Performance aware open-world software in a 3-layer architecture

Diego Perez-Palacin, *Universidad de Zaragoza, Spain*

José Merseguer, *Universidad de Zaragoza, Spain*

Simona Bernardi, *Università di Torino, Italy*

Context

- Open world software
  - Publish-subscribe; SOA; grid computing; etc.
  - Key idea: software made of services.
    - Third parties providers; interplay without authorities.
  - Performance problems
    - Are valid the current assumptions in SPE?
    - Can we trust in these third-parties?
  - Challenge
    - Self- adaptation or self-management

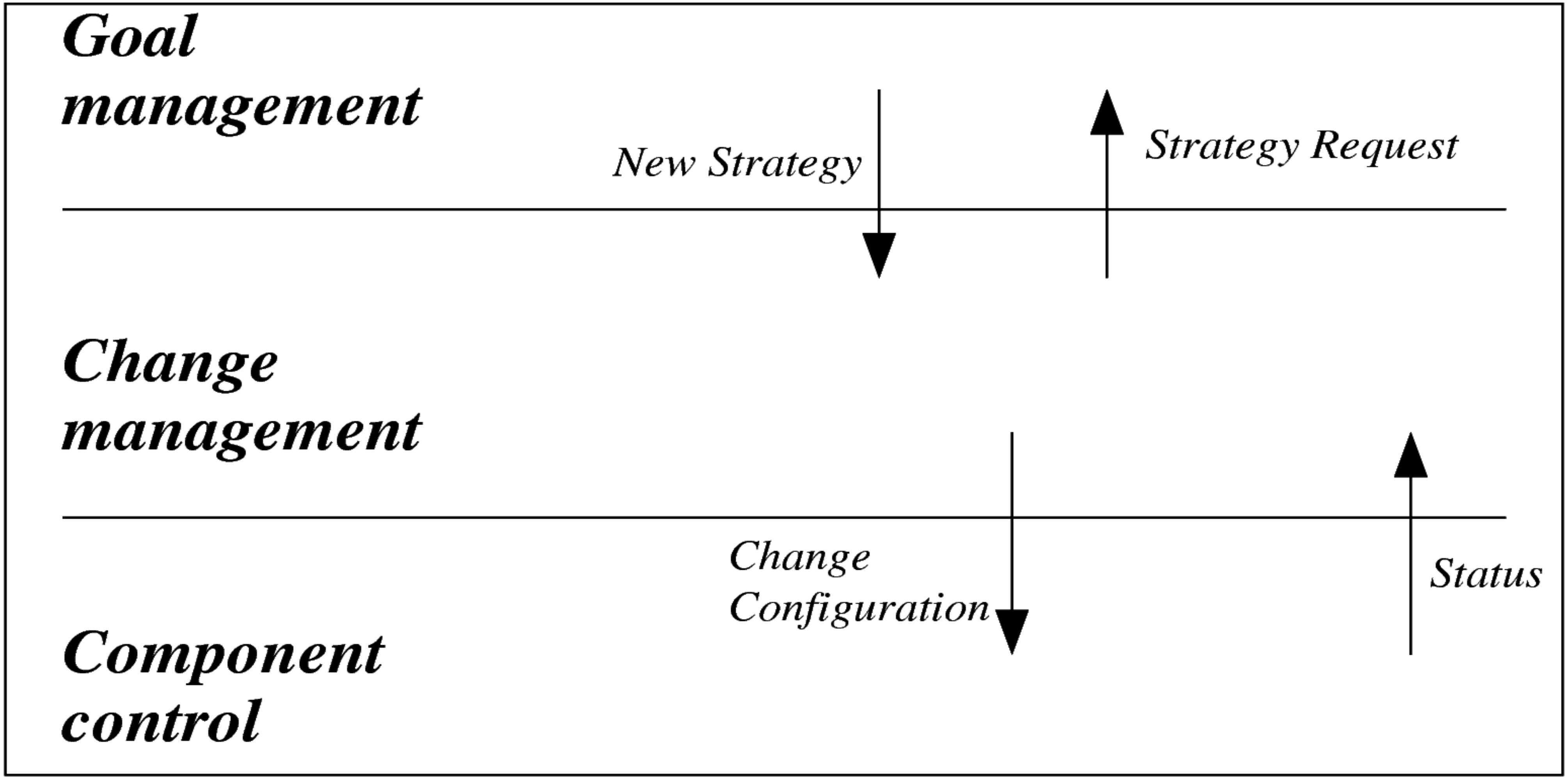
Ghezzi et al. "*Toward open world-software: Issues and challenges*", IEEE Computer 2006

Context (2)

- Kramer & Magee proposal
  - Architecture for self- managed systems
    - Reference architecture.
    - Three layers → KM-3L
    - Benefits:
      - Scalability, abstraction, etc.
    - Inspired in autonomic systems (robotics), since they are self-managed systems.

Kramer and Magee “*Self-managed systems: an architectural challenge*”, FOSE 2007

KM-3L



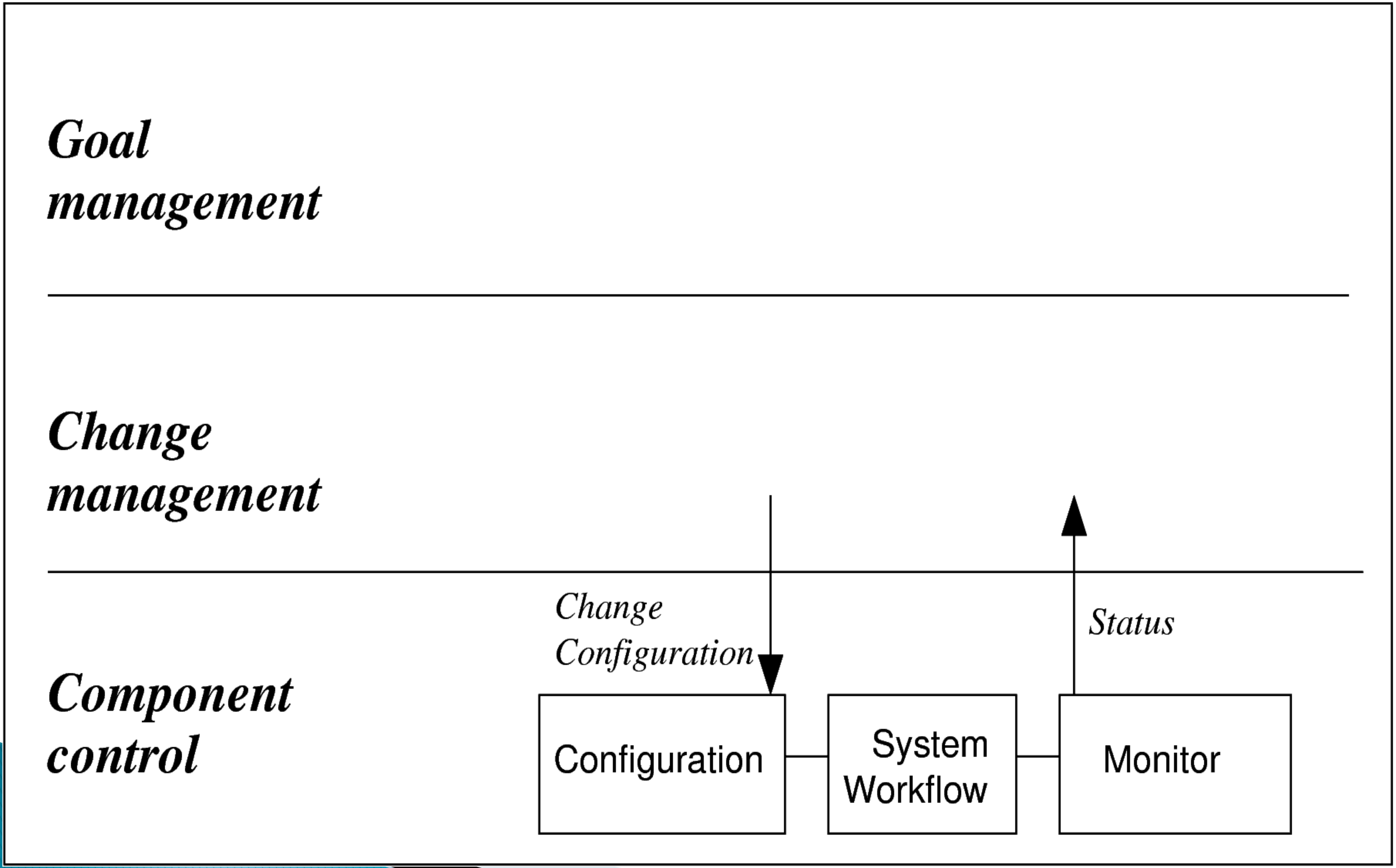
Goal management
New Strategy
Strategy Request
Change management
Change Configuration
Status
Component control

KM-3L

- Idea
  - Identify what a self-managed system needs to carry out its mission, without human intervention.
- *Component control*
  - Carries out the system *mission.*
  - Sense environment; report *status.*
- *Change management*
  - Has the *strategy* to carry out the mission.
  - With a new *status*, executes the *strategy* to produce a new system *configuration.*
  - If the new configuration does not fulfill the mission then asks for *a new strategy.*
- *Goal management*
  - Produces strategies that satisfy the mission and consider the current configuration.

Challenge: exploit KM-3L for the open-world to incorporate performance

KM-3L-4-OpenWorld: Component Control

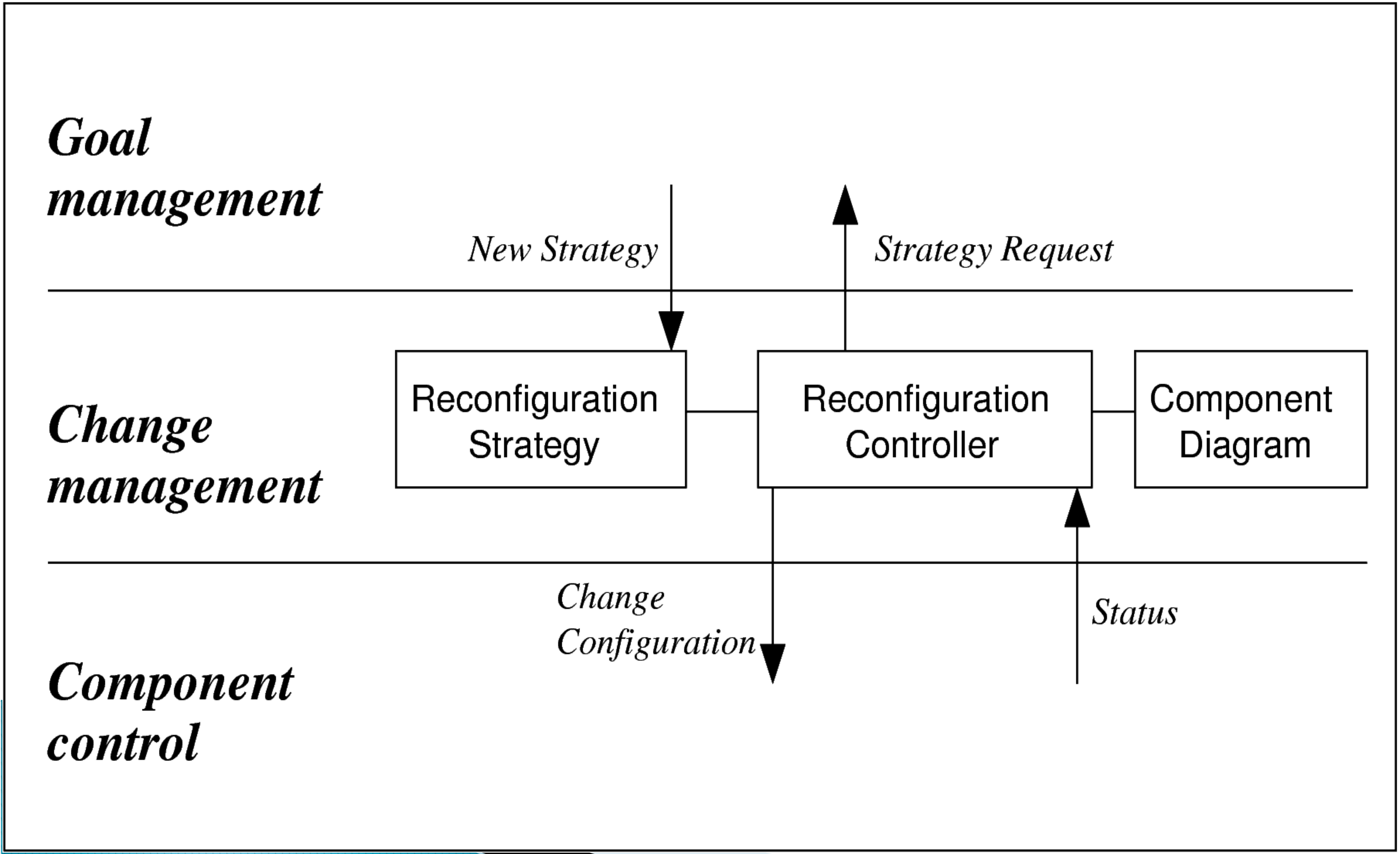


KM-3L-4-OpenWorld: Component Control

- **Responsibilities:**
  1. Tracking performance of components.
  2. Discover new components.
  3. Discover which components are no longer available.
  4. Bind & unbind components.
- **Key: monitor module**
  - (1) Measure time elapsed in the service calls.
  - (2,3 and 4.) As usual in open-world.

- **Other needs:**
  - Workflow (e.g., UML activity diagram)
  - Syste
m
c
onfiguration (e.g., UML component diagram)
- **Output:**
  - Curr
en
t
status (monitored time,unreachable service)
- **Input:**
  - New configuration

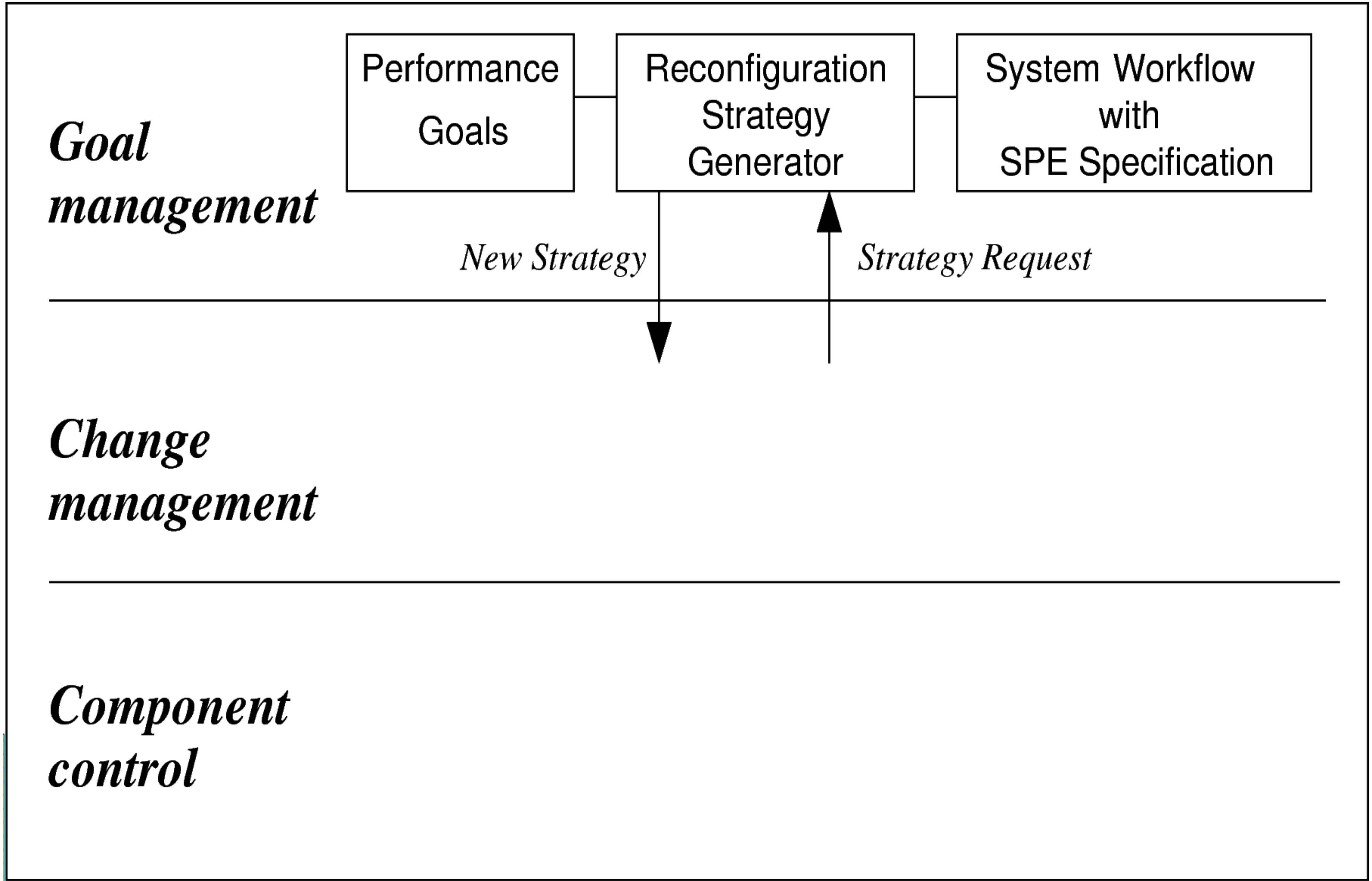
KM-3L-4-OpenWorld: Change Management



KM-3L-4-OpenWorld: Change Management

- **Key:**
  - Reconfiguration controller module.
- **Output:**
  - New system configuration.
- **Input:**
  - System status.
  - A new strategy.
- **Actions:**
  - A component is no longer available or *degraded*.
    - Executes the strategy to find a proper substitute.
    - Reports new configuration.
  - A new component is available for a given service.
    - Updates the current system configuration.

KM-3L-4-OpenWorld: Goal Management

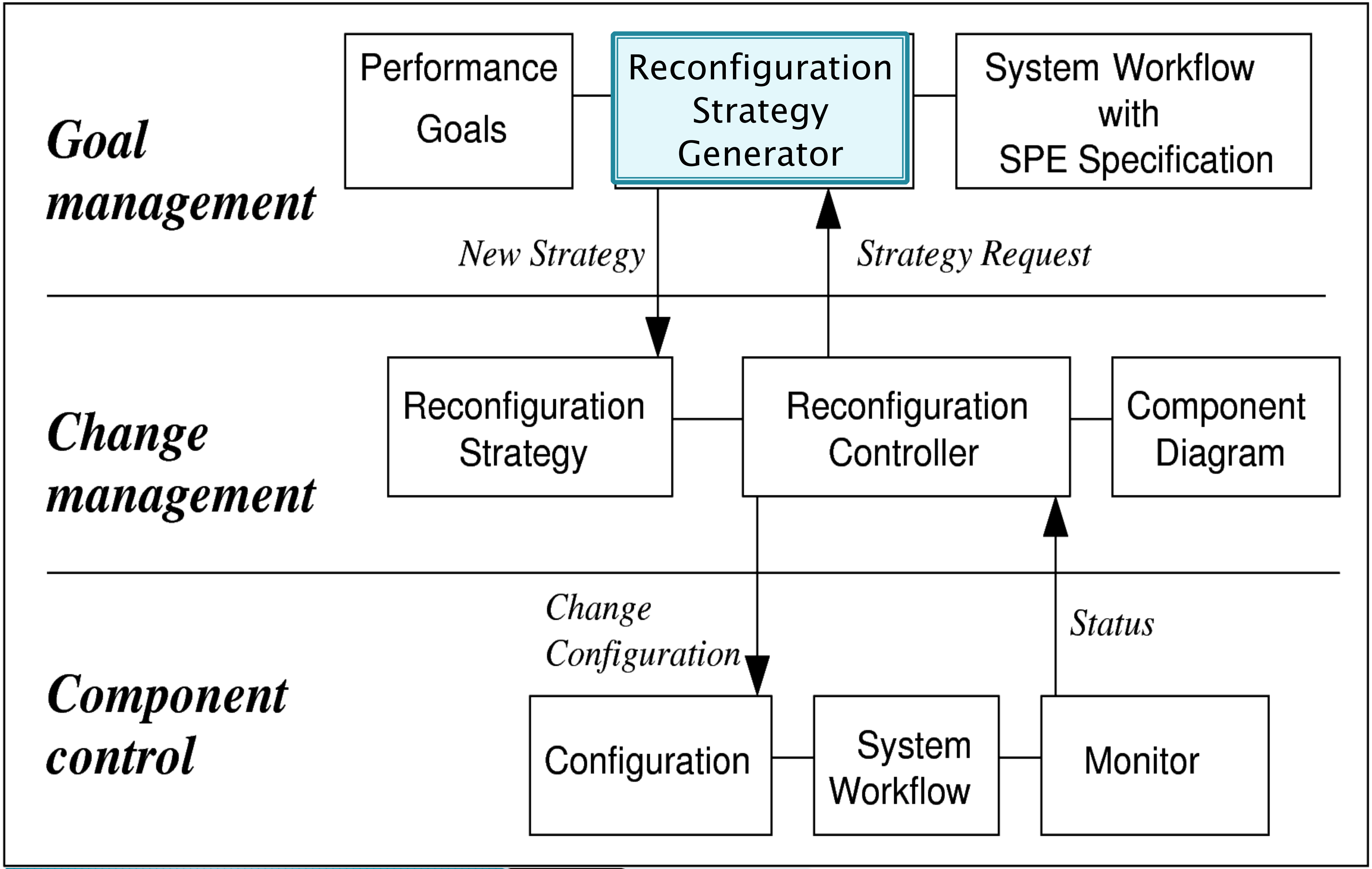


KM-3L-4-OpenWorld: Goal Management

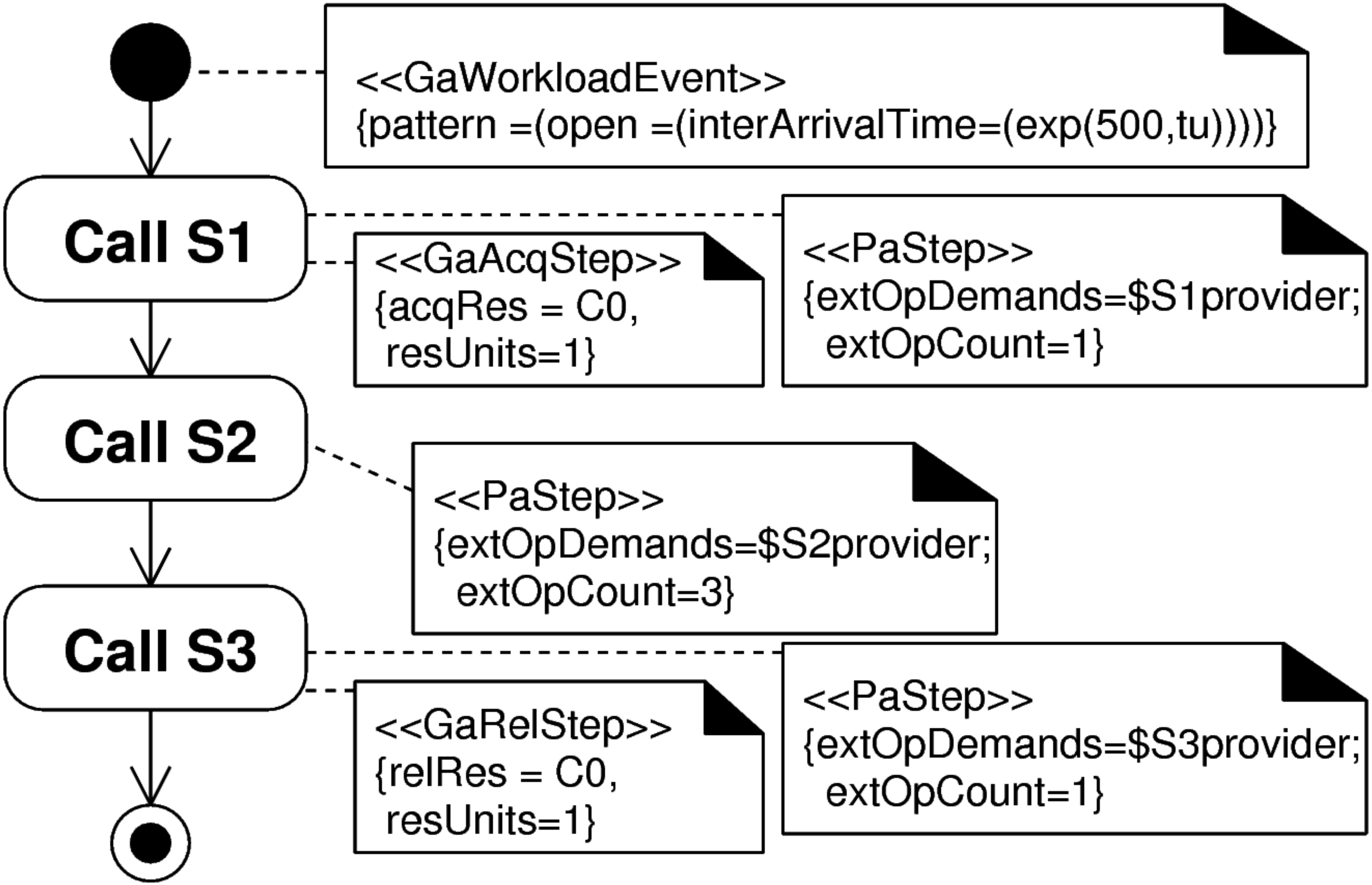
- **Responsibility:**
  - Produce performance aware reconfiguration strategies.
- **Key:**
  - Strategy generator module.
- **Approaches:**
  - ✗ Library of strategies.
  - ✓ Produce the strategy *on demand*.

- **Output:**
  - Strategy that meets the
    targe
    t
    performance goal (e.g., response time)
- **Input:**
  - The performance goal.
  - The workflow specification.
  - The current configuration.
- **Discussion point:**
  - T
    o
    m
    eet other goals (e.g., availability, price).

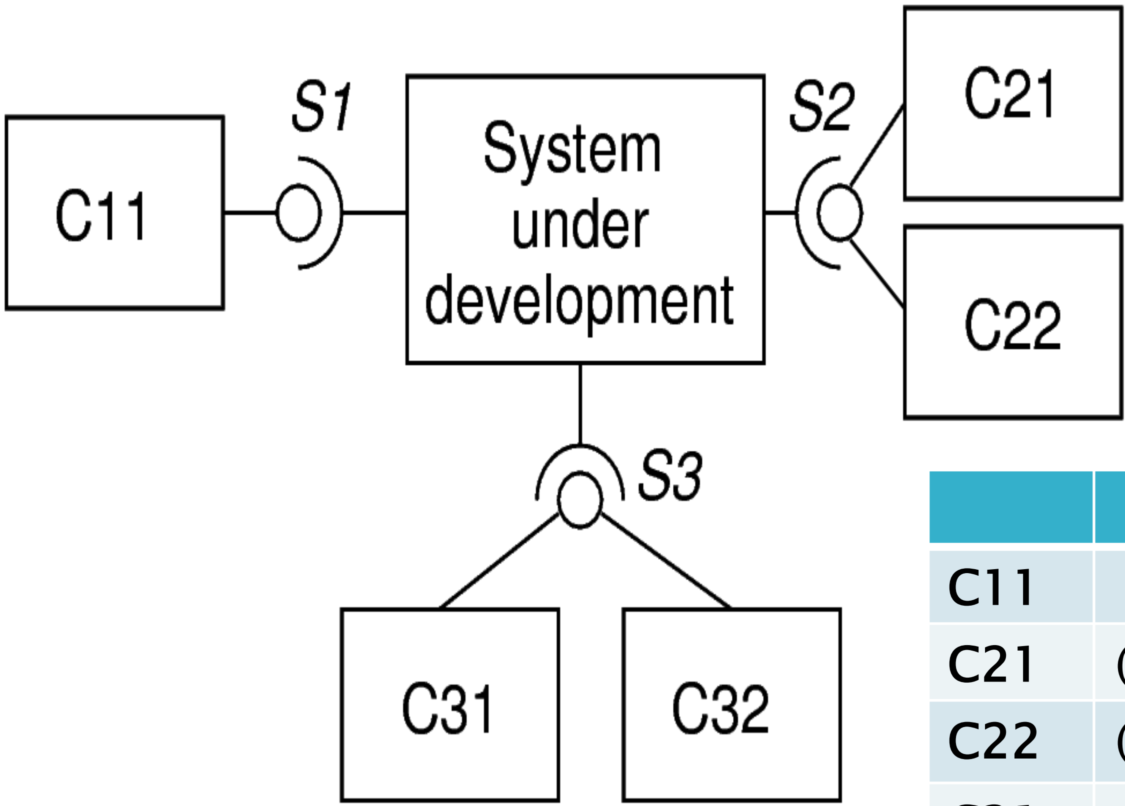
KM-3L-4-OpenWorld



Example (inputs)



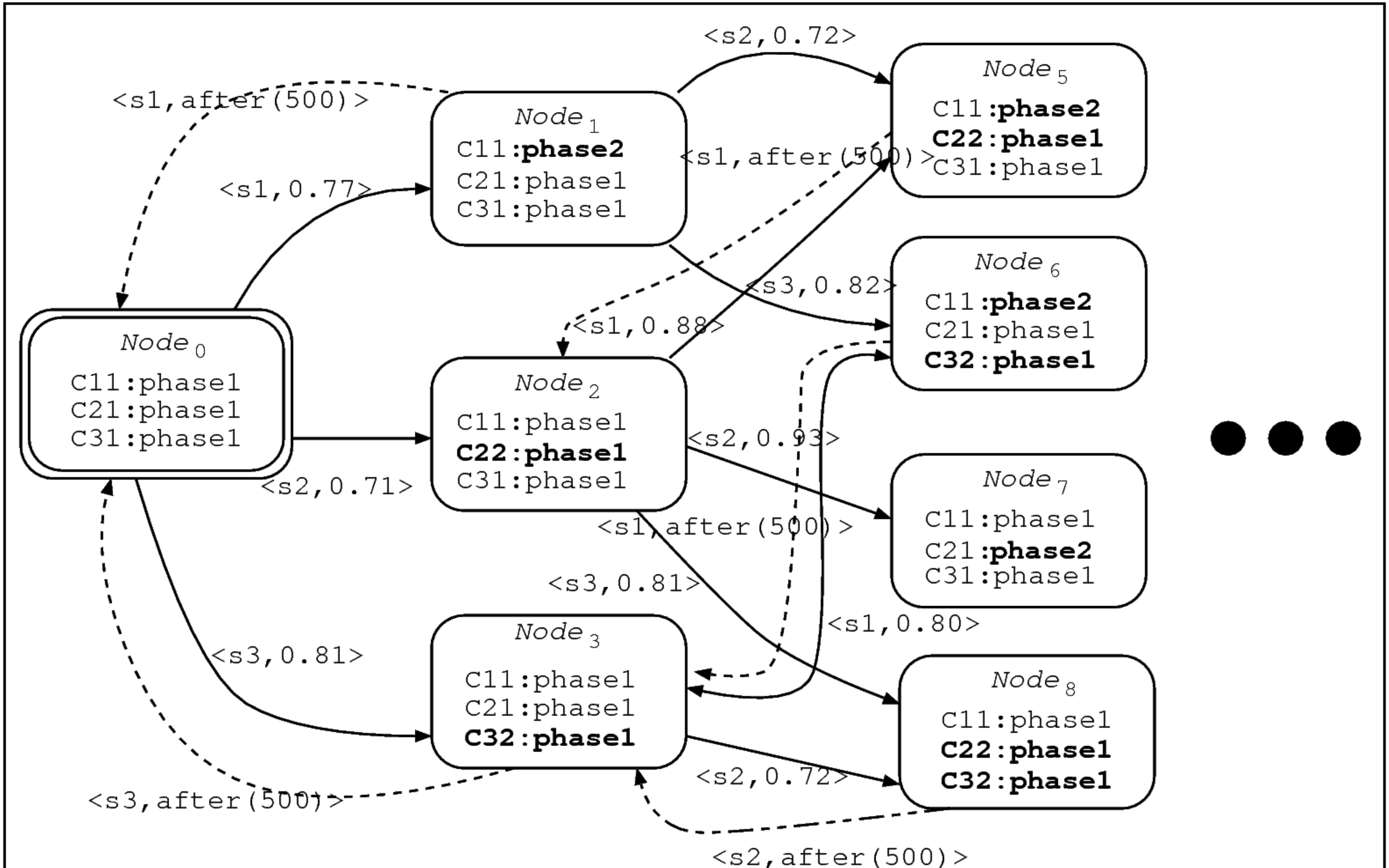
Example (inputs)



Time Table

	phase1	*phase2*	*phase3*
C11	(5,3000)	(20,6000)	
C21	(10,6000)	(70,2000)	(250,2000)
C22	(35,6000)	(140,4000)	
C31	(20,2000)	(70,2000)	
C32	(30,∞)		
(MeanServiceTime, MeanSojournTime)			

Example (output)



Example (strategy graph)

- Reconfiguration strategy → directed graph
- *Nodes* are system configurations
- *Edges* represent changes of configurations
  - *Forward* edges:
    - Replacement of a component.
    - Phase change of a component.
    - Labels → confidence levels.
  - *Backward* edges:
    - Timeouts to bring back the system to a previous configuration.

Example (1st Step: create initial node)

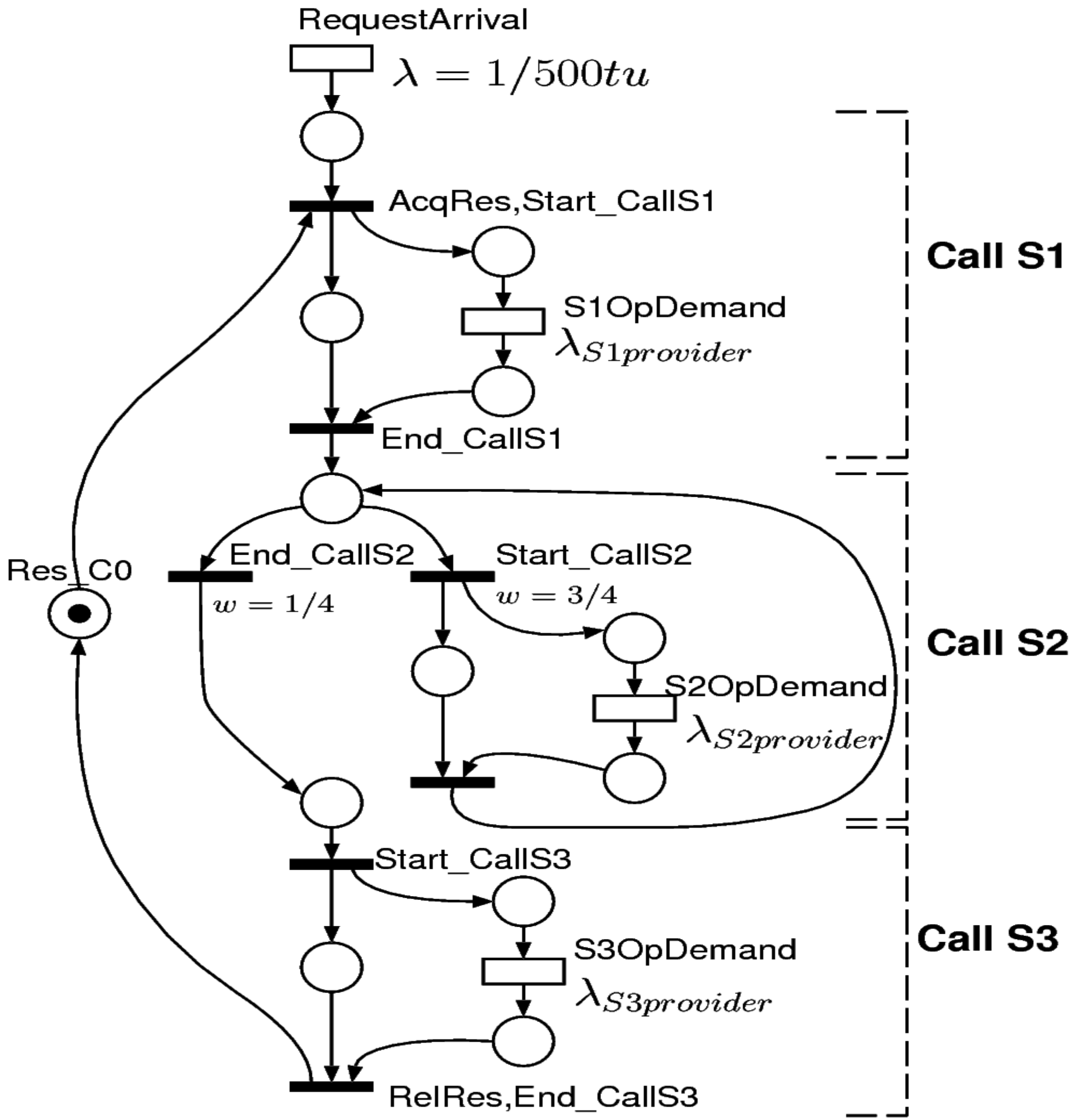
- Assume each provider works in best mode, i.e., minimum mean service time
- Four possible configurations in the example.

Mean response time estimation			
C11:ph1	C21:ph1	C31:ph1	60.5
C11:ph1	C22:ph1	C31:ph1	177.6
C11:ph1	C21:ph1	C32:ph1	72.5
C11:ph1	C22:ph1	C31:ph1	193.8

- Each configuration parameterizes the Petri net.
- Solve the Petri nets and choose the best configuration.

Example (Petri net)

$$\mathcal{GSPN} = (\mathcal{N}, \{\lambda_{S1provider}, \lambda_{S2provider}, \lambda_{S3provider}\})$$



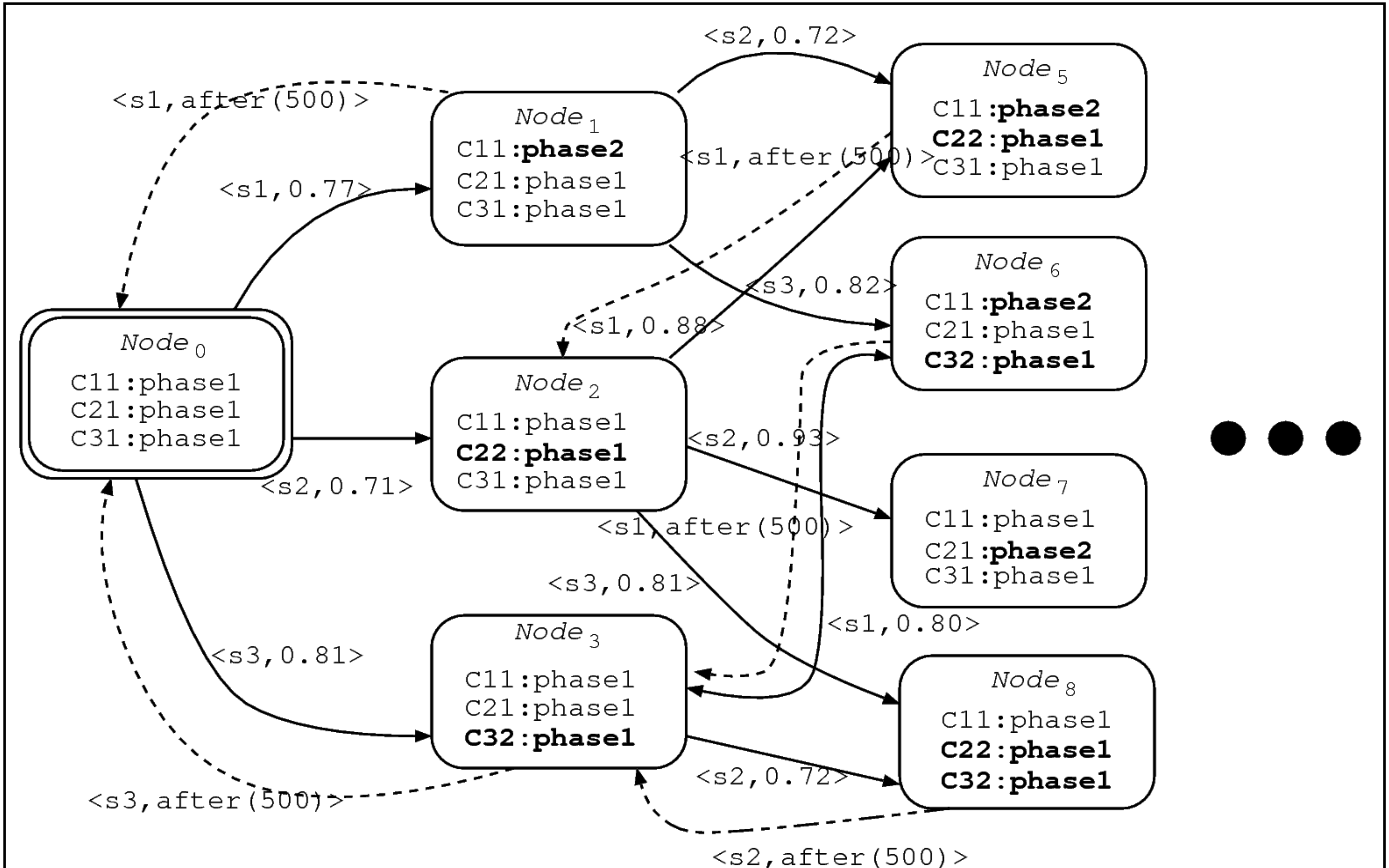
Example(2nd Step: create adjacent nodes)

- Consider that current

  pro

  viders can degrade their performance → 3 adjacent nodes
- *Node1* (provider one degraded)
  - No choice → only one provider
  - Solve the Petri net using *phase2* of C11.
  - Is the performance goal achieved?
- *Node2* (provider two degraded)
  - Alternatives: use C22 or C21 in *phase2*.
  - Again four possible configurations.
  - Solve the Petri net.

Example (output)



Example(3rd Step: Labels)

- Rational:
  - Our *confidence* in a configuration change.
  - Ad-hoc *heuristic* under the open workload assumption.
- Confidence = Improvement/(Improvement+Lost)
  - Impro

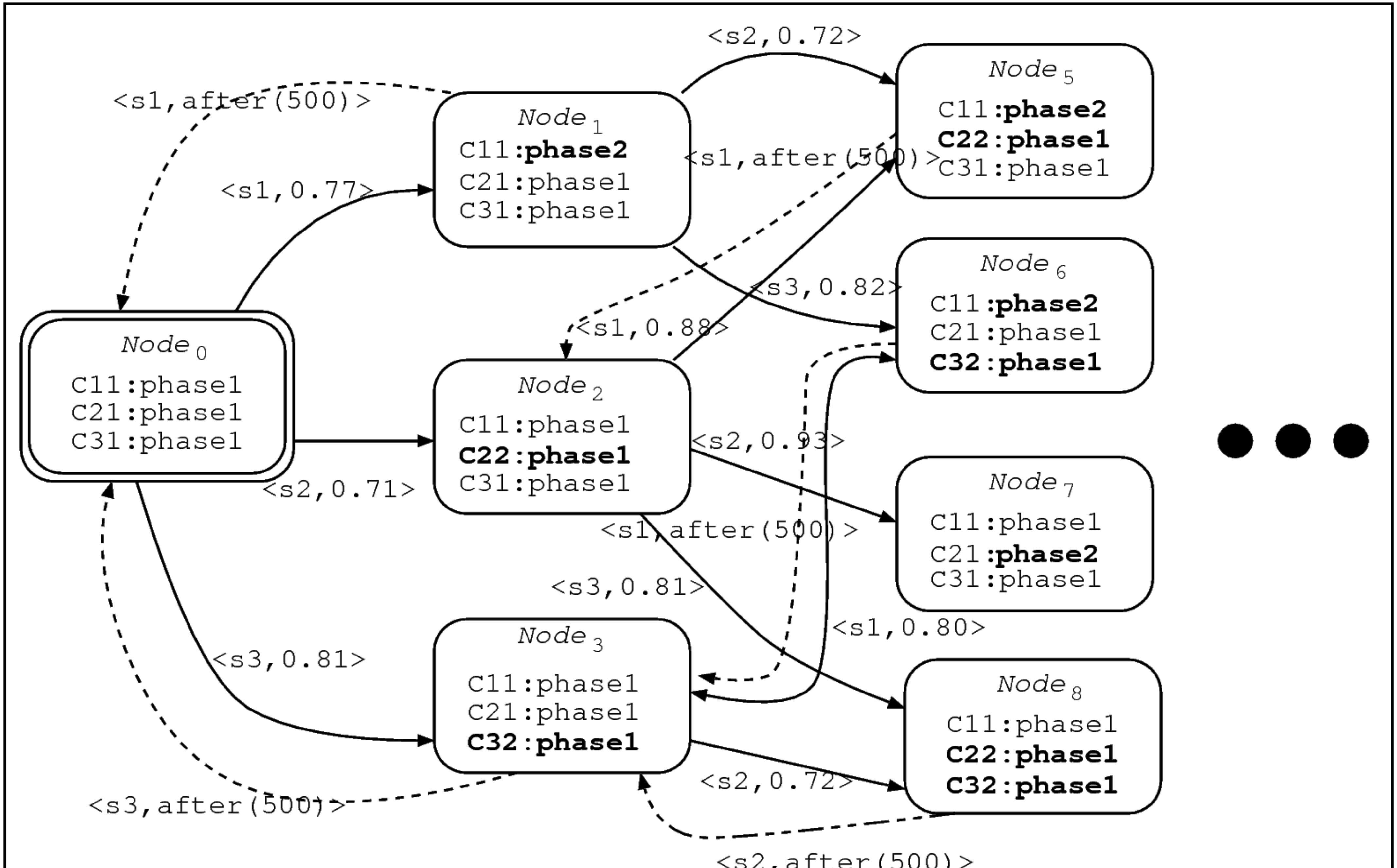
    ve

    ment = RT_source_ch_phase- RT_target (*OK reconfiguratio*
  - Lost = RT_target - RT_source (*wrong reconfiguration*)

Example (4th Step: backward edges)

- Rational
  - We can perform erroneous reconfigurations.
  - So, after a timeout *bring back the system* to a state that performs better:
    - Identify nodes where components perform in their worst phase.
  - Ideal timeout? → Future work

Example (output)



Example (validation)

- We analyzed the system *without reconfigurations* and using the components with their best mean response times→ Response time: *494 tu*
- We analyzed the system *using the strategy graph* → Response time: *436 tu*
- Improvement: *11%*

Related works

- [5] Ghezzi & Tamburrelli ``Predicting
  perfo
  r
  mance properties for open systems with kami”, QoSA, 2009
  - ✓ Performance evaluation in
    open-
    w
    orld. Assuming components evolving independentely and unpredictably.
  - ➢ Queuing networks.
  - ➢ Does not address the problem of generate strategies.
- [10,11,15]
  Menascé'
  s
  works (ICWS'07, Performance Evaluation'07 and WOSP'05)
  - ✓ Evaluate service-based software

Conclusion

- **Original idea**
  - Introduce a reference architecture from self-managed systems in the open-world context.
- **Contributions**
  - Adapt KM-3L to open-world software → focuss the Performance problem.
  - Proposal for reconfiguration strategies module.
- **Challenge**
  - From models to real implementations → software with the ability to reconfigure itself.
  - Problem → run-time Petri net evaluation with *exact analysis tecniques*.
  - Solution → Use Petri net *bounds*.
- **Final Remark**
  - The algorithm has been implemented.

Thanks!