IFML Certification Guide

Outline

Which parts of the specification are part of the certification and which aren’t

Summary of salient aspects of the specification

Useful info on certification

- place, duration, number of questions
- types of questions
- examples for every type of question (and caveats)

IFML Certification - OMG Specs Chapters

To be prepared for the IFML Certification test you need to read carefully these chapters of the OMG Specification

1. Scope

6. Additional Information (except 6.4)

7. IFML Specification

8. IFML Metamodel

Scope

Topic: **Interaction Flow Modeling Language (IFML)**

Target audience: system architects, software engineers, and software developers

Goal: modeling the front-end of an application, i.e.:
- view part (view containers and view components)
- state (and references to business logic actions)
- binding (of view components to data objects and events)
- control logic (to execute actions after an event triggers)
- tiers (for control, data and business logic)

Business Motivation

Common needs in B2C, B2B, B2E

- interaction, browsing, navigation, multimedia, and personalization
- only partly satisfied by HTML 5

Wide spectrum of technologies, platforms, devices

Need for a Platform Independent Model (PIM) to tackle proliferation of devices and technologies

- better efficiency, reuse, portability and less coding

Business Motivation

Benefits of using IFML for PIM-level interaction flow modeling:

- covers several front-end perspectives: content, interface composition, interaction, navigation, connection with business logic, presentation
- separates front-end interaction design from implementation
- enables early communication with non-technical stakeholders like subject matter experts (SMEs) and clients

Design Principles

Usability, understandability, simplicity, and extensibility

Conciseness: minimal set of diagrams and concepts for interface and interaction design

Automatic application of **model inference rules** that apply default modeling patterns and details whenever possible from the context

- e.g., automatic inference of the parameters that need to be passed

Design Principles

Extensibility in the definition of new concepts

Implementability: map the PIM into a PSM (and then executable code) through model transformation frameworks and code generators

Model-level reuse: definition of design patterns that can be stored, documented, searched and retrieved, and re-used in other applications

IFML Artifacts

The IFML specification consists of **four main technical artifacts**:

1. **IFML metamodel**: structure and semantics of the IFML constructs
2. **IFML UML profile:** UML-based syntax for expressing IFML models
3. **IFML visual syntax** for expressing IFML models concisely through a unique diagram capturing all relevant aspects of the user interface
4. **IFML XMI:** IFML model exchange format, for portability

Key concepts of IFML

IFML supports the platform-independent description of graphical user interfaces for multiple systems

- Focus on the application as perceived by the end user

IFML and Model-View-Controller (MVC)

- **View** (IFML's main focus): view composition and description of elements used for user interaction
- **Controller**: specification of the effects of interactions and events on the application
- **Model**: references to data objects published in the interface and actions triggered by user interaction

IFML can be complemented with external models for aspects not directly related to the user interface

IFML in a nutshell

An IFML diagram contains one or more top-level **view containers**, internally structured in a **hierarchy of sub-containers**

A view container can contain **view components** (such as a data entry form), which can have input and output parameters

A view container and a view component can be associated with **events** (for user interaction)

IFML in a nutshell

The effect of an event is represented by an **interaction flow**

An event can also cause the *triggering of an action*

Navigation flows are associated with **parameter bindings** (input-output dependencies between view elements)

Extensibility

IFML uses UML's extensibility mechanisms to allow the definition of stereotypes, tagged values and constraints

The **Extensions package** contains concepts that extend concepts from the **Core package**

New packages may also be introduced to model new concepts (either platform-independent or platform-specific)

Extensions are meant to refine the semantics of the IFML concepts (ViewContainer, ViewComponent, ViewComponentPart, Event) and not modify it

Concept List

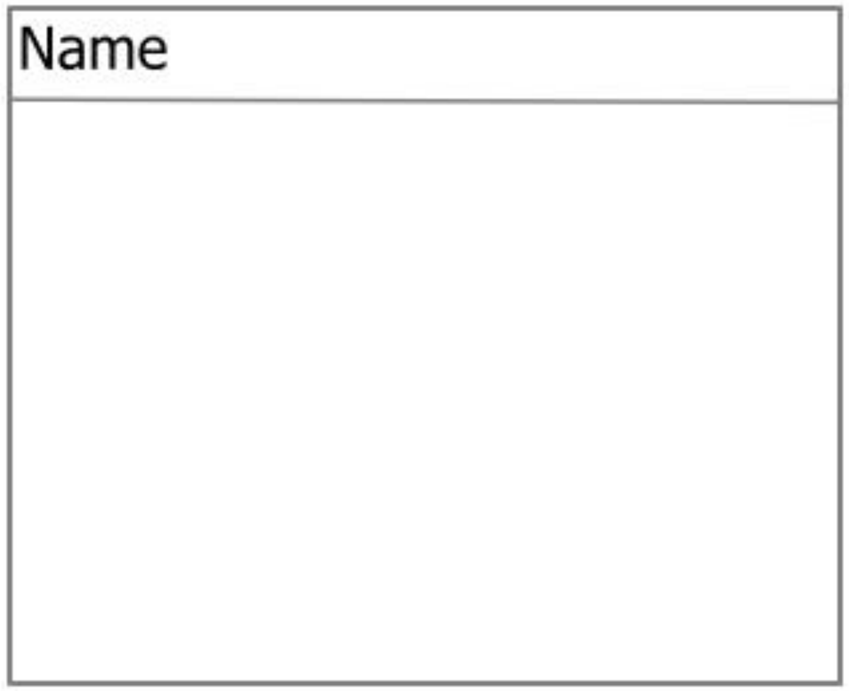
The core concepts of IFML include:

- View Container (possibly XOR, Landmark, Default)
- View Component and View Component Part
- Event and Action
- Flow (Navigation or Data)
- Parameter, Parameter Binding and P.B. Group
- Module
- Expression (Activation or Interaction Flow)
- Port (Input or Output)

IFML also features some extension concepts

Core Concepts: View Container

An element of the interface that comprises elements displaying content and supporting interaction and/or other ViewContainers.

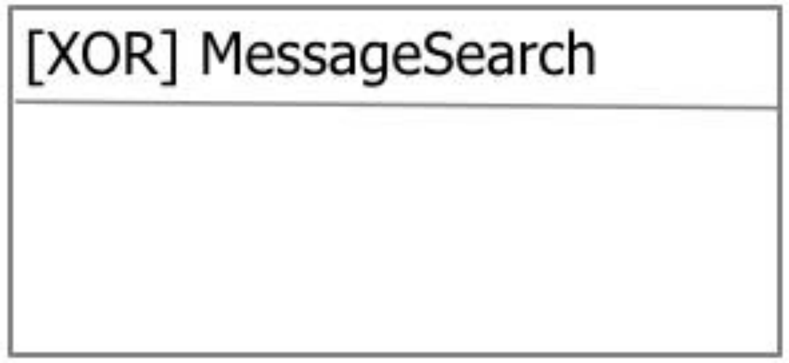


Examples:

- Web page
- Window
- Pane

Core Concepts: XOR View Container

A ViewContainer comprising child ViewContainers that are displayed alternatively

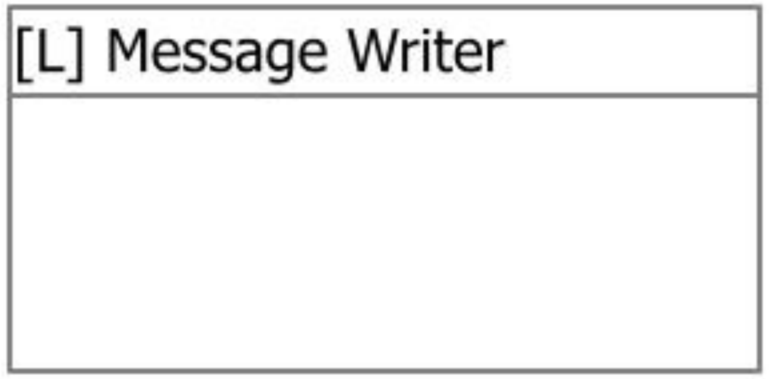


Examples:

- Tabbed panes in Java
- Frames in HTML

Core Concepts: Landmark View Container

A ViewContainer that is reachable from any other element of the user interface without having explicit incoming InteractionFlows

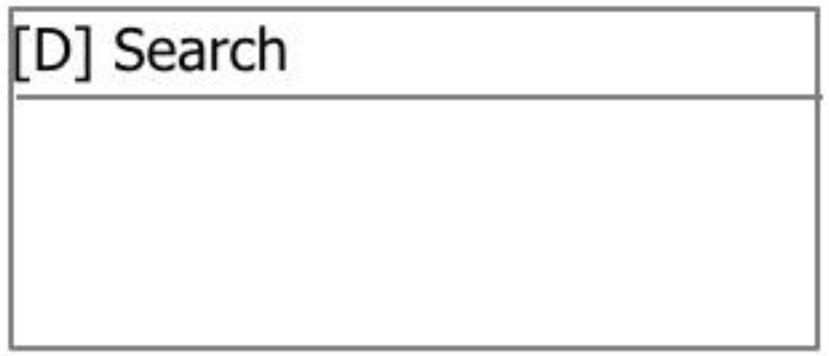


Examples:

- A logout link in HTML sites visible in every page

Core Concepts:
Default View Container

A ViewContainer that will be presented by default to the user, when its enclosing container is accessed

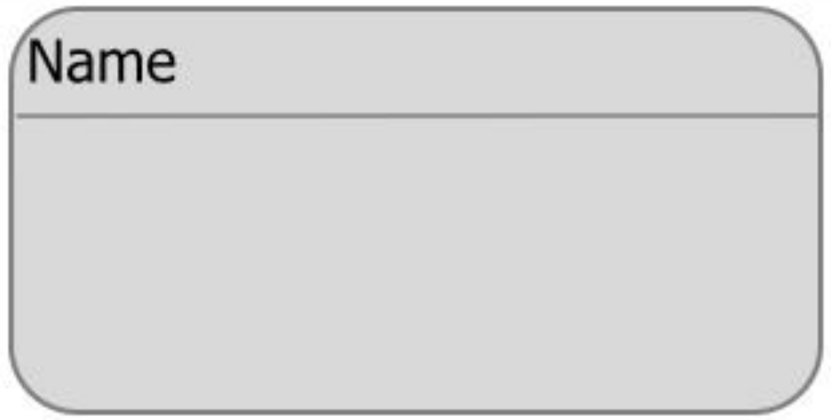


Examples:

- A welcome page

Core Concepts: View Component

An element of the interface that displays content or accepts input

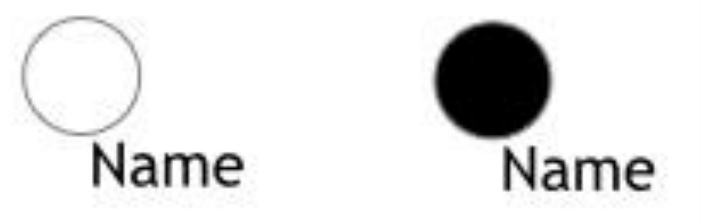


Example:

- An HTML list
- A JavaScript image gallery
- An input form

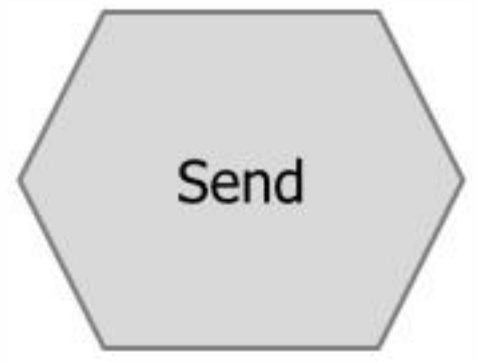
Core Concepts: Event

An occurrence that affects the state of the application



Core Concepts: Action

A piece of business logic triggered by an event; it can be server-side (the default) or client-side, denoted as [Client]



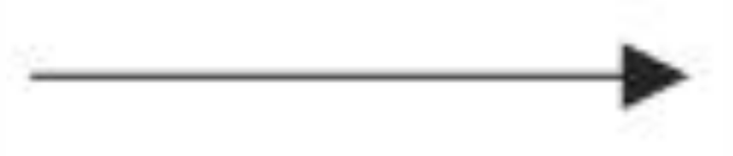
Example:

- A database update
- The sending of an email
- The spell checking of a text

Core Concepts: Flow

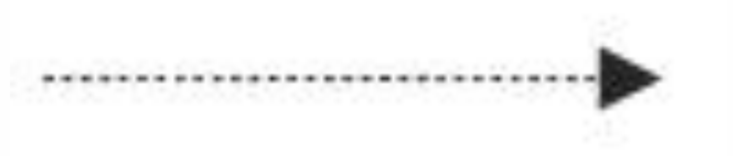
Navigation flow:

An input-output dependency. The source of the link has some output that is associated with the input of the target of the link



Data flow:

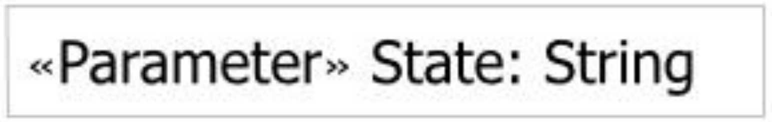
Data passing between ViewComponents or Action as consequence of a previous user interaction



Core Concepts: Parameter, Parameter Binding, Parameter Binding Group

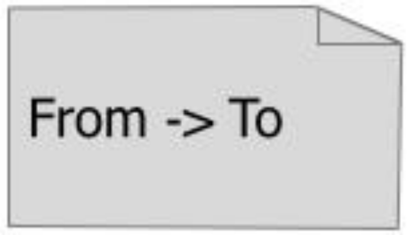
Parameter:

A typed and named value



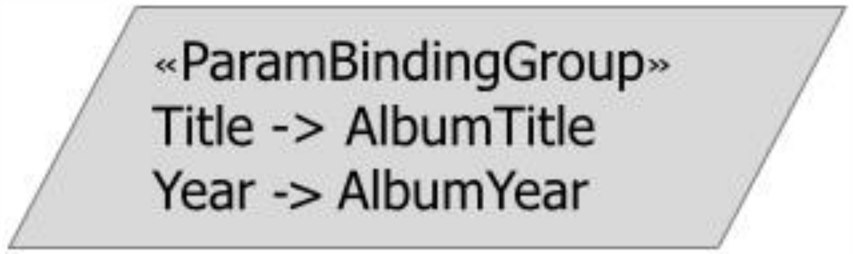
Parameter Binding:

Specification that an input parameter of a source is associated with an output parameter of a target



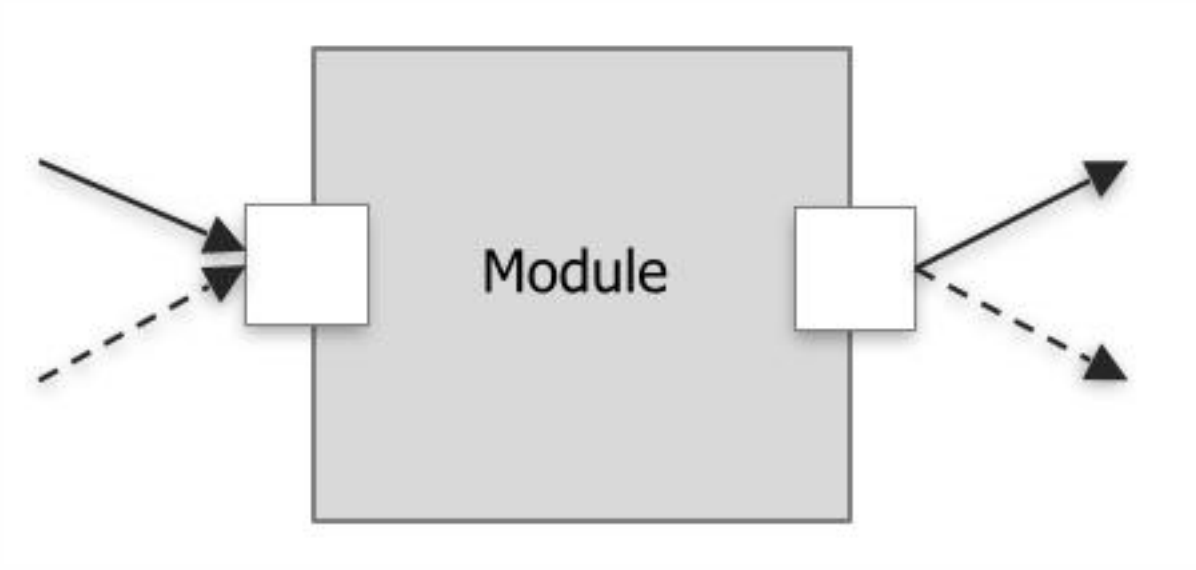
Parameter Binding Group:

Set of ParameterBindings associated with an InteractionFlow



Core Concepts: Module

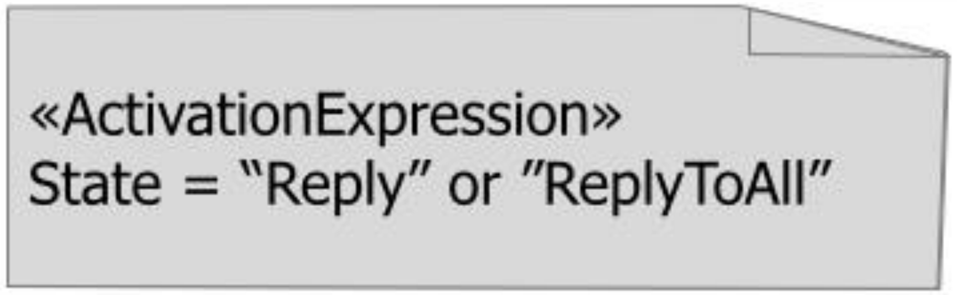
Piece of user interface and its corresponding actions, which may be reused for improving IFML models maintainability
InteractionFlow



Core Concepts: Expression

Activation Expression:

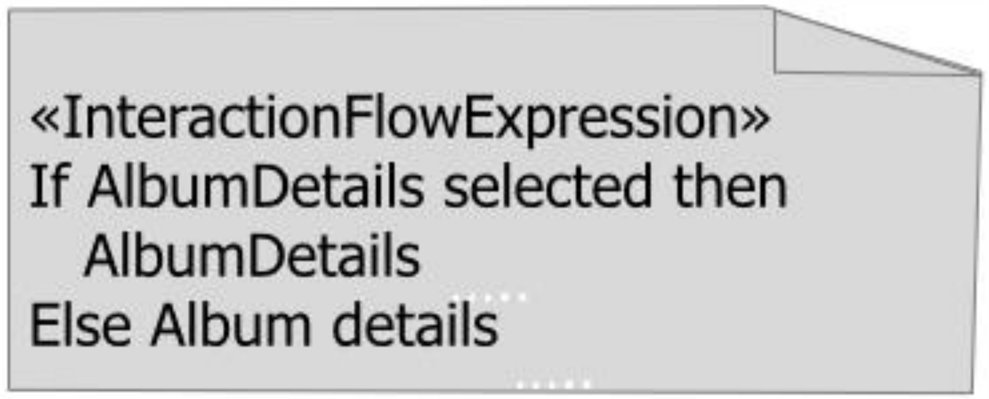
Boolean expression associated with a ViewElement, ViewComponentPart or Event: if true the element is enabled



«ActivationExpression»
State = "Reply" or "ReplyToAll"

Interaction Flow Expression:

Determines which of the InteractionFlows are going to be followed as consequence of the occurrence of an Event

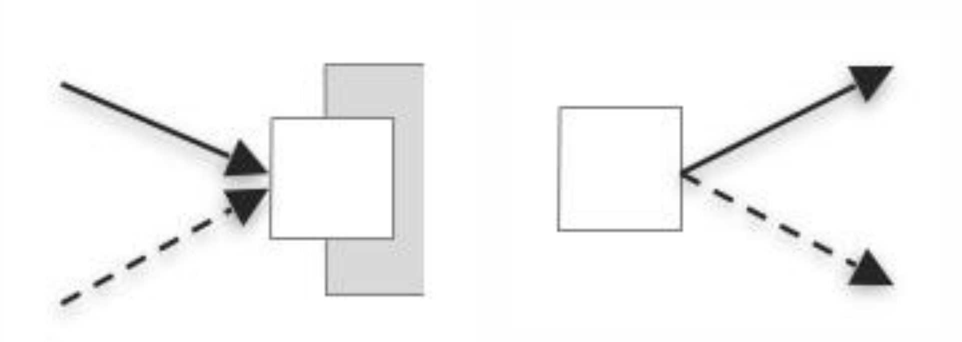


«InteractionFlowExpression»
If AlbumDetails selected then
AlbumDetails
Else Album details

Core Concepts: Port

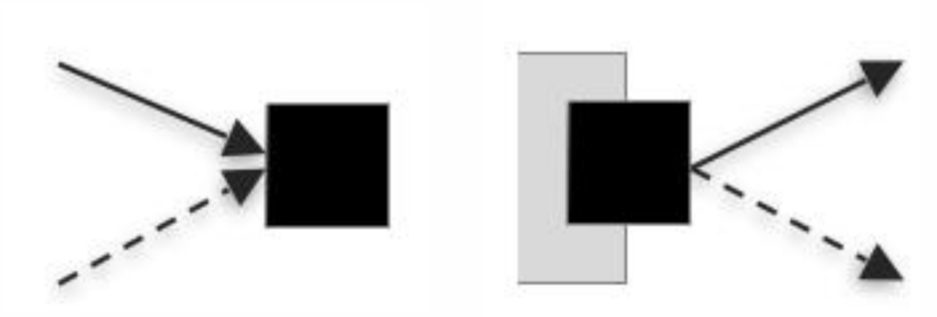
Input port:

An interaction point between a Module and its environment that collects InteractionFlows and parameters arriving at the module



Output port:

An interaction point between the Module and its environment that collects the InteractionFlows and parameters going out from the module



IFML metamodel: High-level description

The IFML metamodel is divided in three packages:

- **Core**: concepts for interaction infrastructure (InteractionFlowElements, InteractionFlows and Parameters)
- **Extension**: extends Core concepts with more complex behaviors
- **DataTypes**: custom data types defined by IFML

The IFML metamodel uses the basic data types from the UML metamodel

- Specifically, the IFML DomainModel is represented in UML

IFML Model

IFMLModel represents an IFML model and is the top-level container of all the rest of the model elements, i.e.:

- an **InteractionFlowModel**: user view of the whole application
- a **DomainModel**: business domain view of the application, comprising DomainElements (further specialized as DomainConcept, FeatureConcept, BehaviorConcept, and BehavioralFeatureConcept)
- (optionally) **ViewPoints**: specific aspects facilitating the comprehension of the system by means of sets of InteractionFlowModelElements (abstract class generalizing every element of an InteractionFlowModel)

IFML Model

Element is the most general class in the model

- specialized by NamedElement (elements that have a name)
  - specialized by IFMLModel, InteractionFlowModel, DomainModel, DomainElement and ViewPoint, and other subclasses

- for any Element, Constraints and Comments can also be specified

Interaction Flow Model

An **InteractionFlowModel** contains all the elements of the user view

Defined by **InteractionFlowModelElements**, whose subtypes include:

- **InteractionFlowElement**: the building block of interactions
- **InteractionFlow**: a directed connection between two InteractionFlowElements
- **Parameter**: a typed name whose instances hold values
- **ParameterBinding**: binds an input Parameter of a target with an output Parameter of a source InteractionFlowElement
- **ParameterBindingGroup**: groups ParameterBindings
- **Module**: reusable collection of InteractionFlowModelElements
- **Expression**: statement that evaluates to a single instance, a set of instances, or an empty result

Interaction Flow Elements

InteractionFlowElements represent pieces of the system involved in InteractionFlow connections:

- ViewElements
- ViewComponentParts
- Ports
- Actions
- Events

InteractionFlowElements contain Parameters

Parameters flow between InteractionFlowElements as a consequence events (ViewElementEvents, ActionEvents or SystemEvents).

Interaction Flow Elements

InteractionFlows are specialized into

- NavigationFlows: navigation or change of ViewElement focus, when Events are triggered
- DataFlow: passes context information between InteractionFlowElements (no navigation)

ViewContainers can contain ViewElements (namely other ViewContainers or ViewComponents) or Actions.

View Elements

View Elements: elements of an IFML model visible at the user interface

- **ViewContainers**: containers of other ViewContainers or ViewComponents
  - If marked landmark, it may be reached from any other ViewElement without the need for explicit InteractionFlows
  - If marked default, it will be presented to the user when its enclosing ViewContainer is accessed
  - If marked XOR, the contained ViewElements will be presented to the user only one at the time, as the user interacts with the system

View Elements

- **ViewComponents**: elements of the interface that display content or accept input from the user
  - ViewComponents exist only inside ViewContainers
  - A ViewComponent may correspond e.g. to a form, a data grid or an image gallery
  - A ViewComponent may be built up from ViewComponentParts

Parameter

- A Parameter is a typed element with multiplicity, whose instances hold values
- May be of a primitive type or a complex type such as object or collection
- Parameters are held by InteractionFlowElements and flow between them when Events are triggered
- Parameters may be mapped to a single element of the user interface (ViewComponentPart) or to a complex set thereof
- Parameters can be input, output or input-output
- A ParameterBinding determines to which input Parameter of a target InteractionFlowElement an output Parameter of a source InteractionFlowElement is connected
- ParameterBindings that flow together within an InteractionFlow are grouped by a ParameterBindingGroup

Events

Events are occurrences that can affect the state of the application, and they are a subtype of InteractionFlowElement

Events are classified in two main categories:

- **CatchingEvents** (events captured in the UI that trigger a subsequent interface change)
- **ThrowingEvents** (events that are generated by the UI)

CatchingEvents own a set of NavigationFlows and are further classified as:

- **ViewElementEvents** (owned by their related ViewElements)
- **ActionEvents** (owned by their related Actions)
- **SystemEvents** (stand-alone events)

Expressions

An Expression defines a side-effect free statement that will evaluate to a single instance, a set of instances, or an empty result

The subtypes of Expression are

- **InteractionFlowExpression**: determines which NavigationFlow should be followed if several NavigationFlows come out from an Event
- **ConditionalExpression**: is a ViewComponentPart defined inside a DataBinding to obtain content information from the DomainModel
- **BooleanExpression**: is an expression that evaluates to true or false. Specializations:
  - ActivationExpression: determines if a ViewElement, ViewComponentPart or Event is enabled
  - Constraint: restricts the behavior of any element

Content Binding

ViewComponents may retrieve content by means of ContentBinding

ContentBinding represents any source of content, optionally with an attribute containing the URI of the content resource

ContentBinding is specialized in two concepts:

- **DataBinding**
- **DynamicBehavior**

Content Binding

DataBinding

- references a DomainConcept (for instance, a Classifier in UML)
- is associated with a ConditionalExpression determining the content to be obtained from the source
- contains VisualizationAttributes used by ViewComponents to determine the features that may be shown to the user, such as a data base column

DynamicBehavior: represents a content access or business logic such as a service or method that returns a result after an invocation

Context

Context is a runtime aspect of the system used for configuring and displaying the user interface

The configuration and content of the user interface is determined by the **ViewPoint**, and thus Context is related to ViewPoint

A Context has several dimensions called ContextDimensions (**UserRole, Device, Position**)

When the user context satisfies all the ContextDimensions, access is granted to the ViewElements of the ViewPoint and to the Events that may be triggered on them

ContextVariables can be associated with Context to store primitive values (SimpleContextVariable) or objects (DataContextVariable) that store the state of the system in the current context

Specific View Components

Among the basic extensions of the core elements, ViewComponent includes the following specializations:

- **List**: used to display a list of DataBinding instances
  - If associated with an Event, each DataBinding displayed by the component may trigger that Event
  - The Event will cause the passing of the corresponding parameter values

- **Details**: used to display detailed information of a DataBinding instance
  - If associated with an Event, the Event will cause the passing of the Parameter values

Specific View Components

Form: used to display a form, composed of Fields to display or capture user content

- Fields have Slots that hold their value
- A Field can be a SelectionField or a SimpleField
- SelectionFields also behave as Parameters passed to ViewElements or Actions