

Design of the Controller and the Camera Zoom Behaviour

Werner Van Belle (werner@onlinux.be)

3rd December 2005

Contents

1	Introduction	1
2	component Controller	2
2.1	Description	2
2.2	Use Cases	2
2.2.1	Boot up	2
2.2.2	Component Join & Disjoin	3
2.2.3	Unwanted Component Disconnection & Component System Disconnection	3
2.3	Interfaces	4
2.3.1	port Component (noi=1)	4
2.3.2	port ComponentSystem (noi=1)	4
2.3.3	multiport Controller (noi=1)	4
2.4	Message Traces	5
2.4.1	Camera Component System Boot Up	5
2.4.2	Master Component System Boot Up	6
3	component ZoomBehaviour	7
3.1	Description	7
3.2	Use Cases	7
3.2.1	Logging of zoom-events	7
3.2.2	Selection of zoom events	7
3.2.3	Coupling of Zoom Behaviours	7
3.3	Interfaces	7
3.3.1	port Component (noi=1)	7
3.3.2	port Camera (noi<=1)	7
3.3.3	port ZoomRequest (noi=1)	8
3.3.4	multiport ZoomChange (noi=1)	8
3.3.5	port ZoomEvent (noi>=0)	8
3.3.6	port Behaviour (0<=noi<=1)	8
3.3.7	port UiDescription (0<=noi<=1)	8
3.4	Example Message Sequences	9
3.4.1	Logging of Zoom events	9
3.4.2	Coupling of 2 cameras	9
3.4.3	Coupling of 3 cameras, A with B, B with C	9

1 Introduction

This document describes the design of two components. The Controller component and the CameraZoomBehaviour component. The first component takes care of setting up interconnections between components, and the second takes care of zooming cameras and coupling the zoom behaviour of a number of cameras. All components are part of a test case for the SEESCOA project.

2 component Controller

2.1 Description

The controller his functionalities are described as follows:

- If new components connect or reconnect, all interested components will be notified.
- If components die or disconnect, the interested components will be notified.
- The controller takes care of distribution errors.
- The controller takes care of booting the system
- When new component systems connect to the component infrastructure, the controller will upload the necessary components to the target.

The controller is *not* responsible for the following:

- The controller does *not* handle runtime events and does not synchronise between components, nor does it contain any application logics.
- The controller is *not* responsible for security, account management, nor choosing between different events.

The controller in the second period of this project (year 2 to 4) can be extended to

- Offer component migration
- Remote update of components

2.2 Use Cases

2.2.1 Boot up

The boot up of the camera system contains two phases. The first phase covers the startup of the master-server, which also boots the controller. In the second phase, all cameras are looked up and new component systems can connect.

Phase 1: Master Component System Boot up

MCS	starts up with on the command line as only argument the controller component
MCS	loads the controller component
C	reads the initialisation file which contains a list of components to load
C	sends a create message to the MCS for all listed components
MCS	loads all components and initialises them. Normally, these component will not make connections with other components.
C	reads from the initialisation phase which components should connect to whom.
C	sends 'faked' Connect messages to all those components.

Phase 2: Client boot up

CCS	starts up with only the IP-address of the Master Component System on the command line.
CCS	connects to the MCS on the given address. Also connects to the Master Component System component.
MCS	accepts the connection, sends out a connect message to all interested parties (In this case, the controller)
C	looks up which components should exist at the given address.
C	sends out creation messages to the CCS for these components
CCS	creates the components (loads the classes from the server)
C	after creation of the new components, the controller will send the correct 'faked' connect messages.

2.2.2 Component Join & Disjoin

Components can subscribe themselves to retrieve notification of certain new components within the system. For example, a user interface component would like to subscribe to new cameras. The user interface component will receive from the controller a connect message when a new camera joins (or is created).

Disconnecting component can be initiated from anywhere. Every delete command must be sent to the controller, which will ask all components to disconnect themselves. If they don't the controller will take action. We will now illustrate connecting and disconnecting

Connecting

UI	is loaded and requests the controller a join for 'cameras'
CC	The camera component is loaded (by the controller)
C	Sends a HasJoined message to the UI
UI	Sends a connect message to the camera

Disconnecting

UI	wants to disjoin, sends a message to the controller
C	sends a disconnect to the UI
UI	disconnects all its ports

2.2.3 Unwanted Component Disconnection & Component System Disconnection

At the moment a component disconnects because there was an error (crash in the component) or a network failure, all dependent components will be notified with a HasDisjoined message. The messages which trigger such an action are

- ComponentSystemDisconnect
- ComponentSystemFailed
- ComponentFail
- ComponentDisconnected

2.3 Interfaces

2.3.1 port Component (noi=1)

in Init() is send by the component system. This is guaranteed the first message a component will receive

in Connect(Port:<String>, With:<String>) connects the sender his port With with the port Port of the receiver.

in Disconnect(Who:<String>, From:<String>) disconnect Who from port From at receiver side

2.3.2 port ComponentSystem (noi=1)

in ComponentSystemDisconnect(ComponentSystem) see description of the ComponentSystem component

in ComponentSystemFailed(ComponentSystem) see description of the ComponentSystem component

in ComponentSystemConnect(ComponentSystem) see description of the ComponentSystem component

in ComponentSystemQueueOverflow(ComponentSystem, Reason) see description of the ComponentSystem component

in ComponentDisconnect(Component) see description of the ComponentSystem component

in ComponentConnect(Component) see description of the ComponentSystem component

in ComponentFailed(Component) see description of the ComponentSystem component

out CreateComponent(Blueprint, Name) is used to create all the necessary components

2.3.3 multiport Controller (noi=1)

in LookingFor(NameSubstring) Requests the controller to look for components with a name which contains the substring Name. In response to this message, the controller will send back all existing components and will from then on notify the requester of new components mathcing the given name.

out HasJoined(Who) is send to notify everybody that Who has joined. This message is only send to all people subscribed to the given NameSubstring.

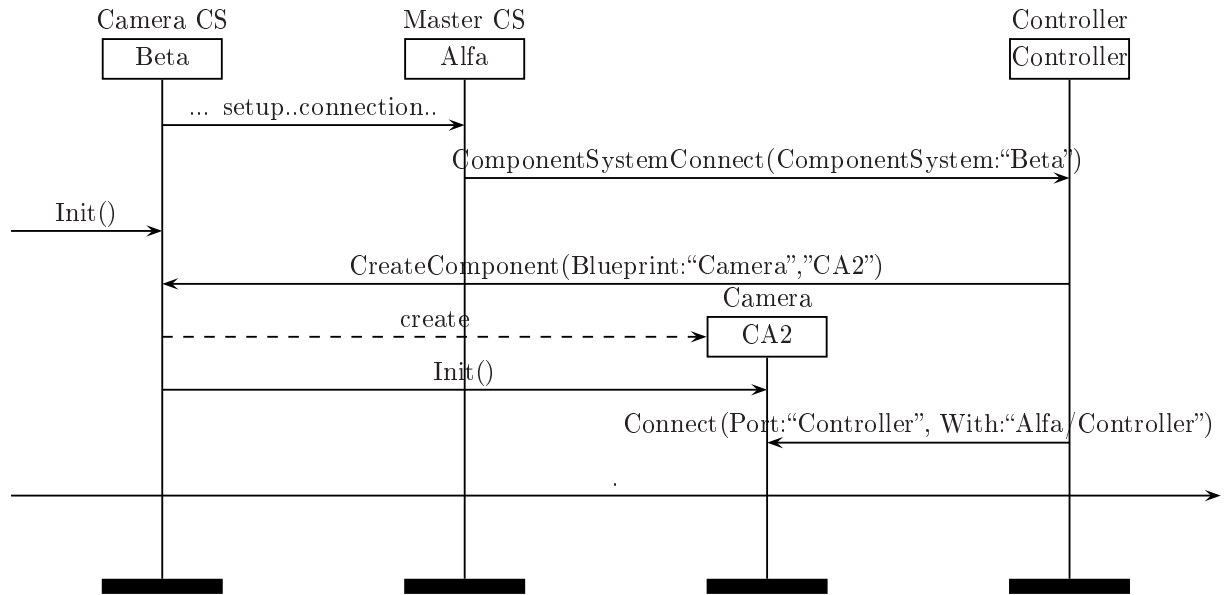
out HasDisjoined(Who) is send as a notification of a disjoin. Is send only to subscribes components.

out Alive(Who) is send out to check whether other component systems are still alive.

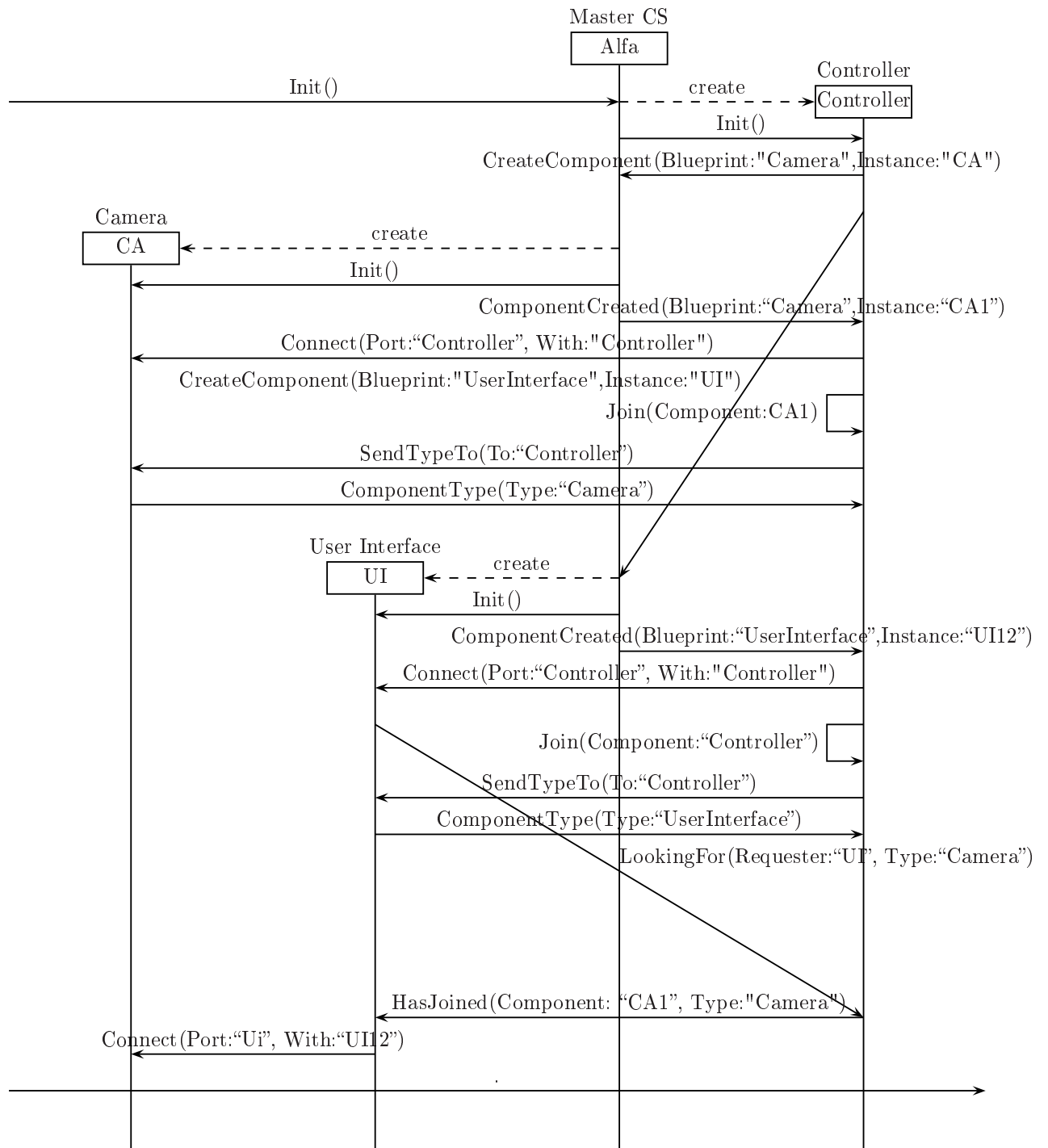
in AreYouAlive() response of previous message

2.4 Message Traces

2.4.1 Camera Component System Boot Up



2.4.2 Master Component System Boot Up



3 component ZoomBehaviour

3.1 Description

This is one of the plug in components, which is added to the system to show its flexibility.

- The zoom behaviour component is able to notify all subscribed components of a zoom changed notification.
- Zoom behaviour components can be coupled such that one camera zooms in while another zooms out and vice versa.

3.2 Use Cases

3.2.1 Logging of zoom-events

It is possible to log zoom events. If we want this we should connect to the zoom behaviour zoom changed port.

3.2.2 Selection of zoom events

It should be possible to select (prioritise) between a number of camera zoom events. If for example, 2 users are zooming in at the same time, together with an operator on the camera, the zoom behaviour controller should select one user and stick to him during a certain period of activity.

3.2.3 Coupling of Zoom Behaviours

It should be possible to couple two (or more) cameras, such that the first camera zooms in while the second one zooms out. The constraints placed upon the zoom will be simple linear equations.

3.3 Interfaces

3.3.1 port Component (noi=1)

in Init() is send by the component system. This is guaranteed the first message a component will receive

in Connect(Port:<String>, With:<String>) See ComponentSystem component

in Disconnect(Who:<String>, From:<String>) See ComponentSystem component

3.3.2 port Camera (noi<=1)

We will use a port to the camera status to obtain information about the maximum zooming parameters. This port is temporary. After obtaining the values this port is not used anymore.

out GetMax(Type:<String>) requests the maximum zoom

out GetMin(Type:<String>) requests the minimum zoom

in Value(Type:<String>, Value:<Integer>) answers to both questions above

3.3.3 port ZoomRequest (noi=1)

This is the port offered by all cameras, which we have to implement in order to intercept certain zoom events.

in ZoomChangeRequest(From:<String>, Value:<Integer>) incoming message from the camera, to ask whether the zoom can be changed to the given value. If this is allowed this component will send back a ZoomChangeAction.

out ZoomChangeAction(NewValue:<Integer>) this message is send back to the camera when the zoom should actually change.

3.3.4 multiport ZoomChange (noi=1)

This port is necessary for anybody interested in zoom-change events. This is a stripped down interface of the camera-settings port and will be used by other ZoomBehaviour components.

out ZoomChanged(Camera:<String>, Value:<Integer>, >InitiatedFrom<:[]) send out whenever the camera reports a change of value of the zoom. Camera is the name of the controlling camera. InitiatedFrom is an array with cameras, which has already send out a ZoomChanged.

3.3.5 port ZoomEvent (noi>=0)

This port is necessary for anybody who wants to report a ZoomChanged event at this controller.

in ZoomChanged(Camera:<String>, Value:<Integer>, >InitiatedFrom<:[]) Camera is the name of the camera for which the zoom has changed to value Value. In response to this message, this camera can change its own zoom factor. InitiatedFrom is an array of names which already sent a ZoomChanged.

3.3.6 port Behaviour (0<=noi<=1)

This port is used to change the formule used in each controller to react to ZoomChanged events.

in SetFormule(A:<Integer>[], B:<Integer>) The formule is a linear equation. The integer array contains the coefficients. The integer B contains the extra value added to all this.

3.3.7 port UiDescription (0<=noi<=1)

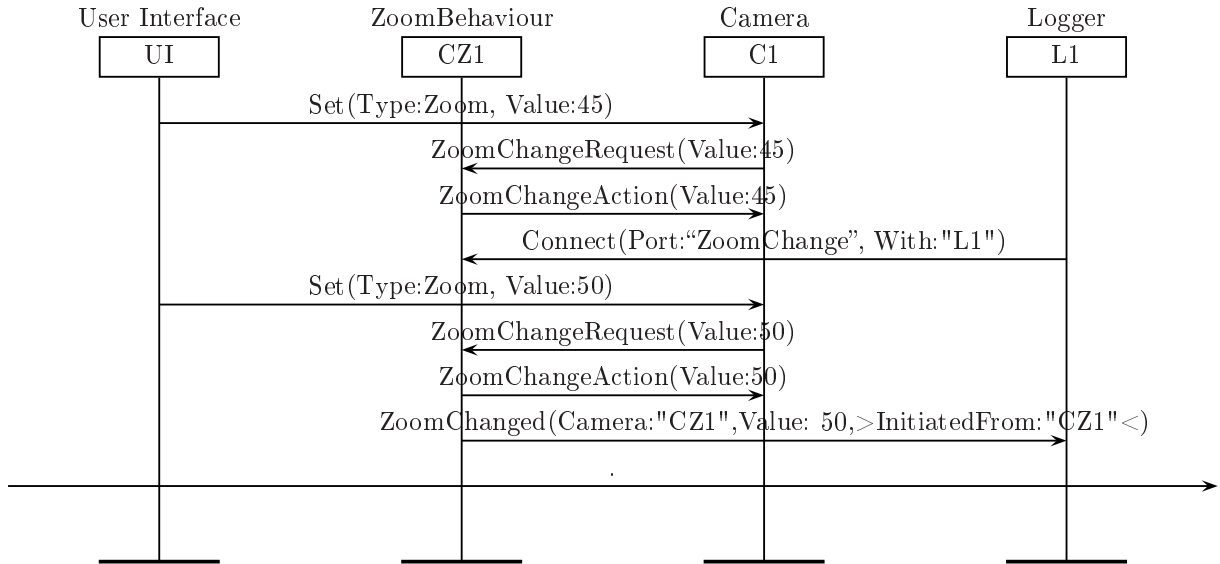
This port can be used by anybody to receive an XML description of the behaviour port.

in GetUiDescription() received from somewhere. In response we have to send back an XML tree which contains the possible behaviour of this component.

out PutUiDescription(Xml:<String>) Send out. Will probably contain the parameters of the linear equation.

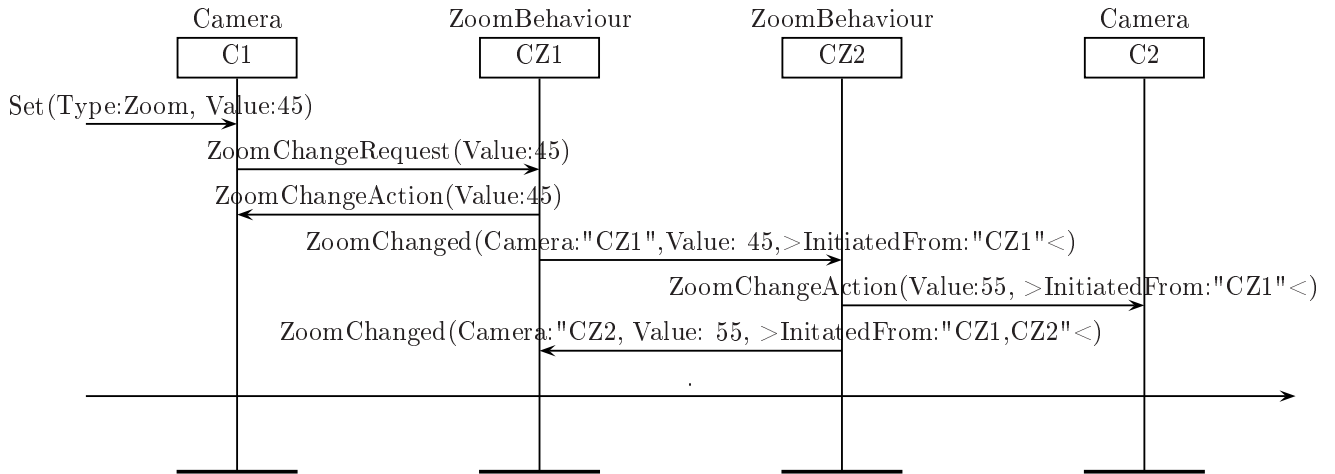
3.4 Example Message Sequences

3.4.1 Logging of Zoom events



3.4.2 Coupling of 2 cameras

This is an example of two cameras coupled together.



3.4.3 Coupling of 3 cameras, A with B, B with C

Below is an example of 3 cameras coupled together. The first is coupled to the second, the second is coupled to the third. The third camera is not immediately connected with the first.

