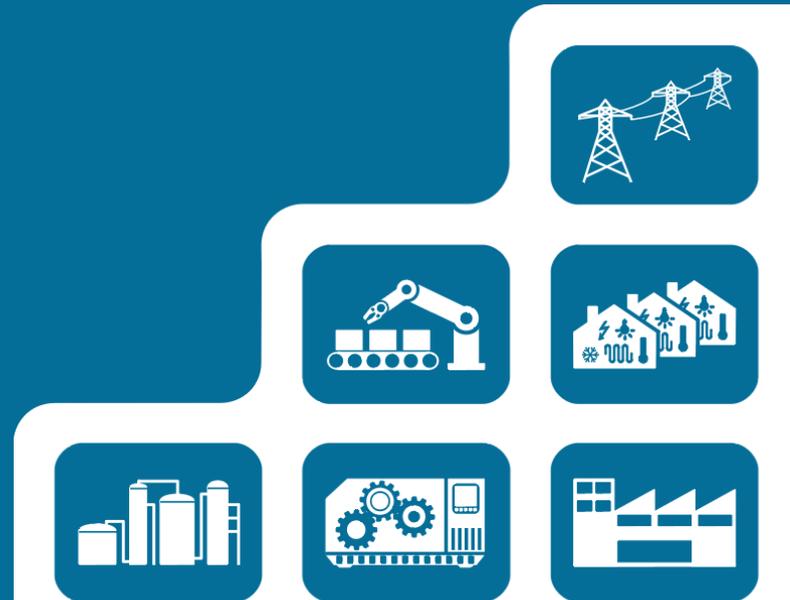


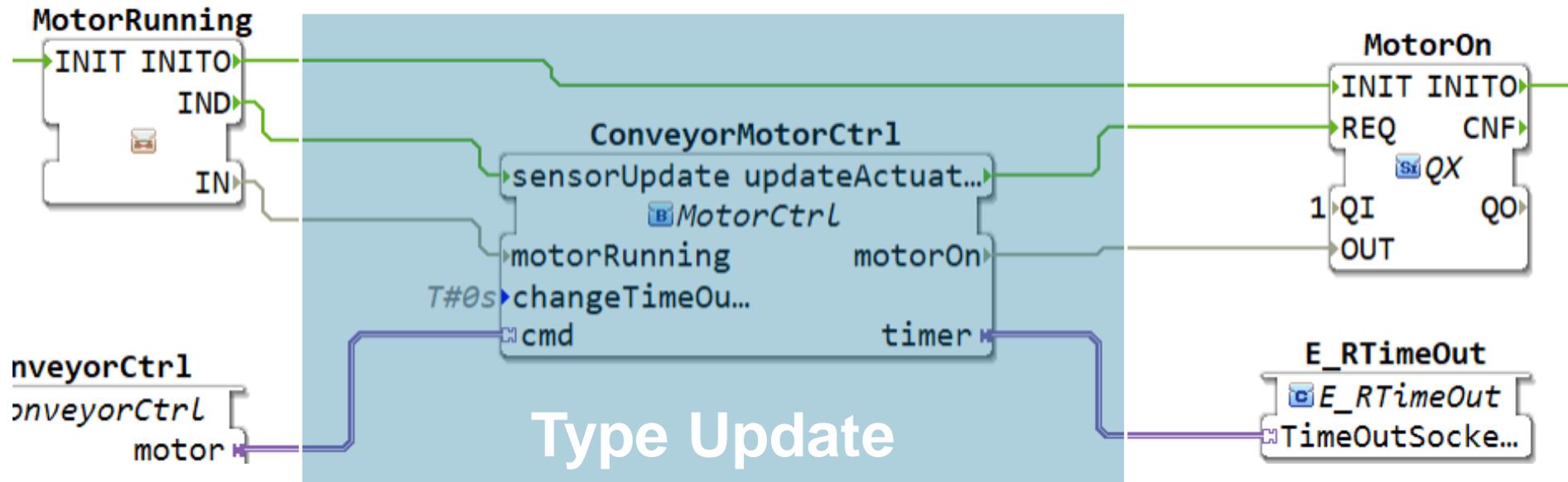
Improving Developer Experience with Refactoring Mechanisms for IEC 61499 Applications



DI Dr. Bianca Wiesmayr, DI Michael Oberlehner
LIT | Cyber-Physical Systems Lab
Johannes Kepler University Linz

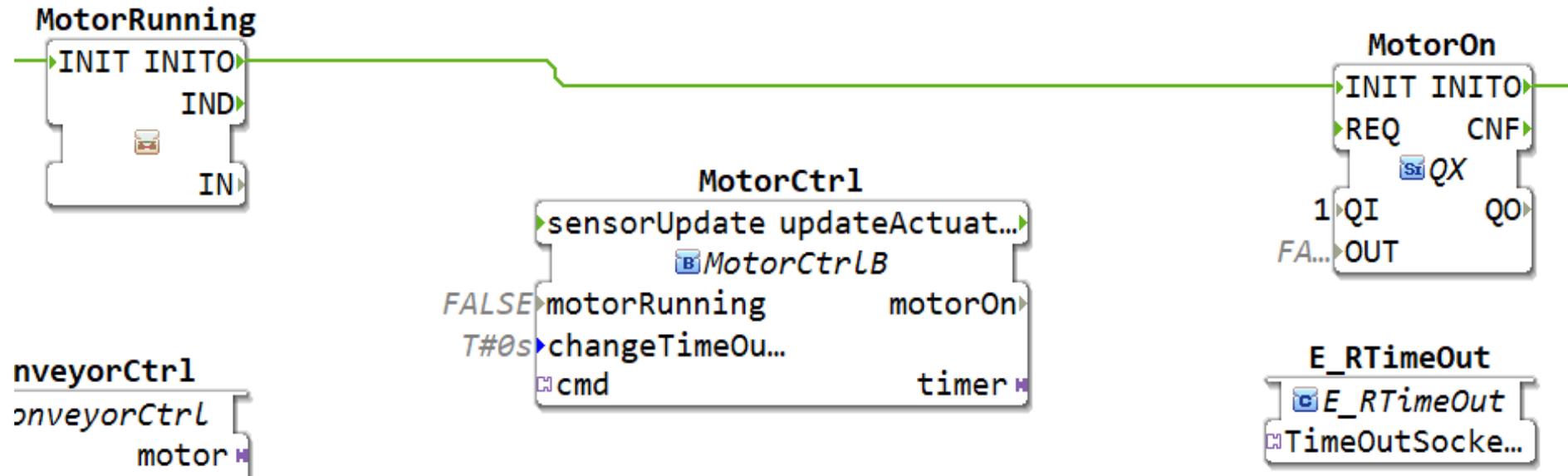


Motivation: Reduce Development Effort



Change of block / interface also requires changes to connections!

Option A: Fully Manual Process

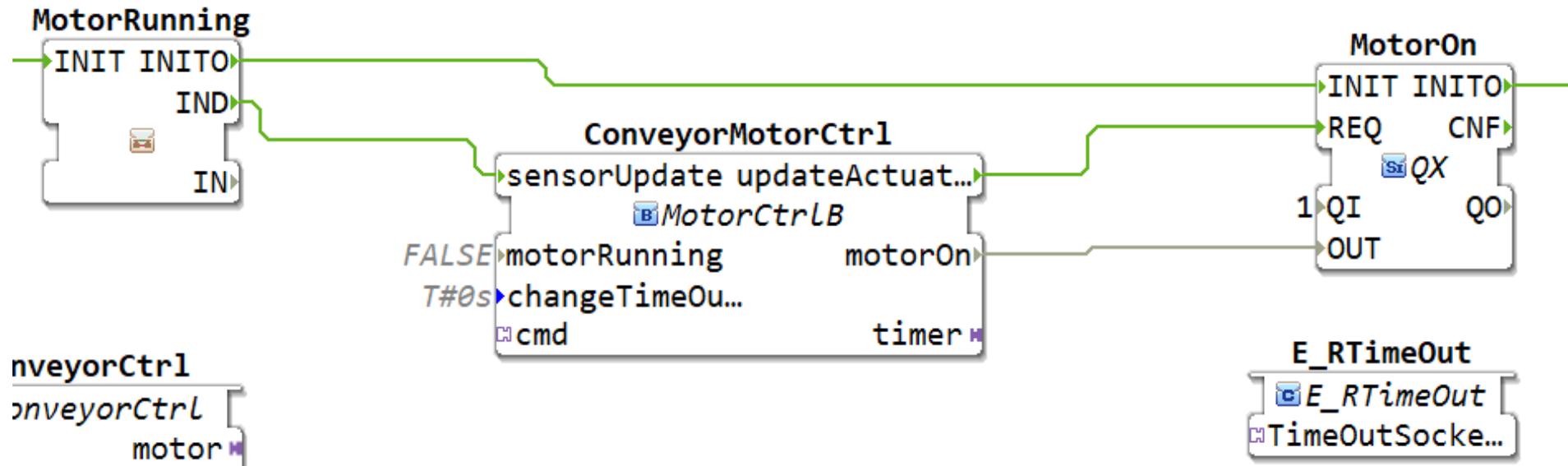


High viscosity

High error proneness

Hard mental operation

Option B: Update and Attempt Reconnect

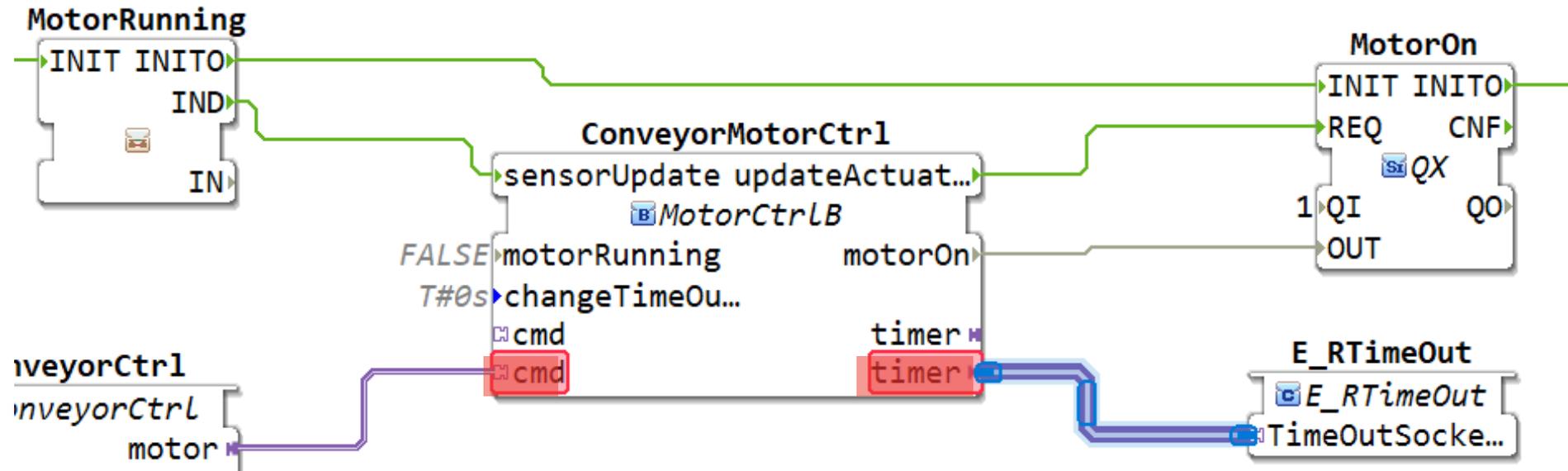


Reduced viscosity

Even higher error proneness

Hard mental operation

Option C: Update and Retain Connections

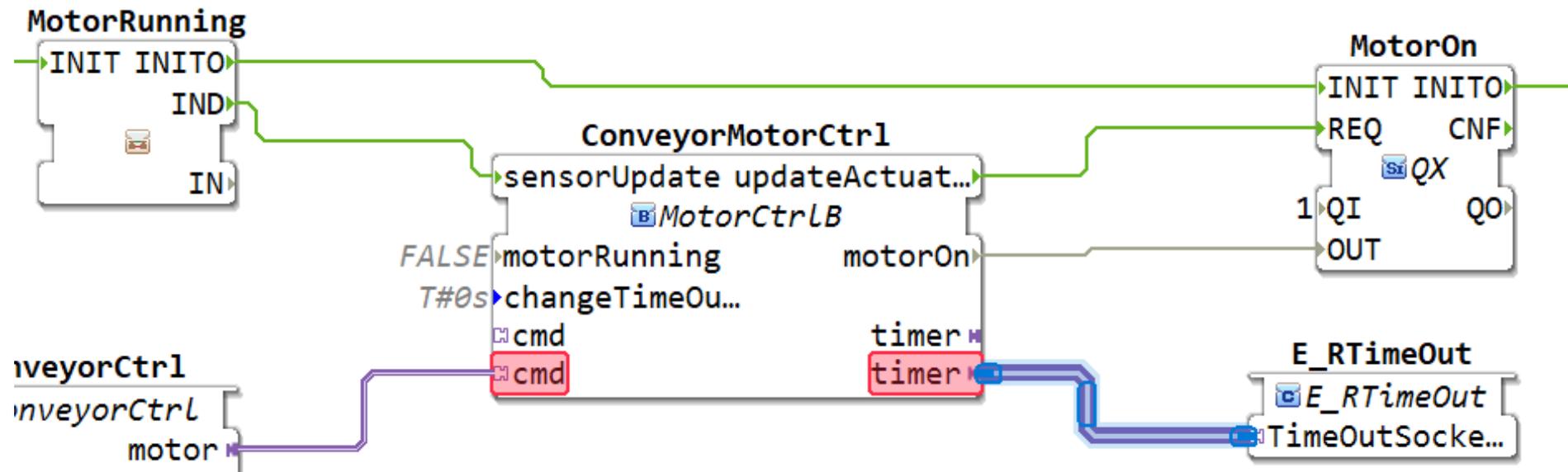


Reduced viscosity

Lower error proneness

Not so hard mental operation, but **low role expressiveness**

Option C: Update and Retain Connections



Reduced viscosity

Lower error proneness

Not so hard mental operation, but **low role expressiveness**

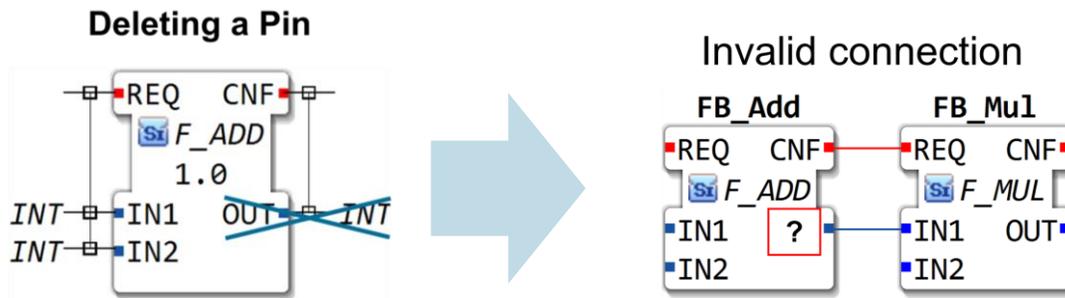
Problem: Inconsistent Application after Type Update

RQ

How to reduce inconsistencies after type update triggers?

Strategies Against Inconsistencies

- Refactoring Operations
- Error Visualization
- Safe Type Update
- Repair Operations



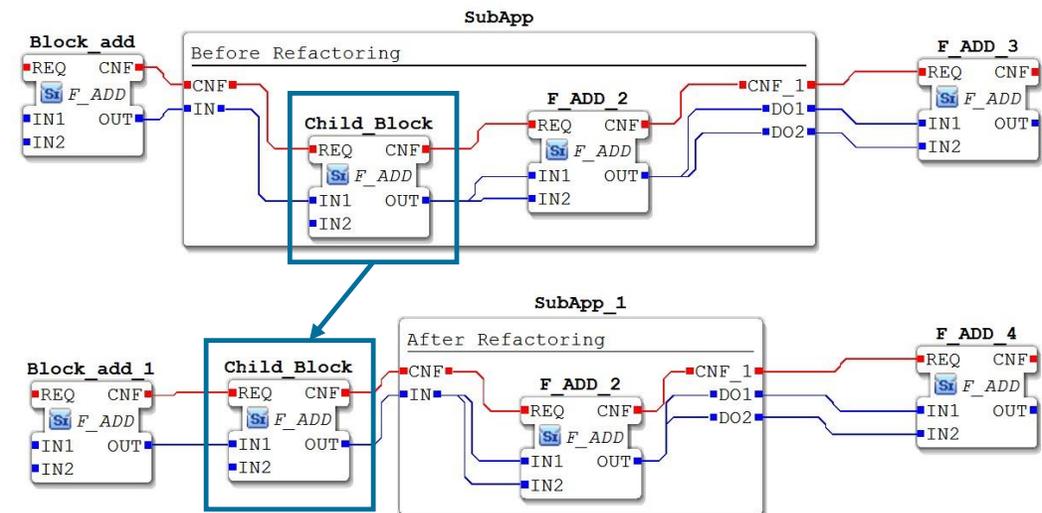
Introduction Refactoring & IEC 61499

Definition Refactoring

„A change made to the **internal structure** of software to make it **easier to understand** and **cheaper to modify** without changing its observable behaviour“ (Fowler M., 1999)

Refactoring in IEC 61499

Restructuring a graphical function block diagram without changing its functionality



Refactoring Operations

RQ

Can we establish a refactoring catalog for IEC 61499 by searching for existing Refactoring Operations in other languages using a generic meta-model as an orientation?

- 29 new IEC 61499 Refactoring Operations
 - 5 from BPMN
 - 16 from UML
 - 8 from Simulink
- 12 new Refactoring Operations for IEC 61131-3
 - 1 new Refactoring Operations for BPMN
 - 1 new Refactoring Operation for Simulink
- Refactoring Operations are implemented in Eclipse 4diac [2] 

[2] <https://eclipse.dev/4diac/>

Example Refactoring – Delete Structured Type

The screenshot shows a software development environment with several windows:

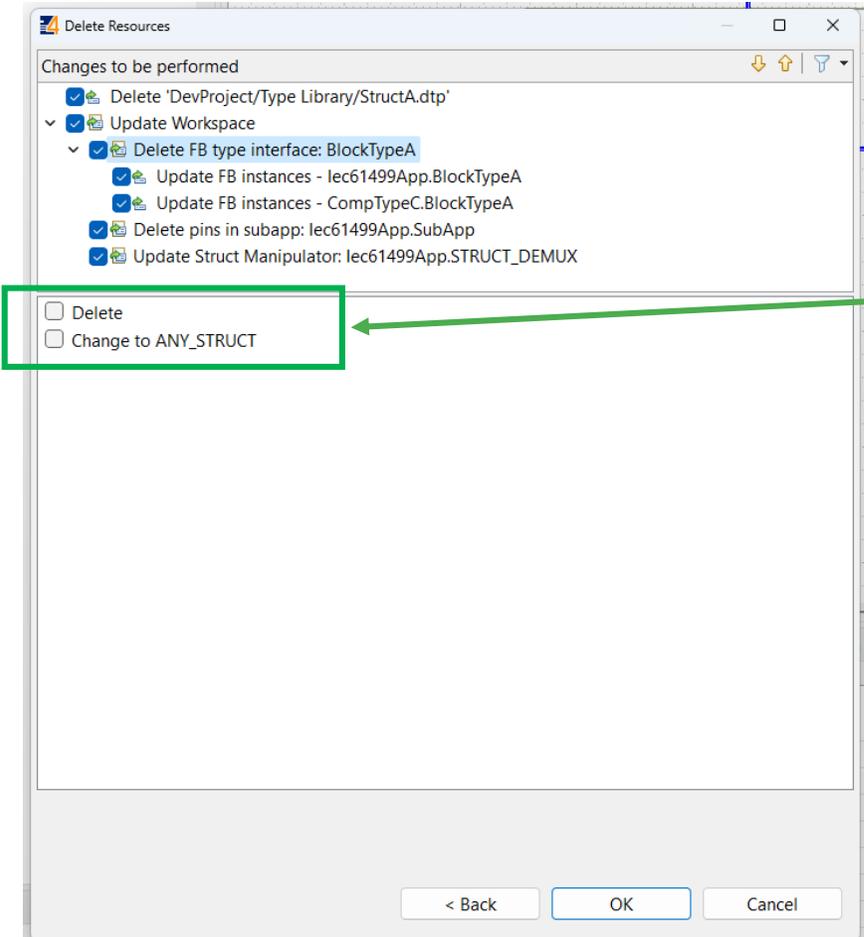
- lec61499System**: A main system diagram showing interconnected blocks like *BlockTypeA*, *BlockTypeB*, *CompTypeC*, and *STRUCT_DEMUX*. A green box highlights *StructA* within the *STRUCT_DEMUX* block.
- BlockTypeA**: A detailed view of a block type with inputs (REQ, CNF, DI1, DI2, DI3) and outputs (Event, Execu). A green box highlights *StructA* in the output section.
- StructA.dtp**: A STRUCT Editor window showing a table of variables:

Name	Type	Comment	Initial Value
1 VAR1	BOOL		FALSE
2 VAR2	BOOL		FALSE
3 VAR3	BOOL		FALSE
- StructB.dtp**: Another STRUCT Editor window showing a table of variables:

Name	Type	Comment	Initial Value
1 VAR1	BOOL		FALSE
2 VAR2	BOOL		FALSE
3 VAR3	BOOL		FALSE
4 VAR4	BOOL		FALSE
5 VAR5	StructA		(VAR1 := FALSE, VAR2 := FALSE, VAR3 := FALSE)

Green arrows indicate the flow of the refactoring: from the *StructA* in the system diagram to its definition in *StructA.dtp*, and from its use in *StructB.dtp* to its removal from the table.

Delete StructA - Refactoring Preview



- The preview shows all affected instances of the regarding type
- It offers the user the possibility to apply repair operations during refactoring

Delete StructA - After Refactoring

The screenshot displays a software development environment with four main panes:

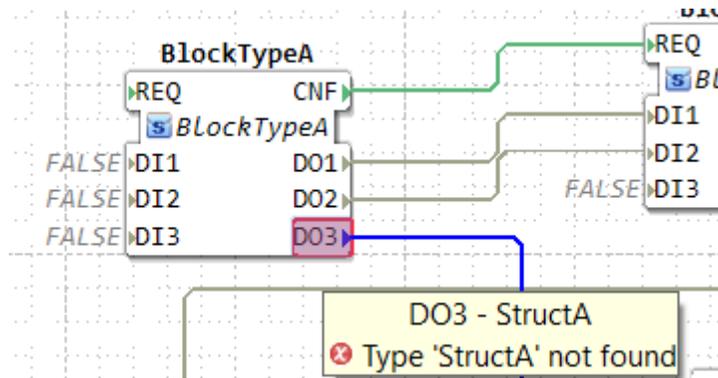
- Top-Left:** A block diagram showing three blocks: BlockTypeA, BlockTypeB, and CompTypeC. BlockTypeA has ports REQ, CNF, DI1, DI2, DI3, D01, D02, D03. BlockTypeB has ports REQ, CNF, DI1, DI2, DI3, D01, D02, D03. CompTypeC has ports INIT, INITO, REQ, CNF, QI, QO. Connections are shown between the blocks.
- Top-Right:** A detailed view of BlockTypeA. It shows a table of data elements:

Normal Execution Request	Event	REQ	CNF	Event	Execution Confirmation
		BlockTypeA	1.0		
-	BOOL	DI1	D01	BOOL	-
-	BOOL	DI2	D02	BOOL	-
-	BOOL	DI3	D03	StructA	-
- Bottom-Left:** A different view of the block diagram, showing BlockTypeB and BlockTypeA. BlockTypeB has ports REQ, CNF, DI1, DI2, DI3, D01, D02, D03. BlockTypeA has ports REQ, CNF, DI1, DI2, DI3, D01, D02, D03. Connections are shown between the blocks.
- Bottom-Right:** A 'STRUCT Editor' table with a red highlight on 'StructA':

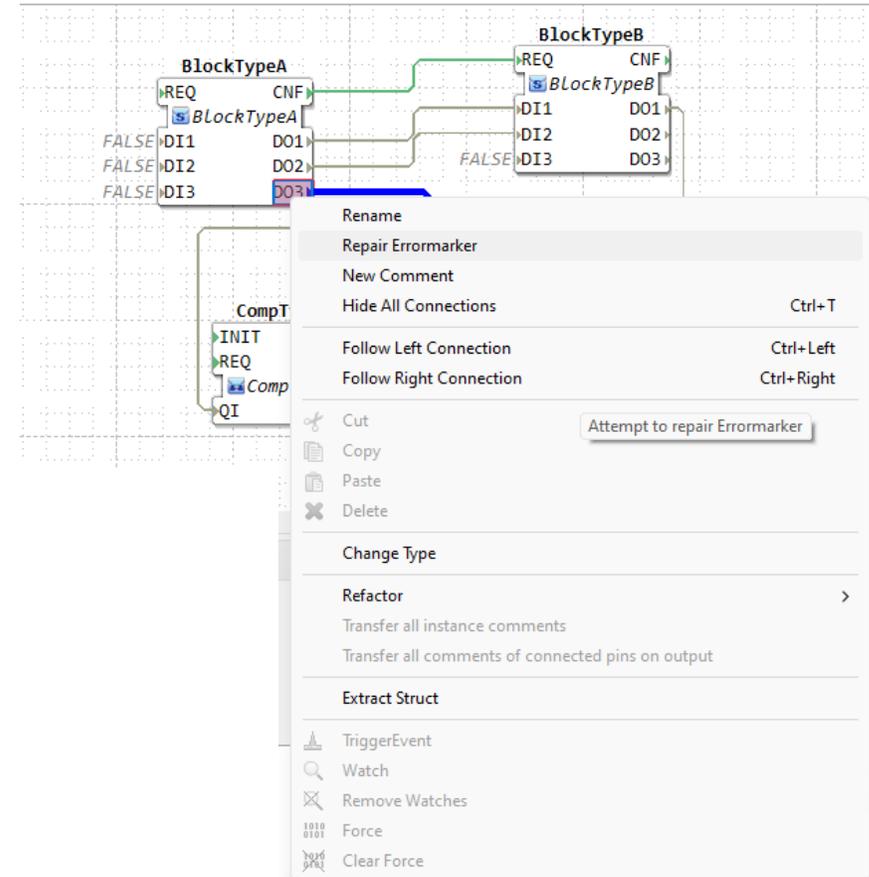
Name	Type	Comment	Initial Value
1 VAR1	BOOL		FALSE
2 VAR2	BOOL		FALSE
3 VAR3	BOOL		FALSE
4 VAR4	BOOL		FALSE
5 VAR5	StructA		

- The editor gives a visual hint to the user which elements are inconsistent after the refactoring
- It is also possible to show them in a collected view

Example – Repair Again by using Leftover Information



4diac IDE shows a dedicated error message and offers repair mechanisms



Next Steps

- Evaluation of Usability of Refactoring Operations
 - Is it easy and intuitive for the user to operate?
- Compare Maturity with other Tools
 - Can we compete with the Refactoring features of other tools?
- Implement Recommender System
 - Additional support for the user (e.g. refactoring hints for code smell)

Thank you!

Michael.oberlehner@jku.at
LIT | Cyber-Physical Systems Lab
Johannes Kepler University Linz

