Modelling and Verifying an Evolving Distributed Control System Using an Event-based Approach

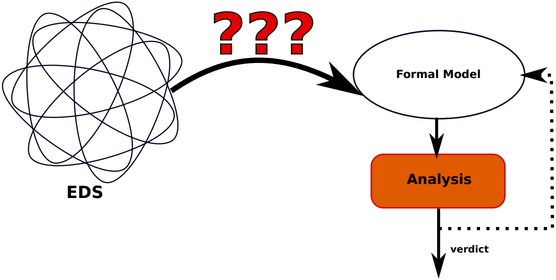
Christian ATTIOGBE

University of Nantes – AeLoS / LINA – UMR CNRS 6241

ISoLA, Corfu, 8-11 October, 2014

What are we doing?

Figure : An evolving distributed system



Agenda

Outline

Context

- Distributed systems
  have been the subject of years of research and development
- Dynamic environments
  involve **additional difficulties at various levels:**
  - identification of the hosts,
  - structuring of the exchanged messages,
  - structure of the overall architecture of the system: the dynamic aspect of the links,
  - messages are exchanged between peers and their environment.

☞ **Modelling and analysis** should not consider the precise structures of interacting entities but their virtual counterpart.

Motivations

- Design of decentralized or highly interacting components: *evolving distributed control systems*. **How to control?**
- Virtualized distributed environments (aka *cloud*) push difficulties at applications level. **How to design?**
- Heterogeneity of components (physical devices, software, various models) is a specific feature of these systems. **How to compose?**
- Modelling and reasoning on software systems which will be implemented as evolving dynamic distributed systems. **How to design?**

Objectives and contributions

The contributions of this work:

- An event-based approach for the modelling and the analysis of dynamic distributed control systems.
  - The approach is based on the use of a virtual network of processes, event-based modelling and the refinement.
- A case study: a Closed-Circuit Television (CCTV) control system.
  - A pattern to be reused for similar cases.

Outline

The example of a CCTV system

Figure : A CCTV control room

- The remote control of industrial plants, homes, various distributed equipments/devices.
- The video captured by cameras are displayed on dedicated screens which may have their own controllers.
- A central or a decentralized control room.
- Dynamic configuration of the devices.

☞ Evolving distributed system

Features of evolving distributed systems

- They are made of an undefined number of peers which cooperate to provide services.
- The peers may be mobile, linked and unlinked to different other peers; peers may leave the system, new peers may be involved.
- The architecture of the system is therefore continuously changing.
- The message-passing technique with explicit naming and peers identification is not tractable in such an adhoc context.
- An implicit message-passing is needed instead.
- The required functionalities and properties of the distributed system should be preserved.

☞ We adress modelling and analysis

The CCTV case study: abstraction (1)

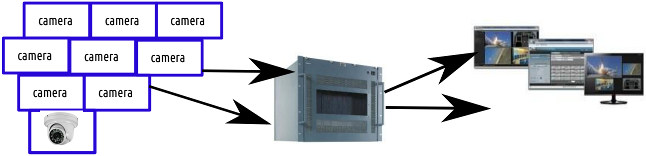


Figure : An abstraction of the CCTV

- A controller is linked with one or several (input) cameras; it displays its output on one or several screens, and it may have a Digital Video Recorder for storage.

The CCTV case study: abstraction (2)

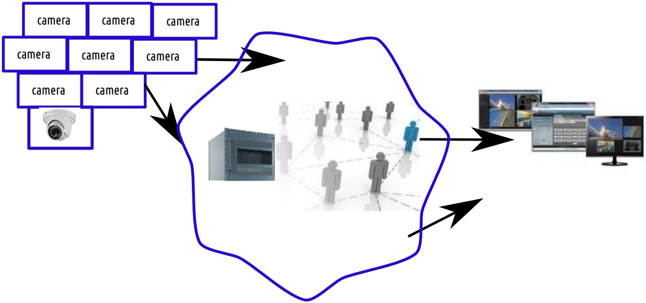


Figure : An abstraction of the evolving CCTV system

- Initially there is no recording of video;
  it can be decided to record and save the video while displaying them.
- A digital video recorder component will be introduced in the system.

The requirements of the case study

FReq_AreaOK	*All the area to be supervised are under control*	**?**
FReq_CtrlNCam	*Each controller can manage several cameras*	**?**
FReq_Cam1Ctrl	*Each camera is managed by only one controller*	**?**
FReq_DispOk	*All the captured videos are displayed on some screens*	**?**
FReq_RecDispOK	*All the active cameras should be under control*	**?**
FReq_RecOK	*All video received from the cameras are recorded*	**?**
FReq_NewDev	*It should be possible to add new cameras and screens*	**?**

Table : Synthesis of the functional requirements

☞ How to model and design?

Outline

An event-based modelling method

We proposed *P-B (2006-2009) to deal with multiprocess modelling in Event-B.

An event-based composition is a weak coupling of processes that interact through a common state space: **a virtual net of processes**.

The model of a global system consists of

- a set of *process types*: the identified components of the global system.
- a set of abstract channels to support the communication between the processes.
- a global state space (**invariant predicate**): the data identified in the requirements.
- a set of behavioural descriptions of the process types (as **guarded events**).
- each process P_i has a state space S_i, a nondeterministic event-based behaviour:

$$P_i \;\widehat{=}\; \langle S_i, E_i, Evt_i \rangle$$

Virtual net of interacting components

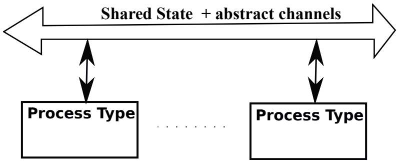


Figure : Virtual net of process types (a)

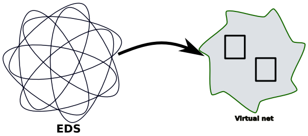


Figure : Virtual net of process types (b)

Handling the evolution of architecture.

- An architecture: a set of processes of various types connected to the channels.
- An instance of a process type may join/leave the configuration at any time.

Interaction

Composition of the processes.

- The described processes P_i are combined by a *fusion* operation $\biguplus$.
  The semantics of the fusion operator comes from the conjunction of processes paradigms (Zave, Jackson, Abadi).
- The fusion operator merges the state spaces and the events of the processes into a single global system Sys_g which has the conjunction of the invariants.

$$Sys_g \mathrel{\widehat{=}} \biguplus_i P_i = \biguplus_i \langle S_i, E_i, Evt_i \rangle = \langle S_g, E_g, Evt_g \rangle$$

- An **incremental**, bottom-up view is adopted.
- Approaches such as UPPAAL, work this way but the behaviours are more constrained (automata).

Outline

Constructing the model (1)

- The identified components (P_i): cameras, screens, controllers, DVR, Abstract channels to model the shared communication: videoChannel, $\cdots$.
- Two levels of abstractions:

  First abstraction: mastering the main requirements and properties.

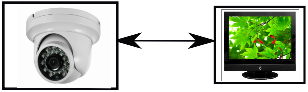


Figure : Abstraction of the functionalities

Refinement: introduction of the details and other required properties.

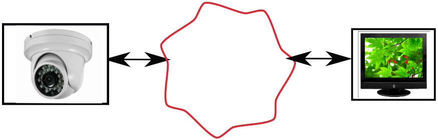


Figure : Refinement of the functionalities

Constructing the model (2)

At the first abstract level, we consider only the Cameras and the Screens

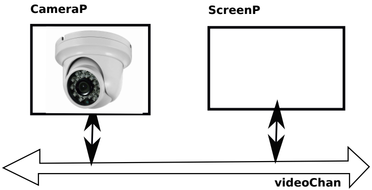


Figure : A net of two process types

Constructing the model (3)

At this first abstract level, we combine the Cameras and the Screens.

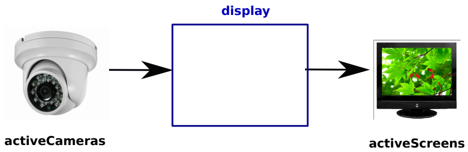


Figure : Abstraction of the functionalities

The virtual net is:

$$Sys_g \mathrel{\widehat{=}} \biguplus_i \{CameraP, ScreenP\}$$

$$display \in activeCameras \twoheadrightarrow activeScreens$$

Active cameras are connected to the active screens, all the active screens are used.

Modelling the behaviour of the Camera

The behaviour of a camera is to send a stream of captured signals to the linked screens via the control units.

Camera behaviour	
Event	Description
addCamera	a new camera is added
activateCamera	one camera is activated
sendVideo	a camera sends a video
rmvCamera	a camera is removed

Table : Camera handling events

A specific language can be introduced: Event-B as the target experimental support.

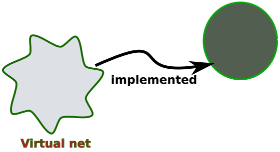


Figure : Implementation of the net

Implementation in Event-B

The Event-B specification to manage a Camera:

MACHINE *CameraHdl*
SETS CAMERA, VIDEO
VARIABLES
  connectedCameras, *activeCameras*, *display*, *videoChan*, *nootherPriority*
INVARIANT
  /* state space predicate */
INITIALISATION
  connectedCameras, *activeCameras*, *display*, *videoChan*,
  nootherPriority := $\emptyset, \emptyset, \emptyset, \emptyset, \emptyset$
EVENTS
  `addCamera` $\widehat{=}$ $\cdots$ /* guarded logical substitution */
; `activateCamera` $\widehat{=}$ $\cdots$
; `sendVideo` $\widehat{=}$ $\cdots$
; `rmvCamera` $\widehat{=}$ $\cdots$
END

Figure : Structure of the Camera abstract machine

Modelling the behaviour of the Screen

The behaviour of a screen consists in visualising the streams of signals received from the cameras.

Screen behaviour	
Event	Description
addScreen	a screen is added
getVideo	a screen gets a video
displayVideo	a screen displays a video
rmvVideo	a screen is removed

Table : Screen handling events

Implementation of the virtual net in Event-B

MACHINE *CameraHdl* $\cdots$

INVARIANT /* state space predicate */

$connectedCameras \subseteq CAMERA$

/* set of connected cameras */

$\wedge$ $activeCameras \subseteq CAMERA$

$\wedge$ $activeCameras \subseteq connectedCameras$

/* the active cameras */

$\wedge$ $videoChan \in \mathcal{P}(VIDEO \times CAMERA)$

/* abstract channel*/

$\wedge$ $\cdots$ /* more properties */

MACHINE *ScreenHdl* $\cdots$

INVARIANT

$connectedScreens \subseteq SCREEN$

/* the set of connected screens */

$\wedge$ $activeScreens \subseteq SCREEN$

$\wedge$ $activeScreens \subseteq connectedScreens$

$\wedge$ $videoChan \in \mathcal{P}(VIDEO \times CAMERA)$

/* abstract channel */

$\wedge$ $\cdots$

/* more properties */

$\wedge$ $display \in activeCameras \twoheadrightarrow activeScreens$

/* the active cameras are connected to active screens */

/* All the active screens are used **FReq_Cam1Ctrl** and **FReq_DispOK** */

Implementation in Event-B: merging

MACHINE *CameraHdl* $\cdots$

INVARIANT /* state space predicate */

$connectedCameras \subseteq CAMERA$
/* the set of connected cameras */

$\wedge$ $connectedScreens \subseteq SCREEN$
/* the set of connected screens */

$\wedge$ $activeCameras \subseteq CAMERA \wedge activeScreens \subseteq SCREEN$

$\wedge$ $activeCameras \subseteq connectedCameras$
/* the active cameras are part of the connected ones */

$\wedge$ $activeScreens \subseteq connectedScreens$

$\wedge$ $display \in activeCameras \twoheadrightarrow activeScreens$
/* the active cameras are connected to active screens */
/* All the active screens are used **FReq_Cam1Ctrl** and **FReq_DispOK** */

$\wedge$ $videoChan \in \mathcal{P}(VIDEO \times CAMERA)$
/* abstract channel: set of video+cameraId */

$\wedge$ $videoStream \in VIDEO \nrightarrow activeScreens$
/* the video are displayed on one screen */

$\wedge$ $displayedVideo \in \mathcal{P}(VIDEO)$

$\wedge$ $\cdots$ /* more properties are added below */

Analysis: verifying the required properties

FReq_RecDispOK: (restructured)
When the DVR is installed, all the displayed video are recorded and saved to ensure the DCA.

The property FReq_RecDispOK is modelled as follows:

$$((activeDVR \neq \{\} \wedge videoStore \neq \{\}) \Rightarrow$$
$$((\mathrm{dom}(videoStore) \subseteq displayedVideo)$$
$$\vee$$
$$(\mathrm{dom}(videoStore) \setminus (displayedVideo \cap \mathrm{dom}(videoStore)) \subseteq \mathrm{dom}(videoStream))))$$

The properties are included in the invariant.

Increasing the model by refinement

Any configuration of the distributed system should preserve the initial requirements; the link between cameras and screens through the controllers is detailed.

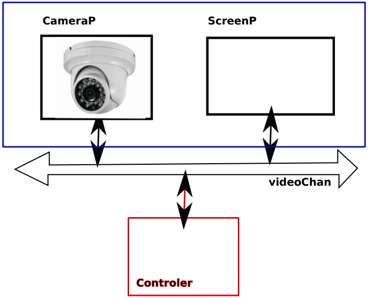


Figure : Hierarchical composition

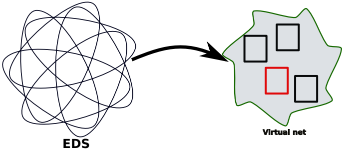


Figure : A larger virtual net

Increasing the model by refinement

One possible configuration of the distributed system: it is a refinement.

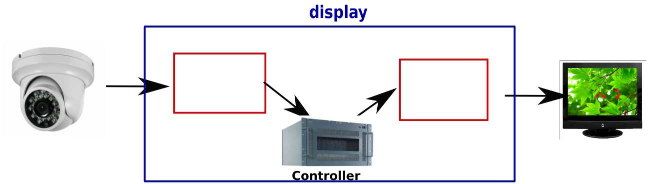


Figure : Refinement of the functionalities

How to design and prove it?

Verification by refinement: a pattern

The refinement we have used is summarized as follows:
a function $f : A \twoheadrightarrow B$ is refined by two relations g and h together with a new set I:

Abstraction	$f : A \twoheadrightarrow B$
Refinement	$g : A \rightarrow I \quad \wedge \quad h : I \twoheadrightarrow B \wedge$ $f \subseteq (g; h)$

I stands for the set of Controllers which have been introduced via the refinement;
g captures the property **FReq_Cam1Ctrl** (Each camera is managed by only one controller)
h captures the property **FReq_CtrlNCam** (Each controller can manage several cameras).

Formal analysis and results

- Using this two modelling levels, we ensure that the structure/policy deployed by the controllers does not impact on the properties to be preserved.
- This answers the initial problem to be solved when considering evolving distributed systems.
- This approach enables us to master the complexity of the model and also to master the verification of the properties.

FReq_AreaOK	✔
FReq_CtrlNCam	✔
FReq_Cam1Ctrl	✔
FReq_DispOk	✔
FReq_RecDispOK	✔
FReq_RecOK	✔
FReq_NewDev	✔

- It can be reused elsewhere as a modelling and verification pattern.

Experimentation with Event-B/Rodin

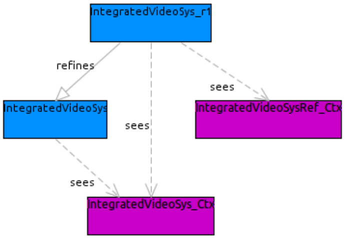


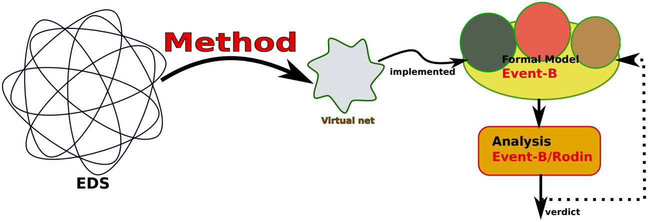
Figure : Structure of the models

ElementName	Total PO	Auto proved
IVS_CCTV (project)	30	30
IntegratedVideoSysRef_Ctx	0	0
IntegratedVideoSys_Ctx	0	0
IntegratedVideoSys	24	24
IntegratedVideoSys_r1	6	6

Table : Statistics

Outline

Conclusion and future works



- An appoach to model an evolving distributed system and to verify its properties.
- An experimentation with the CCTV case study and Event-B/Rodin
- A reusable pattern for similar cases

Future works:

- Enhancement of the heterogeneity (currently handled using Event-B)
- Parameterization to handle design constraints on the evolving architecture.

Thanks

Thank for your attention!

Questions?