Towards a Requirements-Driven Design of Ensemble-Based Component Systems

http://d3s.mff.cuni.cz

Ilias Gerostathopoulos
iliasg@d3s.mff.cuni.cz

Dr. Tomas Bures
Dr. Petr Hnetynka

CHARLES UNIVERSITY IN PRAGUE
faculty of mathematics and physics
“To show that requirement-driven design approaches cannot be used out-of-the-box in the design of ensemble-based component systems”
“To show that requirement-driven design approaches cannot be used out-of-the-box in the design of ensemble-based component systems”
Target domain

Resilient Distributed Systems
Resilient Distributed Systems

Resilient systems: systems that remain *dependable*, even when facing *continuous change* and are *constantly adapting* to it

- Inherently distributed
- Self-aware services pursuing global goals
- Recurring/dynamic changes in the environment
- **Cloud setting: nodes in ad-hoc, dynamic networks**
RDS – an intelligent navigation case study

POI: Work
Time: 7AM-4PM

POI: Cinema
Time: 2PM-4PM

POI: Shopping
Time: 4PM-6PM

POI: Home
Time: 6:45PM

POI: Shopping
Time: 4PM-6PM

POI: Home
Time: 6:30PM
RDS – an intelligent navigation case study

Key Attributes & Challenges

- Distributed
- Open-ended
- Dynamic
- Autonomic, adaptive
- Local knowledge, belief
- Emergent behavior
Ensemble-Based Component Systems
Ensemble-Based Component Systems

- Integrate ideas & concepts from different areas
- Provide abstractions for engineering RDS

DEECo component model:
*Dependable Emergent Ensembles of Components*

- Brings separation of concerns to the extreme
Components are **strictly autonomous** units featuring cyclic periodic execution.
Components are **strictly autonomous** units featuring cyclic periodic execution.
Ensembles are (stateless) **interaction templates** that describe
a) **When** to communicate  b) **What** to communicate
Ensembles are (stateless) interaction templates that describe
a) **When** to communicate   b) **What** to communicate

EBCS design

Issues & Challenges
EBCS design

*Design* of EBCS: The process of identifying and specifying the desired components (knowledge & processes) and ensembles.

**How to design EBCS in a systematic and effective way?**

- How to come up with the desired components and ensembles, according to system requirements?
- How to achieve design validation and traceability?

**Problem**

**Challenge**

Classic requirement-driven design approaches do not suffice.

- Use-cases, user stories, ...
- Goal-oriented approaches
**EBCS design**

*Design* of EBCS: The process of identifying and specifying the desired components (knowledge & processes) and ensembles.

**How to design EBCS in a systematic and effective way?**
- How to come up with the desired components and ensembles, according to system requirements?
- How to achieve design validation and traceability?

**Classic requirement-driven design approaches do not suffice.**
- Use cases, user stories, ...
- Goal-oriented approaches

Describe “how“ instead of “what”
Inherently less adaptable/evolvable.
Require event anticipation.
EBCS design

**Design** of EBCS: The process of identifying and specifying the desired components (knowledge & processes) and ensembles.

**How to design EBCS in a systematic and effective way?**

- How to come up with the desired components and ensembles, according to system requirements?
- How to achieve design validation and traceability?

**Classic requirement-driven design approaches do not suffice.**

- Use cases, user stories, ...
- Goal-oriented approaches

Describe “how” instead of “what”
Inherently less adaptable/evolvable.
Require event anticipation.

Can the high-level concepts (goals, actors, ...) provide a remedy?
Requirements modeling – KAOS

Goal-oriented method for eliciting and analyzing the requirements of a software system.

- Goals have a prominent role
- Formal methods are used \textit{when} and \textit{where} needed

Goal model
Agent model
Object model
Operation model
Behavior model

KAOS specification

Applicability in design of EBCS:

+ captures the (intended) system behavior at a high level
+ allows for automatic formal reasoning
- does not align requirements with architecture
- is intended for requirements analysis and documentation, not system design
Requirements modeling – Tropos

Methodology for building agent-oriented software systems that uses the i* notation.

- Agent and related notions (goals, plans, intentions) have prominent role
- Focus on early stages of SWD and on the organizational context

Goal models ➔ Enhanced goal models

BDI architecture ← Actor-Agent mapping

Applicability in design of EBCS:
+ aligns the requirements phase with architecture and implementation phases
+ preserves a manageable set of concepts throughout the software development phases
- comprises a number of models with manual mappings between them
- does not cope with the emergent architecture requirement
Requirements modeling – Tropos

Methodology for building agent-oriented software systems that uses the i* notation.

- Agent and related notions (goals, plans, intentions) have prominent role
- Focus on early stages of SWD and on the organizational context

Goal models → Enhanced goal models

BDI architecture ← Actor-Agent mapping

Applicability in design of EBCS:

+ aligns the requirements phase with architecture and implementation phases
+ preserves a manageable set of concepts throughout the software development phases
- comprises a number of models with manual mappings between them
- does not cope with the emergent architecture requirement
EBCS design

Proposed Solution
**Operational normalcy & Invariant**

*Operational normalcy*: the property of being within the limits of normal operation

The valuation of the components’ knowledge evolves as a result of their autonomous behavior and of knowledge exchange

*Invariant*: condition on the knowledge valuation of a set of components that captures the operational normalcy to be maintained by the system-to-be
Invariant Refinement Method

**Idea:** Iteratively refine and concretize invariants at the system level up to the point where they can be mapped to:

- Component processes (process invariants)
- Knowledge exchange (exchange invariants)

---

**System Level**

**Ensemble Level**

**Component Level**

**Implementation**

Refinement step

Exchange invariants

Ensembles’ specification

Process invariants

Components’ specification
IRM refinement tree

1. All Vehicles meet their calendar
2. Up-to-date V::plan, w.r.t. information from P, reflecting V::calendar is available
3. V::position is in alignment with the V::plan
4. An up-to-date plan can always be followed by the vehicle and it always schedules reaching the destination in time
5. Up-to-date V::plan, w.r.t. P::availability and V::planFeasibility, reflecting V::calendar is available
6. V::planFeasibility w.r.t. V::energy and V::traffic is determined
7. V::availabilityList - V's belief over P::availability of trip-relevant parking lots - is up-to-date
8. Up-to-date V::plan, w.r.t. V::availabilityList and V::planFeasibility, reflecting V::calendar is available
9. V::energy and V::traffic are monitored
10. V::planFeasibility w.r.t. the monitored V::energy and V::traffic is determined
IRM refinement tree

## Conclusion

<table>
<thead>
<tr>
<th></th>
<th>KAOS</th>
<th>TROPOS</th>
<th>IRM</th>
</tr>
</thead>
<tbody>
<tr>
<td>High-level modeling</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Alignment of requirements with architecture</td>
<td>✗</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Dynamic, emergent architecture support</td>
<td>✗</td>
<td>✗</td>
<td>✔</td>
</tr>
</tbody>
</table>

**Novelty:** “IRM allows reasoning along the line of what needs to *hold* in the system at every time instant (invariants), instead of what needs to *be performed* (actions) or *achieved* (goals)”
Future Work

IRM designer: model-based tool implementing the IRM features

Support for dynamic adaptation based on alternative refinements (when different assumptions hold)

Design rules and constraints for guiding the developer in design decisions and structural validation

Formal structure and semantics of IRM $\Rightarrow$ system analysis
Thank you!

Questions?