SAVE ORCA

Formal Modelling, Safety Analysis, and Verification of Organic Computing Applications

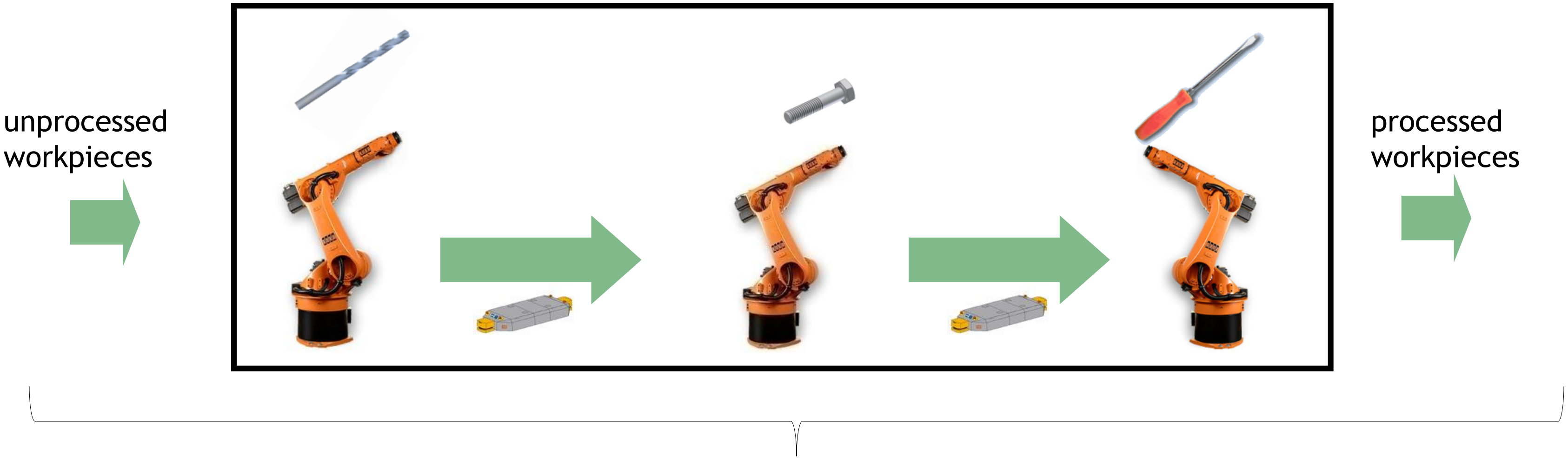
Wolfgang Reif, Florian Nafz, Hella Seebach, Jan-Philipp Steghöfer

Universität Augsburg

Outline

- Motivation, goals and challenges
- Target systems
- Software engineering for Organic Computing
- RIA: Restore Invariant Approach
- Formal modelling and verification
- ORE: ODP Runtime Environment
- Summary and outlook for next phase

Example: adaptive production cell



unprocessed workpieces

processed workpieces

+
Application specific software solution

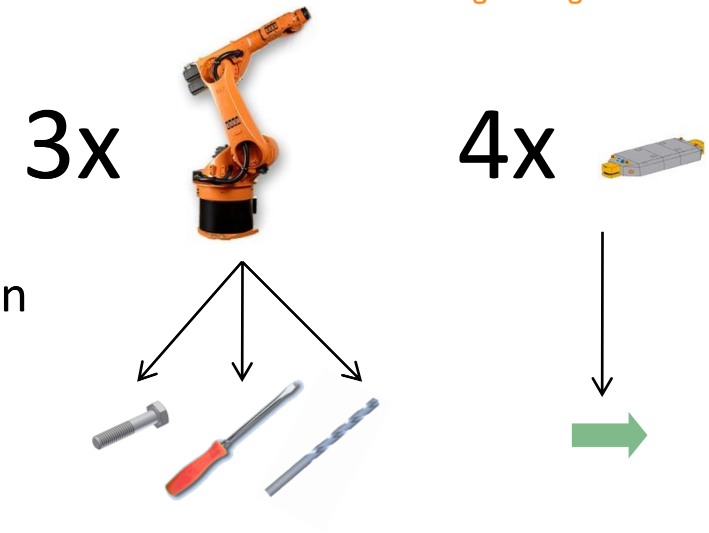
- Traditional control:
  - Addition of robots for better efficiency only manually.
  - Failure of one component leads to system failure.
  - Adaptation to new workpieces needs significant adaptation of control.

Example: adaptive production cell

- Use of flexible HW components
  - Flexible robots
  - Flexible transport system
  - Workpieces with RFIDs for identification

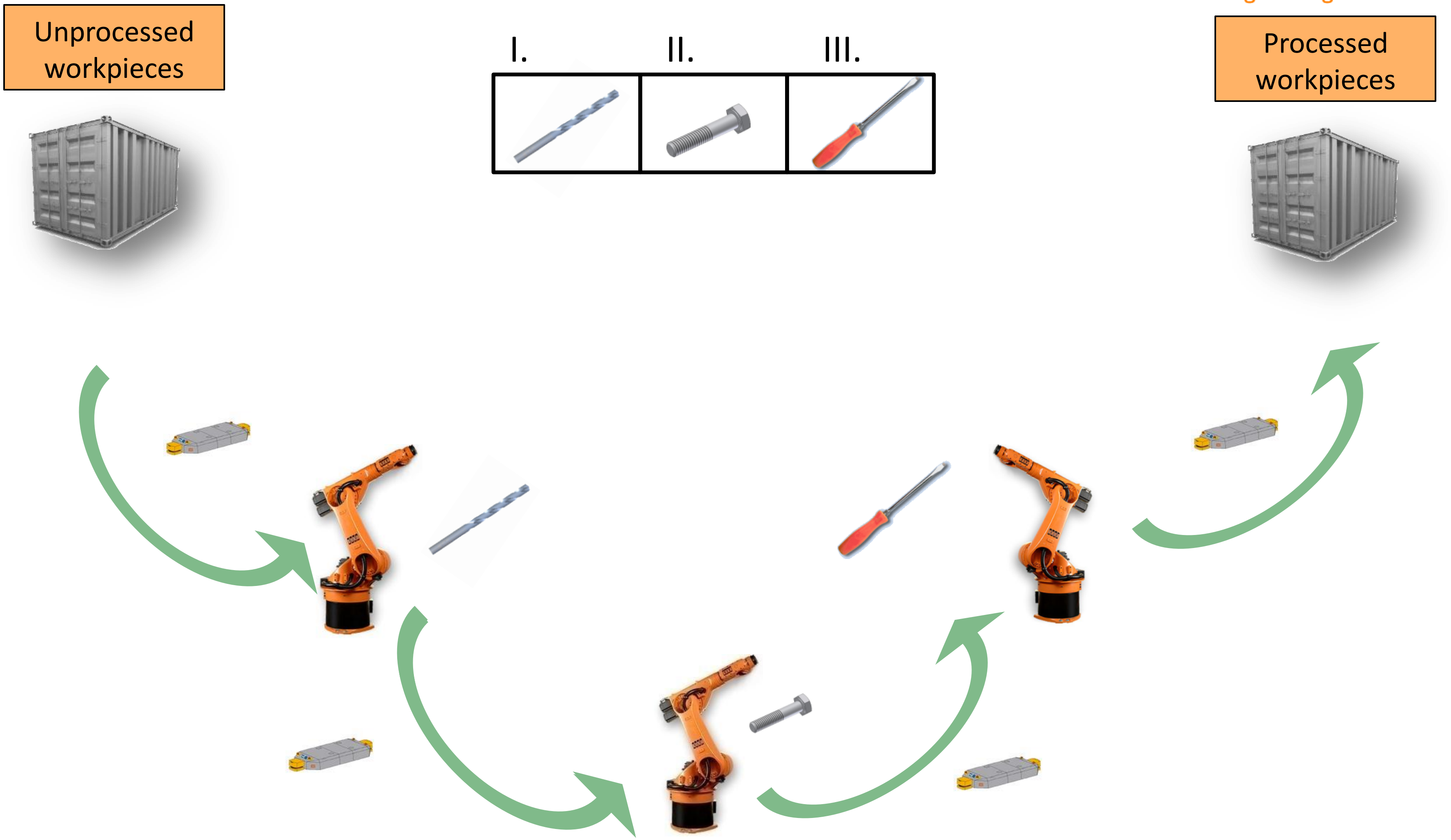
- Addition of degree of freedom (e.g.):
  - Use of different tools
  - Execution of different transport commands

- Observer/Controller:
  - Monitoring of workpieces and components in the system
  - Role distributions (re-)configuration of components



Universität
Augsburg

Example: adaptive production cell



Unprocessed workpieces

I. II. III.

Processed workpieces

Goals and Challenges

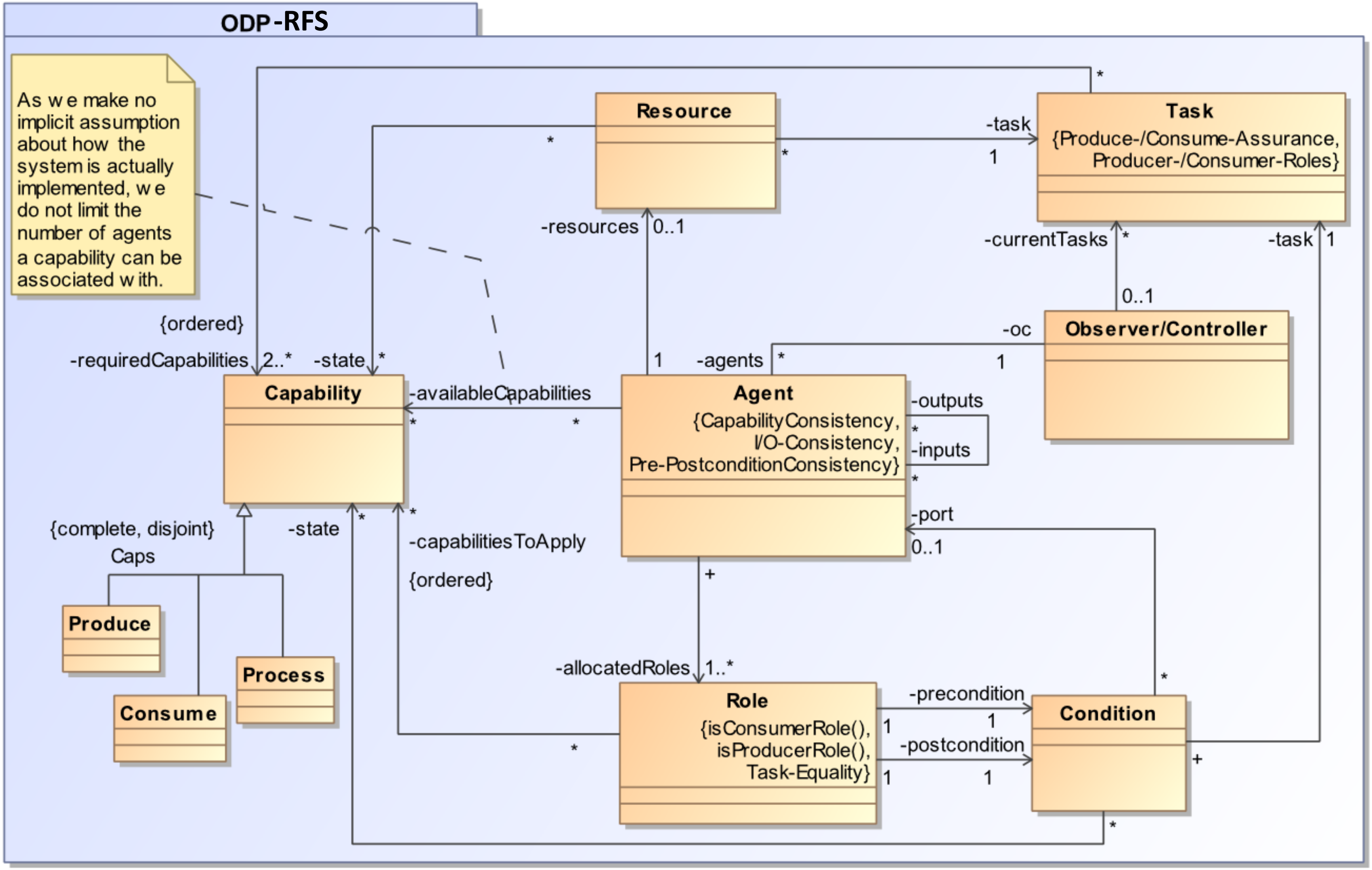
- Software & Verification Co-Design for highly reliable Organic Computing applications
  - Design and construction
    - Top-Down design methodology
    - Extensible generic runtime environment
    - Integrated software engineering process
  - Formalization of self-x
    - What does self-x mean in the context of the considered system class?
  - Methods and tools for formal analysis and verification
    - Correctness – and behavioral guarantees despite of self-organization
    - Qualitative and quantitative analysis

Target Systems

- Software intensive applications that are
  - particularly resistant against disturbances and component failures (w.r.t. functional correctness, safety, security)
  - adaptive to changing requirements and modified tasks
- Resource-flow systems
  - Production automation
  - Logistics
- Agent / Role based systems
  - Each agent has several capabilities
  - Each task needs different processing steps
  - Processing steps are a given sequence of capabilities

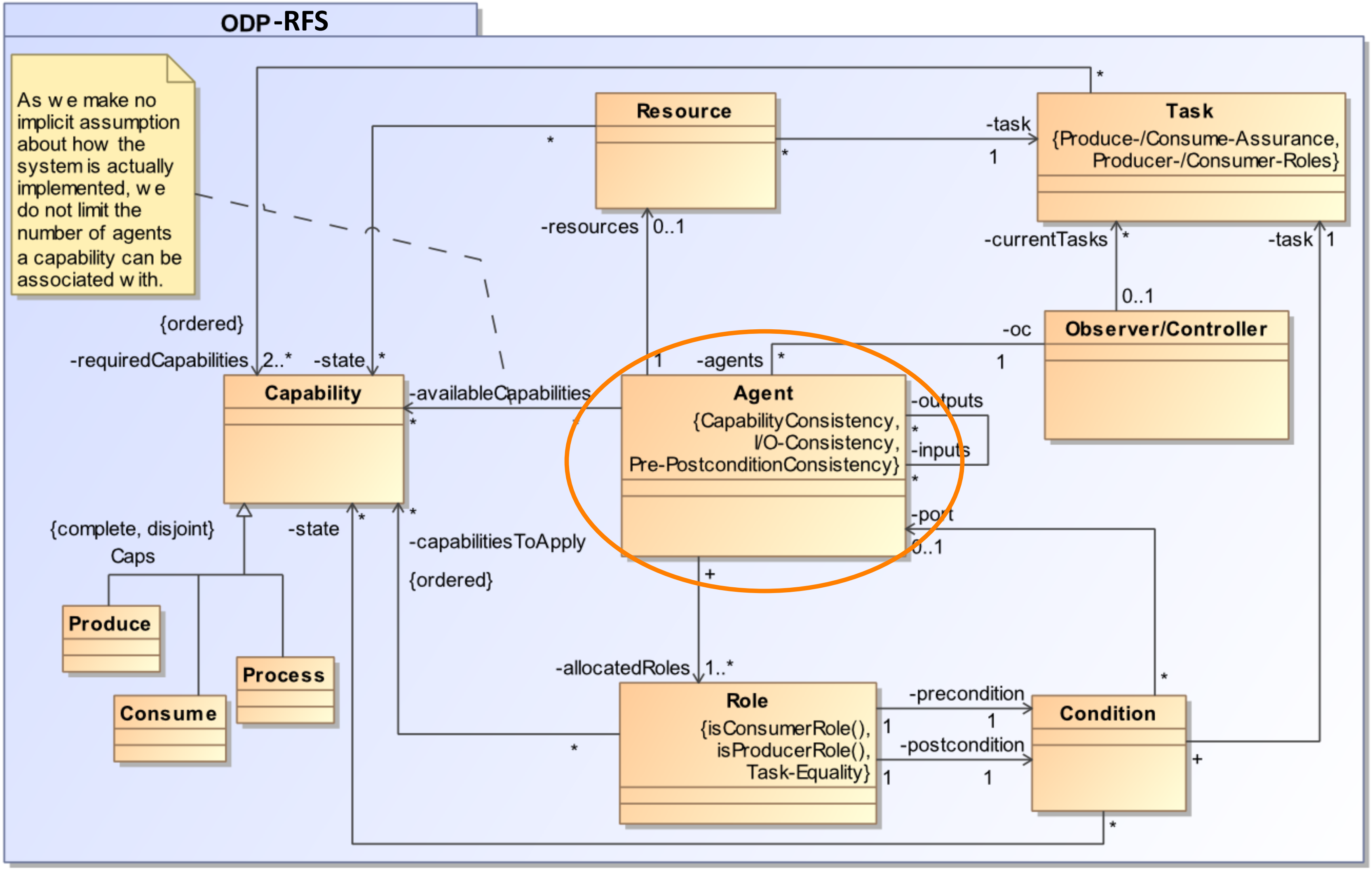
Organic Design Pattern for resource-flow systems

Institute for Software & Systems


ODP-RFS
As we make no implicit assumption about how the system is actually implemented, we do not limit the number of agents a capability can be associated with.
Resource
Task
{Produce-/Consume-Assurance, Producer-/Consumer-Roles}
-task
1
*
-resources
0..1
-currentTasks
-task
0..1
-oc
Observer/Controller
{ordered}
-requiredCapabilities
2..*
-state
Capability
-availableCapabilities
Agent
{CapabilityConsistency, I/O-Consistency, Pre-PostconditionConsistency}
-agents
-outputs
-inputs
-port
{complete, disjoint}
Caps
Produce
Consume
Process
-capabilitiesToApply
{ordered}
+
-allocatedRoles
1..*
Role
{isConsumerRole(), isProducerRole(), Task-Equality}
-precondition
-postcondition
Condition

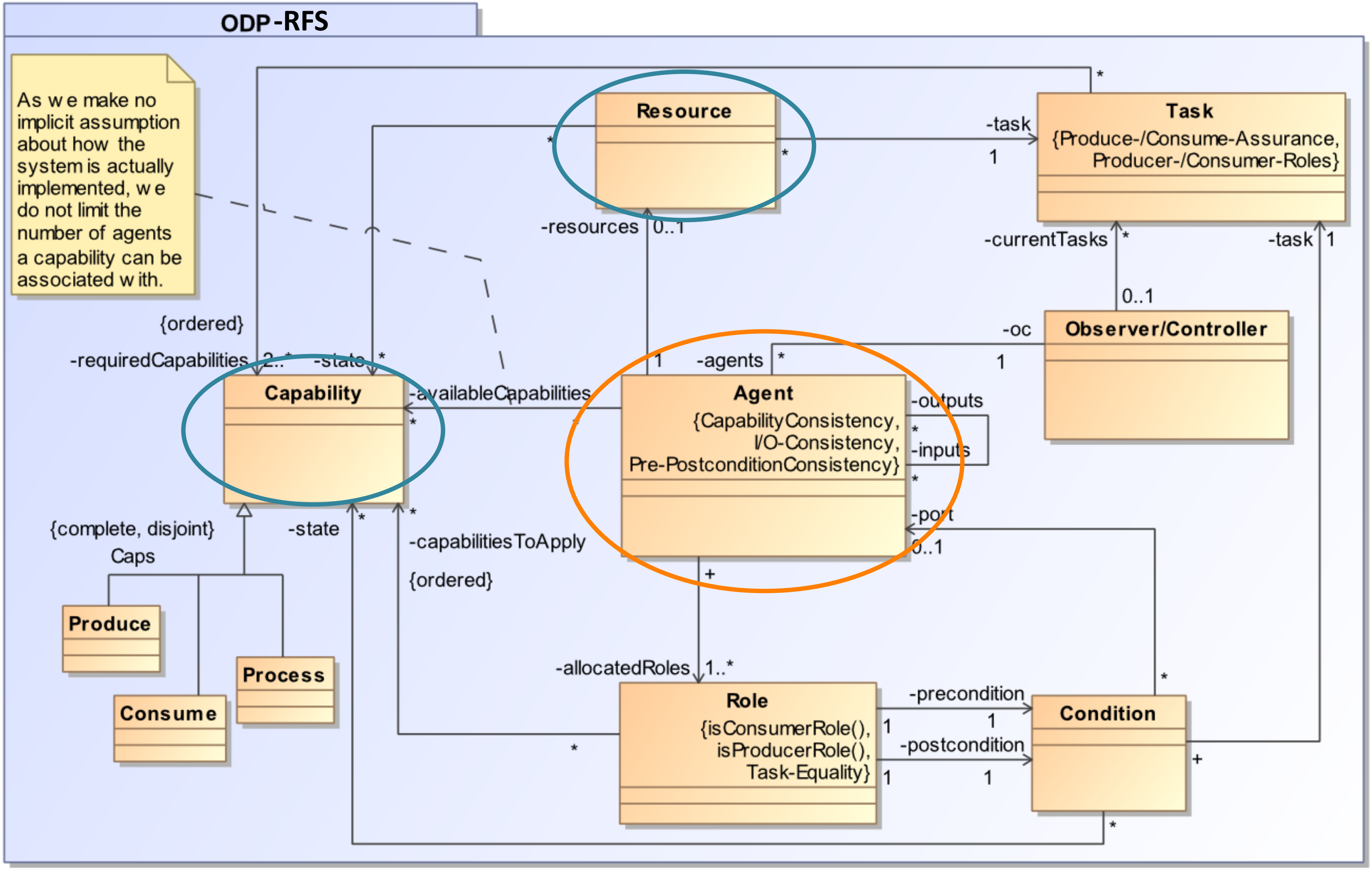
Organic Design Pattern for resource-flow systems

Institute for Software & Systems



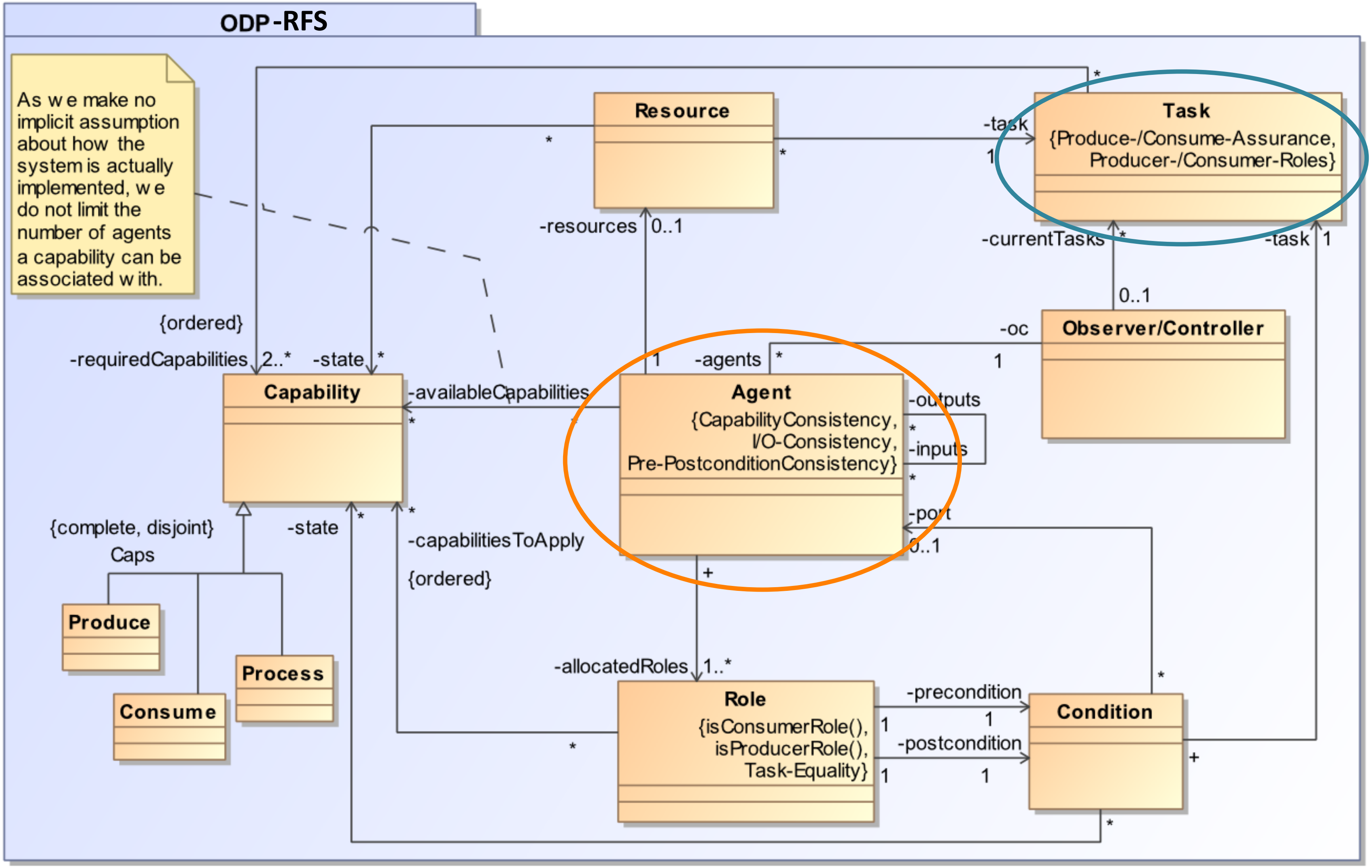
Organic Design Pattern for resource-flow systems

Institute for Software & Systems

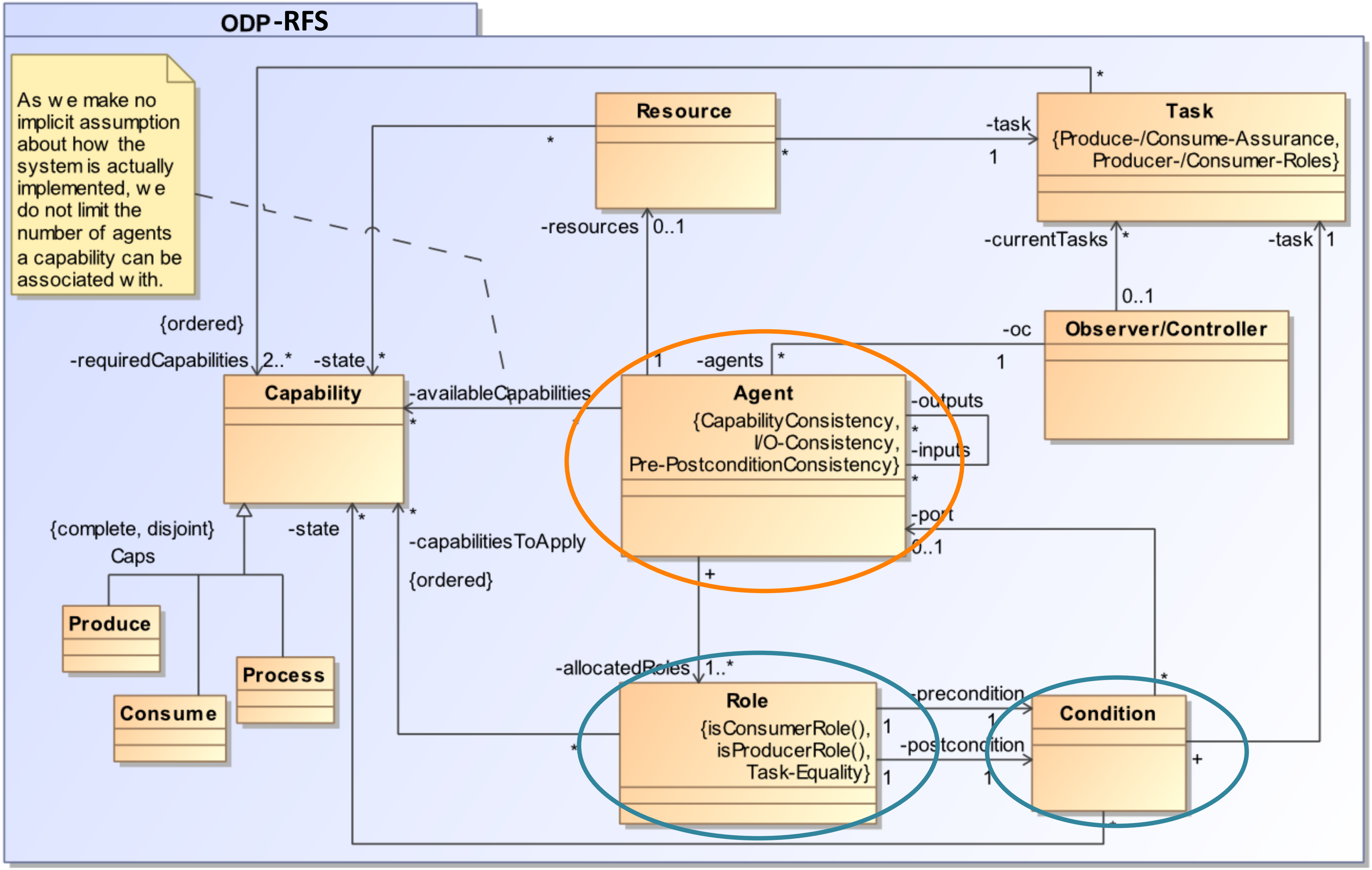


Organic Design Pattern for resource-flow systems

Institute for Software & Systems

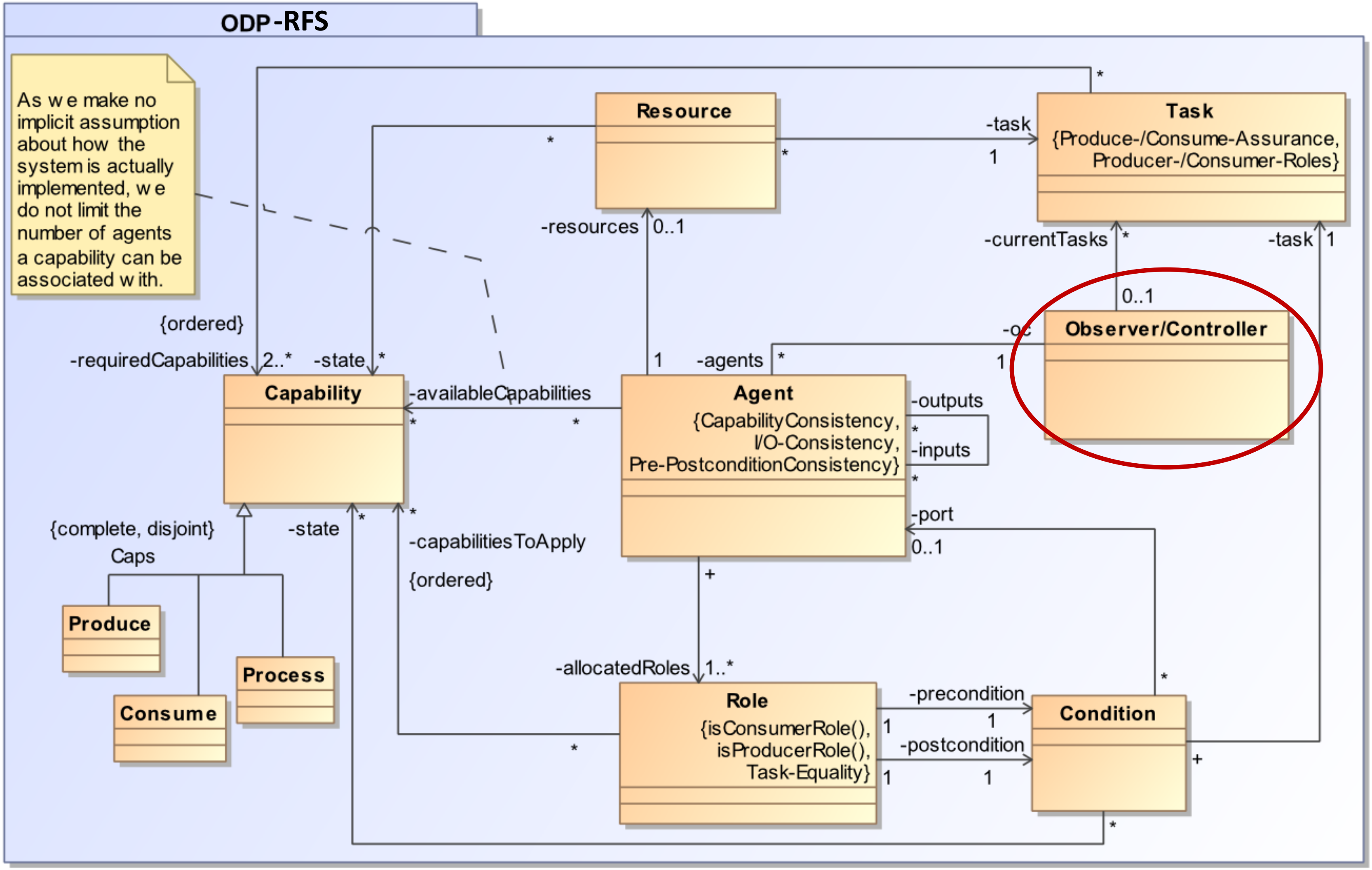


Organic Design Pattern for resource-flow systems


As we make no implicit assumption about how the system is actually implemented, we do not limit the number of agents a capability can be associated with.

Organic Design Pattern for resource-flow systems

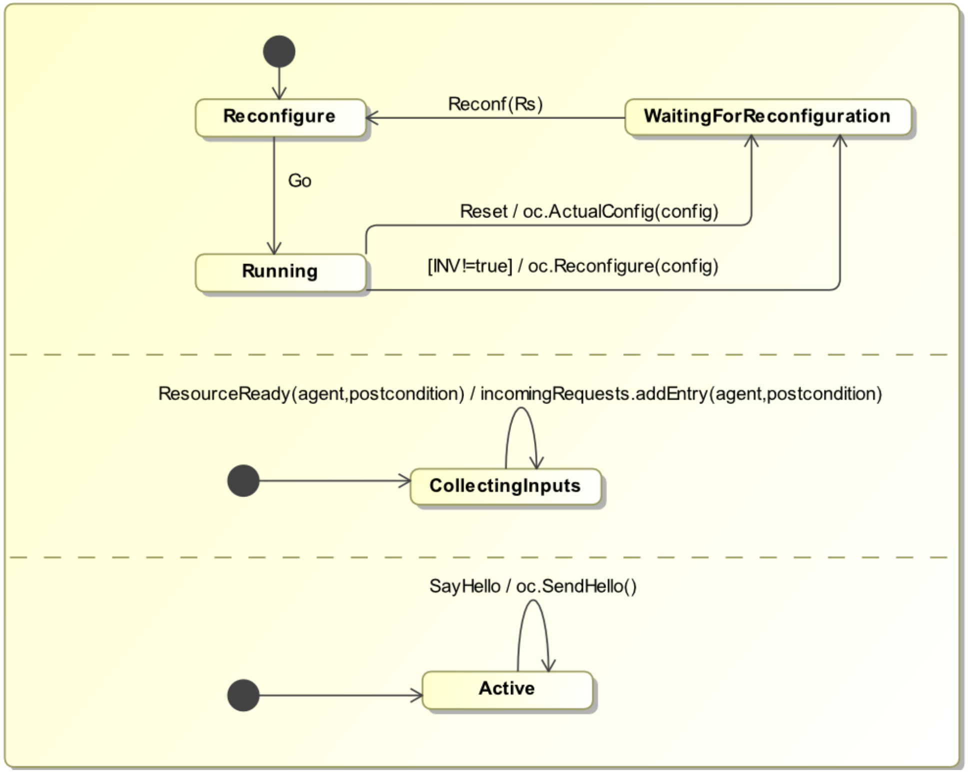
Institute for Software & Systems



Dynamics

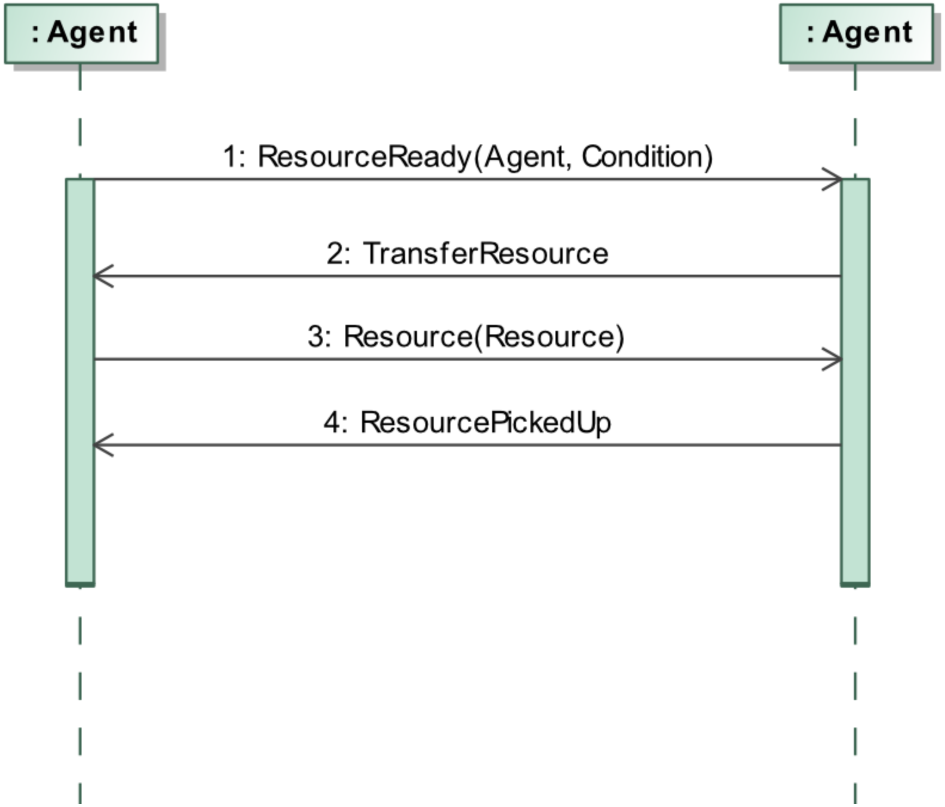
Institute for Software & Systems Engineering

- Generic behavior specified on system class level
- Hierarchical statemachines for agents and observer / controller
- Underlying SOS-semantics for formal model


Reconfigure
Reconf(Rs)
WaitingForReconfiguration
Go
Reset / oc.ActualConfig(config)
Running
[INV!=true] / oc.Reconfigure(config)
ResourceReady(agent,postcondition) / incomingRequests.addEntry(agent,postcondition)
CollectingInputs
SayHello / oc.SendHello()
Active

Communication

- Modelled as sequence diagramms
- One protocol for each communication act
- Three for the class of resource-flow systems

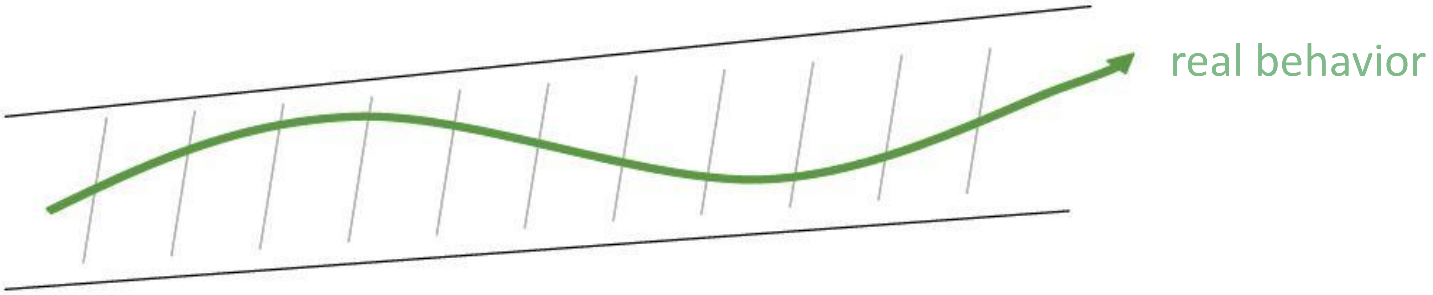


RIA: Restore Invariant Approach

- How to reconcile behavioral guarantees despite self-X ?

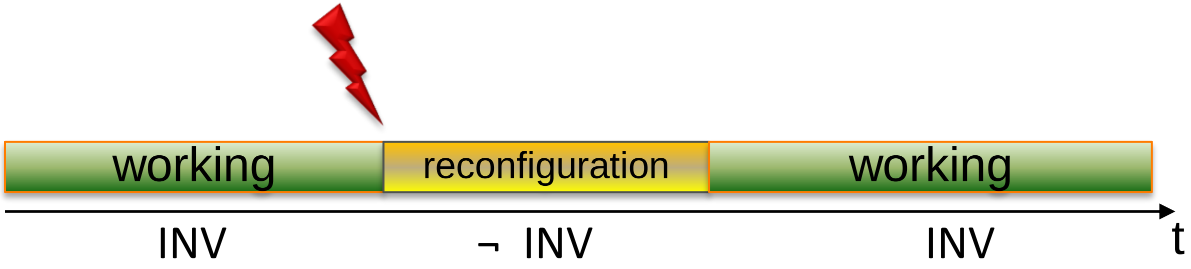
Approach

- Define a functional corridor of acceptable behavior
  - Invariants that have to be maintained by the system
- Within the corridor: let the system go



RIA: Restore Invariant Approach

- Specification of reconfiguration
  - o/c reconfiguration is triggered by invariant violation
  - o/c tries to restore invariant



- Reconfiguration can be specified as constraint solving problem
  - ❖ Universal reconfiguration with SAT-Solver/Constraint-Solver

Specification of corridor

- OCL constraints
  - Part of the pattern
  - Specifying „correct“ role allocations which imply wanted behavior
  - Are transformed into predicate logic formula for reconfiguration and formal specification of the o/c

- Some examples:
  - Only available capabilities are assigned

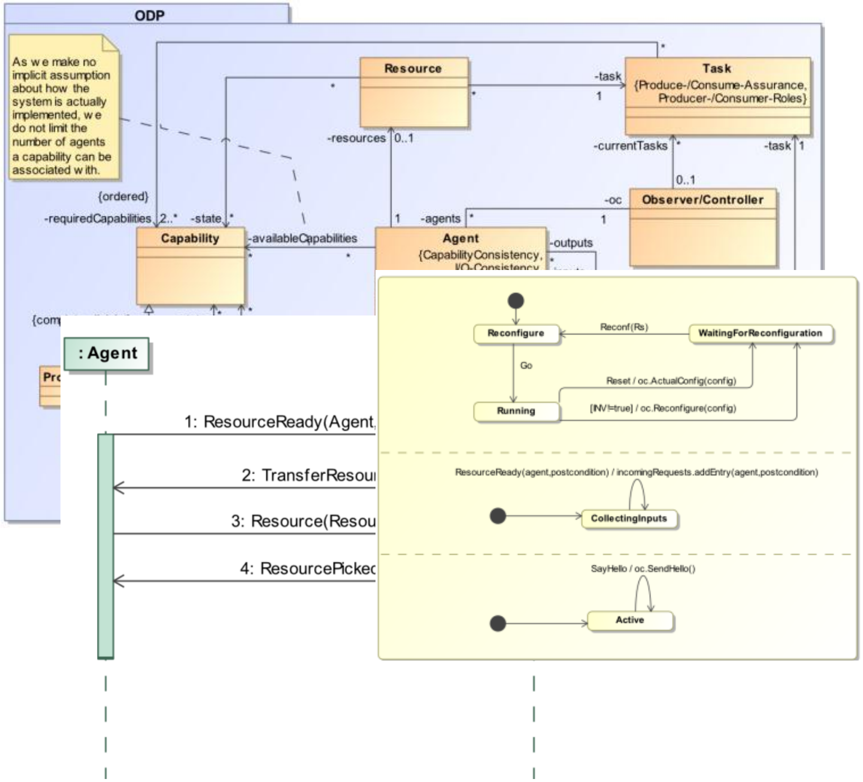
    inv: self.availableCapabilities

    -> includesAll(self.allocatedRole.CapabilityToApply)

  - Ports must be consistent with input/putput
  - Agents who exchange resources need to be connected
  - All needed capabilities must to be assigned

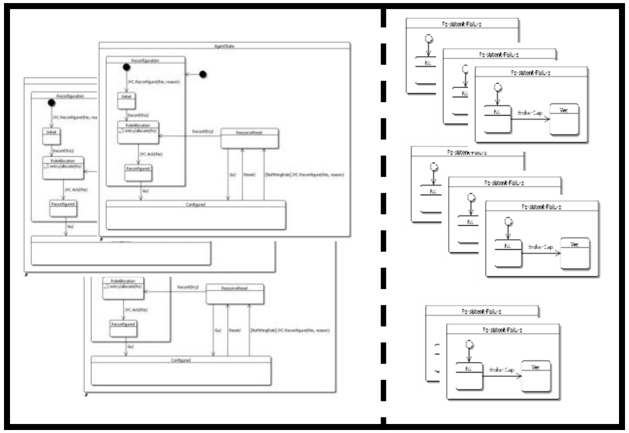
Formal Model

Software Engineering Models



As we make no implicit assumption about how the system is actually implemented, we do not limit the number of agents a capability can be associated with.

Parameterized formal model



Formal Verification

- Based on parameterized formal model
  - Generic verification of system class properties
    - Verified once and for all
    - „ Resource-flow is correct“
    - „ Leaving resources have been processed according to their task“
    - „ Agents behave according to their roles“
  - Application specific extensions
    - Need to be verified once per application
    - Using instantiated parameterized model

Formal Analysis

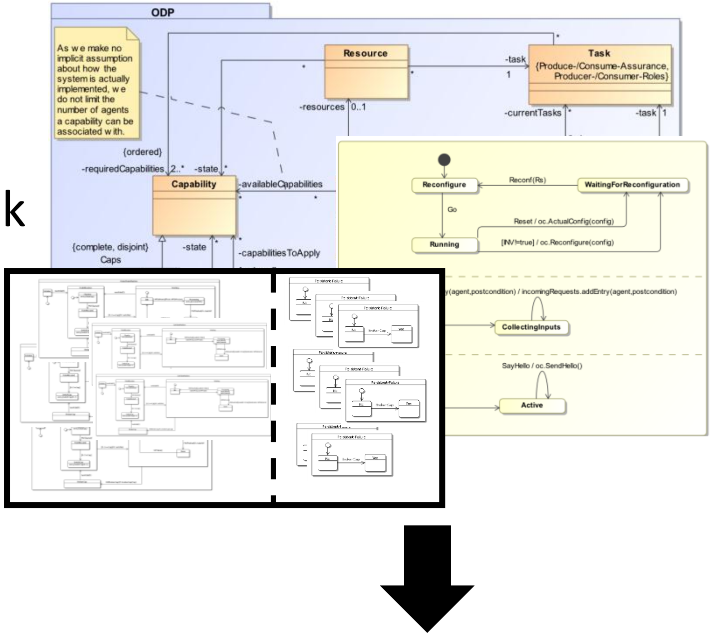
- Adaptive DCCA answers the question:

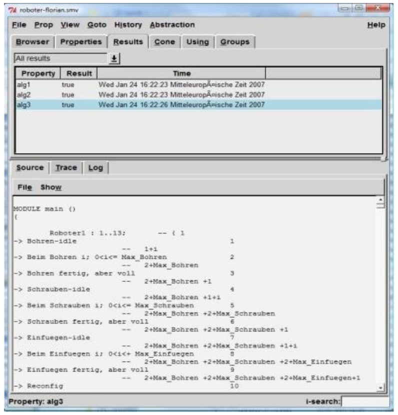
  “Which minimal combination of losses of capabilities can prohibit fulfillment of the task permanently?”

  in other words:

  “How much **self-healing** is in the system?”

- Process:
  - Translate the model into the language of a verification engine (here SMV)
  - aDCCA can then be formulated as (automatically solvable) deduction problem


ODP
As we make no implicit assumption about how the system is actually implemented, we do not limit the number of agents a capability can be associated with.
Resource
-task
1
Task
{Produce-/Consume-Assurance, Producer-/Consumer-Roles}
-resources 0..1
-currentTasks
-task 1
{ordered}
-requiredCapabilities 2..*
-state
Capability
-availableCapabilities
{complete, disjoint}
Caps
-state
-capabilitiesToApply
Reconfigure
Reconf(Rs)
WaitingForReconfiguration
Go
Reset / oc.ActualConfig(config)
Running
[INV!=true] / oc.Reconfigure(config)
y(agent,postcondition) / incomingRequests.addEntry(agent,postcondition)
CollectingInputs
SayHello / oc.SendHello()
Active


roboter-florian.smv
File Prop View Goto History Abstraction Help
Browser Properties Results Cone Using Groups
All results
Source Trace Log
File Show
i-search:

Property	Result	Time
alg1	true	Wed Jan 24 16:22:23 MitteleuropÃ¤ische Zeit 2007
alg2	true	Wed Jan 24 16:22:23 MitteleuropÃ¤ische Zeit 2007
alg3	true	Wed Jan 24 16:22:26 MitteleuropÃ¤ische Zeit 2007

```
MODULE main ()
{

        Roboter1 : 1..13;        -- { 1
-> Bohren-idle                                    1
                          --   1+i
-> Beim Bohren i; 0<i<= Max_Bohren                2
                          --   2+Max_Bohren
-> Bohren fertig, aber voll                       3
                          --   2+Max_Bohren +1
-> Schrauben-idle                                 4
                          --   2+Max_Bohren +1+i
-> Beim Schrauben i; 0<i<= Max_Schrauben          5
                          --   2+Max_Bohren +2+Max_Schrauben
-> Schrauben fertig, aber voll                    6
                          --   2+Max_Bohren +2+Max_Schrauben +1
-> Einfuegen-idle                                 7
                          --   2+Max_Bohren +2+Max_Schrauben +1+i
-> Beim Einfuegen i; 0<i<+ Max_Einfuegen          8
                          --   2+Max_Bohren +2+Max_Schrauben +2+Max_Einfuegen
-> Einfuegen fertig, aber voll                    9
                          --   2+Max_Bohren +2+Max_Schrauben +2+Max_Einfuegen+1
-> Reconfig                                       10
```

Property: alg3

(Formal) definition of self-x properties

- A system SYS, which is modeled as an instance of the organic design pattern is called

self-configuring for a task T, if the system is put into running mode with an arbitrary role allocation σ_{arb} then it will eventually come to a role allocation σ_G in which T will be achieved.

self-healing for a given set C of capabilities and a task T, if after failure/loss of any capability $c \in C$, then it will eventually come to a role allocation in which T will be achieved again.

self-adapting for a given set $T = \{t_i\}$ of tasks, if there is a change of tasks from t_1 to t_2 and $t_1, t_2 \in T$, then the system will eventually come to a role allocation in which the new task t_2 will be performed.

self-optimizing for a given task T and a given rating function $f: \sum \mapsto \mathbb{R}$ self-optimizing (where $\sum$ denotes the space of all eligible role allocations), if the system eventually comes to a role allocation σ in which $f(\sigma)$ is (locally) minimal over the set $\sum$.

ODP Runtime Environment (ORE)

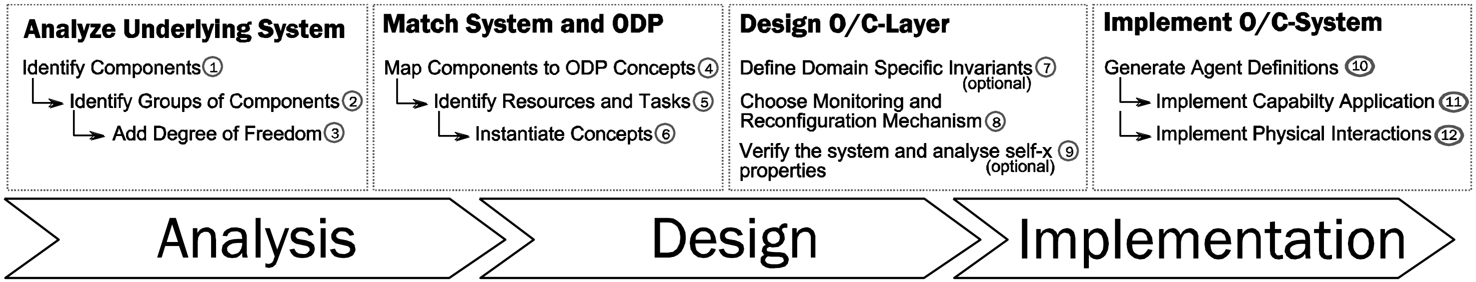
- Complete implementation and execution framework for the class of resource-flow systems
- Functionality common to all ODP agents is provided:
  - Communication
  - Role selection and execution
  - Reconfiguration
  - Data models and messages
- Domain and application-specific extension points
- Code transformation from models available
  - From domain model: agent definitions and capabilities
  - From instance model: bootstrapping scripts and initial configuration

Summary

Institute for Software & Systems Engineering

- For the class of resource-flow systems
  - Definition of how the application is an instance of the pattern
  - Code is generated (OC wrapper + observer/controller)
  - Class has an integrated invariant and behavioural corridor, which is verified
  - Can be attached to existing system components
  - Application-specific extensions need to be verified and implemented

➢ Generic top-down approach for this class of systems



Analyze Underlying System
Identify Components ①
↳ Identify Groups of Components ②
↳ Add Degree of Freedom ③

Match System and ODP
Map Components to ODP Concepts ④
↳ Identify Resources and Tasks ⑤
↳ Instantiate Concepts ⑥

Design O/C-Layer
Define Domain Specific Invariants ⑦ (optional)
Choose Monitoring and Reconfiguration Mechanism ⑧
Verify the system and analyse self-x properties ⑨ (optional)

Implement O/C-System
Generate Agent Definitions ⑩
↳ Implement Capabilty Application ⑪
↳ Implement Physical Interactions ⑫

Analysis → Design → Implementation

SPP-OC Phase III

- Adding self-adaptation
  - Removal/addition of agents during runtime
- Integration of self-optimization
  - Increase MTTF/MTBF by choosing roles with higher quality
  - Higher throughput
- Observer/Controller
  - Centralized ✓
  - Decentralized
    - With global knowledge at each agent ✓
    - Local monitoring ✓
    - Local reconfiguration ongoing
- Deadlock avoidance strategies
- Complete specification of software engineering process

Publications

Institute for Software & Systems

[ICSE09]A generic software framework for role-based Organic Computing systems

Florian Nafz, Frank Ortmeier, Hella Seebach, Jan-Philipp Steghöfer and Wolfgang Reif

SEAMS 2009: ICSE 2009 Workshop Software Engineering for Adaptive and Self-Managing Systems

[ATC09] A universal self-organization mechanism for role-based Organic Computing systems

Florian Nafz, Frank Ortmeier, Hella Seebach, Jan-Philipp Steghöfer and Wolfgang Reif

Proceedings of the Sixth International Conference on Autonomic and Trusted Computing (ATC-09)

[SASO08] A specification and construction paradigm for Organic Computing systems

M. Güdemann, F.Nafz, F.Ortmeier, H.Seebach and W.Reif

Proceedings of the Second IEEE International Conference on Self-Adaptive and Self-Organizing Systems (SASO 2008), IEEE Computer Society Press (2008)

[HINF08] Organic Computing for Health Care Systems

F. Nafz, F. Ortmeier, H. Seebach, and W. Reif

Proceedings of International Conference on Health Informatics

[ENASE08] Implementing Organic Computing Systems with Agentservice

Florian Nafz, Frank Ortmeier, Hella Seebach, Jan-Philipp Steghöfer and Wolfgang Reif

3rd International Conference on Evaluation of Novel Approaches to Software Engineering

[CEC07]Design and Construction of Organic Computing Systems

Hella Seebach, Frank Ortmeier, Wolfgang Reif

Proceedings of 2007 IEEE Congress on Evolutionary Computation, IEEE Computer Society Press 2007

[ISCAS07]Modeling of self-adaptive systems with SCADE

Matthias Güdemann, Andreas Angerer, Frank Ortmeier, Wolfgang Reif

Proceedings of 2007 IEEE International Symposium on Circuits and Systems, IEEE Computer Society Press 2007

[ISOLA06] Safety and Dependability Analysis of Self-Adaptive Systems

M. Güdemann, F. Ortmeier, W. Reif

Proceedings of ISoLA 2006, 2nd International Symposium on Leveraging Applications of Formal Methods, Verification and Validation, IEEE Computer Society Press 2006

[GI06]Towards Safe and Secure Organic Computing Applications

Matthias Güdemann, Florian Nafz, Wolfgang Reif and Hella Seebach

INFORMATIK 2006 – Informatik für Menschen, volume P-93 of GI-Edition – Lecture Notes in Informatics

[ATC06] Formal Modeling and Verification of Systems with Self-x Properties

Matthias Güdemann, Frank Ortmeier and Wolfgang Reif

Proceedings of the Third International Conference on Autonomic and Trusted Computing (ATC-06)