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

SAVE ORCA

Matthias Güdemann, Florian Nafz, **Frank Ortmeier**
Wolfgang Reif, Hella Seebach

Goal & Challenges

Goal:

(Top-Down) design **framework** for highly reliable and Organic Computing applications including

- Design and construction
- Formalization of self-X
- Methods and tools for formal analysis

Challenges:

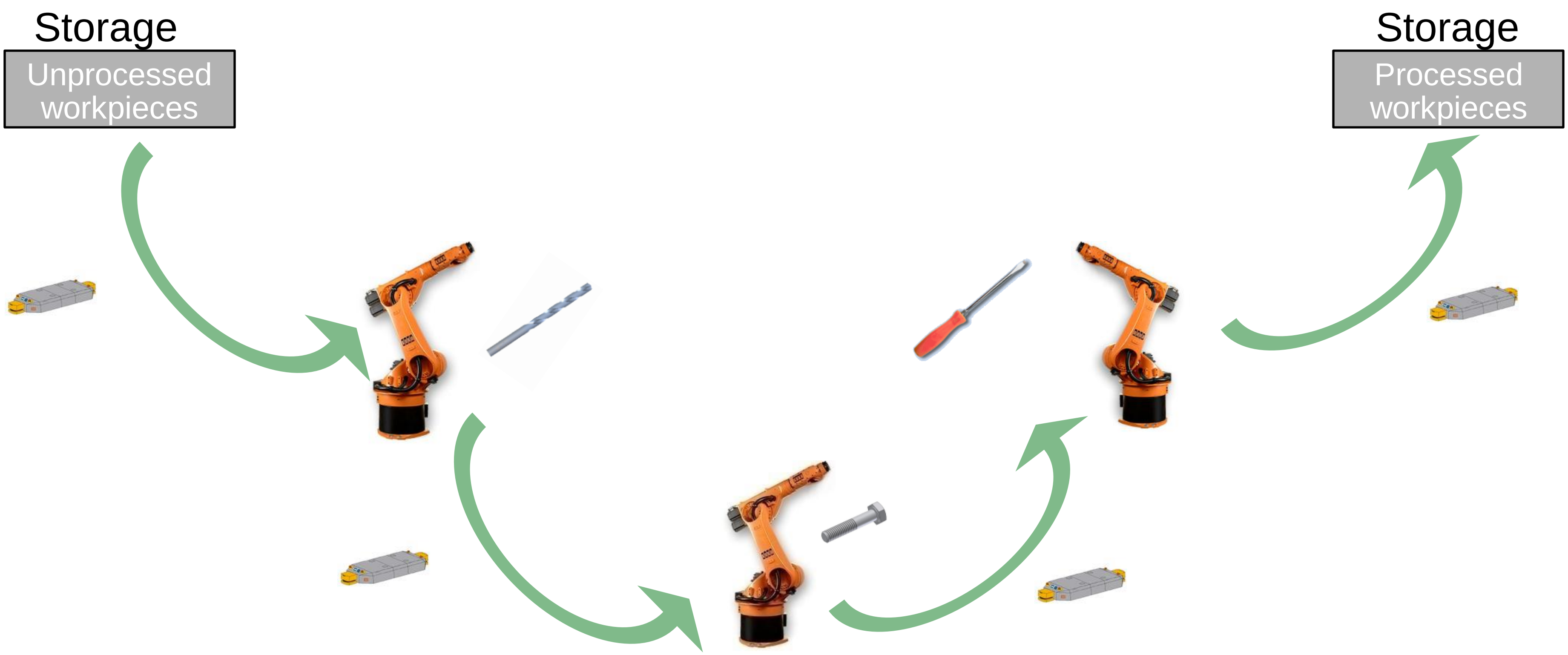
- Guidelines for design and construction of Organic Computing applications from traditional ones
- Process for engineering self-x properties into the application
- Provide tools to give correctness- and behavioral guarantees despite of self-organization
- Develop methods to (Provably) measure the degree of self-X

Target systems

- Embedded, software-intensive applications
- Example: adaptive production cell

Goal: Process workpieces in a given order!

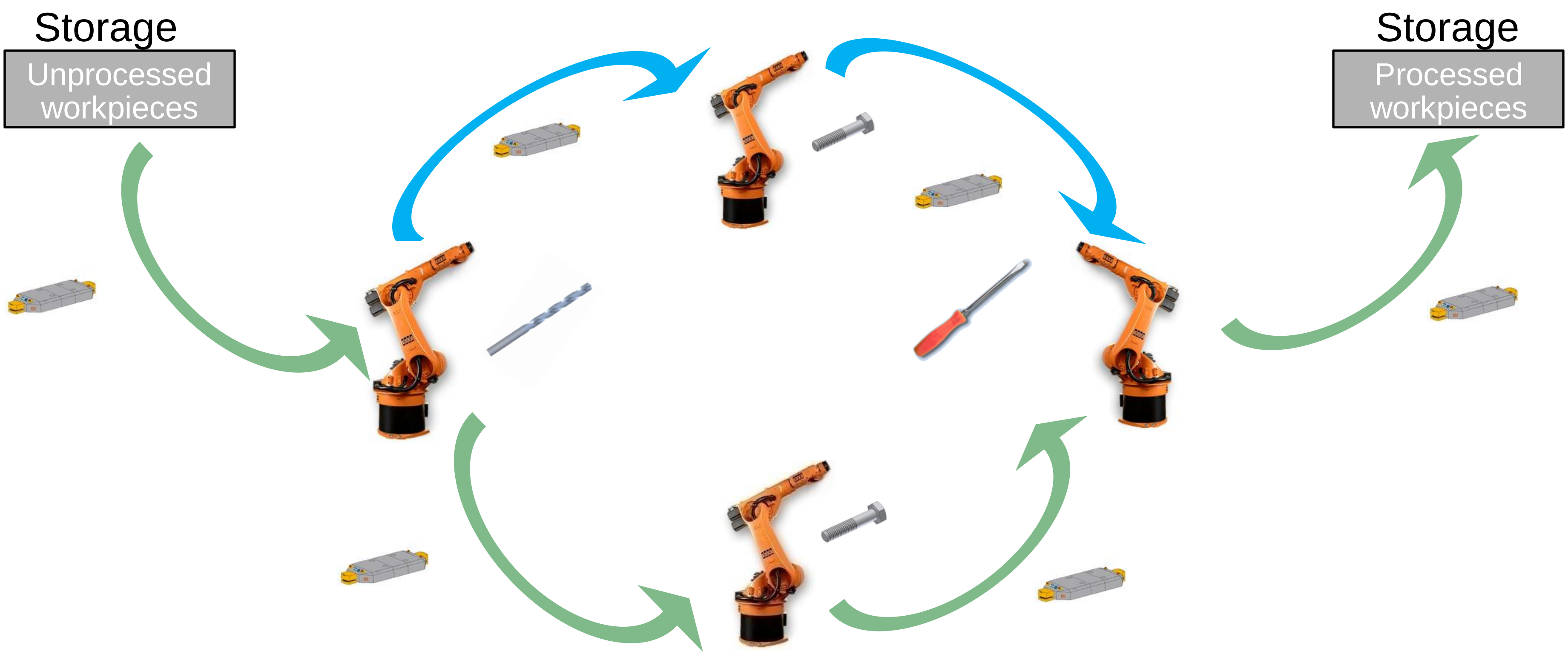
Desired properties of an OC system



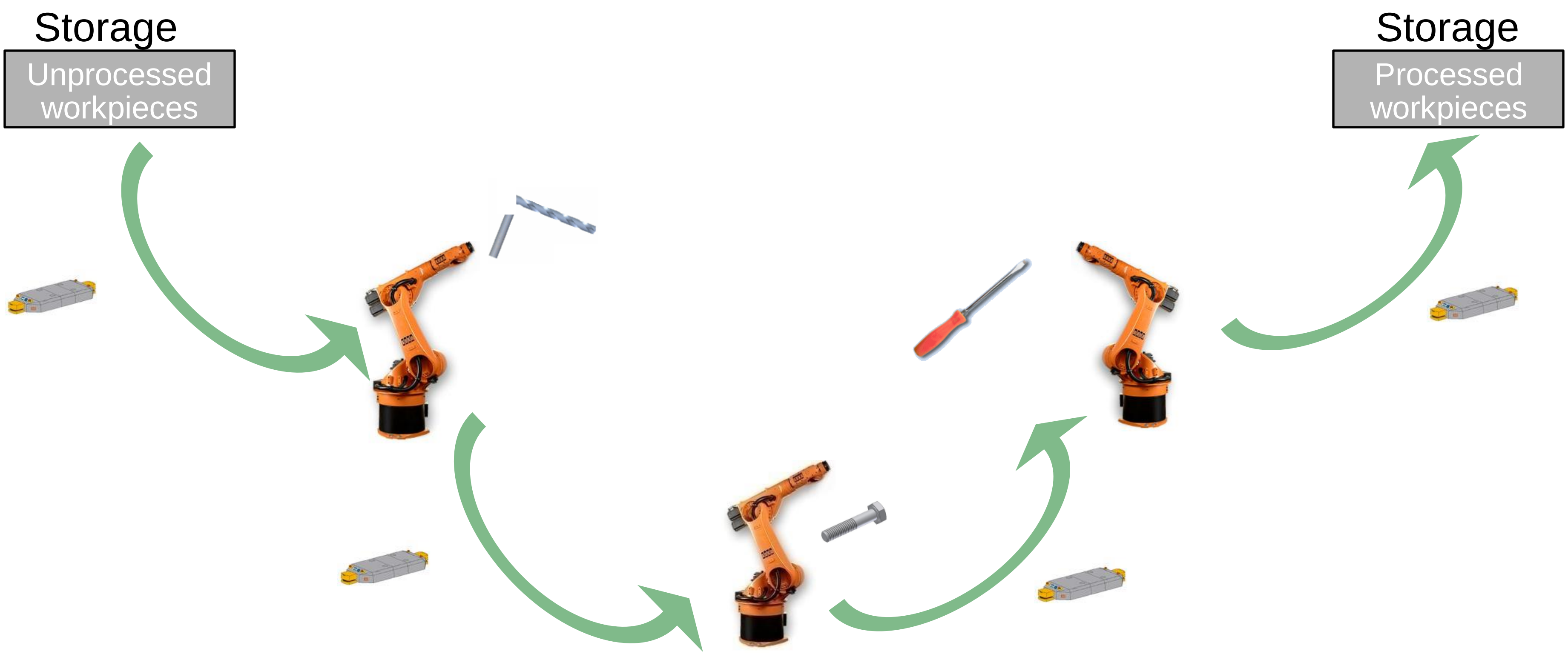
Desired properties of an OC system

Self-configuring

dynamically integrate new robots



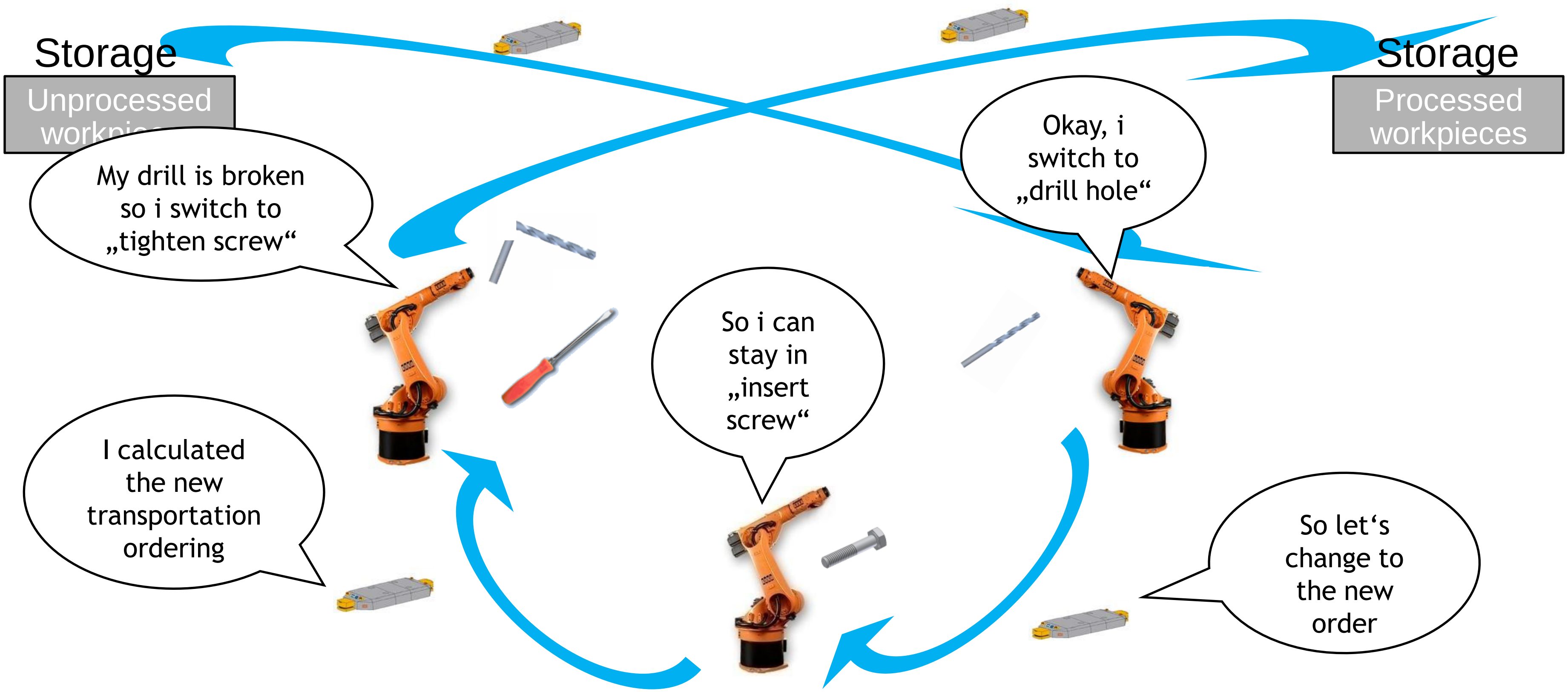
Desired properties of an OC system



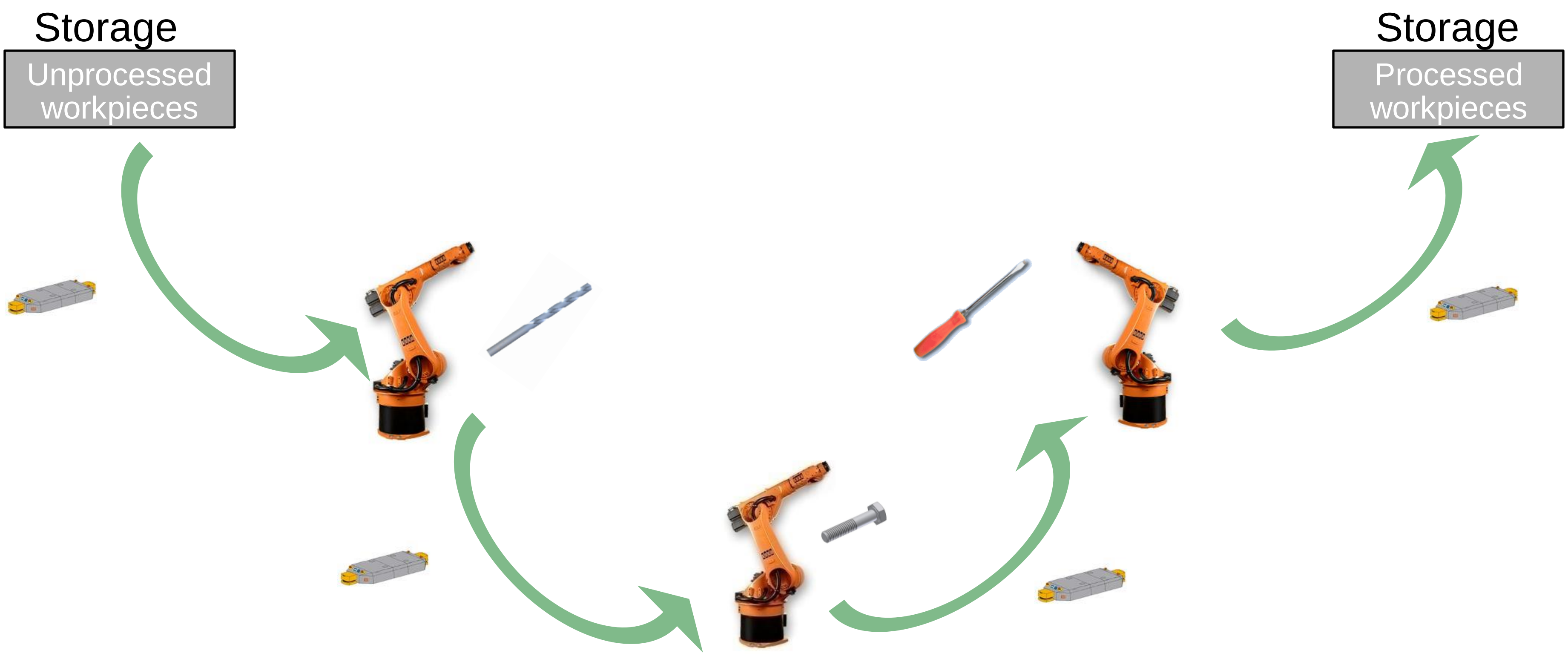
Desired properties of an OC system

Self-healing

Resistant to component failures



Desired properties of an OC system

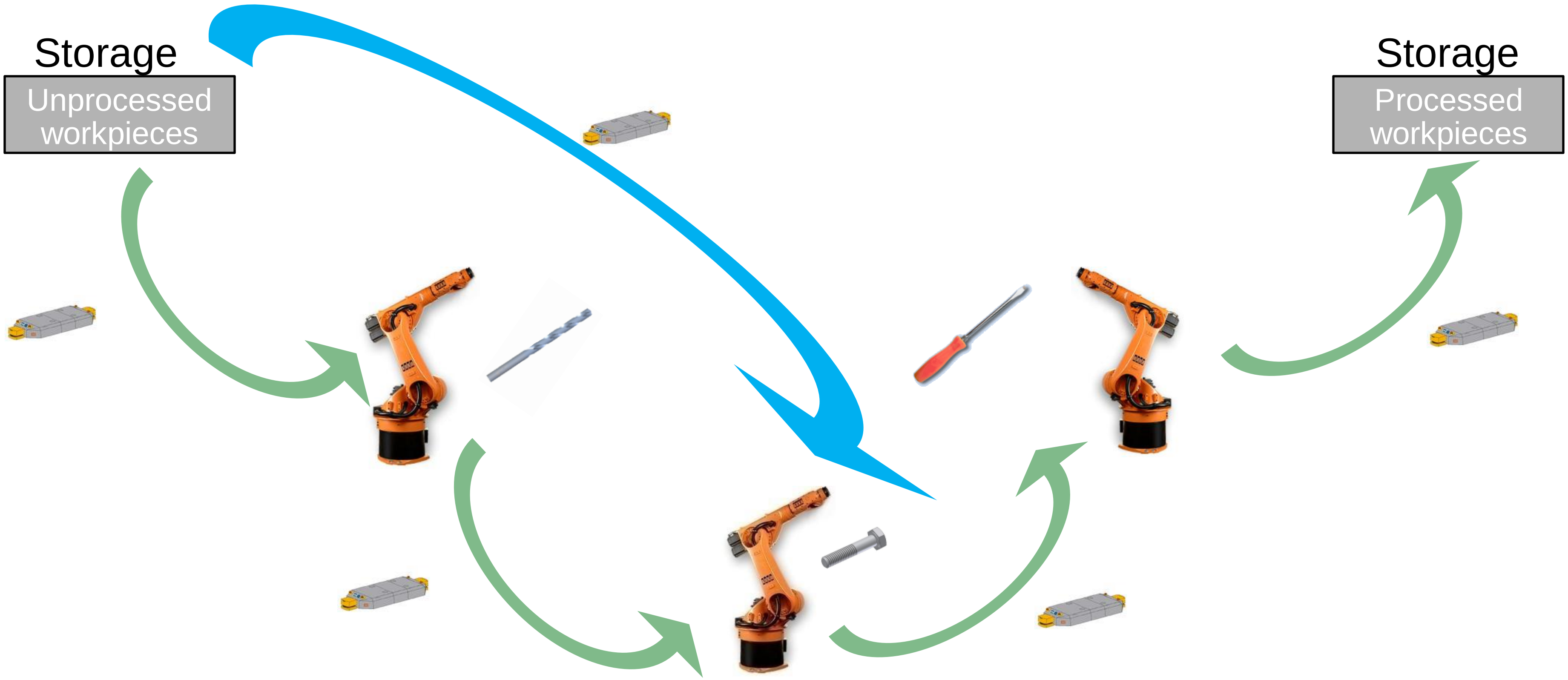


Desired properties of an OC system

Partly processed and unprocessed workpieces with RFID-Tags

Self-adapting

Adapting to new goals/tasks



Desired properties of an OC system

Self-optimizing

Trying to find "optimal" configurations

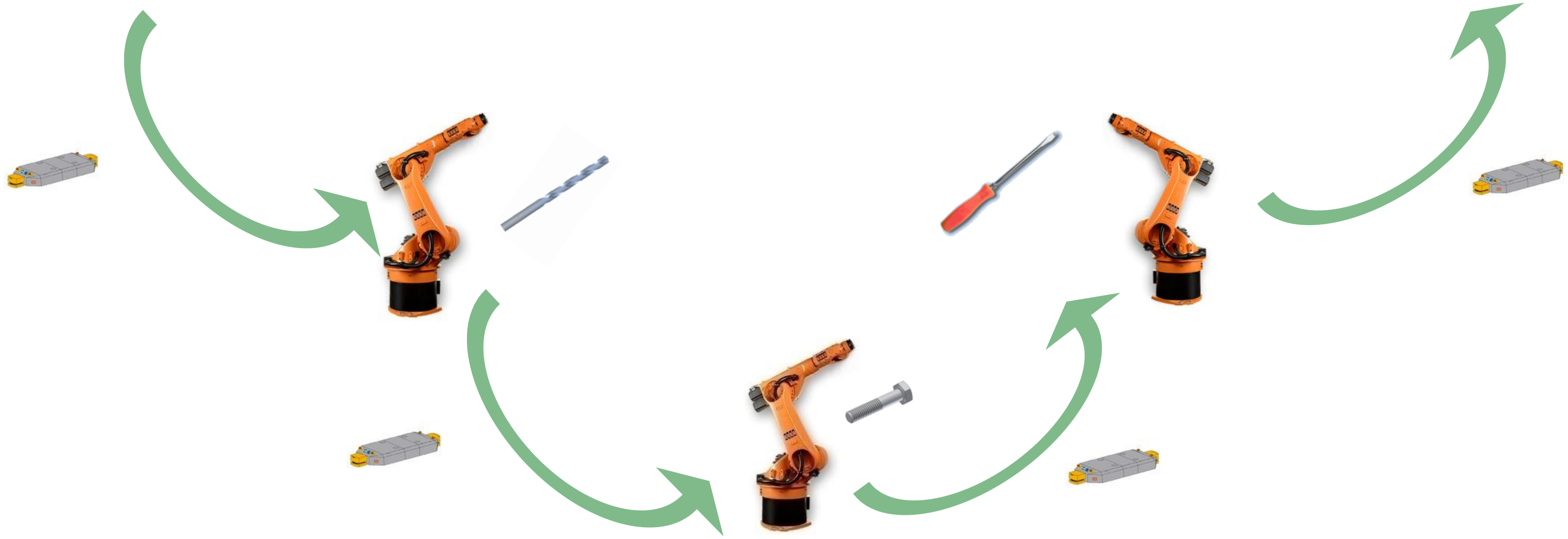
Storage

Unprocessed workpieces

Is this the best solution?

Storage

Processed workpieces



Achievements of the project after phase I:

1. **Design and modeling of Organic Computing systems**

2. Formal foundations for Organic Computing systems

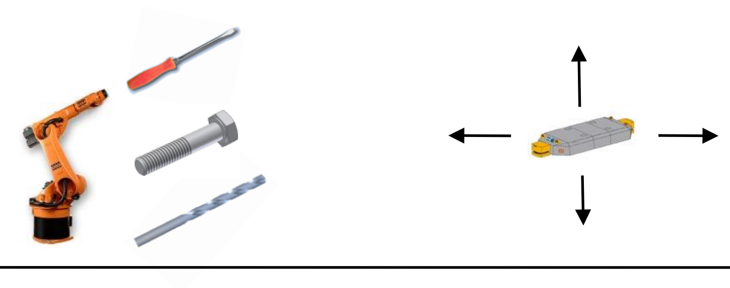
3. Process for construction of Organic Computing systems

4. Techniques for measuring the degree of self-healing

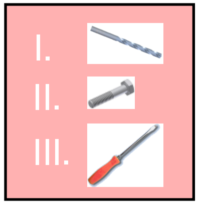
Design and modeling of OC systems

- System consists of agents

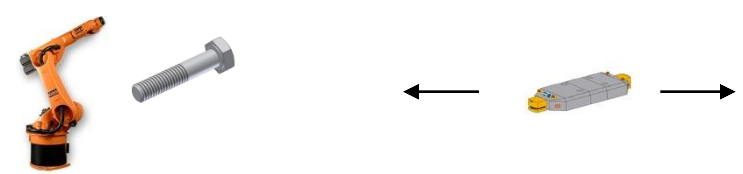
- Agents have capabilities



- Resources are to be processed with capabilities according to given tasks

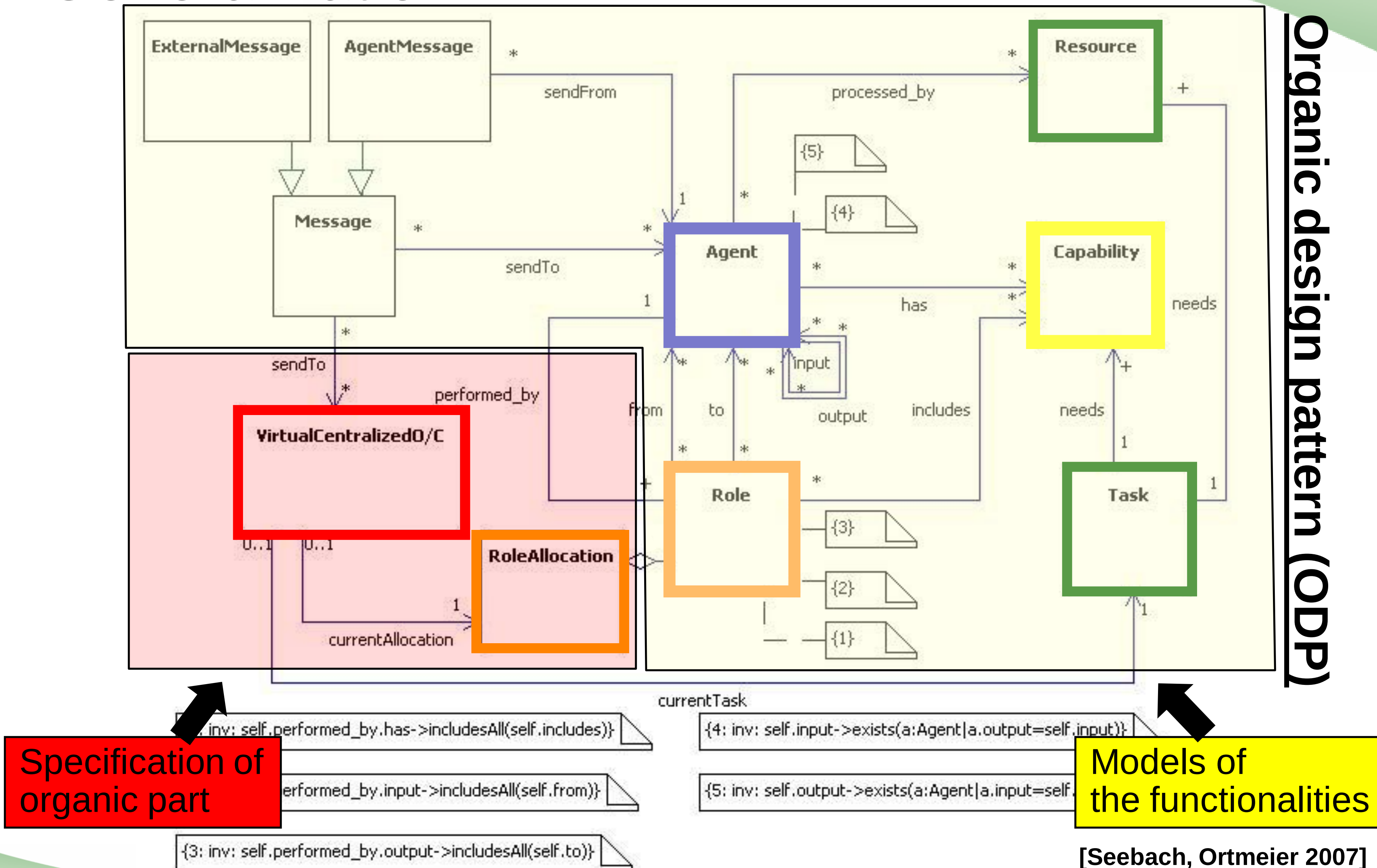


- Roles assign capabilities to agents



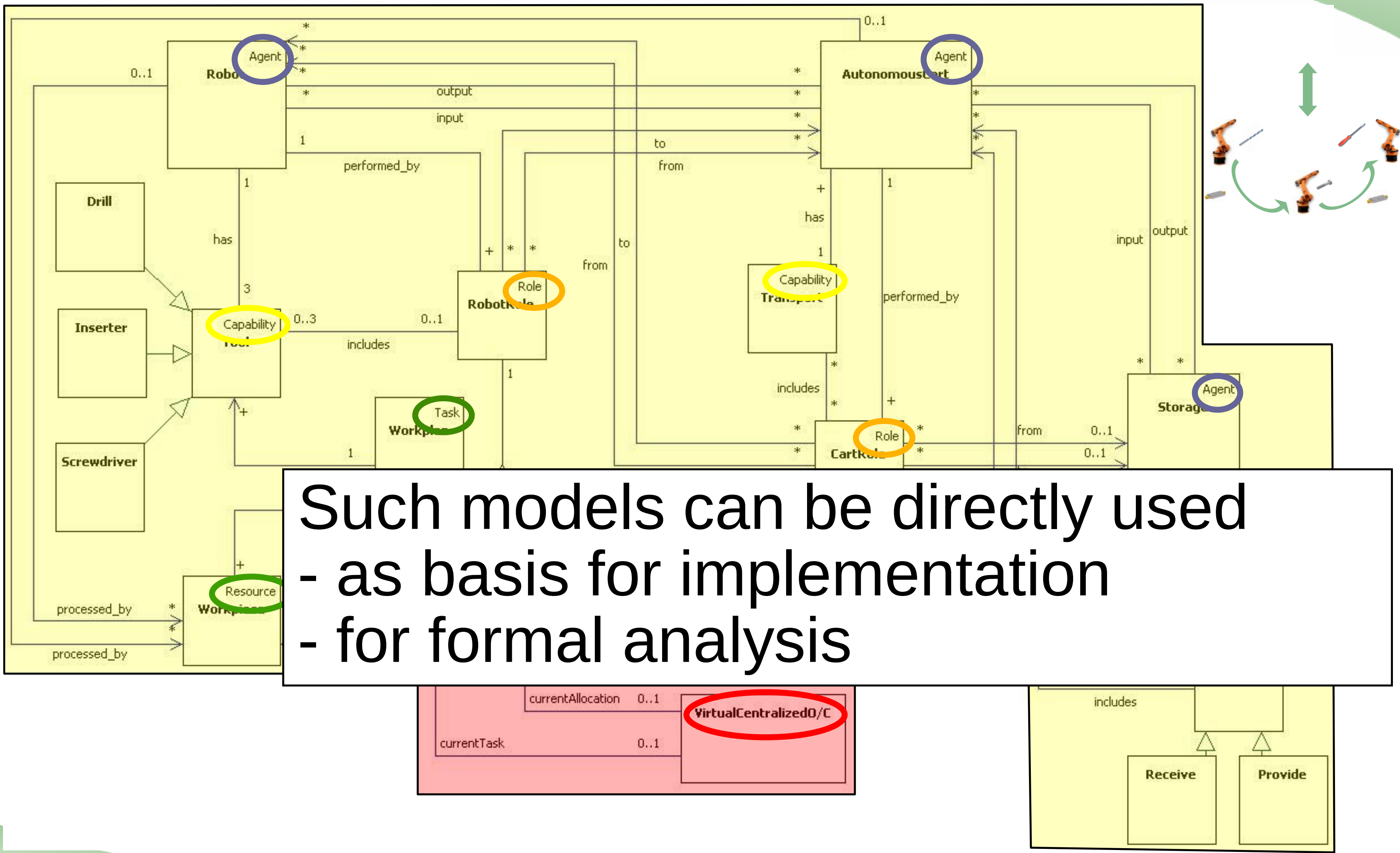
Role distribution is the core element of the organic part of the system

Generalization



[Seebach, Ortmeier 2007]

Example: adaptive production cell



Such models can be directly used
- as basis for implementation
- for formal analysis

Achievements of the project after phase I:

1. Design and modeling of Organic Computing systems

2. **Formal foundations for Organic Computing systems**

3. Process for construction of Organic Computing systems

4. Techniques for measuring the degree of self-healing

Observation

- Many OC systems can be divided into two parts:
  - One part which provides intended functionalities (e.g. collect and relay data, process workpieces, control traffic lights)

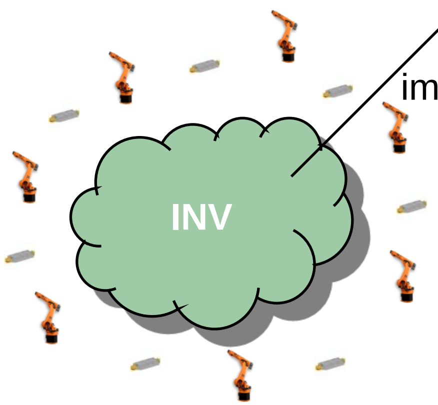
  and
  - One part which provides self-healing, self-adaptation, self-configuration and/or self-optimization capabilities (often implemented as an observer/controller architecture)

- Consequence:
  - This can help for formally describing and specifying Organic Computing systems!

Restore invariant approach

restore invariant approach

1. Design goals captured in INV



Expected guarantees

implies

3. Temporary violation of INV leads to self-organisation (reconfiguration) restoring INV

working

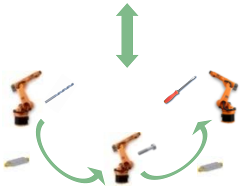
t

Reconfiguration can be understood as a restore invariant problem!

2. OC-system preserves INV as long as ever possible

4. Failure if restoring of INV is no longer possible

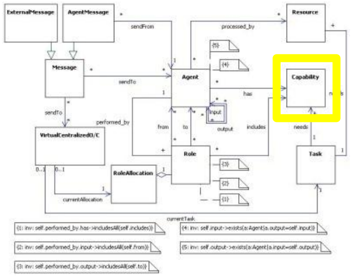
Application to case study



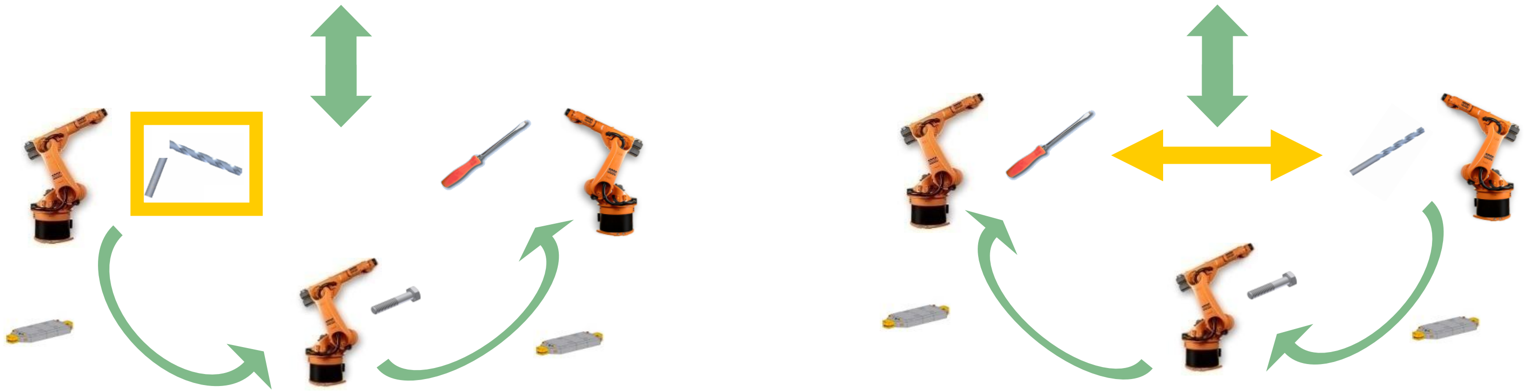
- Case study example
  - 3 robots, 3 different tools each, reconfigurable carts
- Invariants:
  - I_1: “Robot cell still has d-i-t capability”
  - I_2: “Carts are configured correctly”
- Expected properties to prove:
  - P_1: “Workpieces that leave the cell are processed with all tools”
  - P_2: “Workpieces are never processed in wrong sequence”
- Theorem: (can be proven automatically)
  - P_1 and P_2 are valid under the assumption of a correct reconfiguration algorithm that restores the invariants I_1 and I_2

Defining self-X

- ODP can be used for defining self-x properties
  - Idea:
    - Many self-x properties can be described within the language of ODP
  - Example: **self-healing**



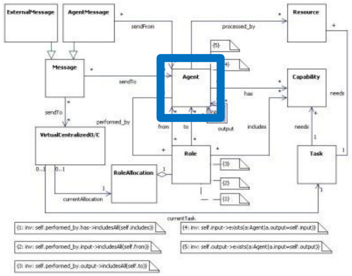
[Seebach, Ortmeier 2007]



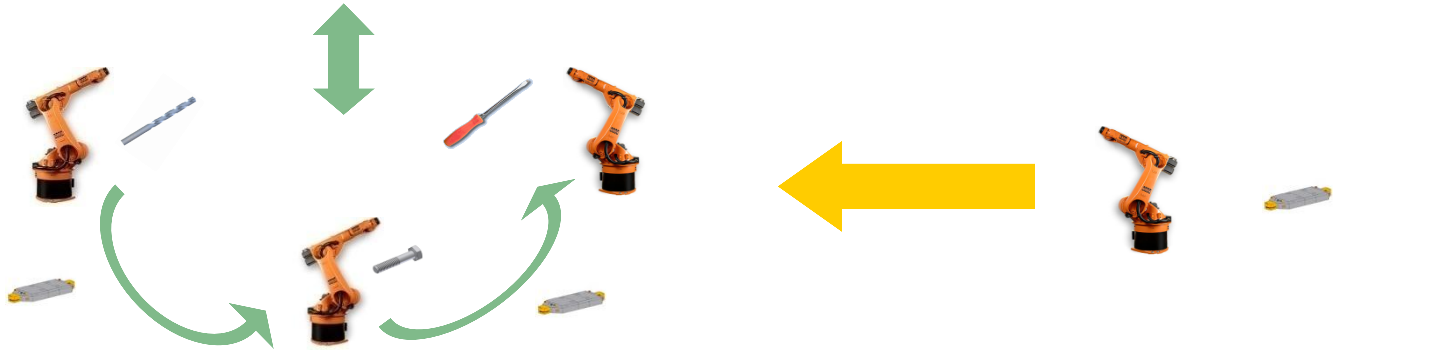
A system SYS, which is modeled as an instance of the organic design pattern is called self-healing for a given set *C* of capabilities and a goal G, if after failure/loss of any capability $c \in C$, then it will eventually come to a role allocation in which G will be achieved again.

Defining self-X

- ODP can be used for defining self-x properties
  - Idea:
    - Many self-x properties can be described within the language of ODP
  - Example: **self-configuring**



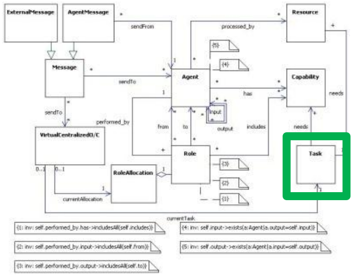
[Seebach, Ortmeier 2007]



A system SYS, which is modeled as an instance of the organic design pattern is called self-configuring for a goal G, 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 G will be achieved.

Defining self-X

- ODP can be used for defining self-x properties
  - Idea:
    - Many self-x properties can be described within the language of ODP
  - Example: **self-adapting**

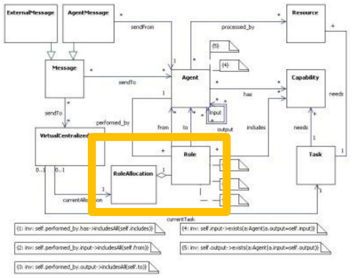


[Seebach, Ortmeier 2007]

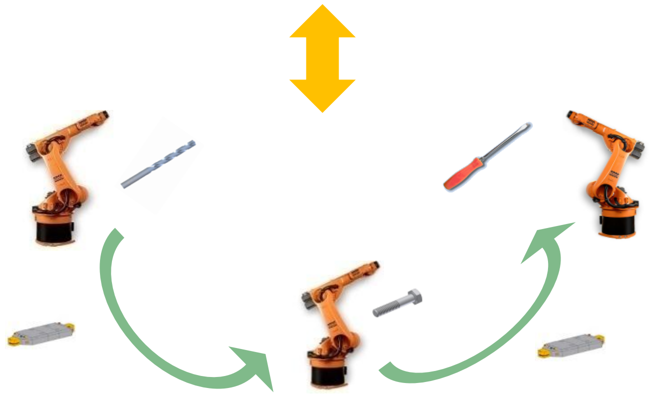
A system SYS, which is modeled as an instance of the organic design pattern is called self-adapting for a given set $T = \{t_i\}$ of tasks, if there is a change of tasks from t_1 to t_2 and $t_1, t_2 \in T$, then the system will eventually come to a role allocation in which the new task t_2 will be performed.

Defining self-X

- ODP can be used for defining self-x properties
  - Idea:
    - Many self-x properties can be described within the language of ODP
  - Example: **self-optimizing**



[Seebach, Ortmeier 2007]



A system SYS, which is modeled as an instance of the organic design pattern is called self-optimizing for a given goal G and a given rating function $f:\Sigma \mapsto \mathbb{R}$ (where Σ 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 Σ.

Achievements of the project after phase I:

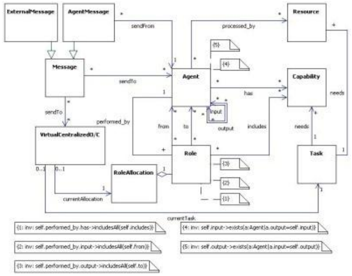
1. Design and modeling of Organic Computing systems

2. Formal foundations for Organic Computing systems

3. **Process for construction of Organic Computing systems**

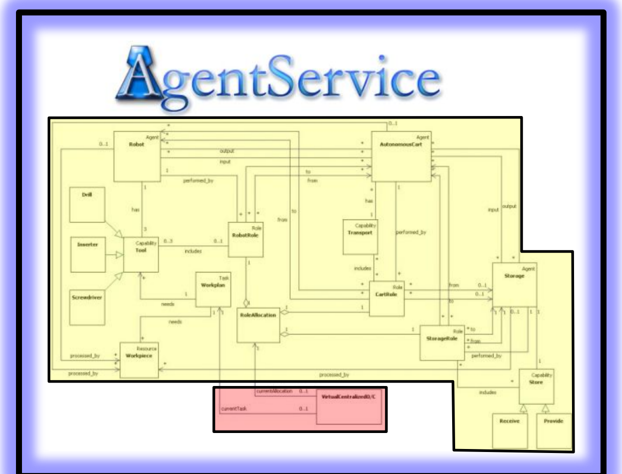
4. Techniques for measuring the degree of self-healing

3. Construction of OC systems



- ODP can be directly used for implementation
  - Idea:
    - Select a communication infrastructure
    - Wrap agents into this infrastructure
    - Define invariants which must be restored
    - Select an algorithm for invariant restoration
  - Example:
    - Communication infrastructure: AgentService
    - ODP entities are wrapped into Agent Service components
    - Hardware is simulated in Microsoft Robotics Studio
    - Algorithms tested:
      - selection of predefined configurations
      - random choice and result checking (work in progress)
      - SAT checking (work in progress)

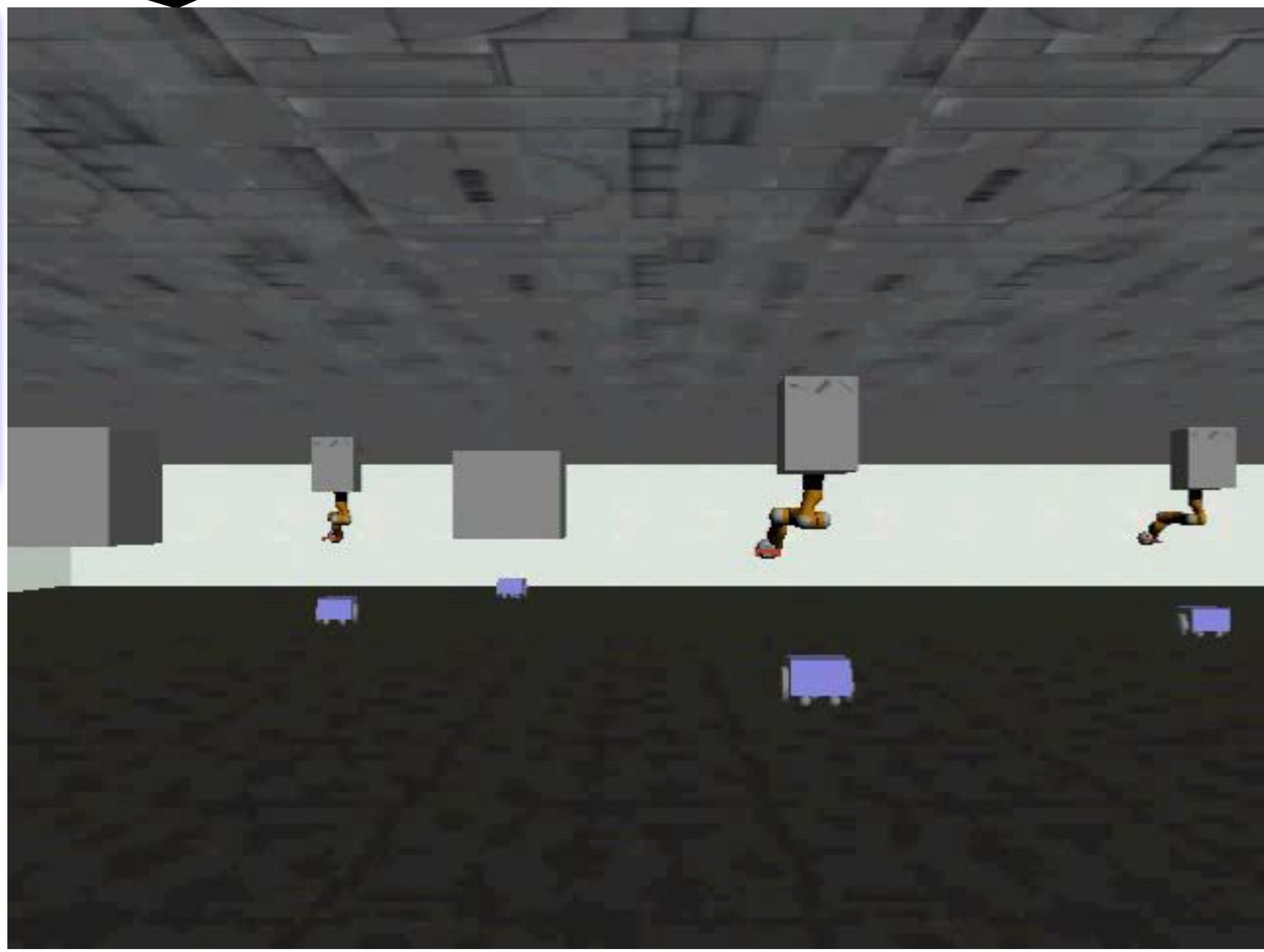
Example: adaptive production cell



+

Microsoft
Robotics Studio

Physical model of
- Robots
- Carts
- Workpieces

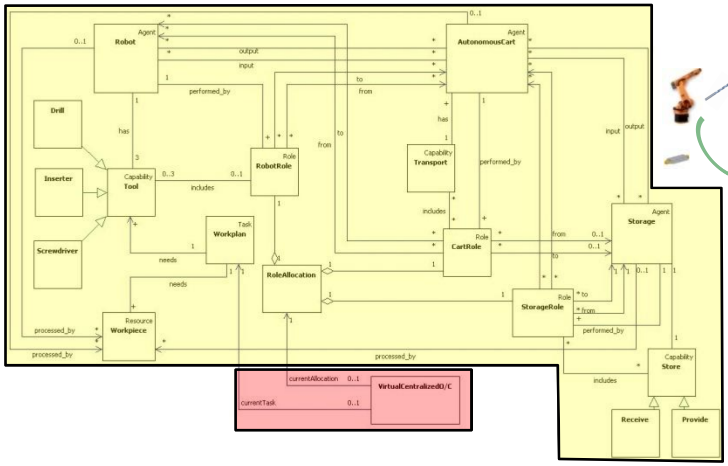


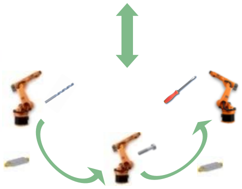
Achievements of the project after phase I:

1. Design and modeling of Organic Computing systems
2. Formal foundations for Organic Computing systems
3. Process for construction of Organic Computing systems
4. **Techniques for measuring the degree of self-healing**

4. Measuring the degree of failure tolerance

- Question:
  - How **self-healing** is this system?
  - How many failures can be tolerated?





- Self-healing:

A system SYS, which is modeled as an instance of the organic design pattern is called self-healing for a given set *C* of capabilities and a goal G, if after failure/loss of any capability $c \in C$, then it will eventually come to a role allocation in which G will be achieved again.

- Developed a theory: Adaptive DCCA

Adaptive DCCA

(DCCA = Deductive Cause Consequence Analysis)

Definition of *minimal critical set:*

Let Γ be a finite set of failure modes, then $\Delta \subset \Gamma$ is called *critical* w.r.t. a given hazard *H* iff

$$SYS^{+} \models \mathbf{E}(\neg (\Gamma \backslash \Delta)\ \textbf{until EG}\ (\neg (\Gamma \backslash \Delta) \wedge H))$$

Γ is called minimal critical if no teal subset is critical

[Güdemann, Ortmeier 2006]

- This means in natural language:

  "There exists a patch such, that eventually H becomes true forever and no failure modes of the set $\Gamma \backslash \Delta$ have appeared before the hazard has become permanent."

- **Theorem:** The set of minimal all minimal critical sets is complete.

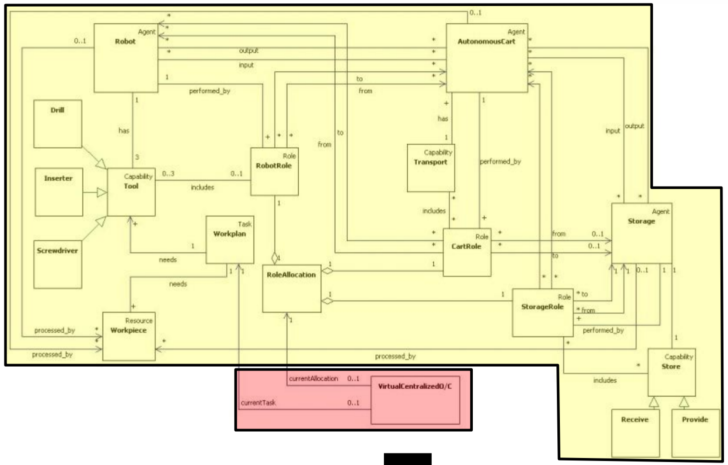
[Ortmeier 2006]

Adaptive DCCA (2)

(DCCA = Deductive Cause Consequence Analysis)

- Failure modes
  - Losses of capabilities
    e.g. drill breaks, arm gets stuck
- Hazard
  - Inability to fulfill a given goal

    e.g. workpieces can not be correctly processed



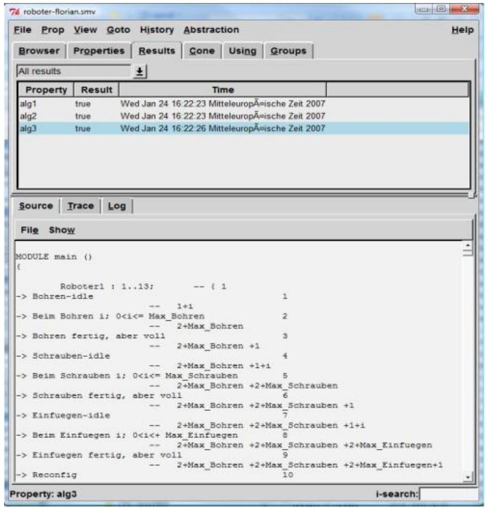
- Adaptive DCCA answers the question:

  “Which minimal combination of losses of capabilities

  can prohibit fulfillment of the goal permanently?”

  in other words:

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



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

Example: Adaptive production cell

- There exist 64 minimal critical sets i.e. combinations of losses of capabilities that can not be self-healed.
  - Tolerable failures: min # = 2 , max # = 7
  - n-point failures:
    - 22 3 – point failures
    - 42 5 – point failures

Verified

[Güdemann, Ortmeier 2006]

- In terms of self-healing:
  - The system can self-heal any single or dual loss of capabilities, it can self heal all but 22 combinations of three lost capabilities, …
- These results can be combined with stochastic data to compute MTF and MTBF rates.

Summary: Status of the project after phase I

- Achievements:
  - Design pattern for modeling Organic Computing systems has been developed
  - Formal foundations for describing Organic Computing systems and their properties have been developed
  - First steps towards a process for the construction of Organic Computing systems have been taken
  - Formal analysis techniques for measuring the degree of self-healing of an Organic Computing have been developed
- Publications:
  - 5 publications, 2 in progress, 2 reports
  - 3 Ph.D. projects started
- Next steps …

Objectives for phase 2:

- Goal 1: Integrate the ODP in an SW engineering process
  - Develop an engineering process to (a) build Organic Computing application and to (b) engineer self-x into existing applications
  - Embed the ODP in a multi-agent or service framework
    - First evaluations with Agent Service
    - Other candidates: JADE, MSRS
  - Evaluate/Integrate existing organic middlewares into the process

- Goal 2: Formal analysis methods
  - Generate invariants form OCL constraints and ODP
  - Develop techniques to formally verify/measure the degree of
    - self-configuration
    - self-adaptation
    - self-optimization

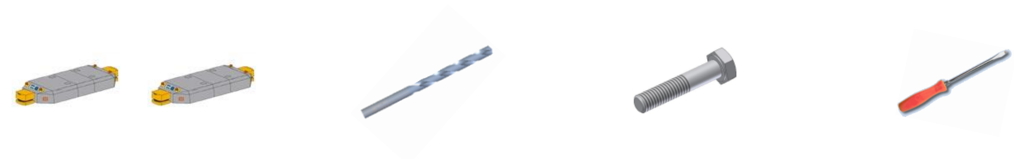
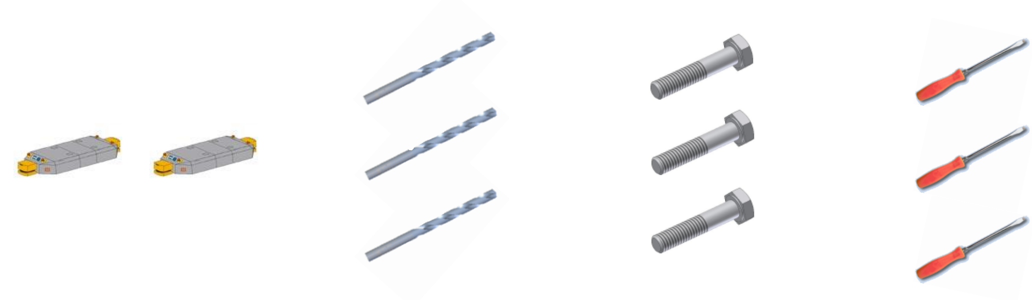
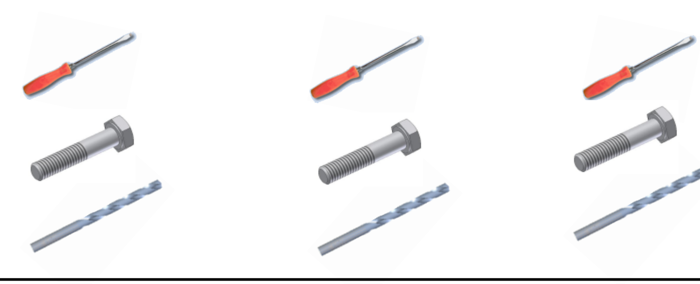
Objective for phase 2:

- Goal 3: Organic algorithms
  - Analyze existing organic algorithms for the class of invariants they can restore
    - Cooperation with OC-μ project appointed
  - Develop an organic algorithm which directly restores invariants
    - First steps/ideas with SAT checking
    - Possible next steps: constraint solvers

- Goal 4: Apply methods to other domains
  - Apply ODP to an autonomous SoC scenario; Cooperation with ASoC and OC-μ projects appointed
  - Analysis of different systems with ADCCA to measure their amount of self-healing
  - Comparison/Integration of ADCCA metric with/into generic metric frameworks

Thank you for your attention …

Comparison of design variants

• Optimistic:	
• Redundancy:	
• Self-x:	  + reconfiguration

Self-x pays off

Results for the example:

	# points of failures				Expected time until system failure
	single point	3 points	4 points	5 points	
Optimistic	8	-	-	-	11 days
Redundancy	5	3	-	-	59 days
Self-x	-	22	-	42	281 days