An extensible autonomous reconfiguration framework for complex component-based embedded systems (ICAC’15)

Johannes Schlatow, Mischa Möstl, Rolf Ernst

{schlatow|moestl|ernst}@ida.ing.tu-bs.de
Motivation

Scenario:
- Single platform with multiple owners
- Extra-functional requirements (safety, security, timing, ...)
  e.g. automotive applications: ISO26262 → freedom from interference

Goals:
- Incremental changes (updates, extensions, customisation)
- Long-term evolution

Lab-centric integration → autonomous in-field “update”
Problem Statement

Incremental integration

Given: current system configuration, set of components to be deployed

Wanted: new and valid system configuration

- A valid configuration must fulfil all requirements.
- (Extra-)functional requirements are specified by component contracts.
- Support additional optimisation objectives (e.g. minimum changes).
- Reuse established models and analyses where possible.

→ Find valid system reconfigurations for incremental change requests.
→ Integrate established formal analyses for (efficient) verification.
Related Work

Related Work

- [Dearle et al. 2004]: constraint-based deployment of distributed applications (mapping and interconnect)
- [Jenson et al. 2010]: SAT encoding for component-level dependencies
- [Panunzio, Vardanega 2014], [Gleirscher et al. 2007]: lab-based design processes covering multiple views
- [Sojka, Hanzalek 2009], [Stein et al. 2011]: single-view contract-based self-adaptation/admission control

Our Contributions

1. Boolean satisfiability (SAT) approach for service-level functional dependencies in component-based systems.
2. Extensible framework for formal (contract-based) admission control of incremental changes w.r.t. multiple design views.
Component Model

Targeted run-time environments (e.g. Genode OS Framework):
- Component-based
- Service-oriented interfaces
- Explicit connections $\rightarrow$ principle of least privilege

$C : \text{set of components}$
$R : \text{set of requirements}$
$P : \text{set of capabilities}$

Relations:
- satisfiedBy $\subseteq R \times P$
- provides $\subseteq C \times P$
- etc...

$(+ \text{optional requirements & cardinality constraints})$

$\rightarrow$ extracted from component contracts
Multi-Change Controller

- Implements contract-based admission control / contract negotiation (multiple applications, multiple aspects)
- Separates (extra-)functional aspects (→ design views)
- Central constraint solver
- Translates model-specific constraints ($\mu$) into configuration constraints ($\sigma$)
Constraint classification:
1) **Necessary and sufficient**: optimal, possibly extensive or not even possible
2) **Only sufficient**: over restrictive → avoid
3) **Only necessary**: under restrictive → requires sanity check
4) **Separation constraints**: incrementally separate unsound solutions

Iterative approach:
- **ENC**: constraint encoding 1)-3)
- **SLV**: central constraint solver
- **CHK**: view-specific analysis engines
  → sanity check + constraints (4)
- **OPT**: optional optimisation engines
  → design-space exploration
Component View

- Solves (service-level) functional dependencies between components.
- Finds functionally valid component compositions.
- Provides a basis for most extra-functional views (e.g. timing).

Requirements

1) A component must be instantiated if explicitly queried.
2) A component must be instantiated if one of its capabilities is used.
3) ...

Encoded as Boolean constraints (SAT), e.g. 2):

\[
\forall (j, k) \in S: \neg s_{jk} \lor p_k \\
\forall (i, k) \in P: \neg p_k \lor c_i
\]

- \( P \): set of capabilities
- \( S \): set of connections
- \( c_i = True \iff \text{component i must be instantiated} \)
- \( p_k = True \iff \text{capability k must be provided} \)
- \( s_{jk} = True \iff r_j \text{ is connected to } p_k \)
Conclusion and Future Work

Conclusion

- Autonomous reconfiguration of component-based systems
- Admission control of in-field updates
- Constraint satisfaction with multiple design views
- SAT approach for service-level functional dependencies

Future Work

- Add views for timing, mapping, functional correctness, safety
- Performance evaluation on realistic, more complex use-cases

Thank you for your attention! Questions?
References

[Dearle et al. 2004]: A. Dearle, G. N. C. Kirby, and A. J. McCarthy

[Jenson et al. 2010]: G. Jenson, J. Dietrich, and H. W. Guesgen

[Panunzio, Vardanega 2014]: M. Panunzio and T. Vardanega

[Gleirscher et al. 2007]: M. Gleirscher, D. Ratiu, and B. Schatz

[Sojka, Hanzalek 2009]: M. Sojka and Z. Hanzalek

[Stein et al. 2011]: S. Stein, M. Neukirchner, and R. Ernst
Component View – Requirements

Requirements

1) A component must be instantiated if explicitly queried.

2) All requirements of an instantiated component must be connected to a capability that satisfies the requirement.

3) All invoked requirements of an instantiated component must be connected to a capability that satisfies the requirement if and only if the invoking capability is connected.

4) A component must be instantiated if one of its capabilities is used.

5) A component should not be present if neither explicitly queried nor providing a required capability.

6) A capability $p$ can be connected to at most $\nu(p)$ requirements.

7) Already deployed components and their connections shall not be modified during reconfiguration.

→ SAT (boolean satisfiability) encoding
Component View – SAT encoding

**Variables**

∀i ∈ C: qi = True ⇔ ci was queried
∀i ∈ C: ci = True ⇔ ci must be instantiated
∀j ∈ R: rj = True ⇔ rj must be satisfied
∀k ∈ P: pk = True ⇔ pk must be provided
∀(j, k) ∈ S: sjk = True ⇔ rj is connected to pk

**Constraints (e.g.):**

∀(j, k) ∈ S: ¬sjk ∨ pk
∀(i, k) ∈ P: ¬pk ∨ ci

Requirement 2: Component is instantiated if one of its capabilities is used.