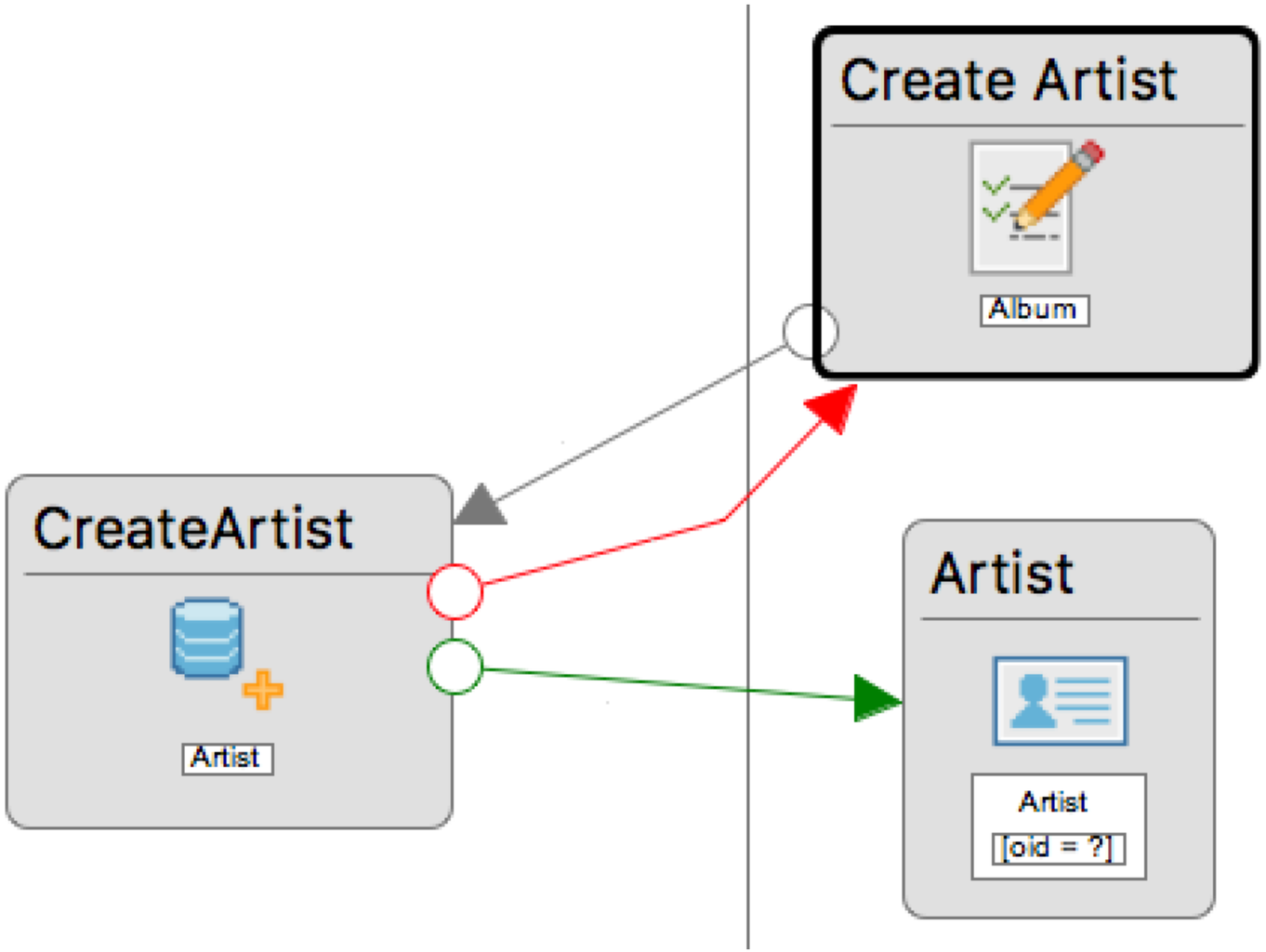
- Generic actions can be defined by the user
- Tool support adds a few very useful (next slide)

More IFML Syntax – Actions: Create

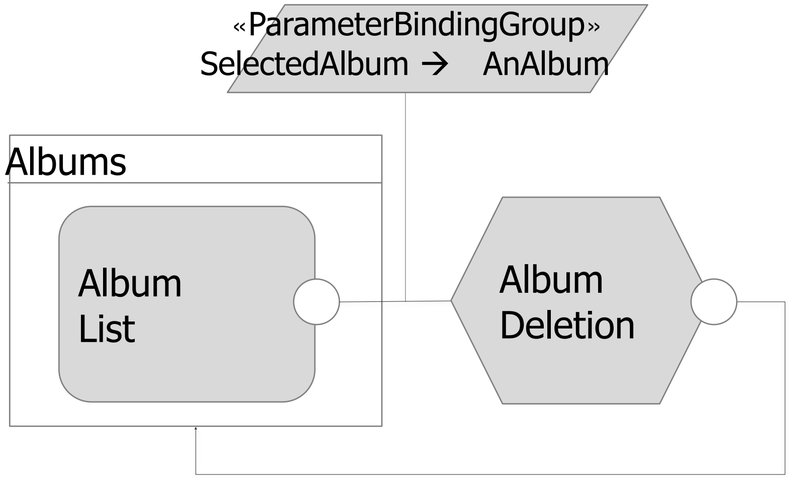


- Create
  - Creates an Entity entry (user selects entity)
  - Receives necessary data through flow and parameter binding

More IFML Syntax – Actions: Create



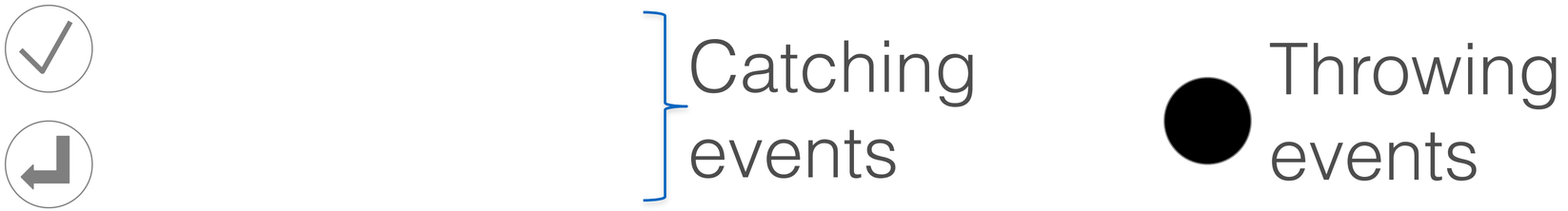
IFML by example



IFML – Subtyping Events

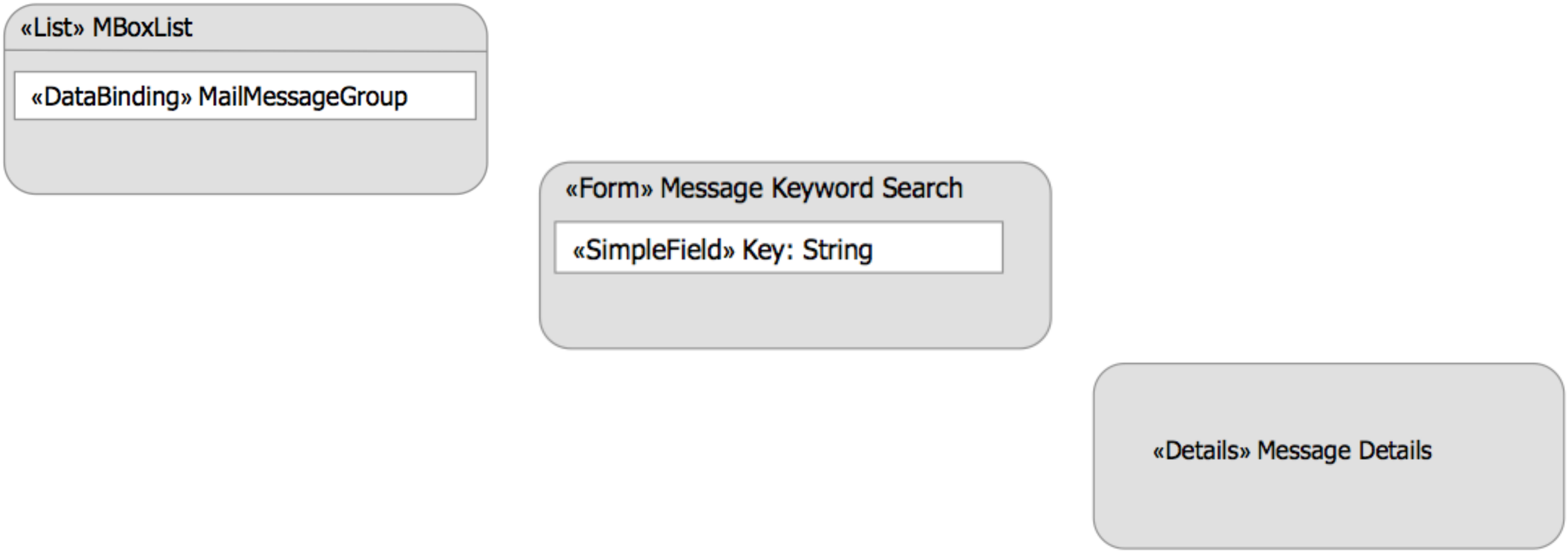
OnSelect event

OnSubmit event



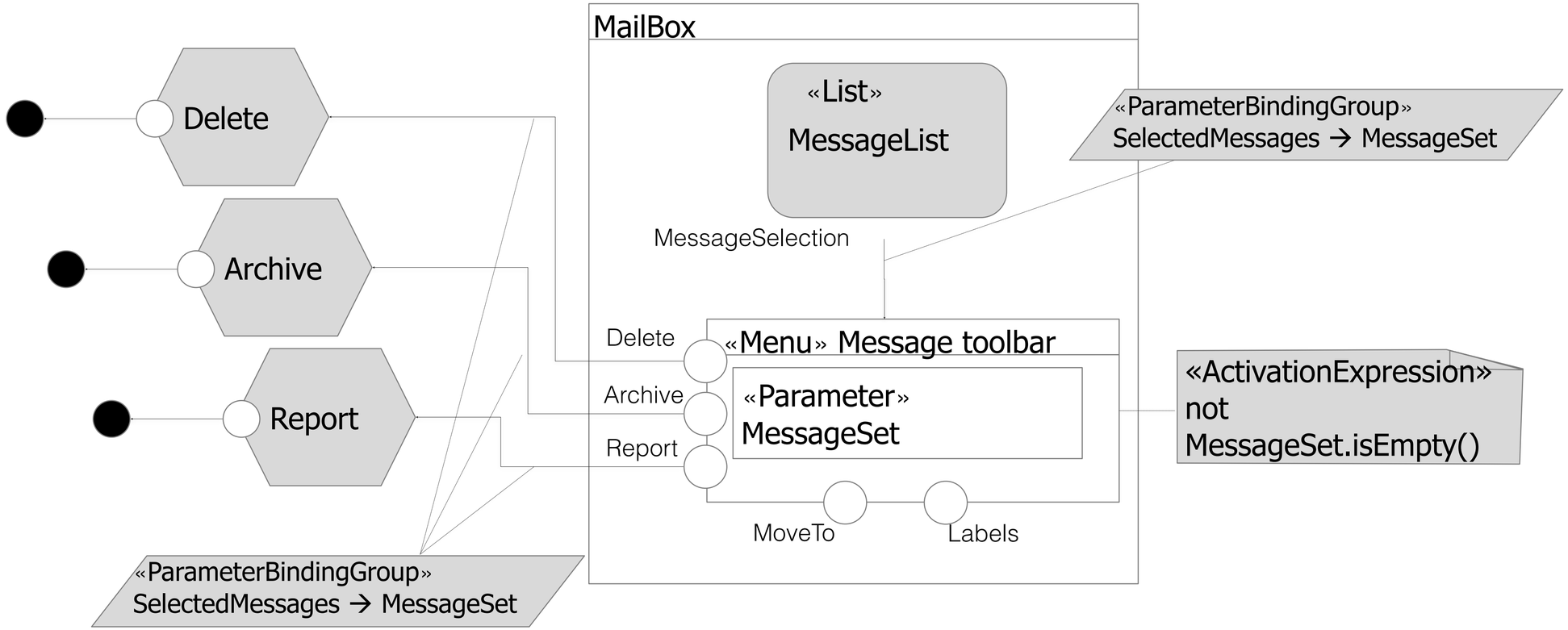
IFML – Subtyping Components

And as many others as you want!

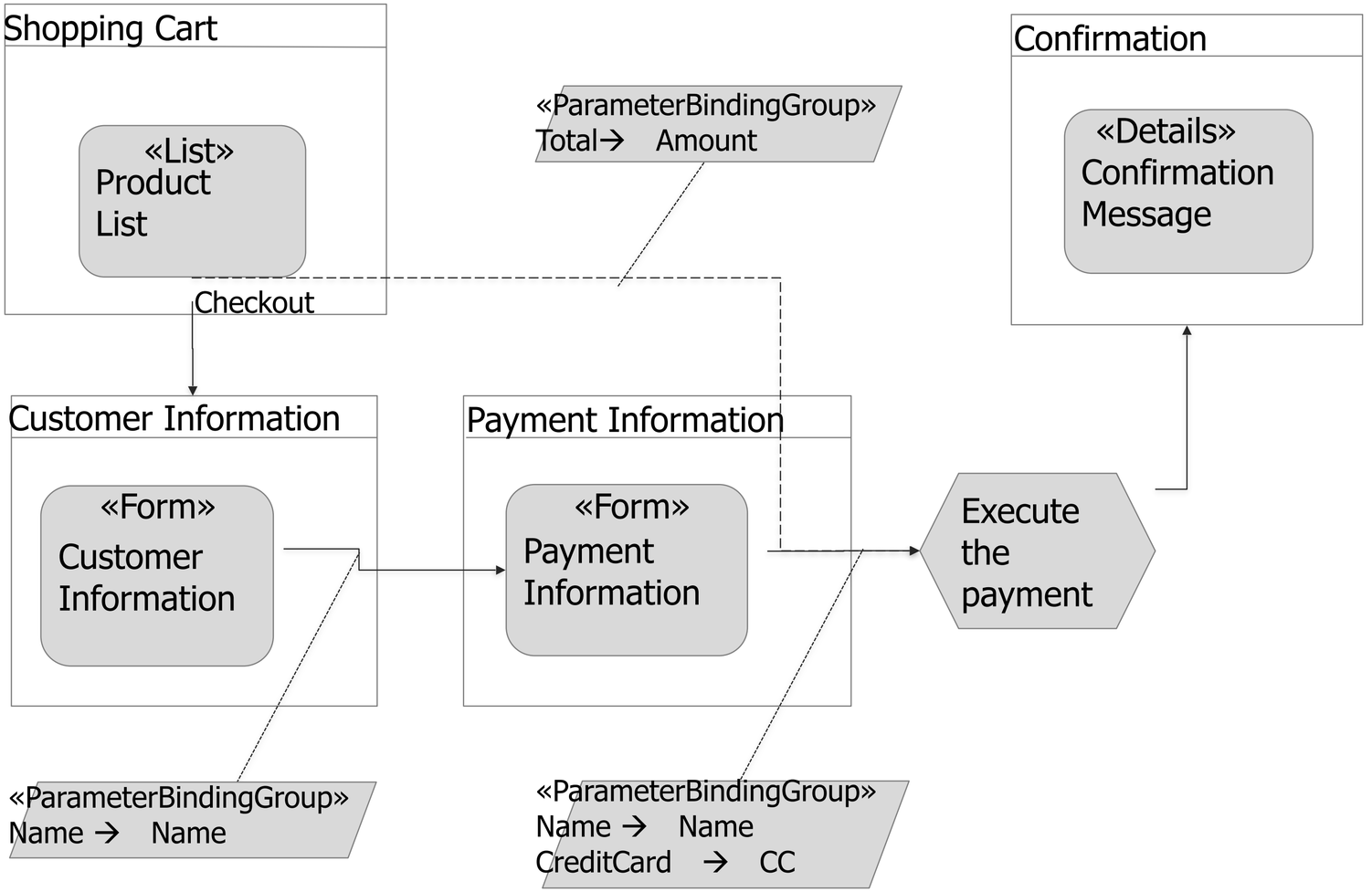


IFML by example – mailbox

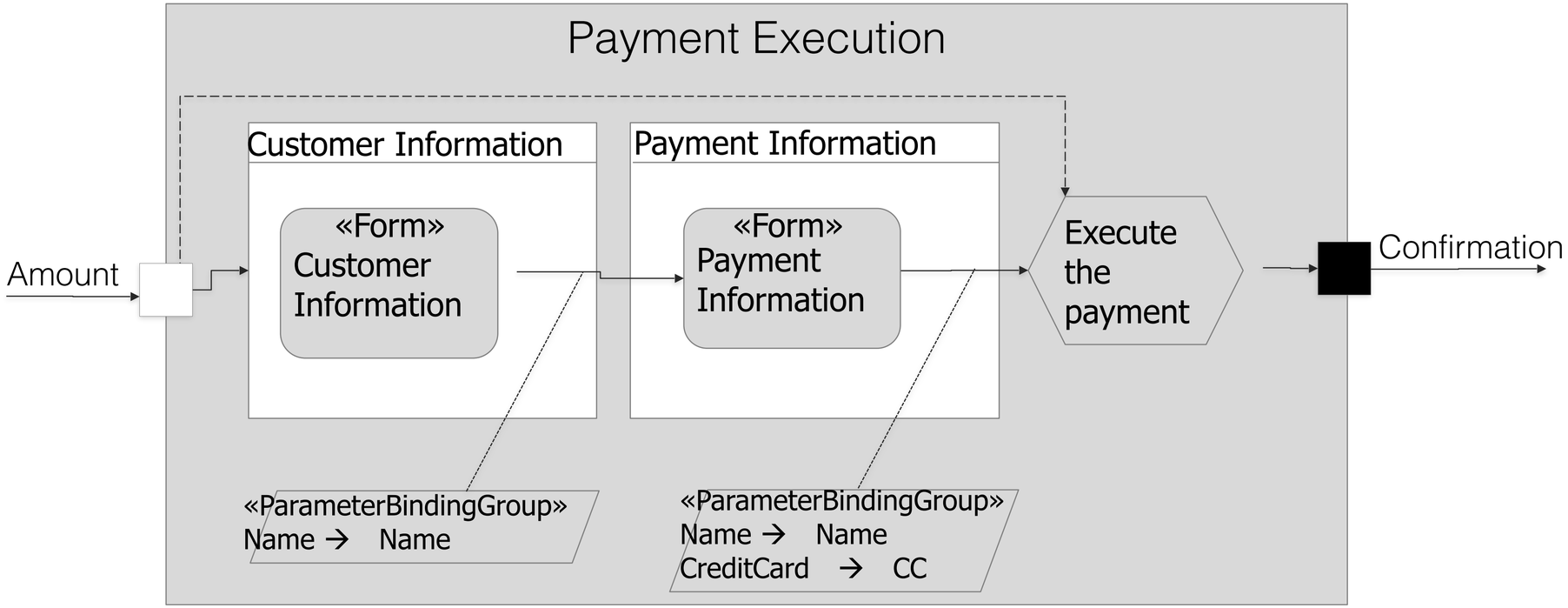
ActivationExpression, SubmitEvent, Event generation



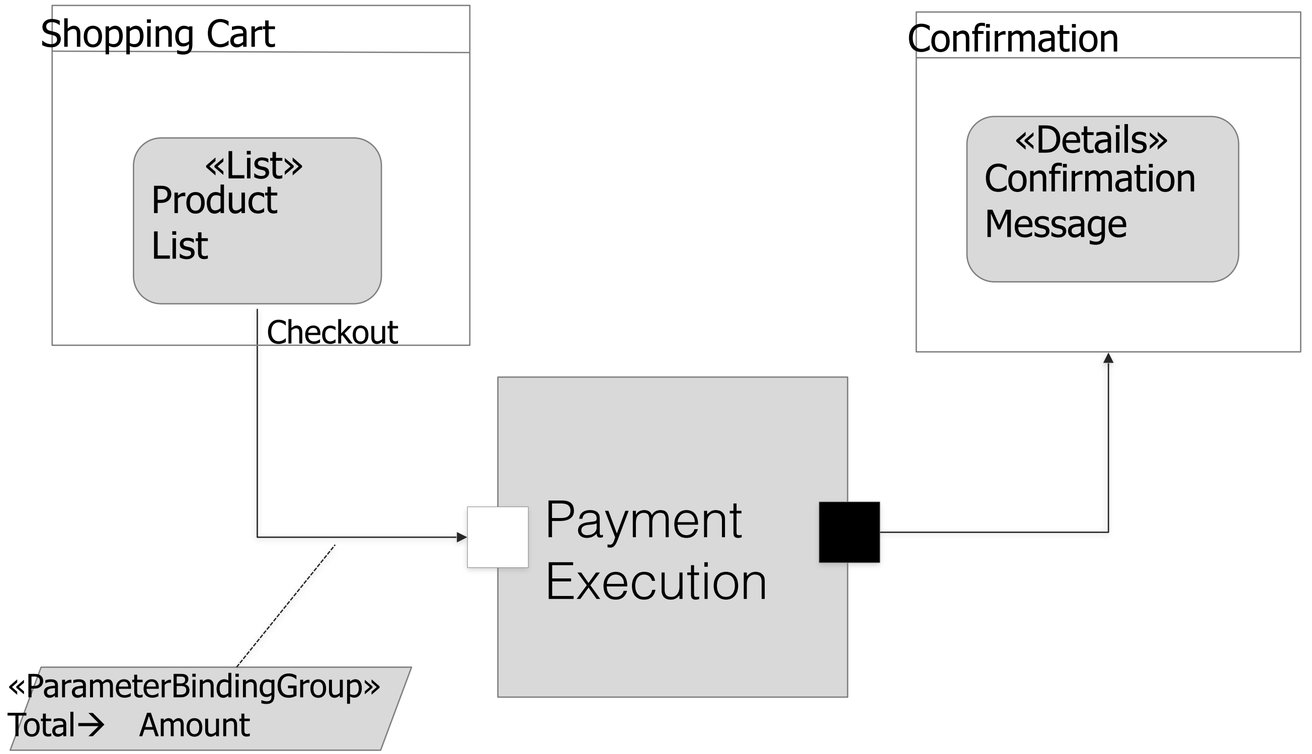
IFML example – online payment



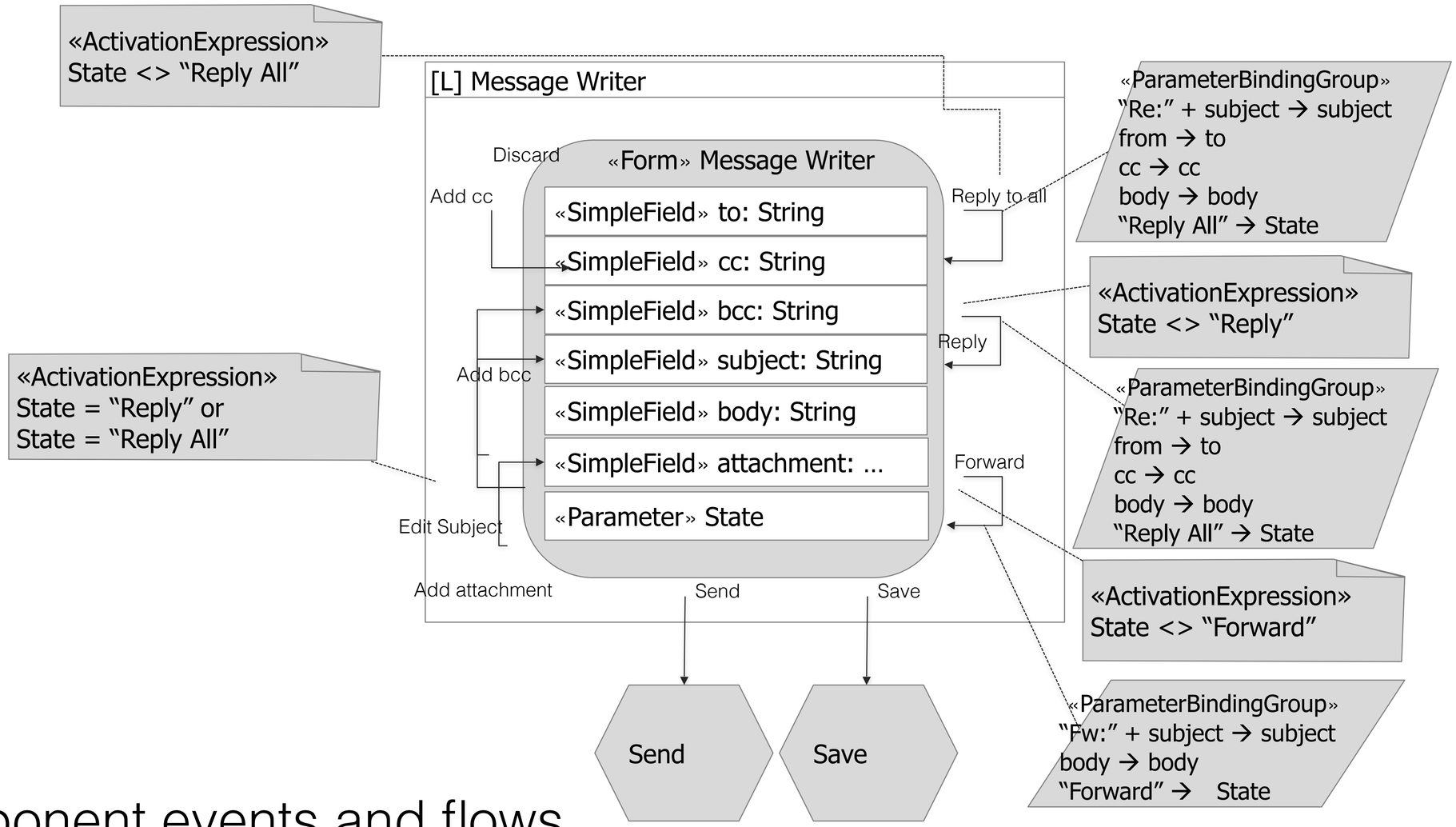
IFML – Modules



IFML – (Re)using Modules



IFML by example – email



intra-component events and flows

Tools & docs

- IFML page: ifml.org
- Eclipse Modelling Tools : http://www.eclipse.org/downloads/packages/
- IFML Eclipse editor: http://ifml.github.io/

Internet Applications Design and Implementation

(Lecture 9 - Part 4 - An App end-to-end)

MIEI - Integrated Master in Computer Science and Informatics
Specialization block

João Costa Seco (joao.seco@fct.unl.pt)
Jácome Cunha (jacome@fct.unl.pt)
João Leitão (jc.leitao@fct.unl.pt)

NOVA SCHOOL OF
SCIENCE & TECHNOLOGY

Example for today: Shared Task List

- A project management app where users have a backlog, can create tasks, divide them into sprints, assign them to users, mark them as completed, and see a variety of statistics about tasks and sprints.

Roadmap

1. User stories define a user centred application
2. IFML specifications to specify the (user stories) interactions and components.
3. React components implement each one of the IFML components and views.
4. Interface Mockups define the style and final composition of the UI.

User stories (I)

1. A user creates a new **task** given a name, a description, and due date, and sees the result in the **backlog**.
2. A user creates a new **sprint** given a name, a description, and a due date, and sees the result in the list of sprints.
3. A user selects a task (from the backlog or a sprint), assigns it to a (another) sprint and sees its name in the sprint task list.
4. A user selects a task (from the backlog or a sprint), and sees its name, description, and dates.
5. A user selects a task (from the backlog or a sprint), assigns it to a (another) user and sees the result in the details of the task and on the original list.
6. A user selects a task (from the backlog or a sprint), completes it (in a given date) and sees the end date in original list of tasks.

User Stories (II)

7. A user opens its home page, selects a sprint, and sees the sprint details, statistics and task list.

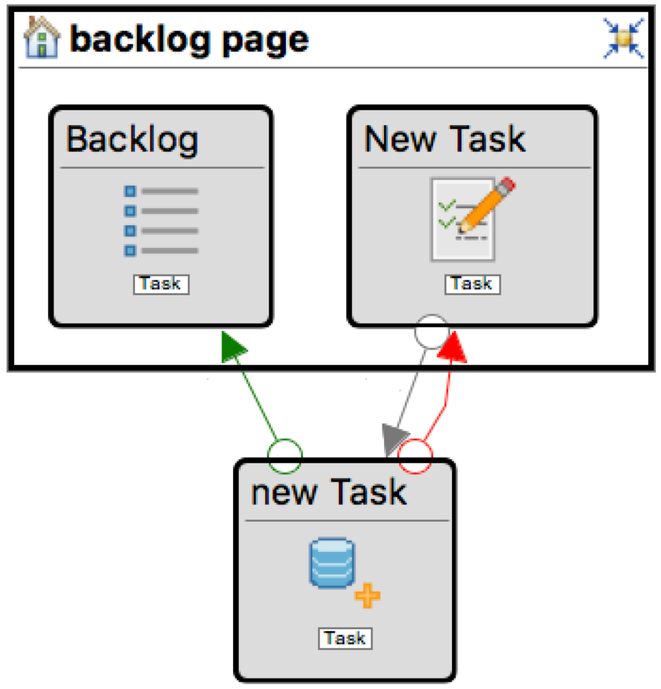
8. A user opens its home page, selects a user, and sees the user details, statistics and tasks list.

9. A user opens its home page, selects the backlog and sees the task list.

10. A user opens its home page, searches the backlog using a text and task properties, and sees the task list filtered by the given criteria.

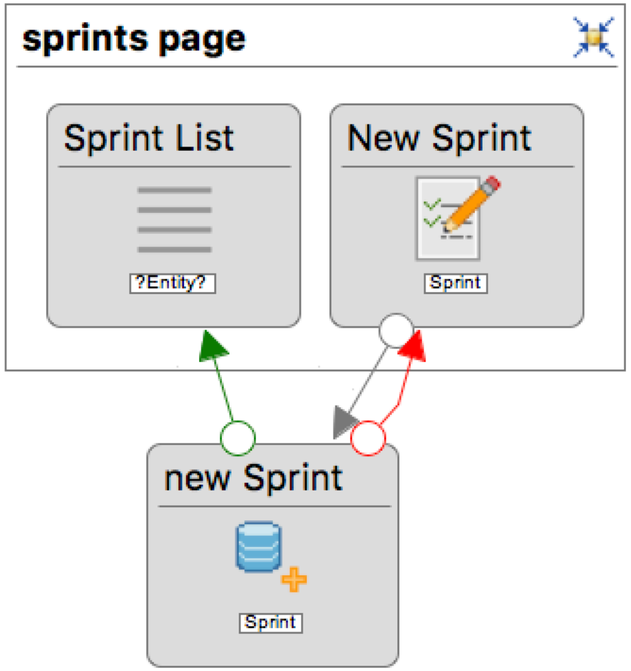
IFML Specification

- A user creates a new **task** given a name, a description, and due date, and sees the result in the **backlog**.



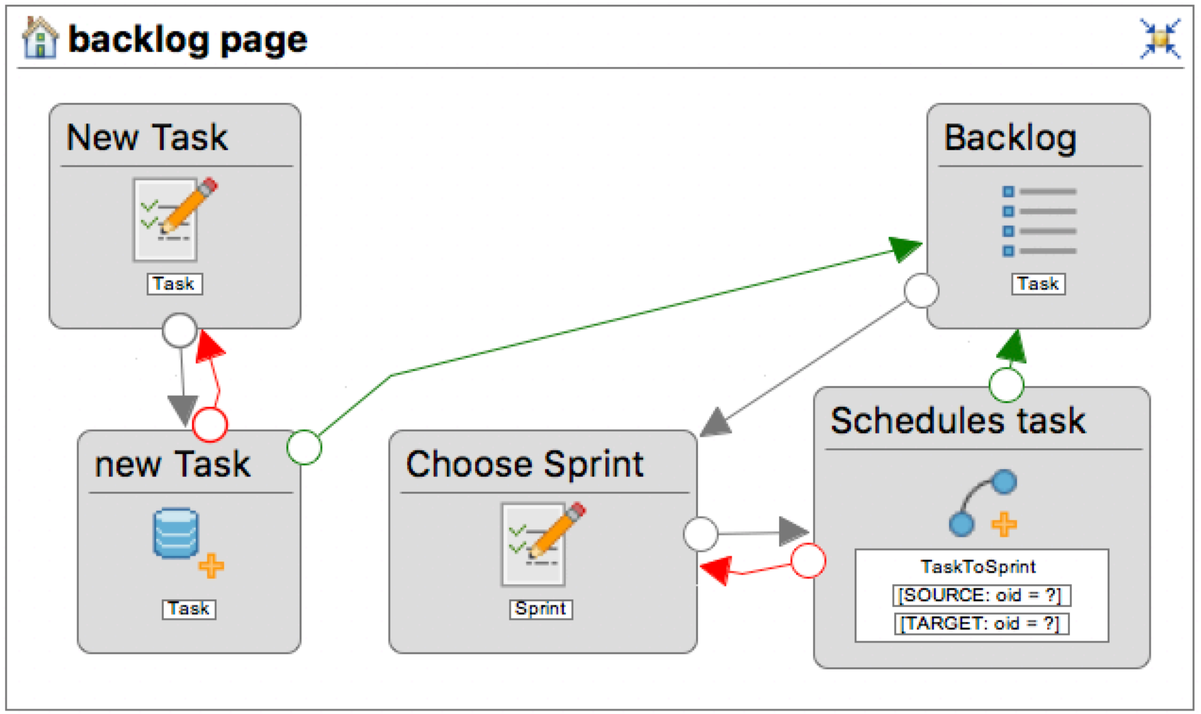
IFML Specification

- A user creates a new **sprint** given a name, a description, and a due date, and sees the result in the list of sprints.



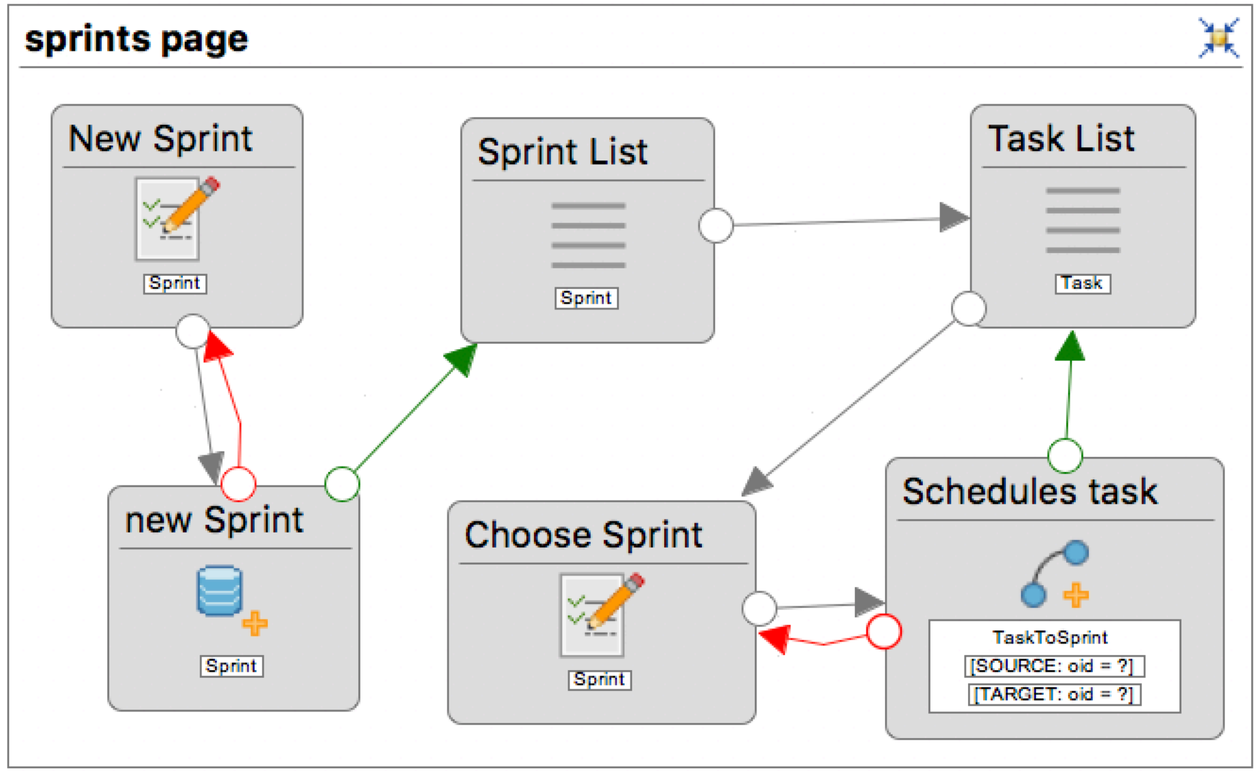
IFML Specification

- A user selects a task from the backlog, assigns it to a (another) sprint and sees its name in the sprint task list.



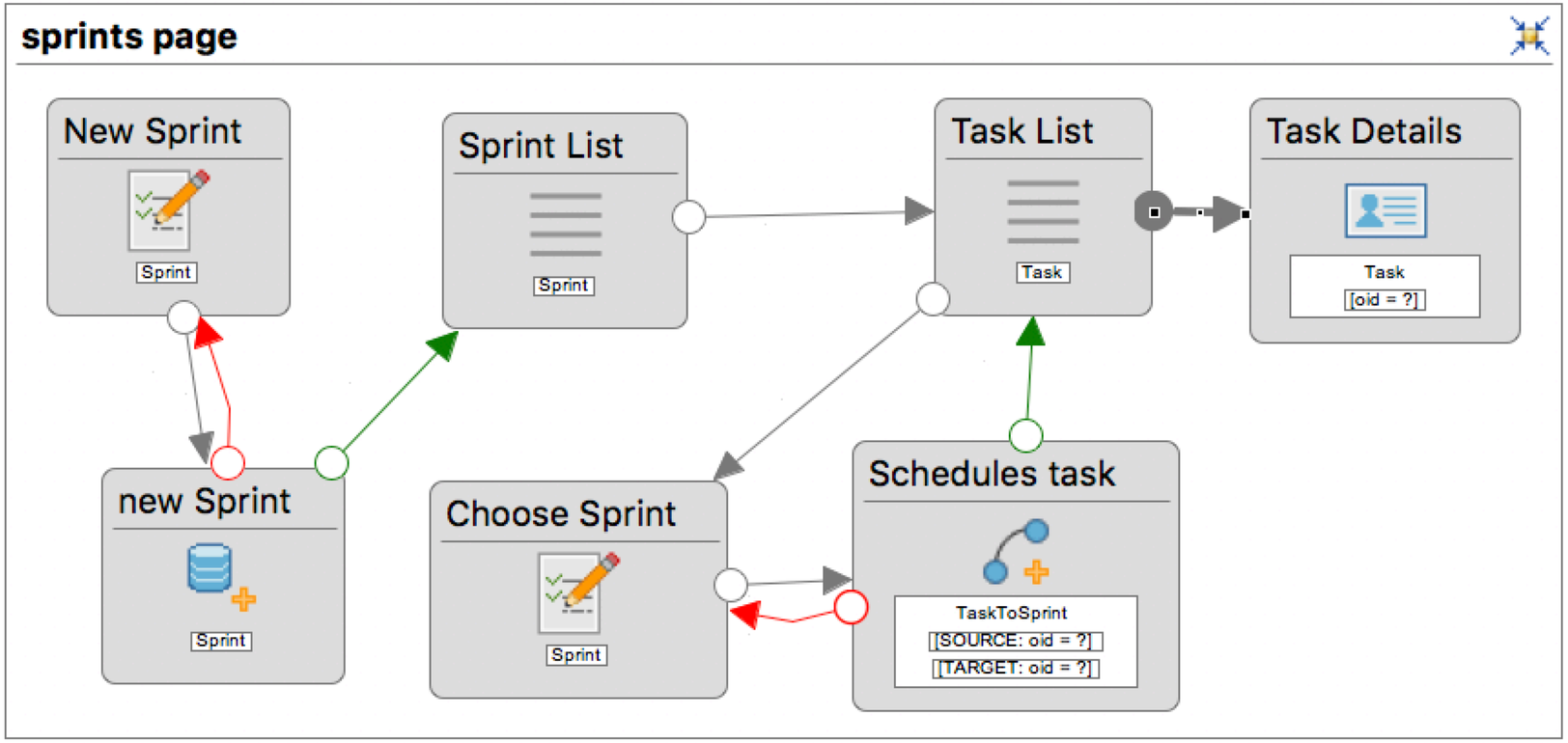
IFML Specification

- A user selects a task from a sprint, assigns it to a (another) sprint and sees its name in the sprint task list.



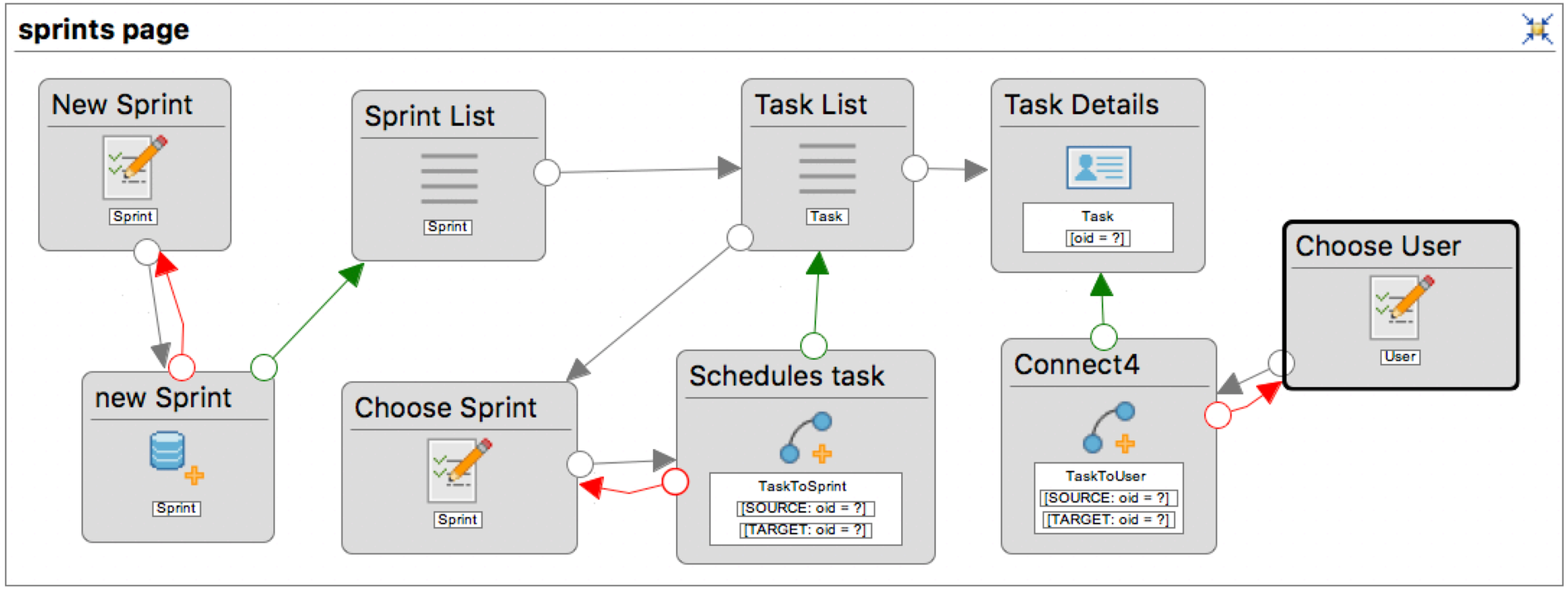
IFML Specification

- A user selects a task from a sprint, and sees its name, description, and dates.



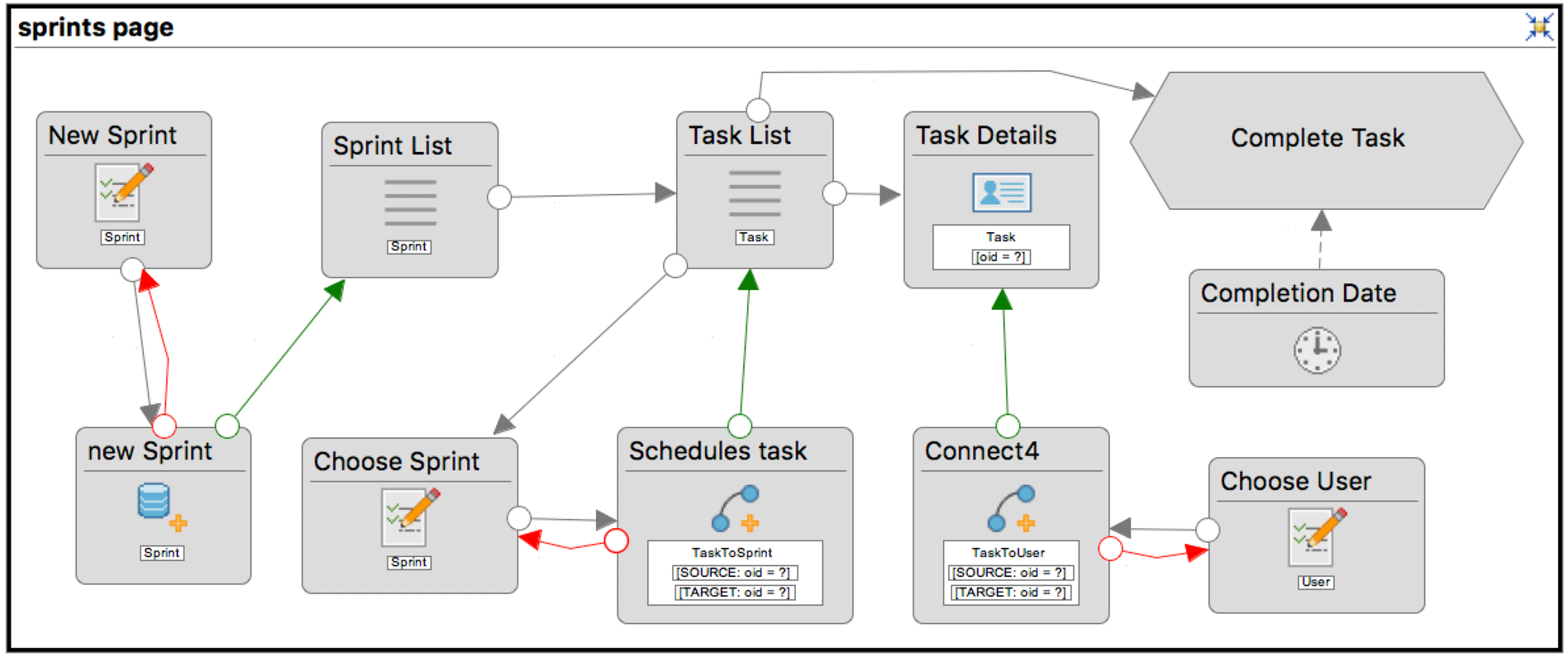
IFML Specification

- A user selects a task from a sprint, assigns it to a (another) user and sees the result in the details of the task and on the original list.
-



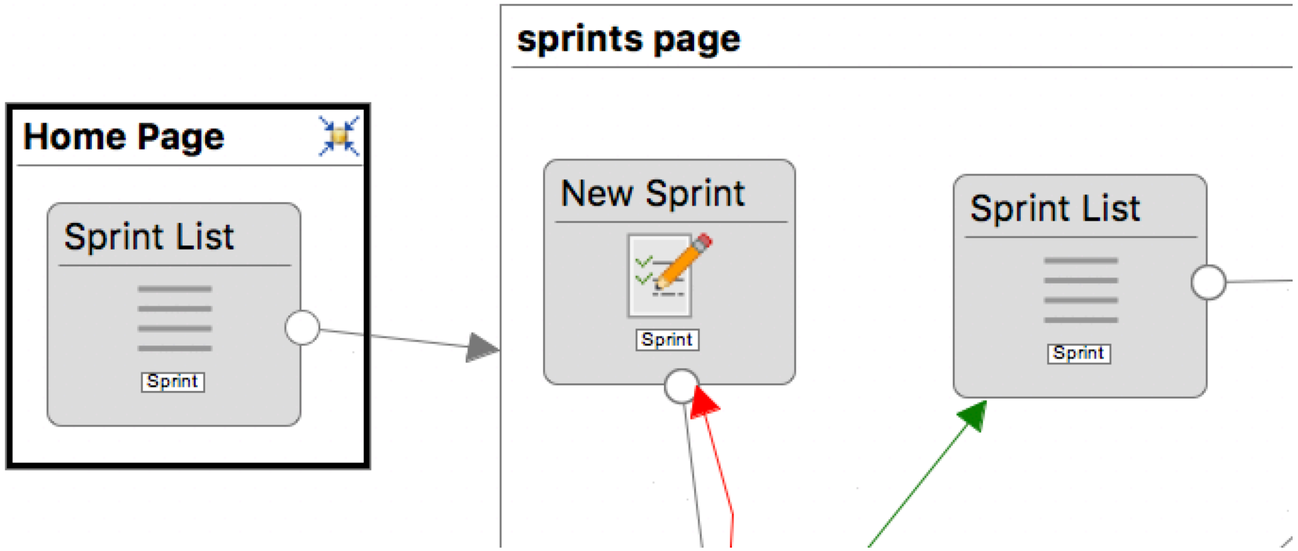
IFML Specification

- A user selects a task from a sprint, completes it (in a given date) and sees the end date in original list of tasks.



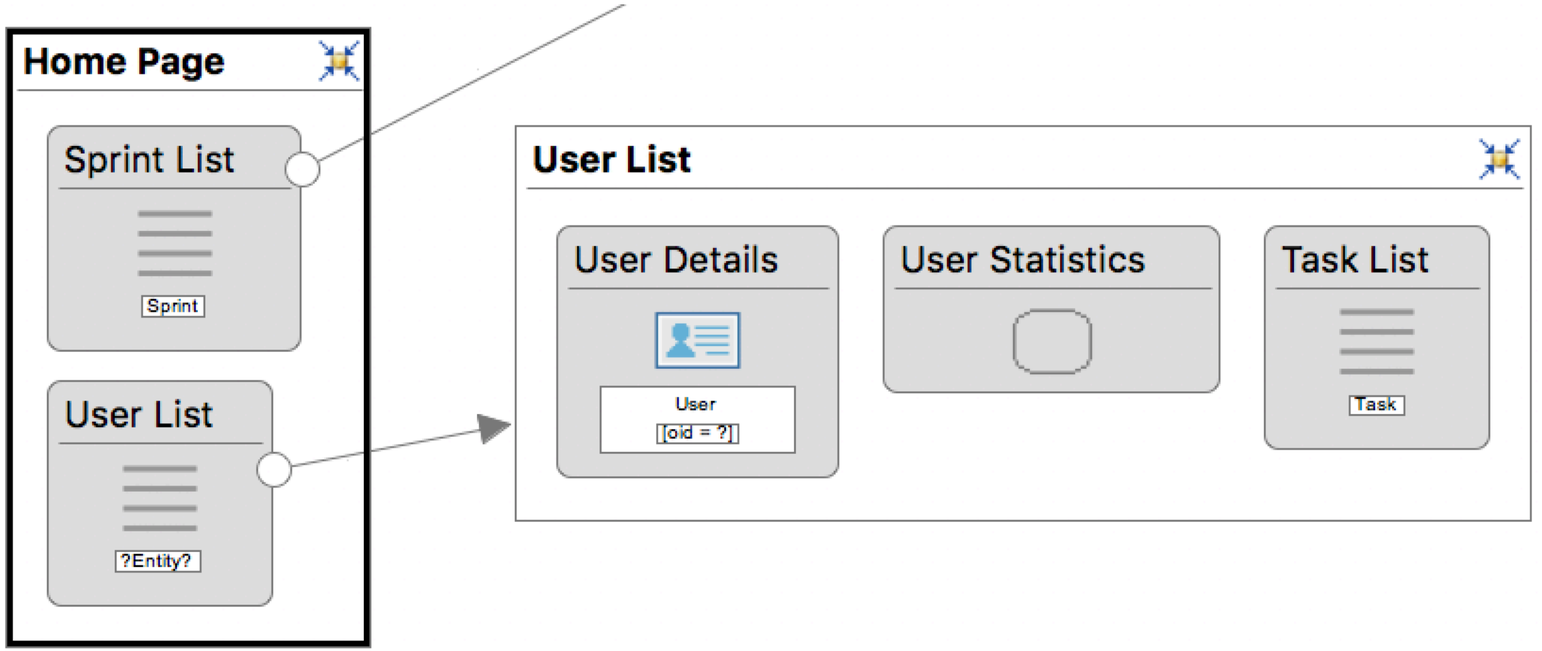
IFML Specification

- A user opens its home page, selects a sprint, and sees the sprint details, statistics and task list.



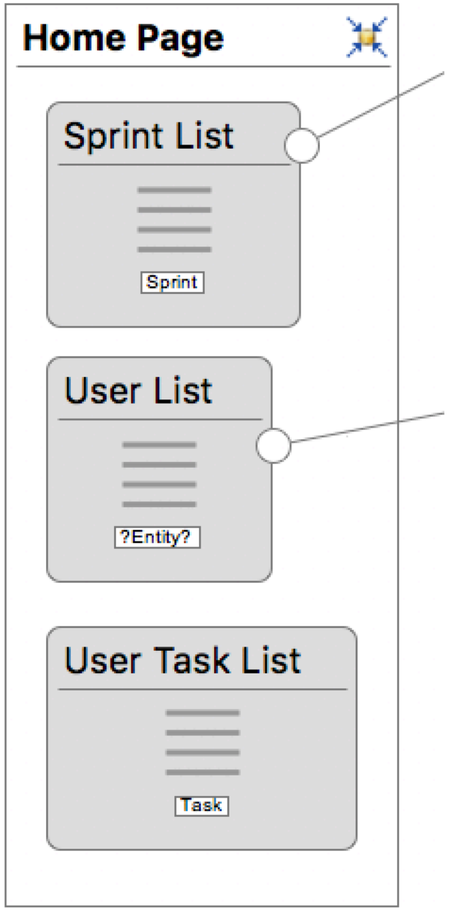
IFML Specification

- A user opens its home page, selects a user, and sees the user details, statistics and tasks list.



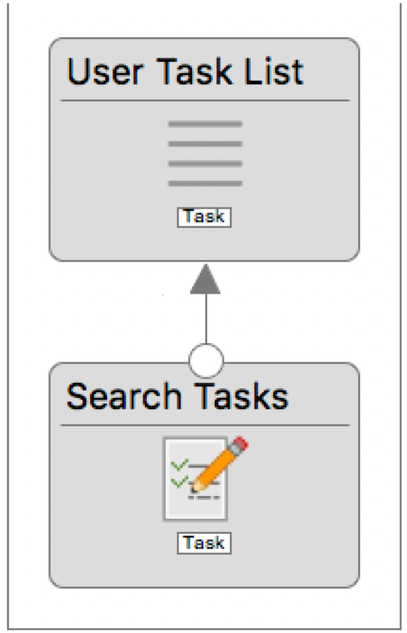
IFML Specification

- A user opens its home page, selects the backlog and sees the task list.

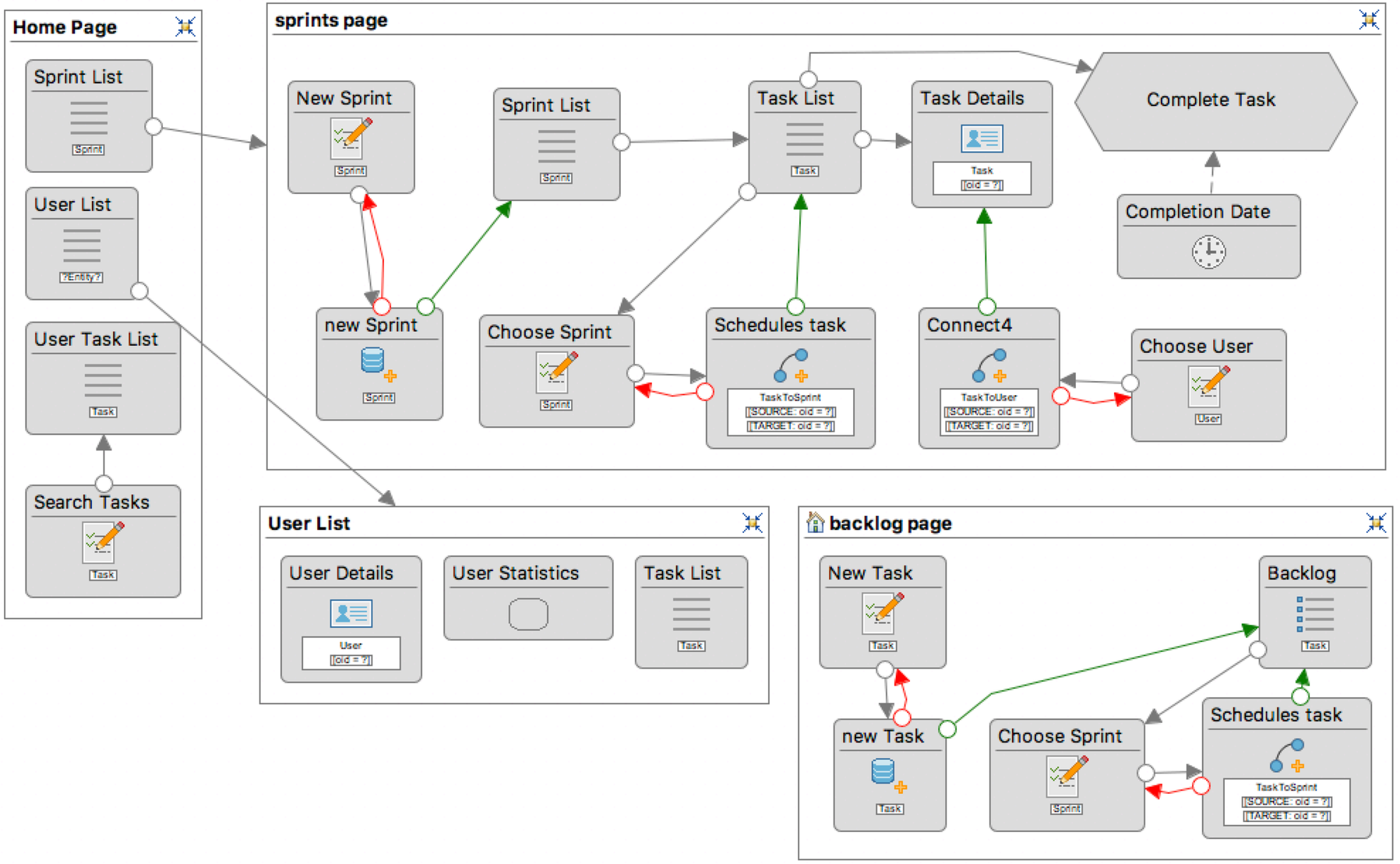


IFML Specification

- A user opens its home page, searches the backlog using a text and task properties, and sees the task list filtered by the given criteria.



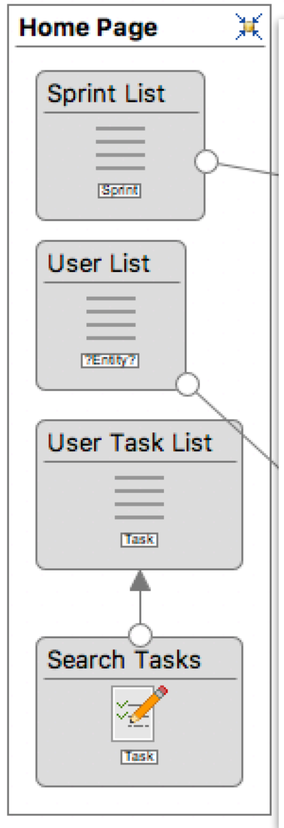
IFML Specification: The full picture



React structure

```
import * as React from "react";
import { BrowserRouter as Router, Route, Link } from "react-router-dom";

export interface HelloProps { compiler: string; framework: string; }

export const Dashboard = () => <div>Dashboard</div>

export const BackLog = () => <div>BackLog</div>

export const Sprints = () => <div>Sprints</div>

export const Users = () => <div>Users</div>

export const App = () =>
    <Router>
        <div className="container">
            <h1>Task List</h1>
            <div>
                <ul>
                    <li><Link to="/"> Home </Link></li>
                    <li><Link to="/backlog"> BackLog </Link></li>
                    <li><Link to="/sprints"> Sprints </Link></li>
                    <li><Link to="/users"> Users </Link></li>
                </ul>
            </div>

            <div>
                <div>
                    <Route exact path="/" component={Dashboard}/>
                    <Route path="/backlog" component={BackLog}/>
                    <Route path="/sprints" component={Sprints}/>
                    <Route path="/users" component={Users}/>
                </div>
            </div>
        </div>
    </Router>
```

React structure




```
const SearchCriteria = () =>
    <form>
        <input type="text" name="search"/>
        <input type="submit" value="Search"/>
    </form>

const SeacheableUserTaskList = () =>
    <div>
        <TaskList/>
        <SearchCriteria/>
    </div>

const Dashboard = () =>
    <div>
        <h1>Dashboard</h1>
        <SprintList/>
        <UserList/>
        <SeacheableUserTaskList/>
    </div>
```

```
const UserList = () =>
    <div>
        <h2>Users</h2>
        <ul>
            <li>User 1</li>
            <li>User 2</li>
            <li>User 3</li>
            <li>User 4</li>
        </ul>
    </div>

const SprintList = () =>
    <div>
        <ul>
            <li>Sprint 1</li>
            <li>Sprint 2</li>
            <li>Sprint 3</li>
            <li>Sprint 4</li>
        </ul>
    </div>

const TaskList = () =>
    <div>
        <h2>Tasks</h2>
        <ul>
            <li>Task 1</li>
            <li>Task 2</li>
            <li>Task 3</li>
            <li>Task 4</li>
        </ul>
    </div>
```

React structure

```
export class App extends Component<any,any> {

    constructor(props: any) {
        super(props);
        this.state = {
            sprints: [],
            tasks: []
        }
    }

    render() {
        return (
            <Router>
                <div className="container">
                    <h1>Task List</h1>
                    <div>
                        <ul>
                            <li><Link to="/"> Home </Link></li>
                            <li><Link to="/sprints"> Sprints </Link></li>
                            <li><Link to="/backlog"> BackLog </Link></li>
                            <li><Link to="/users"> Users </Link></li>
                        </ul>
                    </div>

                    <div>
                        <div>
                            <Route exact path="/"
                                   component={() => <DashboardPage sprints={this.state.sprints}/>}/>
                            <Route path="/sprints" component={Sprints}/>
                            <Route path="/backlog" component={BackLogPage}/>
                            <Route path="/users" component={UsersPage}/>
                        </div>
                    </div>
                </div>
            </Router>);
    }
}
```

Summary

Pros and Cons of using specs and frameworks

- Formal Design and Method
- Communication between stakeholders
- Documentation for future reference
  - It’s important to use tools that sync specs and code
- Maintenance and evolution
  - Allows the analysis of the impact of code evolutions