

# *Enabling Self-adaptivity in Component-based Streaming Applications*

Onur Derin, Alberto Ferrante

Advanced Learning and Research Institute  
Faculty of Informatics  
Università della Svizzera italiana  
Lugano, 6900, Switzerland  
`name.surname@alari.ch`

APRES'09  
October 11, 2009



## *Outline*

*A model of self-adaptive systems*

*SACRE Framework*

*Adaptation support in SACRE*

*Conclusion*



### Definition

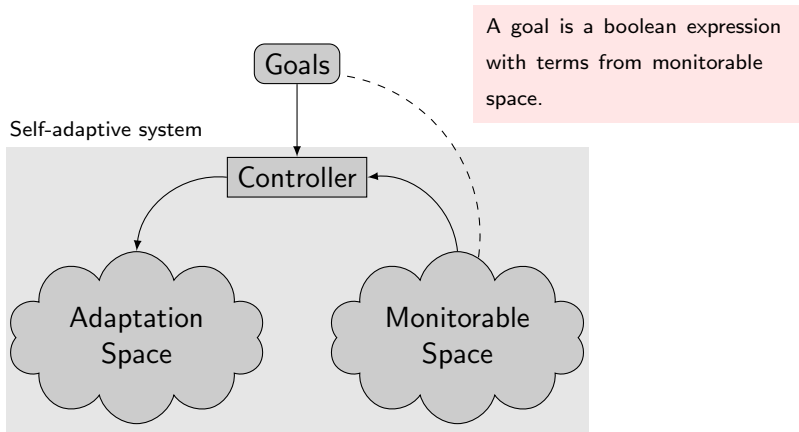
**Self-adaptivity** is the capability of a system to adapt itself dynamically to achieve its goals.

### Why self-adaptation?

- ▶ Changing internal and/or external conditions  
e.g. moving with a portable device between wired and wireless networks, switching to battery power
- ▶ Increasing complexity of systems and difficulties in integration  
(self-organization - specification tradeoff principle)
- ▶ Some information is available only at run-time  
(application specific vs. general purpose, design space exploration vs. run-time adaptation)



## Monitor-Controller-Adaptor Paradigm



e.g. different implementations-  
parameters, available cores, available  
HW functional units, clock frequency

e.g. frame size, resource utilization,  
cache miss rate, power consumption

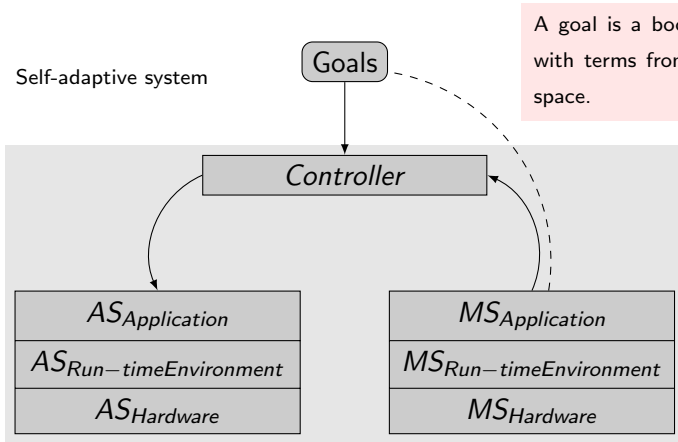


## Quality measures for a self-adaptive system

- ▶ Adaptation coverage  
How big is the adaptation space?
- ▶ Separation of concerns  
Is the application programmer concerned with self-adaptivity aspects?
- ▶ Adaptation management  
How good is the controller?
- ▶ Adaptation requirements (goal) specification  
How big is the monitorable space? How are goals specified?



## Layered Model



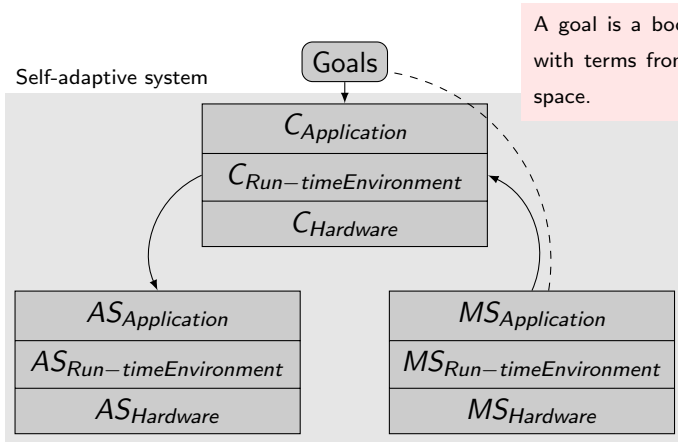
A goal is a boolean expression with terms from monitorable space.

e.g. different implementations-parameters, available cores, available HW functional units, clock frequency

e.g. frame size, resource utilization, cache miss rate, power consumption



## Layered Model



A goal is a boolean expression with terms from monitorable space.

e.g. different implementations-parameters, available cores, available HW functional units, clock frequency

e.g. frame size, resource utilization, cache miss rate, power consumption



## *Component-based approach*

Main enabler for application-level and RTE-level adaptation is a **component-based approach**.

- ▶ allows the framework to be aware of the run-time characteristics of components
- ▶ thus separation of concerns

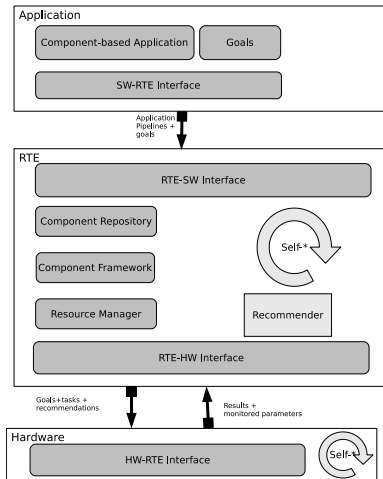


## Model of a self-adaptive heterogeneous system

### Self-adaptive Component Framework

the component run-time environment is extended with an MCA loop

- ▶ does application level adaptations
- ▶ does RTE-level adaptations
  - ▶ Resource manager adapts concurrency, redundancy and mapping.
- ▶ Goals as lower/upper bounds or Min/Max.





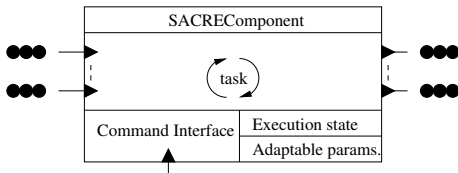
### Component model

- ▶ based on the KPN model of computation
- ▶ suited for streaming applications
- ▶ A SACRE component
  - ▶ extends the *SACREComponent* abstract class
  - ▶ implements the `task()` method
  - ▶ has a thread of its own
  - ▶ specifies its input and output ports
  - ▶ exchanges typed messages through its named input and output ports
  - ▶ is adaptable
    - ▶ has a command interface for adaptations
    - ▶ declares its adaptable parameters
    - ▶ ports can be connected to a different connector at run-time
    - ▶ input and output ports are *hookable*
    - ▶ messages can be tagged



## Connectors

- ▶ unbounded buffers that hold typed messages
- ▶ blocking read, non-blocking write
- ▶ connects one output port to one input port



*Figure:* Visual representation of a SACRE component



## Pipelines

- ▶ created by means of a simple language
  - ▶ ;-separated list of statements
  - ▶ & for parallel, ! for serial composition
  - ▶ & has precedence over !
- ▶ no cycles  $\implies$  deadlock-freedom, otherwise  $\implies$  constraints

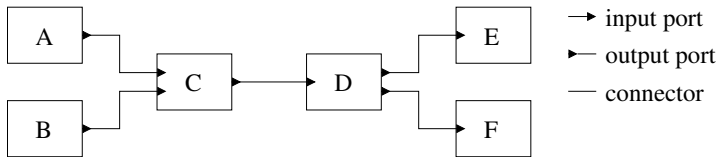


Figure: A & B ! C ! D ! E & F

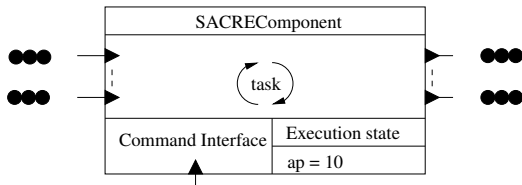


## Possible adaptations at Application Level

- ▶ Parameter adaptation of a component  $C_i(p_i) \Rightarrow C_i(p_j)$
- ▶ Structural adaptation
  - ▶ Replacement of a component  $C_i \Rightarrow C_j$
  - ▶ Parallelization of a component  $C_i \Rightarrow \{C_{ij}\}$
  - ▶ Transformation with adaptation patterns for high level goals such as dependability and security  
 $G_i\{C\} \Rightarrow G_j\{C'\}$

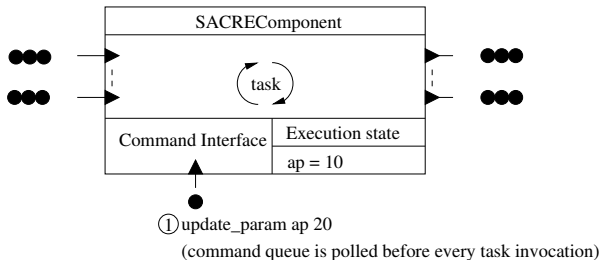


## Parameter adaptation of a component



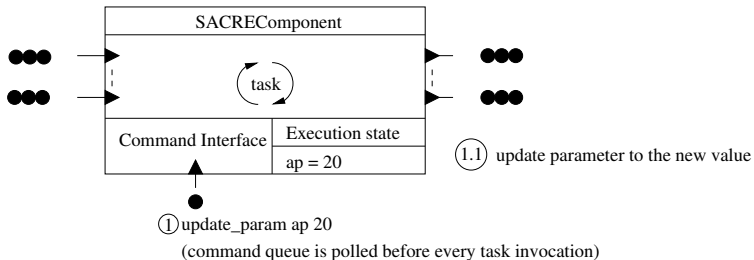


## Parameter adaptation of a component



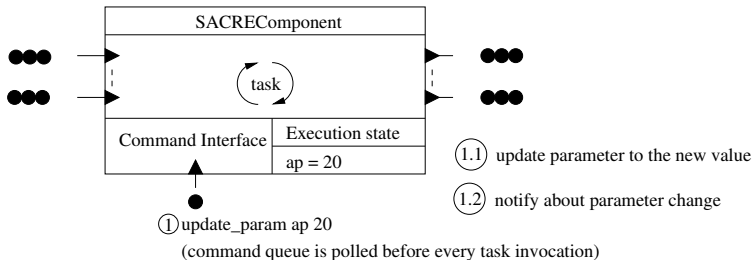


## Parameter adaptation of a component



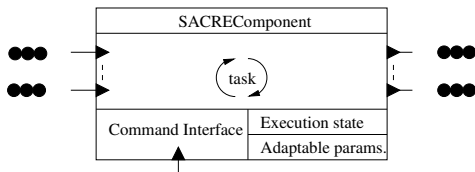


## Parameter adaptation of a component



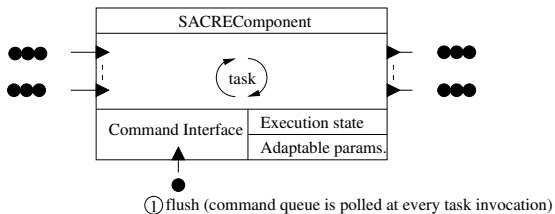


## Structural adaptation: Component replacement





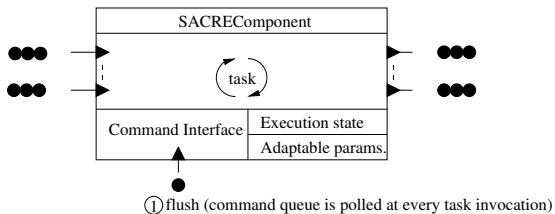
## Structural adaptation: Component replacement





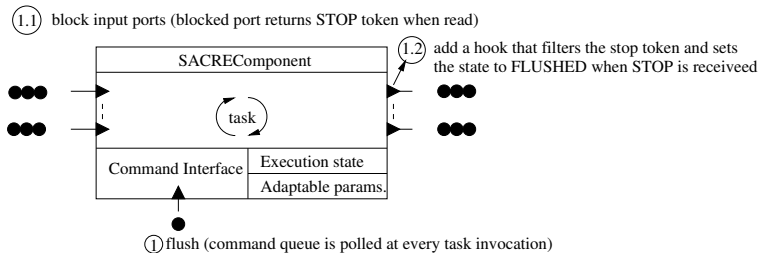
## Structural adaptation: Component replacement

①.1 block input ports (blocked port returns STOP token when read)





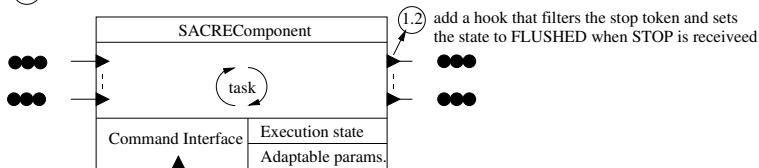
## Structural adaptation: Component replacement



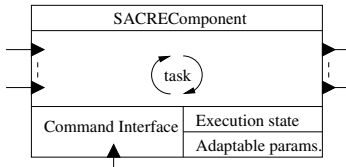


## Structural adaptation: Component replacement

①.1 block input ports (blocked port returns STOP token when read)



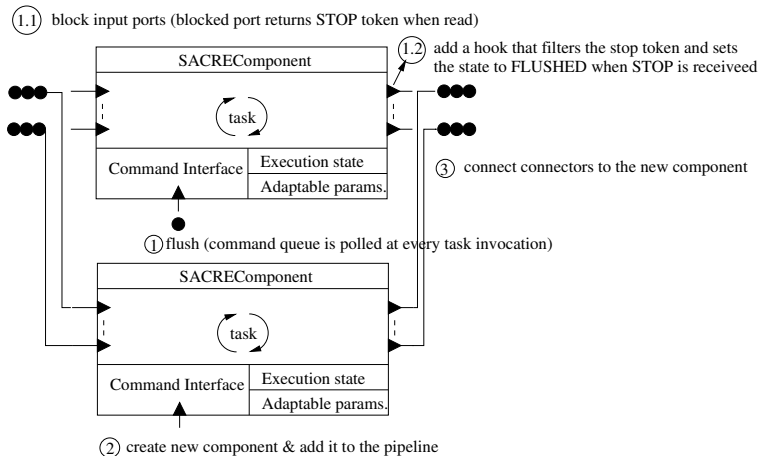
① flush (command queue is polled at every task invocation)



② create new component & add it to the pipeline

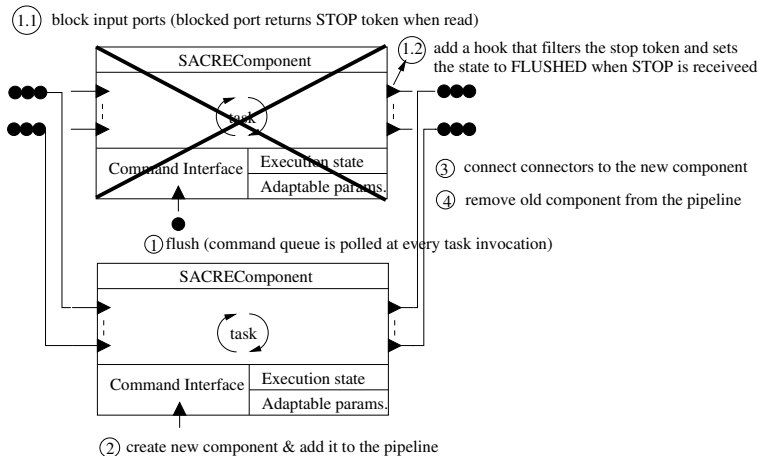


## Structural adaptation: Component replacement



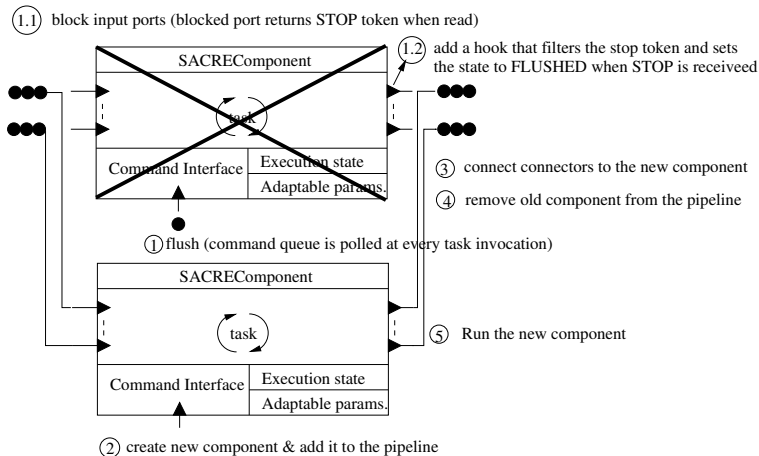


## Structural adaptation: Component replacement



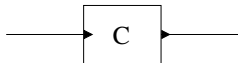


## Structural adaptation: Component replacement



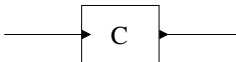


## *Structural adaptation patterns: Dependability*





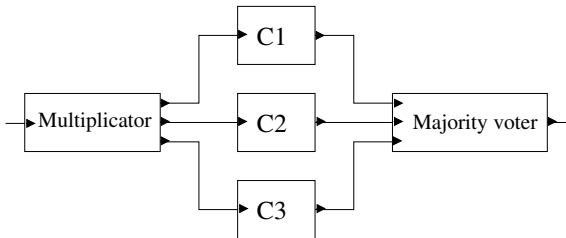
## Structural adaptation patterns: Dependability



1. flush component and disconnect connectors



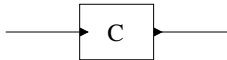
## Structural adaptation patterns: Dependability



1. flush component and disconnect connectors
2. create new instances of the component
3. create as many multipliers as input ports  
(copies input token to output ports and tags them with the same integer ID)
4. create as many majority voter components as output ports  
(matches input tokens with same ID and writes to its output the most recurring message)
5. connect multipliers with redundant instances and the majority voter
6. run all new components

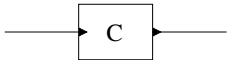


## *Structural adaptation patterns: Parallelization*





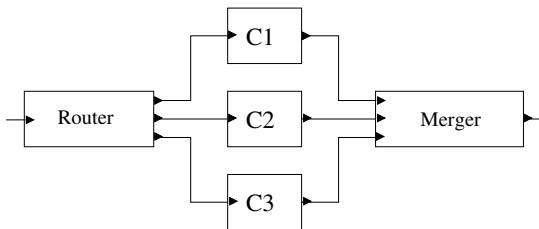
## Structural adaptation patterns: Parallelization



1. flush component and disconnect connectors



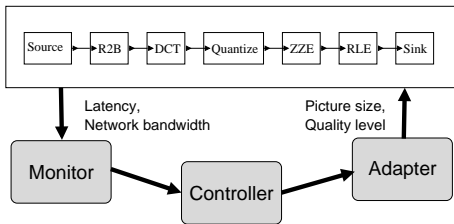
## Structural adaptation patterns: Parallelization



1. flush component and disconnect connectors
2. create new instances of the component
3. create as many routers as input ports  
(routes input token to output ports in round-robin or load balancing fashion by tagging with incremental IDs)
4. create as many merger components as output ports  
(merges input tokens and writes to its output by ordering them according to their IDs (preserves monotonicity))
5. connect routers with parallel instances and mergers
6. run all new components



## Test case: A self-adaptive MJPEG streaming server



### Goal

- ▶  $FPS > FPS^T \Rightarrow Latency < \frac{1}{FPS^T}$  and  $FrameSize < \frac{Bandwidth}{FPS^T}$
- ▶ maximize PictureSize and Quality

### Adaptation & Monitoring Space

$$AS_{Application} = \{PictureSize, EncodingQuality\}$$

$$MS_{Application} = \{Latency, Bandwidth\}$$



## *Conclusion*

- ▶ proposed an approach to implement self-adaptive streaming applications on component frameworks
- ▶ presented the SACRE framework as an implementation of our ideas
- ▶ described a self-adaptive MJPEG streaming server as a test case

## *Future work*

- ▶ currently we support a subset of KPN (processes with no states)
- ▶ automated synthesis of application level controllers and monitors from goals
- ▶ extension of SACRE to simulate self-adaptivity on heterogeneous MPSoC platforms



- O. Derin, A. Ferrante. *Simulation of a self-adaptive run-time environment with hardware and software components. Proceedings of ESEC/FSE Workshop on software integration and evolution @ run-time*, 55(3):170-179, 2009.
- O. Derin, A. Ferrante, A. V. Taddeo. *Coordinated management of hardware and software self-adaptivity. Journal of System Architecture*, 55(3):170-179, 2009.
- O. Derin, A. Ferrante, A. V. Taddeo. *Coordinated management of hardware and software self-adaptivity: What do we need from reconfigurable computing?. presented in Reconfigurable Computing Italian Meeting (RCIM'08)*, Milano, December 2008.
- O. Derin, A. Ferrante. *Enabling self-adaptivity at application level. presented in 2<sup>nd</sup> AETHER-Morpheus Workshop (AMWAS'08)*, Lugano, October 2008.
- A. Ferrante, A. V. Taddeo, O. Derin. *Security in self-adaptive systems. presented in 1<sup>st</sup> AETHER-Morpheus Workshop (AMWAS'07)*, Paris, October 2007.



*Thank you!*