

Evaluation of Static Properties for Component-Based Architectures

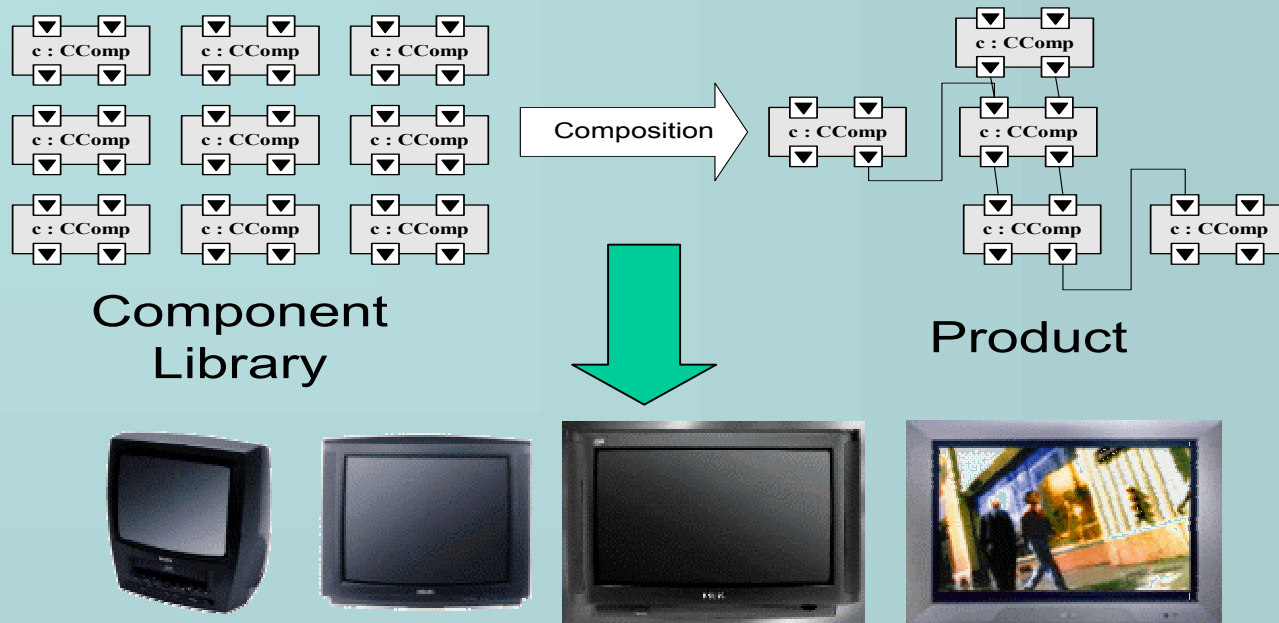
by Evgeni Eskenazi & Alexandre Fioukov



Contents

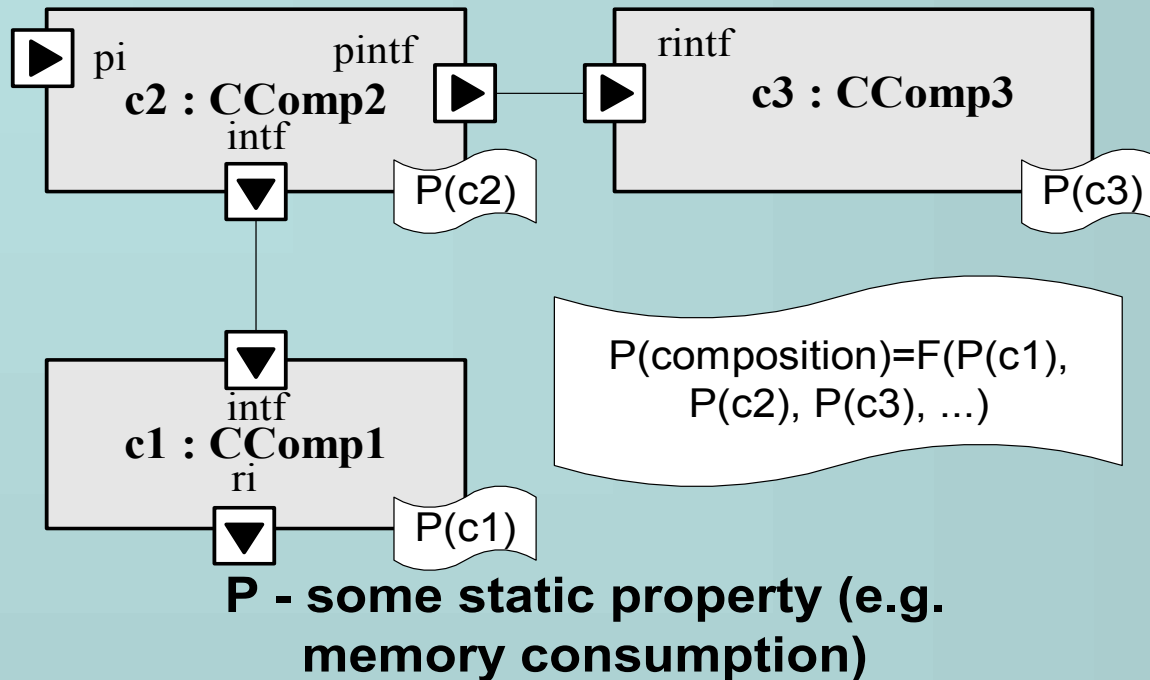
1. Problem statement
2. Estimation framework
3. Specification
4. Evaluation approaches
5. Conclusions

Problem statement



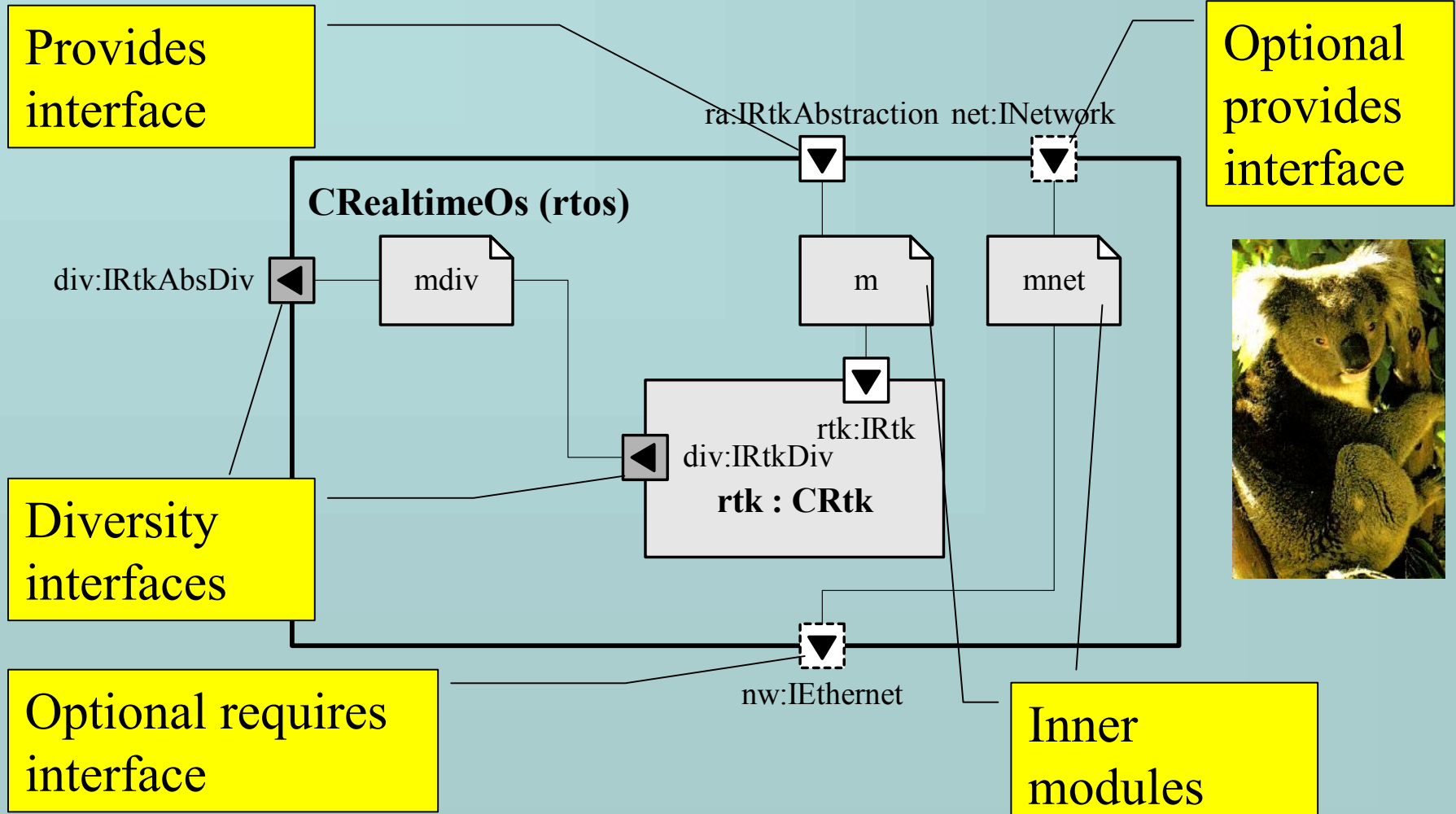
- Product families: *diversity* and commonality
- Resource constraints
- Compositionality

Goals



- Assessment of static properties of a composition based on the properties of the components
- Trade-off between estimation effort and accuracy
- Budgeting should be possible

Example: static memory demand for Koala components

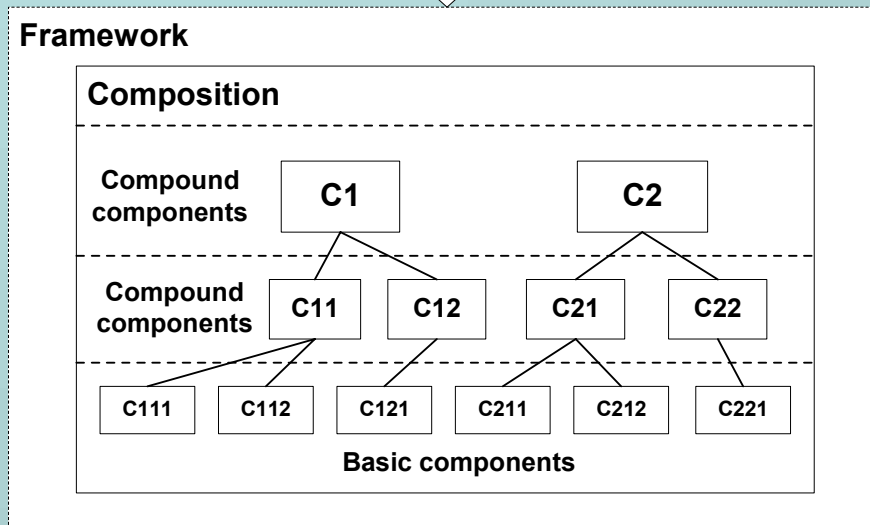


Estimation framework

Defines how static properties are combined

Compositional rules

Calculation is automatic!

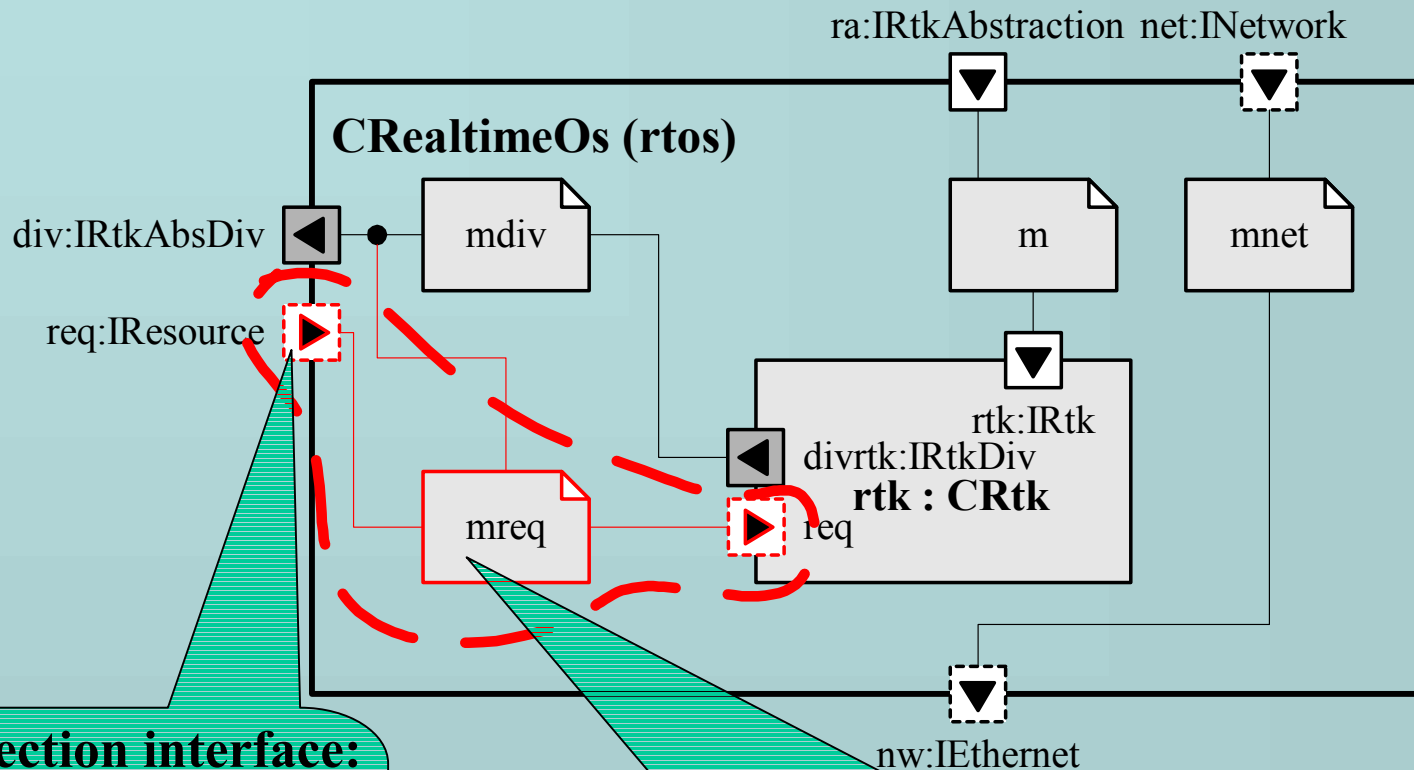


Estimation of composed property

Static property specifications

Specifies contribution to the property for each component

Koala specification example

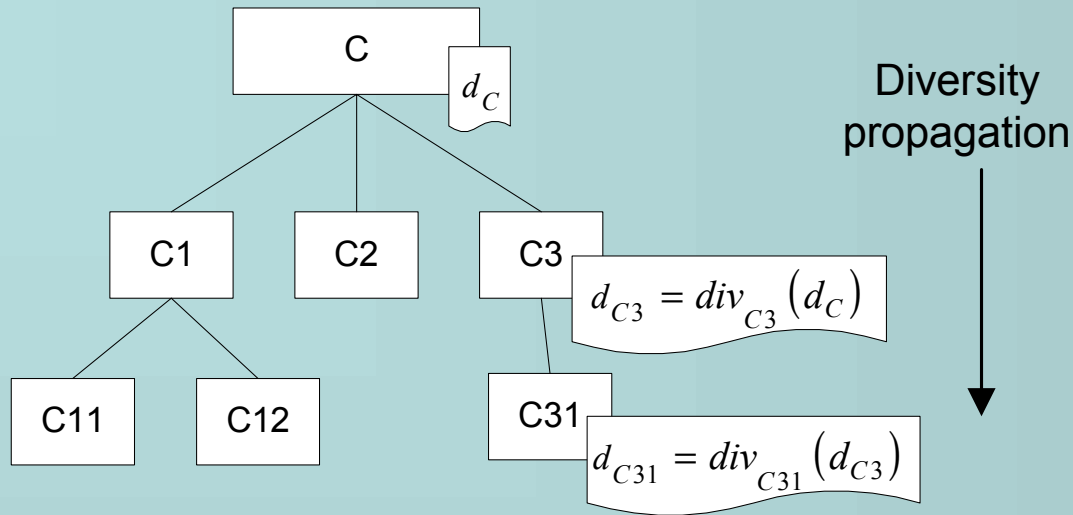


Reflection interface:

```
interface IResource {
    ...
    long XRAM_size;
}
```

IResource Implementation

Diversity propagation



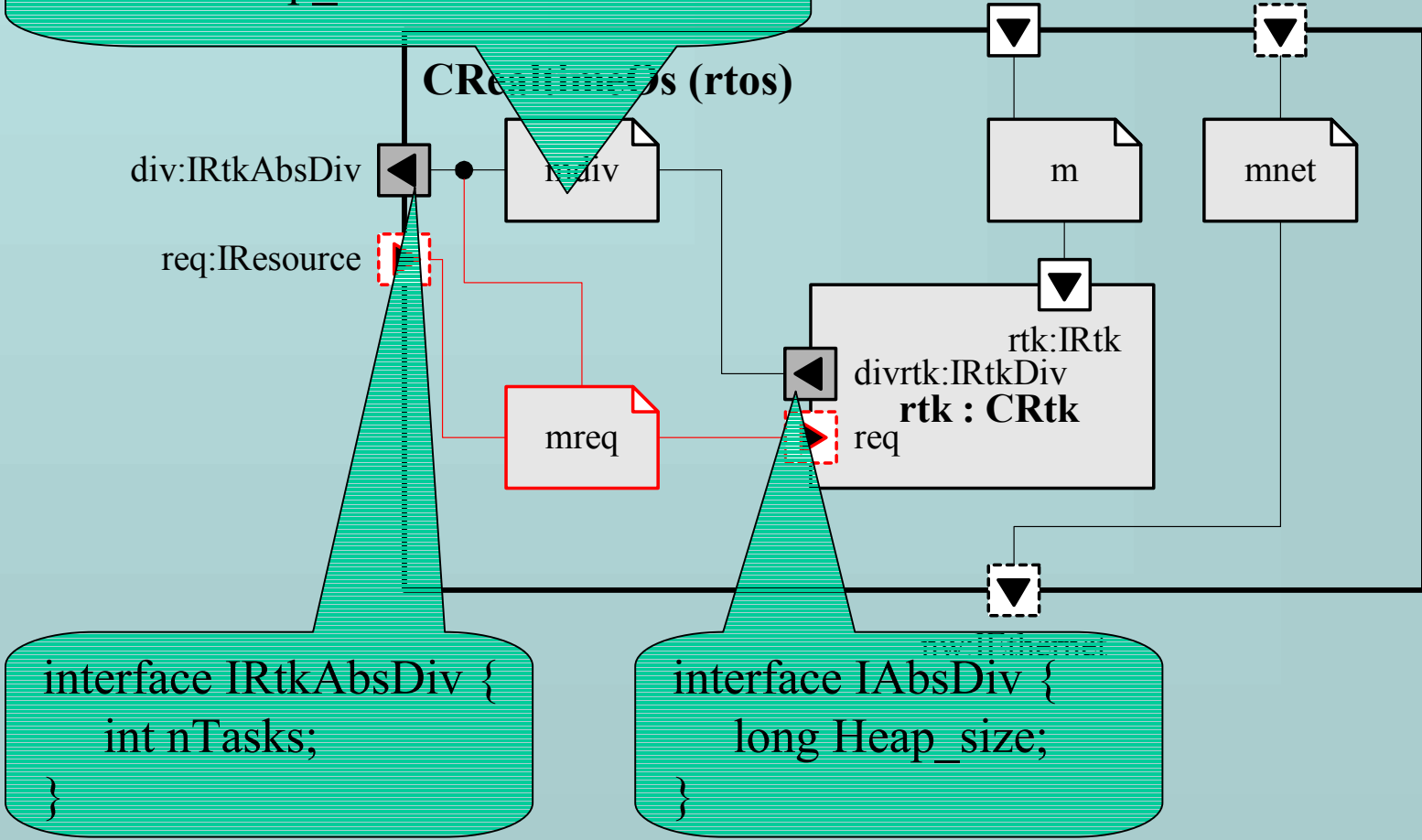
- Diversity d of a component is expressed in terms of diversity of its high-level component:

$$d_{C31} = \text{div}_{C31} \circ \text{div}_{C3}(d_C)$$

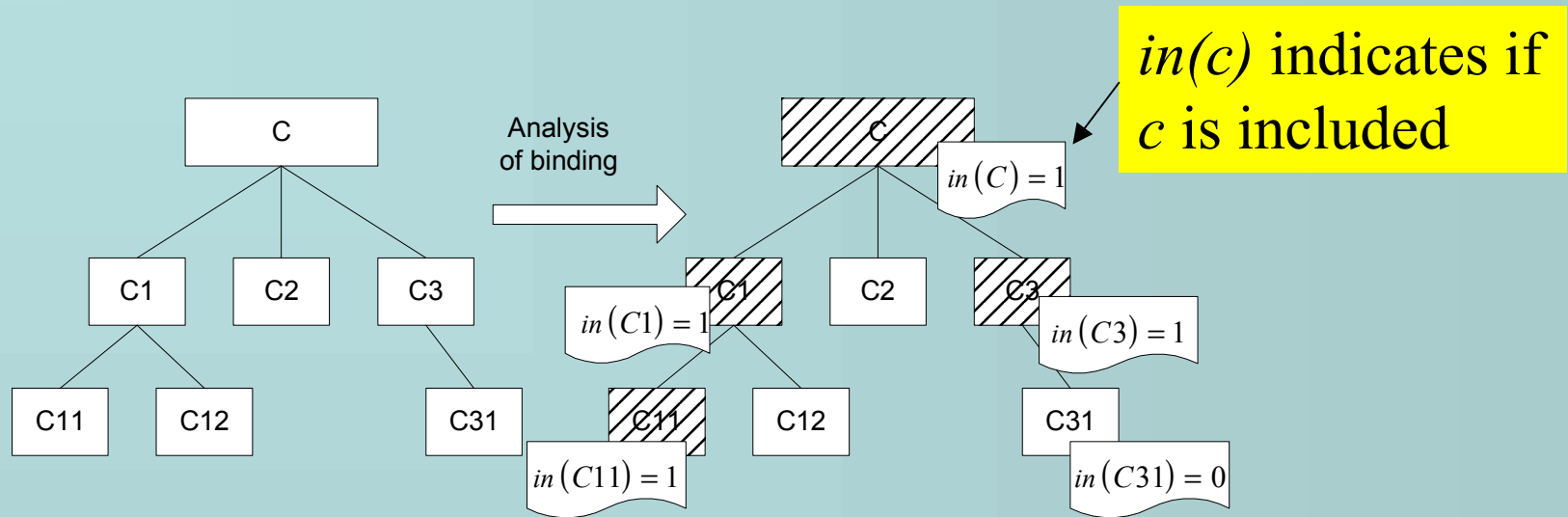
Koala specification example

$$d_{rtos} = div_{rtk} (d_{rtos})$$

Rtk.div.Heap_size=56*divrtk.nTasks



Influence of binding



- Putting it all together:

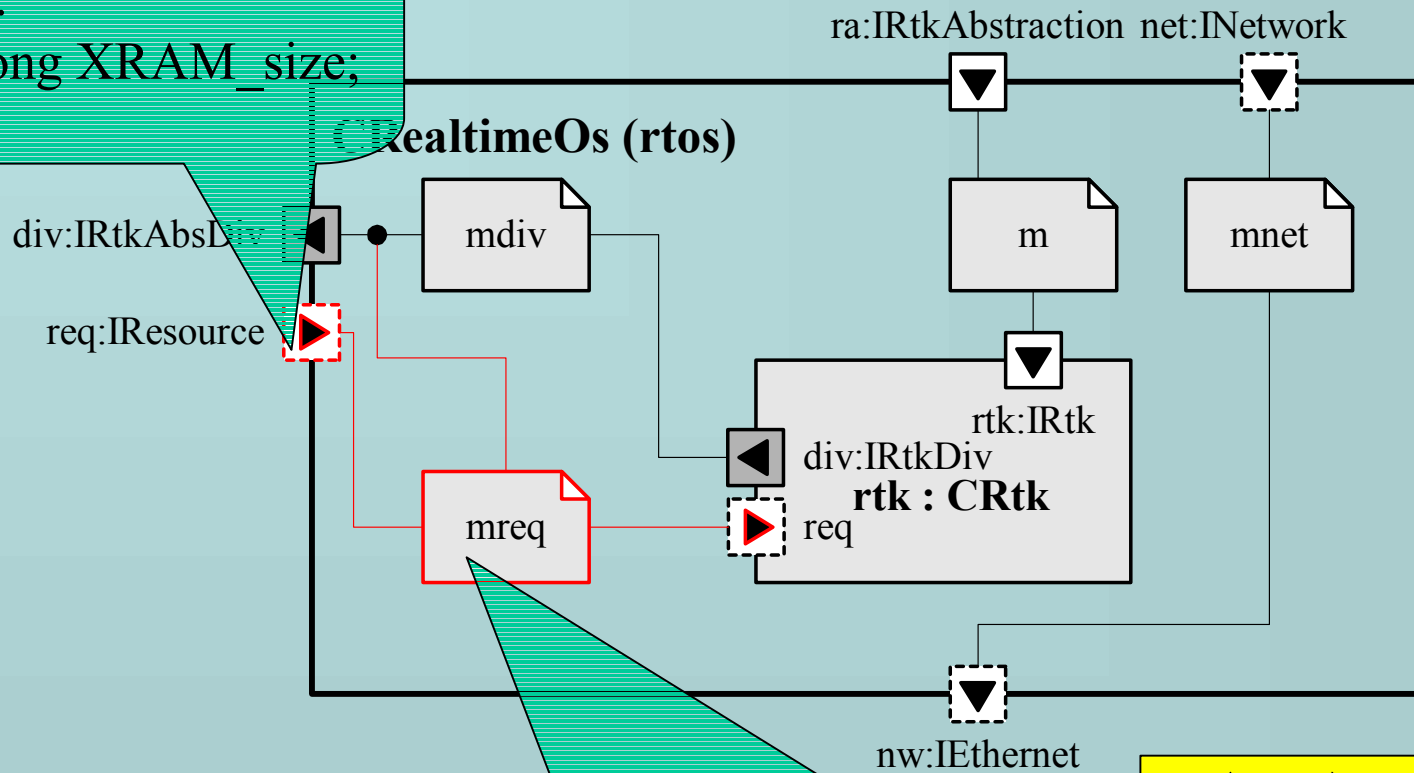
$$P(c, D) = I(c, D) + \sum_{i \in sub(c)} P(i, div_{c,i}(D)) \cdot in(i).$$

$P(c, D)$ - value of the property of component c w.r.t diversity D ,
 $I(c, D)$ - internal contribution of component c to property P

- For non-implemented components $P(c, D)$ is budgeted!

Koala specification example

```
interface IResource {
  ...
  long XRAM_size;
}
```



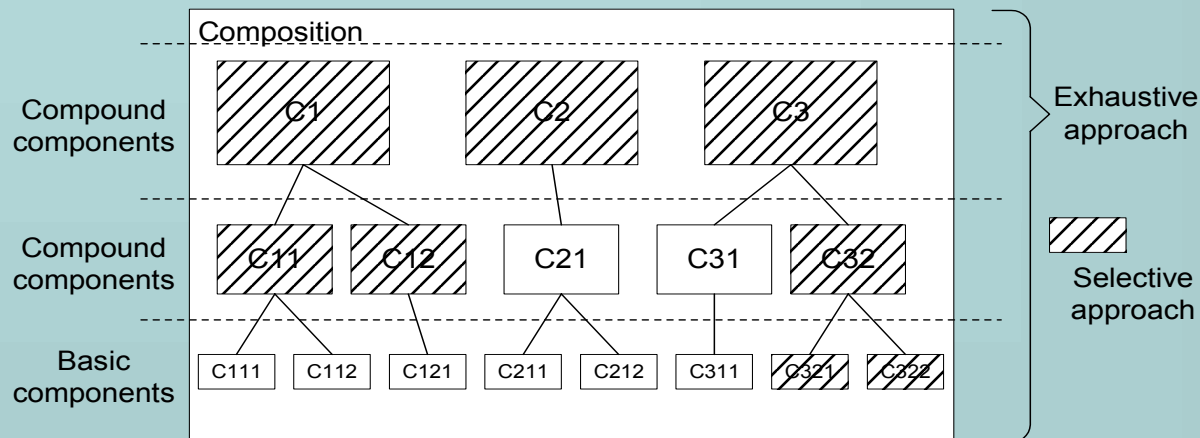
```
req.XRAM_size = 34 + (8+(div.MaxValue+1)*2) +
rtk.req.iPresent() ? rtk.req.XRAM_size:0;
```

$$I(c, D)$$

$$P(i, div_{c,i}(D)) \cdot in(i)$$

Evaluation approaches

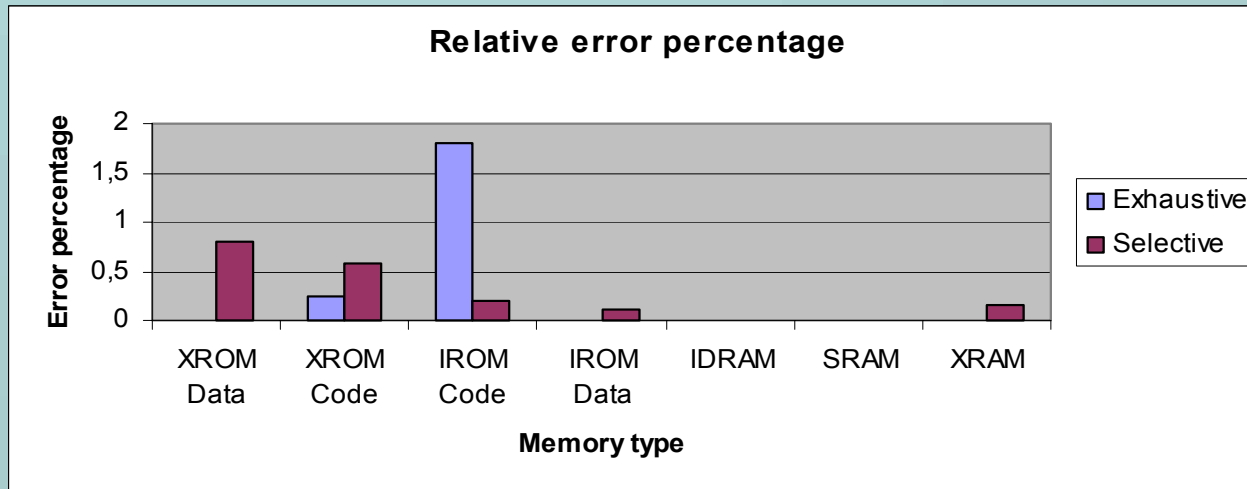
- Two approaches to component selection:
 - *exhaustive* (all components are annotated)
 - *selective* (only selected components are annotated)



- Choice criteria: annotation effort & reusability of components

Approach checking

- Checked with real TV software for both *exhaustive* and *selective* approaches
- Different types of static memory—*code*, *data*, *stack*—were considered
- Case-studies on TV software: accuracy is higher than **95%** for both cases



Conclusions

- Calculation is done **automatically**
- **Budgeting** is supported
- Currently, the technique is **checked with two Koala compositions**
- Specification effort for one component **30-60 min**
- Generic description of resource interface (e.g. in XML) will extend the applicability of the technique to other component models

Questions and discussion?

