Typing Adaptation Monitors for Actor Systems

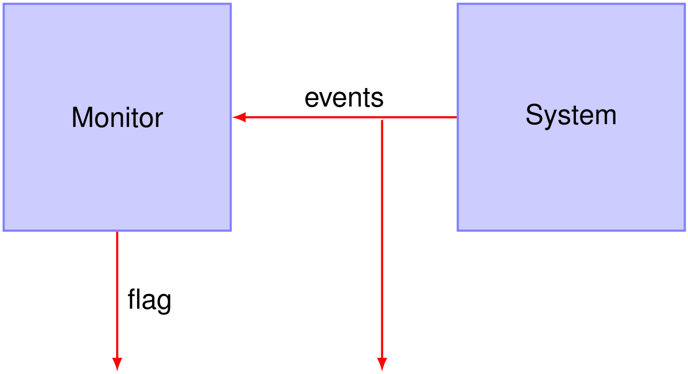
Ian Cassar & Adrian Francalanza

University of Malta

{*ian.cassar.10, adrian.francalanza*}*@um.edu.mt*

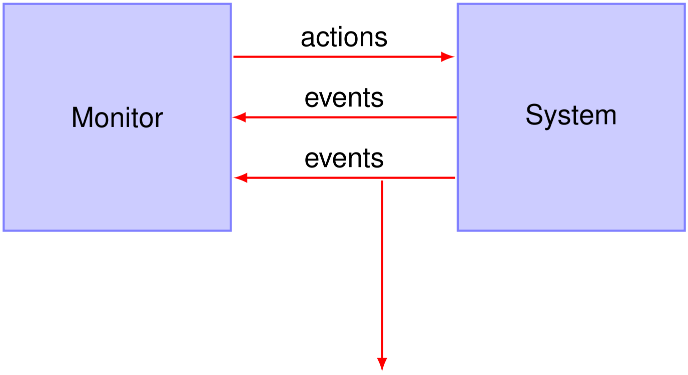
17th March 2016

Monitors Variants (Runtime Verification Monitors)



Detect and Flag Errors then stop.

Monitors Variants (Runtime Adaptation Monitors)



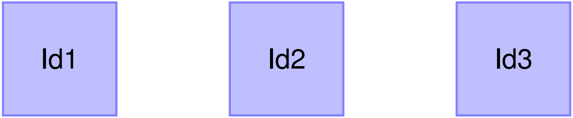
Detect errors, adapt system (automate recovery) then continue.

Verifying Monitor Correctness

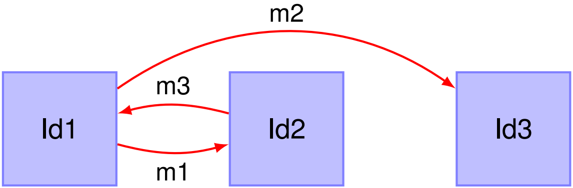
The Monitor should be part of the Trusting Computing Base.

- Monitors rarely come with guarantees — may introduce errors.
- The Monitor's verdict is *useless* if the monitor is *incorrect*.

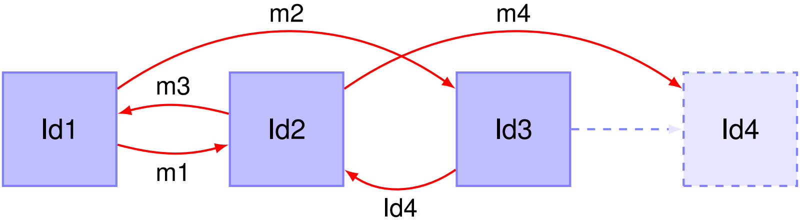
Actor Systems



Actor Systems



Actor Systems



Technologies and Characteristics

Actor Languages/Frameworks

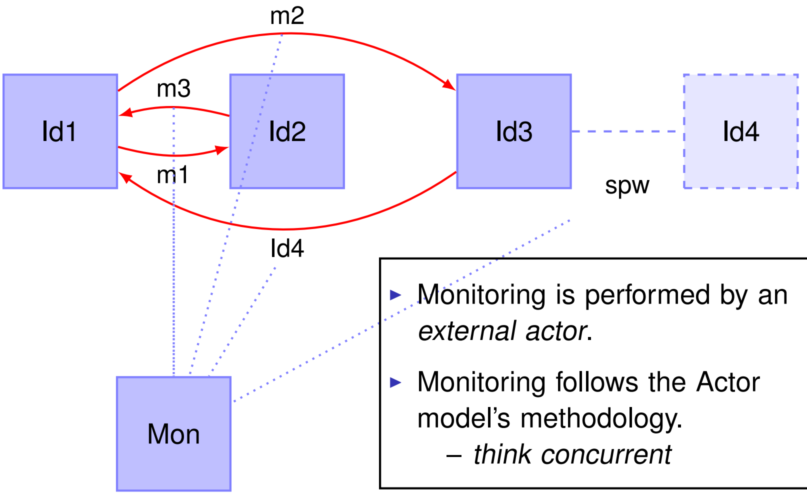
- Erlang
- Akka for Scala and Java
- Salsa (JVM)
- Stage (Python)

Monitoring and Adapting Actor Systems

- Our Monitors take advantage of the fact that Actor Systems are not Monolithic.
- The Monitors can effect specific parts of the system without effecting others.

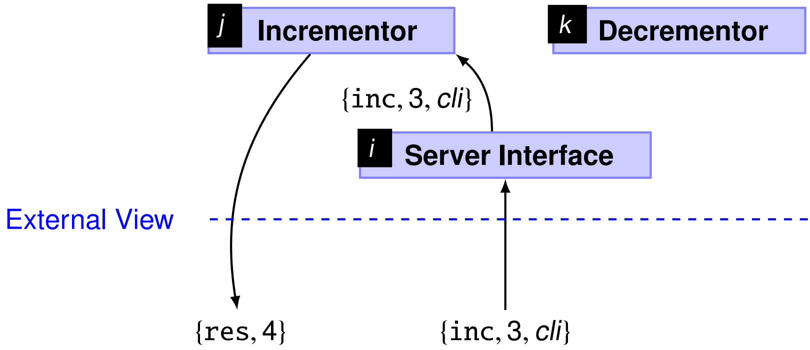
- We start from RV Monitors ...
  - designed to passively observe individual parts of actor systems...
- and we move to RA Monitors ...
  - we introduce control capabilities by which the monitors can influence certain parts of the system based on the observed behaviour.

Monitoring and Adapting Actor Systems

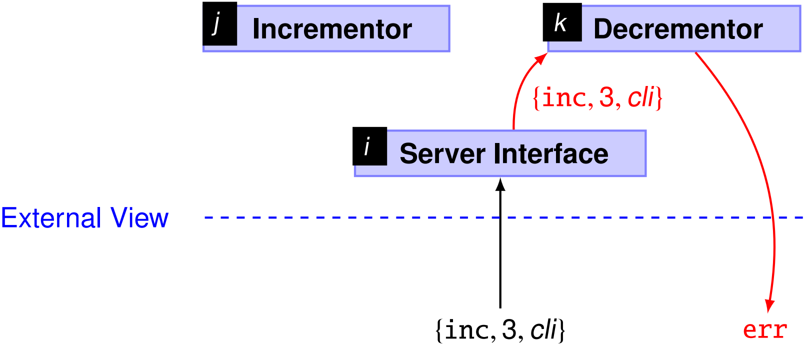


- Monitoring is performed by an *external actor*.
- Monitoring follows the Actor model's methodology.
  – *think concurrent*

Actor System Example



Actor System Example



Example Property (Detection Monitors)

$$
\max Y.\ [i?\{\mathtt{inc}, x, y\}]
\left(\begin{array}{l}
([j \triangleright y\,!\{\mathtt{res}, x+1\}]\,Y) \\
\ \& \\
([z \triangleright y\,!\,\mathtt{err}]\,\mathsf{ff})
\end{array}\right)
$$

Example Property (Detection Monitors)

$$
\max\ Y.\ [i?\{\texttt{inc}, x, y\}] \left(\begin{array}{l} ([j \triangleright y\,!\,\{\texttt{res}, x+1\}]\, Y) \\ \ \& \\ ([z \triangleright y\,!\,\texttt{err}]\, \mathsf{ff}) \end{array} \right)
$$

Example Property (Detection Monitors)

$$\max Y.\ [i?\{\mathtt{inc}, x, y\}] \left(\begin{array}{l} ([j \triangleright y!\{\mathtt{res}, x+1\}]\ Y) \\ \ \& \\ ([z \triangleright y!\mathtt{err}]\ \text{restart}(i).\ \text{purge\_mbx}(z).\ Y) \end{array} \right)$$

Example Property (Adaptation Monitors)

$$\max\ Y.\ [i?\{\mathtt{inc}, x, y\}] \left(\begin{array}{l} ([j \triangleright y\,!\{\mathtt{res}, x+1\}]\ Y) \\ \quad \& \\ ([z \triangleright y\,!\mathtt{err}]\ \mathsf{restart}(i).\ \mathsf{purge\_mbx}(z).\ Y) \end{array} \right)$$

Example Property (Adaptation Monitors)

$$\max Y.\ [i?\{\texttt{inc}, x, y\}] \left(\begin{array}{l} ([j \triangleright y\,!\,\{\texttt{res}, x+1\}]\ Y) \\ \ \& \\ ([z \triangleright y\,!\,\texttt{err}]\ \textcolor{red}{\mathsf{restart}(i).\ \mathsf{purge\_mbx}(z)}.\ Y) \end{array} \right)$$

Issues

- Full-blown synchronous monitoring is not an option:
  1. Actor Systems are inherently **asynchronous**.
  2. Synchronous monitoring carries **huge overheads** [1].
  3. We must synchronise a **minimal subset** to allow correct adaptation.

Monitor Synchronisations for Actor Systems

- We thus introduce mechanisms to incrementally block and unblock actors.
- We can block actors *before* a potential violation occurs
  - We *release* blocked actors when we the monitor collects more information showing that the predicted violation will not occur.
  - We apply *rectifying adaptation actions* immediately when the monitor collects enough information to confirm that the predicted violation has occurred.

Proposed Mechanism for Incremental Synchronisation

$$[\alpha]\ \phi$$

Proposed Mechanism for Incremental Synchronisation

$$[\alpha]^{\text{blist}}\ \phi$$

The Blocking list — A list of actor ids that will be *Blocked* after system action α is performed.

Proposed Mechanism for Incremental Synchronisation

$$[\alpha]^{\text{blist}}_{\color{red}\text{rlist}}\ \phi$$

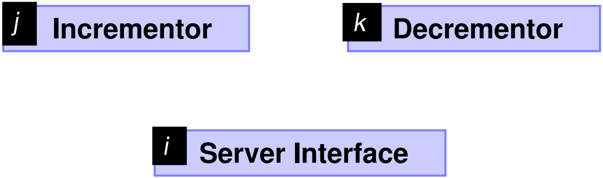
The Release list — A list of actor ids that will be *Released* after system action α is **not** performed.

Proposed Mechanism for Incremental Synchronisation

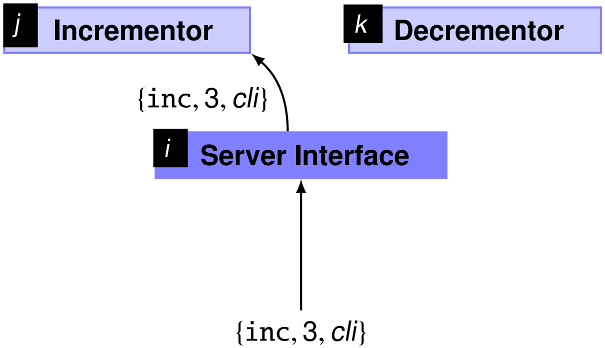
$$[\alpha]^{\text{blist}}_{\text{rlist}}\,\phi$$

$$\max Y.\ [i?\{\mathtt{inc}, x, y\}]^{i} \left(\begin{array}{l} ([j \triangleright y\,!\,\{\mathtt{res}, x+1\}]\, Y) \\ \quad \& \\ ([z \triangleright y\,!\,\mathtt{err}]^{z}_{i}\ \ \text{restart}(i).\ \text{purge\_mbx}(z)_{i,z}.\ Y) \end{array} \right)$$

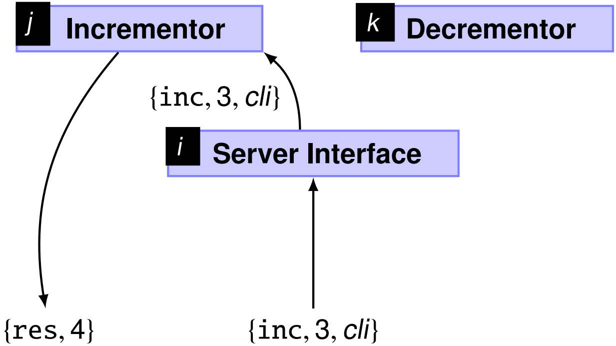
Actor System Example with Incremental Synchronisations



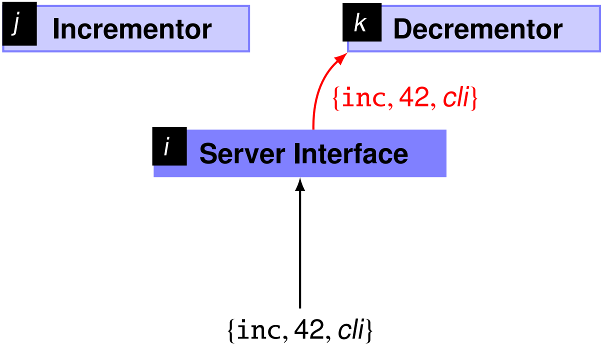
Actor System Example with Incremental Synchronisations



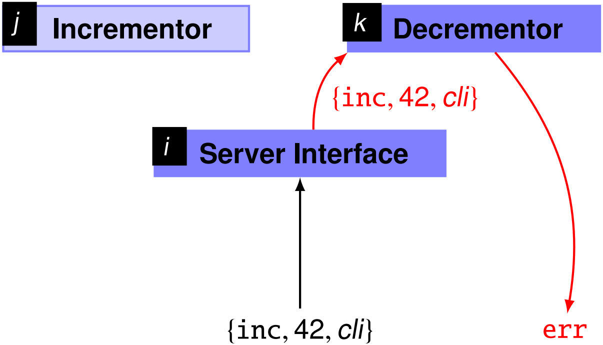
Actor System Example with Incremental Synchronisations



Actor System Example with Incremental Synchronisations



Actor System Example with Incremental Synchronisations



Programming Synchronisations

Rationale

- Hard to infer automatically.
- There are many ways how to carry out incremental synchronisations.

Programming Synchronisations

Rationale

- Hard to infer automatically.
- There are many ways how to carry out incremental synchronisations.

Disadvantage

We allow the possibility for **synchronisation errors**.

Examples of Potential Errors

$$
\begin{array}{l}
\max\ Y.\ [i?\{\texttt{inc}, x, y\}]^{i} \\
\qquad \left(\begin{array}{l}
(\,[j \triangleright y\,!\,\{\texttt{res}, x+1\}]\quad Y) \\
\ \ \& \\
(\,[z \triangleright y\,!\,\texttt{err}]_{i}^{z}\ \ \mathrm{restart}(i).\ \mathrm{purge\_mbx}(z)_{i,z}.\ Y)
\end{array}\right)
\end{array}
$$

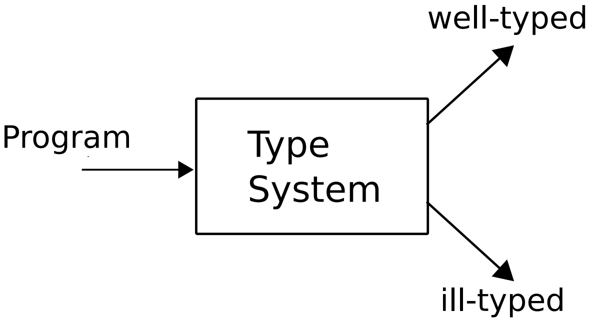
Examples of Potential Errors

$$\max Y.\ [i?\{\texttt{inc}, x, y\}]^{i} \left(\begin{array}{l} (\,[j \triangleright y\,!\,\{\texttt{res}, x+1\}] \quad Y) \\ \ \& \\ (\,[z \triangleright y\,!\,\texttt{err}]_{i}^{z} \quad \text{restart}(i).\ \text{purge\_mbx}(z)_{i,z}.\ Y) \end{array} \right)$$

Examples of Potential Errors

$$
\max\ Y.\ [i?\{\mathtt{inc}, x, y\}]^{i}
\left(
\begin{array}{l}
\left([j \triangleright y\,!\,\{\mathtt{res}, x + 1\}]_{i}\ Y \right) \\
\ \& \\
\left([z \triangleright y\,!\,\mathtt{err}]_{i}^{z}\ \ \mathrm{restart}(i).\ \mathrm{purge\_mbx}(z)_{i,z}.\ Y \right)
\end{array}
\right)
$$

A Type System for Adaptation Scripts



Theorem

Well-Typed Scripts *are guaranteed* ***not to generate synchronisation errors*** *at runtime.*

A Type System for Adaptation Scripts

We introduce two primary types for monitor variables.

- Linear Typed Variables
  - can be used only amongst a single concurrent branch ($\varphi \,\&\, \psi$);
  - used to bind actor ids that can be used in blocking, releasing and adaptation mechanisms.

A Type System for Adaptation Scripts

We introduce two primary types for monitor variables.

- Linear Typed Variables
  - can be used only amongst a single concurrent branch ($\varphi \,\&\, \psi$);
  - used to bind actor ids that can be used in blocking, releasing and adaptation mechanisms.
- Unrestricted Typed Variables
  - can be *shared amongst multiple* concurrent branch ($\varphi \,\&\, \psi$);
  - used to generic data and actor ids that are only used for monitoring purposes.

A Type System for Adaptation Scripts

$$\textsc{tCn1}\ \frac{\Sigma;\Gamma_1 \vdash \varphi \qquad \Sigma;\Gamma_2 \vdash \psi}{\Sigma;(\Gamma_1 + \Gamma_2) \vdash \varphi \,\&\, \psi}$$

- The value type environment Γ is *split* into Γ1 and Γ2.
- They do not share linear identifiers (*lid* and *lbid*).
- They may share unrestricted variables.

A Type System for Adaptation Scripts

$$\text{TNCB}\ \frac{\Sigma; (\Gamma, \mathsf{bnd}(e)) \vdash \mathsf{blk}(b)\, \varphi \qquad \Sigma; \Gamma \vdash \mathsf{rel}(r)\, \mathsf{tt}}{\Sigma; \Gamma \vdash [e]^{b}_{r}\, \varphi}$$

If *e* occurs then the actors in *b* should be blocked, while those in *r* should be released.

A Type System for Adaptation Scripts

$$\textsc{tBlk}\ \frac{\Gamma = \Gamma', w : \mathsf{lid} \qquad \Sigma; (\Gamma', w : \mathsf{lbid}) \vdash \varphi}{\Sigma; \Gamma \vdash \mathsf{blk}(w)\, \varphi}$$

If the actors in w are typed as *linear ids* (*lid*) then their type changes to *linear blocked* upon a block operation.

A Type System for Adaptation Scripts

$$\text{tBLK}\ \frac{\Gamma = \Gamma', w : \mathsf{lid} \qquad \Sigma; (\Gamma', w : \mathsf{lbid}) \vdash \varphi}{\Sigma; \Gamma \vdash \mathsf{blk}(w)\, \varphi}$$

If the actors in *w* are typed as *linear ids* (*lid*) then their type changes to *linear blocked* upon a block operation.

$$\text{tREL}\ \frac{\Gamma = \Gamma', w : \mathsf{lbid} \qquad \Sigma; (\Gamma', w : \mathsf{lid}) \vdash \varphi}{\Sigma; \Gamma \vdash \mathsf{rel}(w)\, \varphi}$$

The vice-versa happens upon a release operation.

A Type System for Adaptation Scripts

$$\text{TADS}\ \frac{\Gamma = \Gamma', w : \mathsf{lbid} \qquad \Sigma; \Gamma \vdash \mathsf{rel}(r)\, \varphi}{\Sigma; \Gamma \vdash \mathsf{sA}(w)_r\, \varphi}$$

Synchronous adaptations may only be applied on *linear blocked* process ids.

A Type System for Adaptation Scripts

$$\textsc{tAdS}\ \frac{\Gamma = \Gamma', w : \mathsf{lbid} \qquad \Sigma; \Gamma \vdash \mathsf{rel}(r)\ \varphi}{\Sigma; \Gamma \vdash \mathsf{sA}(w)_r\ \varphi}$$

Synchronous adaptations may only be applied on *linear blocked* process ids.

$$\textsc{tAdA}\ \frac{\Gamma = \Gamma', w : \mathsf{lid} \qquad \Sigma; \Gamma \vdash \mathsf{rel}(r)\ \varphi}{\Sigma; \Gamma \vdash \mathsf{aA}(w)_r\ \varphi}$$

Similarly, asynchronous adaptations are only applied on *linear* process ids.

Guaranteeing a degree of Monitor Correctness

Definition

$$\text{error}(s \triangleright \phi) \stackrel{\textit{def}}{=} \textit{“A Synchronous Adaptation is applied to an Unblocked actor.”}$$

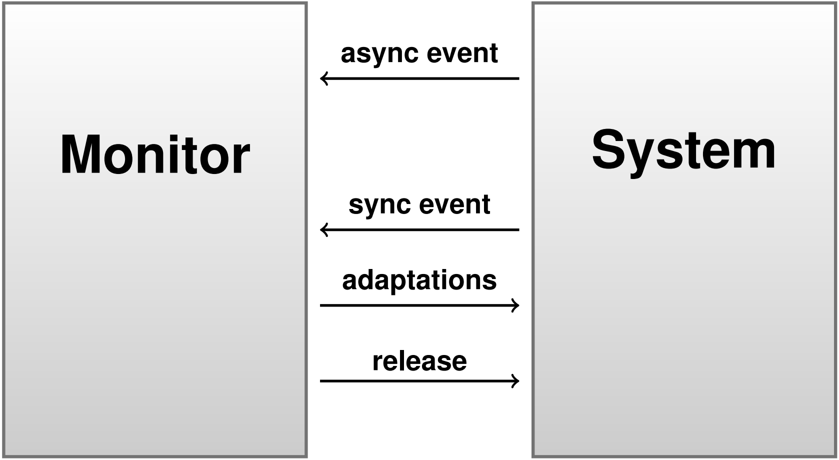
Theorem (Type Soundness)

$$s \triangleright \phi \stackrel{t}{\Rightarrow} s' \triangleright \phi' \quad \textit{implies} \quad \neg \textit{error}(s' \triangleright \phi')$$

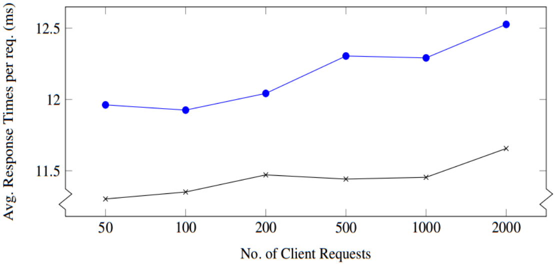
Implementation - AdaptEr

- AdaptEr Repository
  - https://bitbucket.org/casian/adapter
- Based on DetectEr.
- First we introduced *Synchronous Monitoring* through *AOP*.
- Then we added *Adaptations*.

Implementation - Protocol



Implementation - Results



- Performance evaluation was conducted *wrt.* the Yaw Webserver.
- We defined several Adaptation Properties for Yaws to strengthen it.
  - We patched a Directory Traversal Vulnerability.

References

Cassar, I., and Francalanza, A.
On Synchronous and Asynchronous Monitor Instrumentation for Actor Systems.
In *FOCLASA* (2014), vol. 175, pp. 54–68.

Cassar, I., and Francalanza, A.
Runtime Adaptation for Actor Systems.
In *RV* (2015), pp. 38–54.

Cassar, I., and Francalanza, A.
On Implementing a Monitor-Oriented Programming Framework for Actor Systems.
In *iFM* (2016).
(to appear).