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

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)
- Darwin/Tracta (J. Magee, N. Dulay, S. Eisenbach, J. Kramer, 1995 / J. Magee, J. Kramer, D. Giannakopoulou, 1999)
- 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

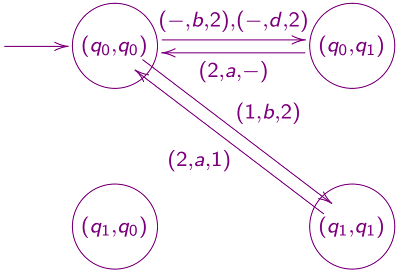
  suitable for specification
- Composition should preserve:
  - hierarchy of components
  - which components synchronize on a particular action
  - choosen properties of partial automata
- Flexible composition according to:
  - type of communication
  - architecture description

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

$$\begin{tikzcd}
{} \arrow[r] & q_0 \arrow[r, shift left, "{(1,b,-),\ (1,c,-)}"] & q_1 \arrow[l, shift left, "{(-,a,1)}"]
\end{tikzcd}$$

Hierarchy: (1)

$$\begin{tikzcd}
{} \arrow[r] & q_0 \arrow[r, shift left, "{(-,b,2),\ (-,d,2)}"] & q_1 \arrow[l, shift left, "{(2,a,-)}"]
\end{tikzcd}$$

Hierarchy: (2)

$$\begin{tikzcd}
{} \arrow[r] & (q_0,q_0) \arrow[r, shift left, "{(-,b,2),(-,d,2)}"] \arrow[d, shift left, "{(1,b,-),(1,c,-)}"'] \arrow[dr, shift left, "{(1,b,2)}"] & (q_0,q_1) \arrow[l, shift left, "{(2,a,-)}"] \arrow[d, shift left, "{(1,b,-),(1,c,-)}"] \\
& (q_1,q_0) \arrow[u, shift left, "{(-,a,1)}"'] \arrow[r, shift left, "{(-,b,2),(-,d,2)}"] & (q_1,q_1) \arrow[l, shift left, "{(2,a,-)}"] \arrow[u, shift left, "{(-,a,1)}"] \arrow[ul, shift left, "{(2,a,1)}"]
\end{tikzcd}$$

Hierarchy: ((1),(2))

Composition of Component-Interaction automata 3/3

$$\begin{tikzcd}
{} \arrow[r] & q_0 \arrow[r, shift left, "{(1,b,-),\ (1,c,-)}"] & q_1 \arrow[l, shift left, "{(-,a,1)}"]
\end{tikzcd}$$

Hierarchy: (1)

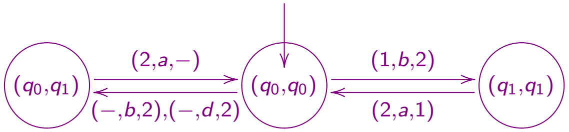
$$\begin{tikzcd}
{} \arrow[r] & q_0 \arrow[r, shift left, "{(-,b,2),\ (-,d,2)}"] & q_1 \arrow[l, shift left, "{(2,a,-)}"]
\end{tikzcd}$$

Hierarchy: (2)

$$\begin{tikzcd}
{} \arrow[r] & (q_0,q_0) \arrow[r, shift left, "{(-,b,2),(-,d,2)}"] \arrow[d, dotted, shift right, "{(1,b,-),(1,c,-)}"'] \arrow[dr, shift left, "{(1,b,2)}"] & (q_0,q_1) \arrow[l, shift left, "{(2,a,-)}"] \arrow[d, dotted, shift left, "{(1,b,-),(1,c,-)}"] \\
& (q_1,q_0) \arrow[u, dotted, shift right, "{(-,a,1)}"'] \arrow[r, dotted, shift left, "{(-,b,2),(-,d,2)}"] & (q_1,q_1) \arrow[u, dotted, shift left, "{(-,a,1)}"] \arrow[l, dotted, shift left, "{(2,a,-)}"] \arrow[ul, shift left, "{(2,a,1)}"]
\end{tikzcd}$$

Hierarchy: ((1),(2))

Executions, traces



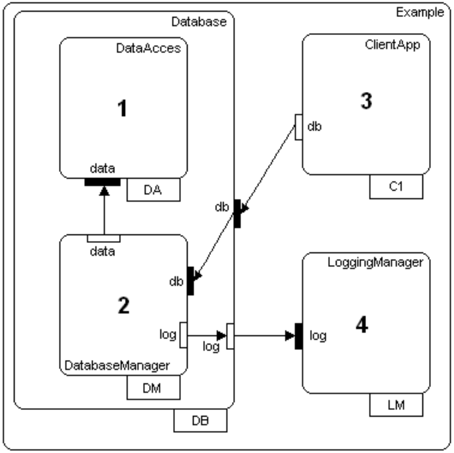
Hierarchy: ((1),(2))

Execution:
$((q_0, q_0), (1, c, -), (q_1, q_0), (-, a, 1))^*$

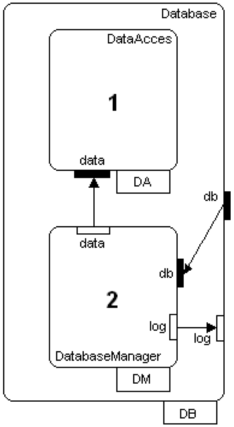
Closed execution:
$((q_0, q_0), (1, b, 2), (q_1, q_1), (2, a, 1))^*$

Trace:
$((1, c, -), (-, a, 1))^*$

Example - database system



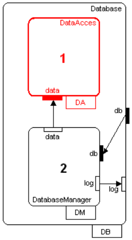
Composed component DB (type Database)



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↓)*
};
```



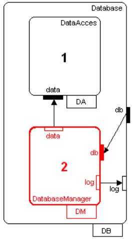
$$\begin{tikzcd} {} \arrow[r] & q_0 \arrow[r, bend left, "{(-,Insert,1),(-,Delete,1),(-,Query,1)}"] & q_1 \arrow[l, bend left, "{(1,Done,-)}"] \end{tikzcd}$$

Hierarchy: (1)

Component DM (type DatabaseManager)

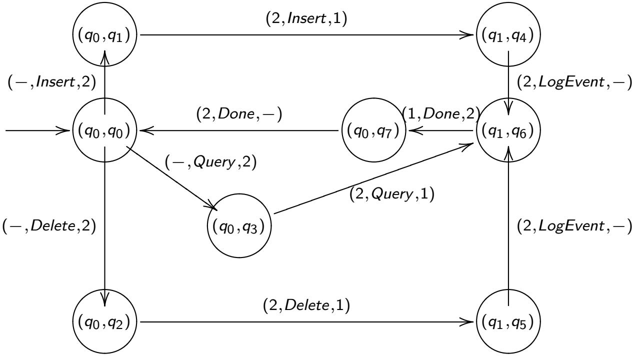
```
frame DatabaseManager {
   ...
   protocol:
      (((?db.Insert↑; !data.Insert↑; !log.LogEvent↑)+
      (?db.Delete↑; !data.Delete↑; !log.LogEvent↑)+
      (?db.Query↑; !data.Query↑ )); ?data.Done↓; !db.Done↓)*
};
```

$$\begin{tikzcd}
 & q_1 \arrow[rr, "(2,\mathit{Insert},-)"] & & q_4 \arrow[d, "(2,\mathit{LogEvent},-)"] \\
{} \arrow[r] & q_0 \arrow[u, "(-,\mathit{Insert},2)"] \arrow[dd, "(-,\mathit{Delete},2)"'] \arrow[dr, "(-,\mathit{Query},2)"'] & q_7 \arrow[l, "(2,\mathit{Done},-)"'] & q_6 \arrow[l, "(-,\mathit{Done},2)"'] \\
 & & q_3 \arrow[ur, "(2,\mathit{Query},-)"'] & \\
 & q_2 \arrow[rr, "(2,\mathit{Delete},-)"] & & q_5 \arrow[uu, "(2,\mathit{LogEvent},-)"']
\end{tikzcd}$$

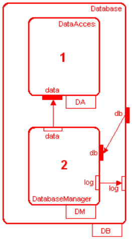


Hierarchy: (2)

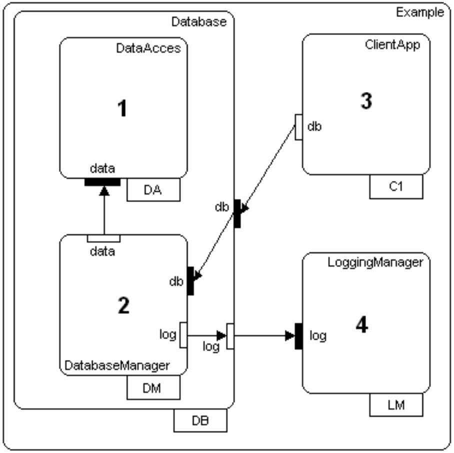
Composed component DB (type Database)



Hierarchy: ((1),(2))



Specification

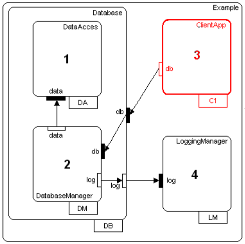


Component C1 (type Client)

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

$$\begin{tikzcd} {} \arrow[r] & q_0 \arrow[r, shift left, "{(3,\mathit{Insert},-),(3,\mathit{Delete},-),(3,\mathit{Query},-)}"] & q_1 \arrow[l, shift left, "{(-,\mathit{Done},3)}"] \end{tikzcd}$$

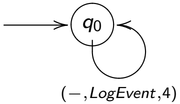
Hierarchy: (3)



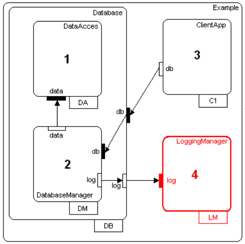
Component LM (type LoggingManager)

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

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



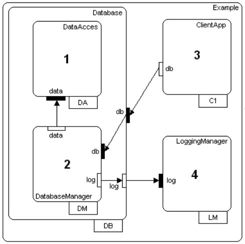
Hierarchy: (4)



Composition of previous components

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

$$\begin{tikzcd}
 & q_{0110} \arrow[rrr, "{(2,Insert,1)}"] & & & q_{1410} \arrow[d, "{(2,LogEvent,4)}"] \\
{} \arrow[r] & q_{0000} \arrow[u, "{(3,Insert,2)}"] \arrow[dr, "{(3,Query,2)}"] \arrow[dd, "{(3,Delete,2)}"'] & & q_{0710} \arrow[ll, "{(2,Done,3)}"'] & q_{1610} \arrow[l, "{(1,Done,2)}"'] \\
 & & q_{0310} \arrow[urr, "{(2,Query,1)}"] & & \\
 & q_{0210} \arrow[rrr, "{(2,Delete,1)}"] & & & q_{1510} \arrow[uu, "{(2,LogEvent,4)}"']
\end{tikzcd}$$



Hierarchy:(((1),(2)),(3),(4))

Properties 1/2

Every action *Insert* sent by the client is followed by action *Done* received by the client.

$$G((3, Insert, 2) \Rightarrow F(3, Done, 2))$$

`true`

Properties 2/2

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

$$G(((3, Insert, 2) \vee (3, Delete, 2) \vee (3, Query, 2))$$
$$\Rightarrow ((\neg(2, Done, 3))\ \mathcal{U}\ (2, LogEvent, 4)))$$

`false`

$((3, Query, 2), (2, Query, 1), (1, Done, 2), (2, Done, 3))*$

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, . . .