Towards Dependable Emergent Ensembles of Components: The DEECo Component Model

Jaroslav Keznikl, Tomáš Bureš, František Plášil, Michal Kit

bures@d3s.mff.cuni.cz
The Scope

- ASCENS EU project
  - FP7 IP FET
  - 14 partners
  - Goal: Self-aware, self-adaptive systems from components

- Case-studies:
  - Self-aware and autonomous robots
  - Resource management in cloud computing
  - Intelligent navigation of electric vehicles (VW)
Challenges

- Dynamic & changing environment, open system
- Massively distributed, ad-hoc networks
  - Components appear/disappear
  - Connections formed to reflect current „physical“ situation
  - Communication is unreliable
  - No notion of the global state of the system

Classical approaches to component architectures do not scale
Component
- independent autonomous unit responsible for computation and sensing/actuating
- local knowledge (tree data structure)
- processes (periodic or event-based) performing computation over the local knowledge
- no proactive (message or procedure call-based) communication with other components
Example

\textbf{component} \textit{CarA} = \\

\textbf{Knowledge}

\begin{itemize}
  \item \textbf{id:} \textit{CarId} = "A";
  \item \textbf{info:} \textit{CarInfo} = \\
    \begin{itemize}
      \item \textbf{position:} \textit{Position};
      \item \textbf{path:} \textbf{list} \textit{Position};
    \end{itemize}
  \end{itemize}

\text{}\textbf{otherCars:} \textbf{map} \textit{CarId} \rightarrow \textit{CarInfo};

\textbf{Processes}

\begin{verbatim}
/* Driving according to common traffic rules */
step: \textit{Process} = \\
  \begin{itemize}
    \item \textbf{function =} \textbf{fun} (\textbf{inout} \textit{i: CarInfo}, \\
          \textbf{in o:} \textbf{map} \textit{CarId} \rightarrow \textit{CarInfo}) \{ ... \} ;
    \item \textbf{input=} \textbf{[info, otherCars]};
    \item \textbf{output=} \textbf{[info]};
    \item \textbf{scheduling =} \textbf{PERIODIC(100ms)};
  \end{itemize}
\end{verbatim}

\end{itemize}
Example

```
component CarA = {
  id: CarId = "A";
  info: CarInfo = {
    position: Position;
    path: list Position;
  };
  otherCars: map CarId -> CarInfo;

  /* Driving according to common traffic rules */
  step: Process = {
    function = fun(inout i: CarInfo,
      in o: map CarId->CarInfo) { ... } ;
    input=[info, otherCars];
    output=[info];
    scheduling = PERIODIC(100ms);
  };
}
```
Ensemble

- group of components cooperating for a common goal
- formed dynamically based on predicate over components’ knowledge
  - membership
- synchronizes parts of components’ knowledge
  - periodic / triggered

![Diagram showing ensemble specification with components exchanging data knowledge and ensemble membership and mapping functions.](image-url)
**Example**

**ensemble** *CarsInCrossingEnsemble* {

**membership**: `fun(in r: ICar, in c: ICar, out ret: Boolean) = {
  ret = bothCarsCloseToSameCrossing(r.info.position,
  m.info.position,
  TRESHOLD);
};`

**mapping**: `fun(inout m: ICar, inout c: ICar) = {
  m.otherCars = m.otherCars.merge(c.otherCars).except(m.id);
  c.otherCars[m.id] = m.info;
};`

`/* Driving according to common traffic rules */
step: Process = {
  function = fun(inout i: CarInfo, in o: map CarId->CarInfo) { ... };
  input = [info, otherCars];
  output = [info];
  scheduling = PERIODIC(100ms);
};`
Example

**Example**: CarsInCrossingEnsemble

```plaintext
ensemble CarsInCrossingEnsemble {

  membership: fun(in r: ICar, in c: ICar, 
                 out ret: Boolean) = {
    ret = bothCarsCloseToSameCrossing( 
      r.info.position, 
      m.info.position, 
      TRESHOLD);
  };

  mapping: fun(inout m: ICar, 
              inout c: ICar) = {
    m.otherCars = m.otherCars.merge( 
      c.otherCars).except(m.id);
    c.otherCars[m.id] = m.info;
  };

  /* Driving according to common traffic rules */
  step: Process = {
    function = fun(inout i: CarInfo, 
                  in o: map CarId->CarInfo) { ... };
    input=[info, otherCars];
    output=[info];
    scheduling = PERIODIC(100ms);
  };
}
```

Towards Dependable Emergent Ensembles of Components: The DEECo Component Model, WICSA 2012,
**Example**

**ensemble** `CarsInCrossingEnsemble` {

**Membership**

`membership: fun(in r: ICar, in c: ICar,` `out ret: Boolean) = {` `ret = bothCarsCloseToSameCrossing(` `r.info.position,` `m.info.position,` `TRESHOLD);` `}`

**Mapping**

`mapping: fun(inout m: ICar,` `inout c: ICar) = {` `m.otherCars = m.otherCars.merge(` `c.otherCars).except(m.id);` `c.otherCars[m.id] = m.info;` `}`

*/ Driving according to common traffic rules */

**Process**

`step: Process = {` `function = fun(inout i: CarInfo,` `in o: map CarId->CarInfo) { ... } ;` `input=[info, otherCars];` `output=[info];` `scheduling = PERIODIC(100ms);` `}`
Example

ensemble CarsInCrossingEnsemble {

  membership: fun(in r: ICar, in c: ICar,
       out ret: Boolean) = {
    ret = bothCarsCloseToSameCrossing(
      r.info.position,
      m.info.position,
      TRESHOLD);
  };

  mapping: fun(inout m: ICar,
       inout c: ICar) = {
    m.otherCars = m.otherCars.merge(
      c.otherCars).except(m.id);
    c.otherCars[m.id] = m.info;
  };

}"
**Example**

**ensemble** *CarsInCrossingEnsemble* {

**Membership**

`membership: fun(in r: ICar, in c: ICar, 
    out ret: Boolean) = {
    ret = bothCarsCloseToSameCrossing(
        r.info.position, 
        m.info.position, 
        TRESHOLD);
};` 

**Synchronization**

`mapping: fun(inout m: ICar, 
    inout c: ICar) = {
    m.otherCars = m.otherCars.merge(
        c.otherCars).except(m.id);
    c.otherCars[m.id] = m.info;
};` 

/* Driving according to common traffic rules */

**step: Process** = {
    function = fun(inout i: CarInfo, 
        in o: map CarId->CarInfo) { ... } ;
    input=[info, otherCars];
    output=[info];
    scheduling = PERIODIC(100ms);
};

Towards Dependable Emergent Ensembles of Components: The DEECo Component Model, WICSA 2012, I
Conclusion & Future Directions

- **DEECo highlights**
  - separation of *autonomous computation* × *communication*
  - more resilient to communication failures
  - inherent support for partial belief of the system
  - facilitates development of highly distributed open systems

- **Ongoing work**
  - Formalization of the behavior
    - SCEL (formal model used in ASCENS)
    - Stochastic model-checking
  - Distributed Java runtime
    - Using JavaSpaces (LINDA-like middleware)
    - [https://github.com/d3scomp/JDEECo](https://github.com/d3scomp/JDEECo)
  - UML-based system-level design
    - [https://github.com/d3scomp/UMLDEECo](https://github.com/d3scomp/UMLDEECo)