Chapter 1. Fundamental Concepts

Table of Contents

This chapter is a brief informal introduction to Kompics. We begin with a discussion of the main concepts in Kompics, then describe how to download and install Kompics, and finally cover the build environment, based on Maven2.

Kompics: Components, Events, Ports and Channels

A component is a reactive unit of computation that communicates with other components asynchronously by passing data-carrying typed events. Events are passive immutable objects that can be serialized and communicated over network links and between different address spaces.

Components are loosely coupled in the sense that a component does not know the component type, availability or identity of any components with which it communicates. Rather, components are endowed with typed communication ports. A component sends events to and receives events from a local port. Ports of the same type on two different components can be connected by a bidirectional channel. This connection enables the two components to communicate.

As a reaction to received events, components execute event handlers, procedures specific to the types of events being received. During execution, event handlers may trigger new events, by sending them to one of the component's ports. An event handler is associated with events of a certain type, received though a certain port, by means of a subscription.

Table 1.1. Kompics Programming Abstractions

Entity	Description
Component	A *component* is a reactive unit of computation that communicates with other components asynchronously by passing data-carrying typed events over ports. Components contain handlers to execute received events, and components can be composed into *composite components*.
Event	An *event* is a passive immutable object that can be serialized and communicated over network links and between different address spaces.
Port	A port represents a bidirectional event interface of a component and it specifies the types of events that flow into or out of the component. The direction in which an event flows through a port is defined as either *positive* or *negative*. A negative event type flows towards the negative side of the port, while a positive event type flows towards teh positive side of the port. A port is illustrated in Figure 1.1, “Example port.”. By convention, the positive pole (+) of a port is understood to be the provided direction for events, while the negative pole (#) of a port is understood to be the required direction for events. When a component implements (or provides) a port, the port is oriented with its + pole to the outside of the component and its # pole inside. Conversely, a (required) port that is used by a component is oriented with its # pole outside and its + pole inside. The types of events flowing through the port from the # pole to the + pole are tagged with + and the types of events flowing from the + pole to the # pole are tagged with #.
Event Handler	An event handler is a procedure that a component executes as a reaction to receiving a certain event.
Subscription	A subscription binds an event handler to a port pole.
Channel	A channel is a first-class bidirectional connection between two ports of the same type. A channel can connect two ports of the same type and of different polarity.

In Figure 1.1, “Example port.”, we can see two Ports containing 2 events and 3 events, respectively. For `PortType1`, `e1` both goes "out" and comes "in". For `PortType2`, `e2` goes "out", while `e2` and `e3` come "in".

Figure 1.1. Example port.

A component with two ports (one outgoing, one incoming), two handlers and two subscriptions is illustrated in Figure 1.2, “Example component.”. The ports in this component are from Figure 1.1, “Example port.”. `PortType1` is provided by the component and `PortType2` is required by the component. We can see how the `subscriptions` map events from `Ports` to `handlers`, while handlers can trigger (or send) an `event` to a port (if the polarity of that port allows that event to be sent in that direction).

So, we can see for `PortType1`, the event `E1` can be both sent and received over this port. For `PortType2`, we can see that a handler inside the component can send either `E2` or `E3` to the port and handler inside the component can subscribe for `E2` (but not `E3`). For handlers or components outside this component (handlers would have to be in a parent component), they can send and receive events of the opposite type. So, for `PortType2`, they could subscribe for `E2` or `E3` events, and send `E2` events to the component.

Figure 1.2. Example component.

INPUt (Implements Negative, Positive Uses)

An easy way to remember whether a PortType refers to the client-side or server-side is to remember the idiom *INPUt* (Implements Negative, Positive Uses). INPUt reminds you that a negative PortType is one that is provided or implemented by a component (server-side), while a positive PortType is one that is used by a component (client-side).

Summary

We introduced a number of concepts for Kompics, and outlined the software requirements for downloading and installing Kompics.