Component-Interaction Automata as a Verification-Oriented Component-Based System Specification

L. Brim, I. Černá, P. Vařeková, and B. Zimmerova

Masaryk University in Brno
Czech Republic

SAVCBS’05
1 Specification of CBSs
   - Specification languages
   - Good specification language

2 Component-Interaction automata
   - Component-Interaction automata language
   - Definitions

3 Example - Database system
   - Specification
   - Interaction properties
   - Verification

4 Conclusion and future work
Motivation

- Component-based systems (CBSs)
- Interaction properties of CBSs
- Verification of interaction properties

Specification languages:
- Architecture description languages (ADLs)
- Automata-based languages
Specification languages - ADLs

- **Wright** (R. J. Allen, 1997)
- **Rapide** (D. C. Luckham, 1996)
- **SOFA** (F. Plasil, S. Visnovsky, 2002)

  + Hierarchical component architecture
  + Supported by tools
  + User friendly

  – Verification of a small fixed set of properties
Specification languages - Automata-based languages 1/2

- **I/O automata** (N. A. Lynch, M. R. Tuttle, 1987)
- **Interface automata** (L. de Alfaro, T. A. Henzinger, 2001)
- **Team automata** (C. Ellis, 1997)

  + Verification of temporal properties
  + Supported by automated verification tools

  - Specification of hierarchical component architecture
Specification languages - Automata-based languages 2/2

- **I/O automata**
  - one type of communication
  - inflexible composition
  - not all automata can be composed

- **Interface automata**
  - one type of communication
  - inflexible composition
  - not all automata can be composed

- **Team automata**
  - composition does not preserve all important information
  - not all automata can be composed
Good specification language

- Automata-based and formal verification algorithms
- ADL description as an input suitable for specification
- Composition should preserve:
  - hierarchy of components
  - which components synchronize on a particular action
  - chosen properties of partial automata
- Flexible composition according to:
  - type of communication
  - architecture description
Component-Interaction automata

- Inspired by Interface automata, I/O automata, and Team automata
- Three types of actions (input, output, internal)
  general used concept
- CCS like synchronization
  inspired by Interface automata
- Flexible composition
  inspired by Team automata
- Close to architecture description languages
  can be semi-automatically transformed into CI automata
- Preserving information
  hierarchy, participants of synchronization
- Close to Büchi automata
  infinite traces
Component-Interaction automata

- States (initial)
- Labels (actions)
- Transitions
- Hierarchy

Hierarchy: ((1),(2))
Composition of Component-Interaction automata 1/3

- Complete transition space
- Transition set of composed automata \( \subseteq \) complete transition space
- Transition set is determined by architecture and other characteristics of the system
Composition of Component-Interaction automata 2/3

Hierarchy: (1)

Hierarchy: (2)

Hierarchy: ((1),(2))
Composition of Component-Interaction automata 3/3

Hierarchy: (1)

Hierarchy: (2)

Hierarchy: ((1),(2))
**Definitions**

**Executions, traces**

![Diagram showing state transitions]

**Execution:**

\[
((q_0, q_0), (1, c, -), (q_1, q_0), (-, a, 1))^* 
\]

**Closed execution:**

\[
((q_0, q_0), (1, b, 2), (q_1, q_1), (2, a, 1))^* 
\]

**Trace:**

\[
((1, c, -), (-, a, 1))^* 
\]
Example - database system
Composed component DB (type Database)
Component DA (type DataAcces)

interface IDatabaseServer {
    void Insert(in string key, in string data);
    void Delete(in string key);
    void Query(in string query, out string data);
    void Done();
};

frame DataAccess {
    provides:
        IDatabaseServer data;
    protocol:
        ((?data.Insert↑ + ?data.Delete↑ + ?data.Query↑);
        !data.Done↓)*
};
Component DM (type DatabaseManager)

frame DatabaseManager {
    ...
    protocol:
        (((?db.Insert↑; !data.Insert↑; !log.LogEvent↑)+
           (?db.Delete↑; !data.Delete↑; !log.LogEvent↑)+
           (?db.Query↑; !data.Query↑ )); ?data.Done↓; !db.Done↓)*
};
Composed component DB (type Database)

Hierarchy: ((1),(2))
Specification

Example - Database system

Outline
Specification of CBSs
Component-Interaction automata

Conclusion and future work
Component C1 (type Client)

frame ClientApp {
    requires:
        IDatabaseServer db;
    protocol:
        ((!db.Insert↑ + !db.Delete↑ + !db.Query↑);
         ?db.Done↓)*
};

Hierarchy: (3)
Component LM (type LoggingManager)

interface ILogging {
    void LogEvent(in string event, in string user);
};

frame LoggingManager {
    provides:
        ILogging log;
    protocol:
        (?log.LogEvent↑)*
};

Hierarchy: (4)
Specifying of CBSs

Component-Interaction automata

Example - Database system

Conclusion and future work

---

**Specification**

Composition of previous components

**Notation:** \( \forall \in \{0, 1\}, j \in \{0, 1, \ldots, 7\} : (q_i, q_j, q_k, q_l) = q_{ijkl} \)

![Diagram showing the composition of previous components with transitions and states labeled with operations like Insert, LogEvent, Query, Delete, and Done.](attachment:diagram.png)

Hierarchy: \(((1),(2)), (3),(4))\)
Every action *Insert* sent by the client is followed by action *Done* received by the client.

\[ G((3, \text{Insert}, 2) \Rightarrow F(3, \text{Done}, 2)) \]

true
Properties 2/2

Every database action (Insert, Delete, Query) sent by the client is logged.

\[ G(((3, \text{Insert}, 2) \lor (3, \text{Delete}, 2) \lor (3, \text{Query}, 2)) \Rightarrow ((\neg(2, \text{Done}, 3)) \cup (2, \text{LogEvent}, 4))) \]

false

\(((3, \text{Query}, 2), (2, \text{Query}, 1), (1, \text{Done}, 2), (2, \text{Done}, 3))\)
Properties - verification

- Verification of given properties by model checking tool DiVinE
- Translation of CI automata to DiVinE input language
- Other verification tools
Conclusion

New specification language Component-Interaction automata

- Automata-based
- Flexible composition
- Structured labels
- Hierarchy
- Verification of interaction properties
Future work

- Practical
  - Automatic transformation from ADL specification (SOFA, ... ) to CI automata
  - Automatic transformation from CI automata specification to input languages of model checking tools (DiVinE, ... )

- Theoretical
  - Theory of CI automata
    - behavioral equivalences, ...