ACVI Workshop
April 5, 2016
Venice, Italy

Contract-Based Architectural Design with OCRA

Stefano Tonetta
Fondazione Bruno Kessler
Center for Scientific and Technological Research
tonettas@fbk.eu

FONDAZIONE BRUNO KESSLER

ES EMBEDDED SYSTEMS

Parts of the material presented in these slides have been contributed by Alessandro Cimatti, Anthony Fernandes Pires, Cristian Mattarei, Marco Gario and other people in FBK

Outline

- Contract-based design with OCRA
- OCRA language
- OCRA-supported analysis
- OCRA and AADL

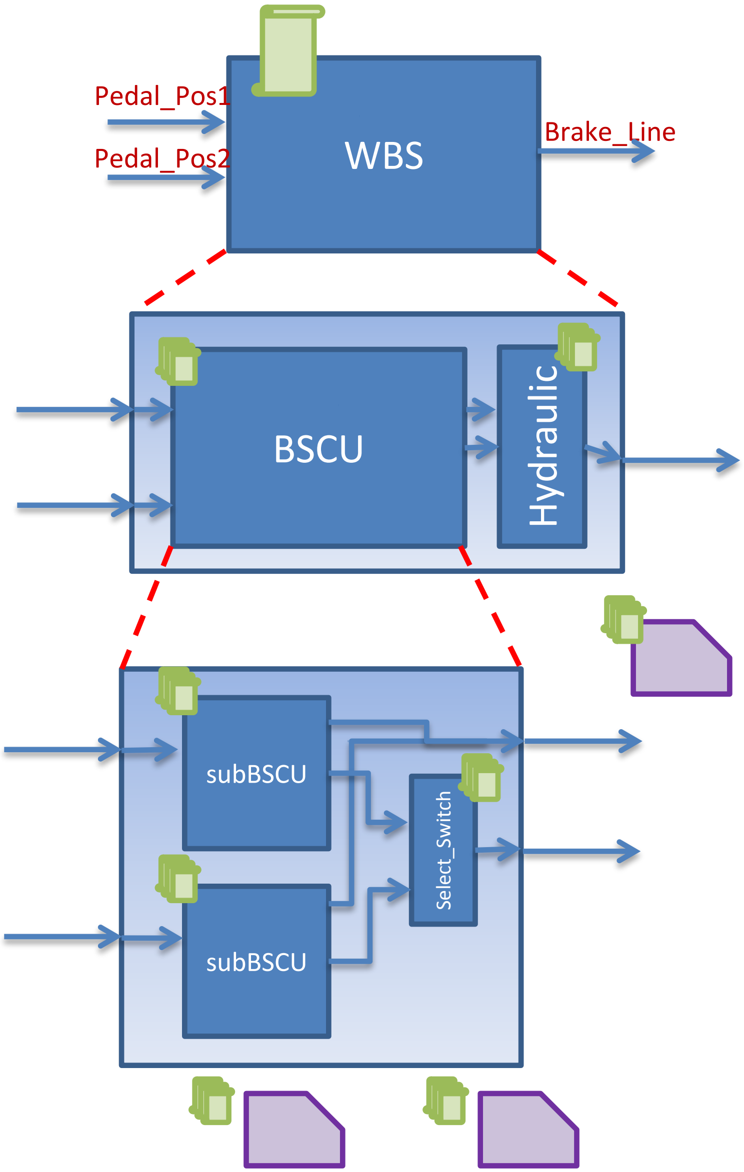
Component-based design

- Embedded systems become more and more complex, networked, interconnected
- Component-based design: popular approach for managing such complexity
- Many languages and tools: (SysML, AADL, AF3, Altarica, …)
- A component can be defined as a unit of composition with contractually specified interfaces
  - Hides internal information
  - Defines interface to interact with the environment
- Component-based design ideal for
  - Independent development
  - Reuse of components

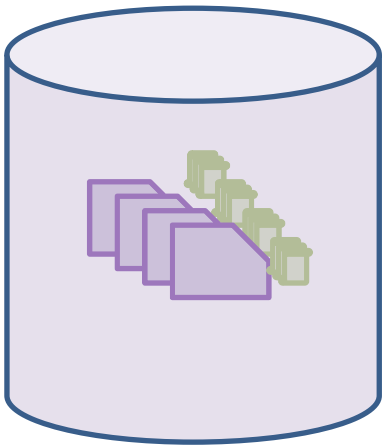
Contract-based design

- Contracts used to specify assumptions and guarantees
  - First conceived for software, now popular also for system architectural design
- Assumptions and guarantees are properties respectively of the environment and of component
- Can be seen as assertions for component interfaces.
- Contracts used to characterize the correctness of component implementations and environments
- Contracts for OO programing are pre-/post-conditions
- For systems, assumptions correspond to pre-conditions, guarantees correspond to post-conditions

Contract-based approach



- Early validation of refinement
- Composition verification
- Ensuring correct reuse



OCRA tool support

- Textual representation of the architecture and contracts
- Built on top of nuXmv for infinite-state model checking
- Integrated with CASE tools:
  - AutoFocus3
    - Developed by Fortiss
    - For synchronous system architectures
  - CHESS
    - Developed by Intecs
    - For SysML and UML modeling
  - COMPASS
    - Developed by FBK and RWTH/AACHEN
    - Variant of AADL
- One of the few tools supporting contract-based design for embedded systems
- Publicly available at https://ocra.fbk.eu

Contract specification language

- Contracts' assumptions and guarantees specified in an extension of LTL
- Both discrete-time and hybrid semantics are supported
- Increase usability with
  - Syntactic sugar
  - English words instead of math symbols:
    - "always" (G)
    - "never" ($G\neg$)
    - "eventually" (F)
    - "next" (X)

OCRA language

```
COMPONENT system
…

COMPONENT A
…

COMPONENT B
…
```

Component interface

```
COMPONENT system
        INTERFACE
                INPUT PORT x: continuous;
                OUTPUT PORT a: boolean;
        …
        REFINEMENT
        …

COMPONENT A
…

COMPONENT B
…
```

Contracts

```
COMPONENT simple system
        INTERFACE
                INPUT PORT x: continuous;
                OUTPUT PORT v: boolean;

                CONTRACT v_correct
                        assume: always x>=0;
                        guarantee: always (x=0 implies v);

        REFINEMENT
        …

COMPONENT A
…

COMPONENT B
…
```

Component refinement

```
COMPONENT simple system
        INTERFACE
                INPUT PORT x: continuous;
                OUTPUT PORT v: boolean;

                CONTRACT v_correct
                        assume: always x>=0;
                        guarantee: always (x=0 implies v);

        REFINEMENT
                SUB a: A;
                SUB b: B;

                CONNECTION a.x := x;
                CONNECTION b.y := a.v;
                CONNECTION v:= b.v;
        …
```

Contract refinement

```
COMPONENT simple system
    INTERFACE
        INPUT PORT x: continuous;
        OUTPUT PORT v: boolean;

        CONTRACT v_correct
            assume: always x>=0;
            guarantee: always (x=0 implies v);

    REFINEMENT
        SUB a: A;
        SUB b: B;

        CONNECTION a.x := x;
        CONNECTION b.vi := a.v;
        CONNECTION v:= b.vo;

        CONTRACT v_correct REFINEDBY a.v_correct, b.pass;
```

Port types

- Port types are either
  - NuSMV types: boolean, enumeratives, ...
  - nuXmv additional types: real, integer, and uninterpreted functions
  - continuous, i.e. real-value ports evolving continuously in time.
  - event, i.e. boolean-value port that is assigned only on discrete transitions.
- Port can be:
  - Input
  - Output
  - Parameter

LTL with SMT predicates

- Use first-order predicates instead of propositions:

  always (x>=a and x<=b)

- Standard LTL operators:

  always (e1 implies in the future (e2 and x=y+z))

- “next” to express changes/transitions:

  always (next(x) = x+1)

  always (next(x) – a <= b)

Continuous time

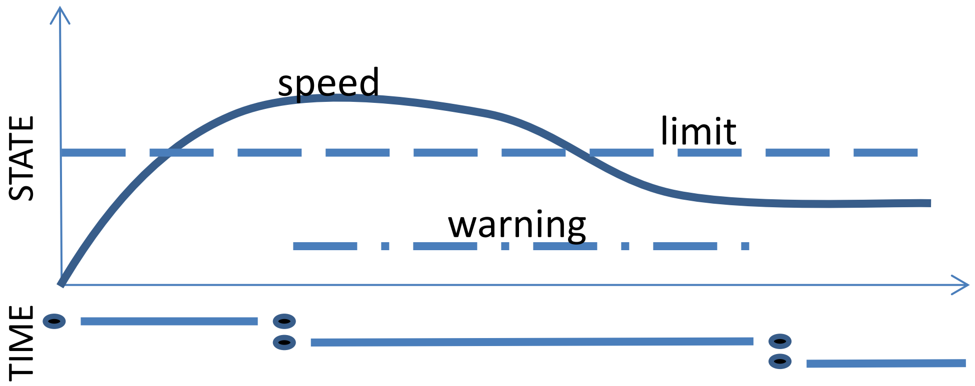
- The derivative of “x” is always less than 2:

  always der(x)<2
- Whenever “a” holds, the derivative of “x” is zero

  always (a implies der(x)<=2)
- Whenever “a” holds, “b” remain true until the derivative of “x” is less or equal to 5

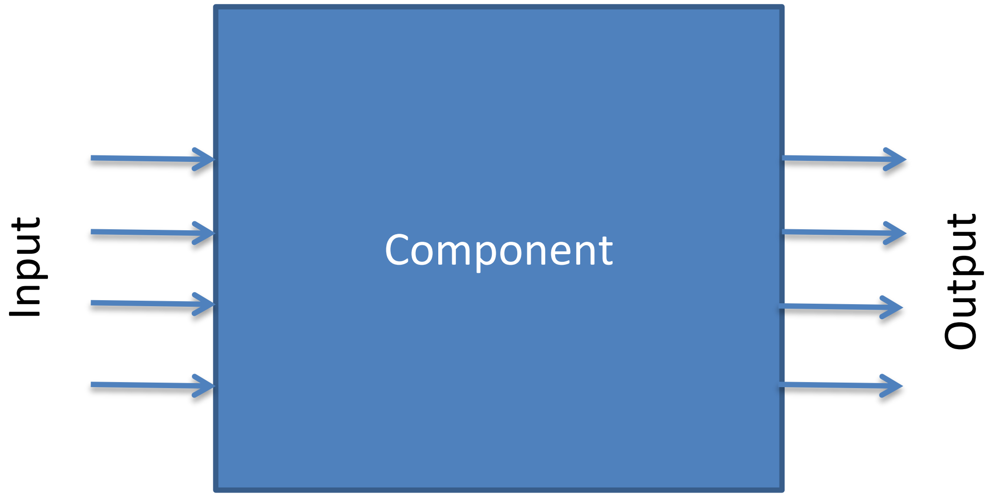
  always (a implies (b until der(x)<=5)
- Reaction time

  always (e1 implies time_until(e2)<=5)



always (speed>limit implies in the future warning)

Trace-based semantics

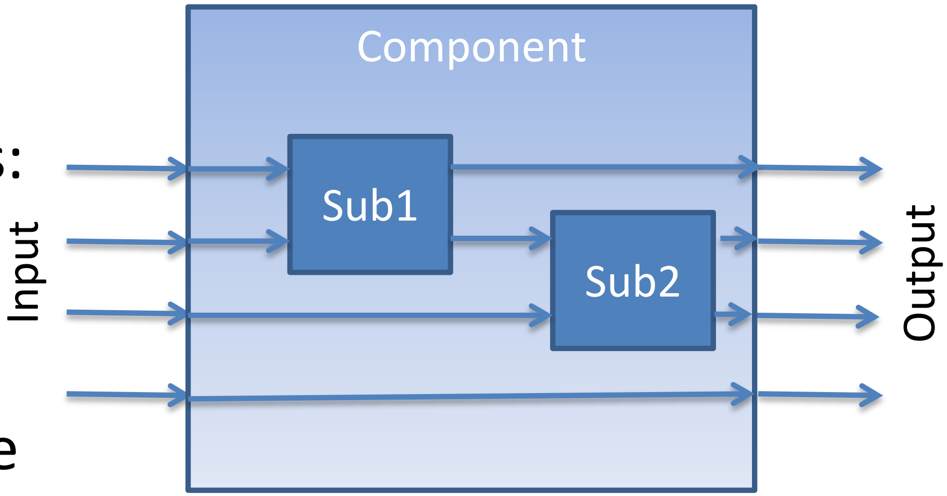


- A component interface defines boundary of the interaction between the component and its environment.
- Consists of:
  - Set of input and output ports (syntax)
    - Ports represent visible data and events exchanged with environment.
  - Set of traces (semantics)
    - Traces represent the behavior, history of events and values on data ports.

Composite components

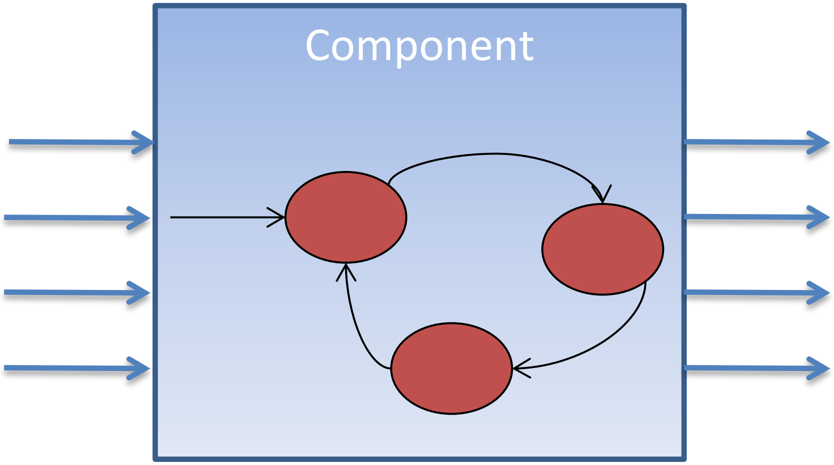
- Components are decomposed into subcomponents
- Different kind of compositions:
  - Synchronous,
  - Asynchronous,
  - Synchronizations
- Connections map (general rule of architecture languages):
  - Input ports of the composite component
  - Output ports of the subcomponents

  Into
  - Output ports of the composite component
  - Input ports of the subcomponents.



Leaf implementations

- External to the OCRA language
- State-machine
  - Internal state
  - Internal transitions
  - Language over the ports
- Hybrid automaton in case of continuous variables



Discrete vs. hybrid

- OCRA is parametrized by the logic
- The expressions can be restricted and interpreted as discrete-time LTL or hybrid LTL
- In discrete-time mode, behavior of leaf components specified in smv (nuXmv)
- In hybrid-time mode, behavior of leaf components specified in hydi (HyCOMP)

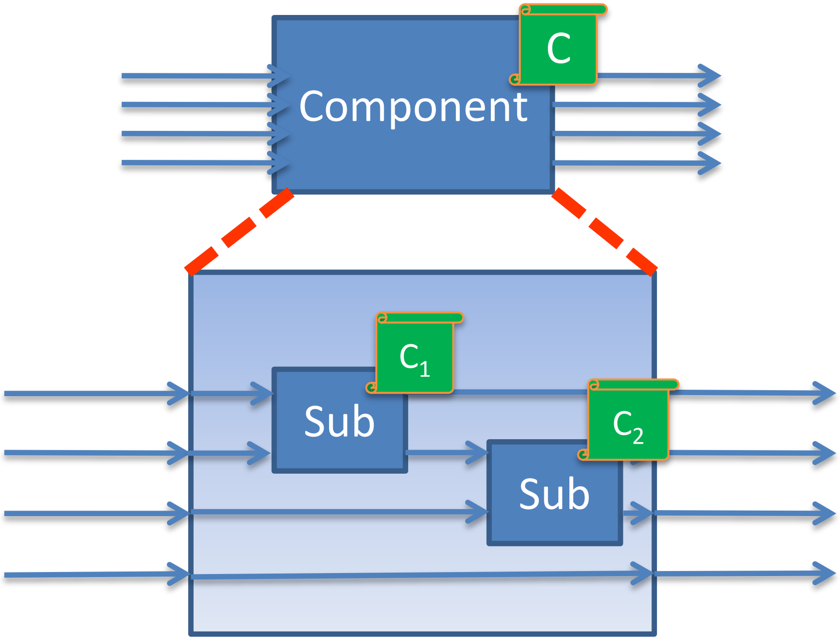
Main Features

- Check refinement
- Validation of contracts
- Check implementation
- Check receptiveness
- Compute fault tree

- Results are shown in textual form or in XML to ease the integration within modeling tools (to map back results)

Trace-based contract refinement

- The set of contracts $\{C_i\}$ refines C with the connection γ ($\{C_i\} \preccurlyeq_\gamma C$) iff for all correct implementations Imp_i of C_i and correct environment Env of C:
  - The composition of $\{Imp_i\}$ is a correct implementation of C.
  - For all k, the composition of Env and $\{Imp_i\}_{i \neq k}$ is a correct environment of C_k.
- Verification problem:
  - check if a given refinement is correct (independently from implementations).

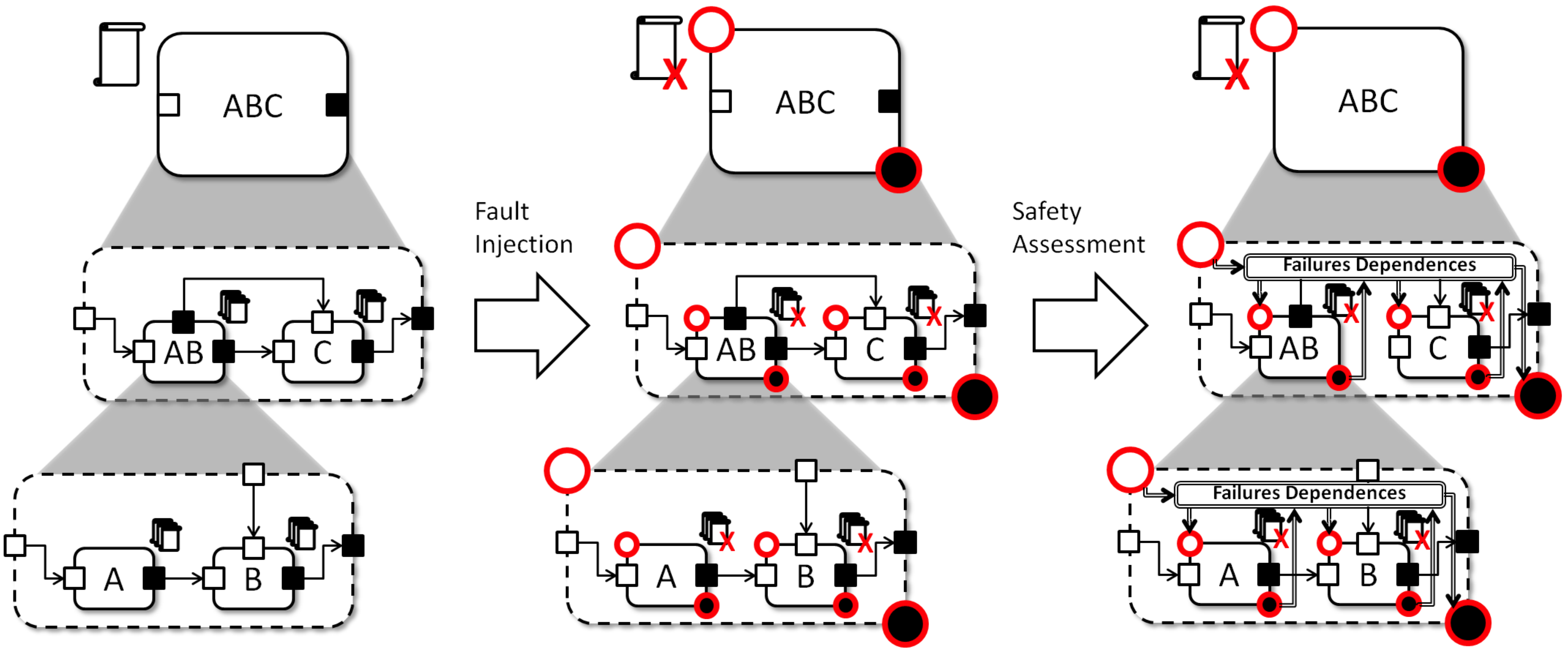


Proof obligations for contract refinement

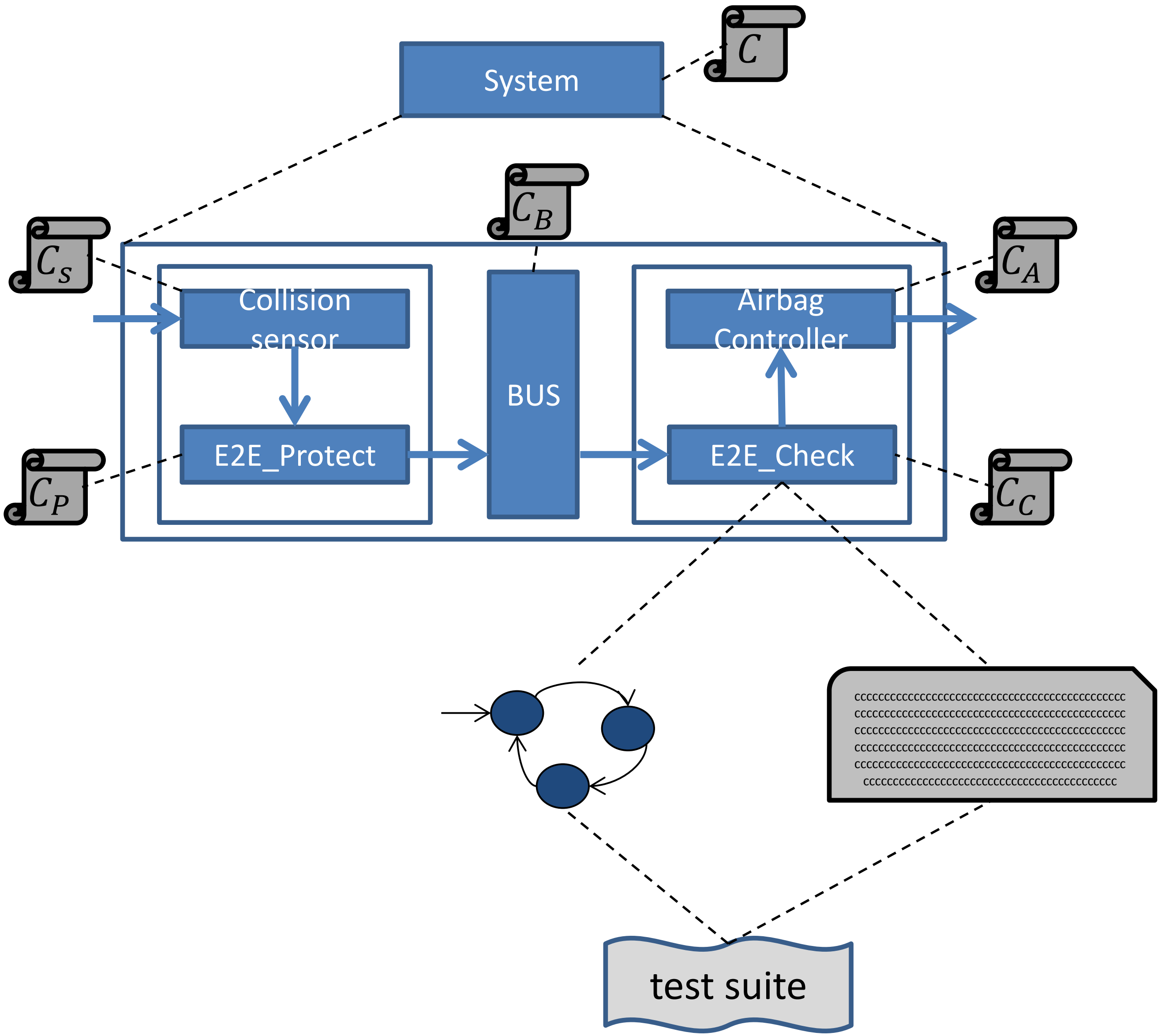
- Given $C1 = \langle \alpha_1, \beta_1 \rangle, \ldots, C_n = \langle \alpha_n, \beta_n \rangle, C = \langle \alpha, \beta \rangle$
- Proof obligations for $\{C_i\} \preccurlyeq C$:
  - $\gamma\Big(\big(\bigwedge_{1 \le j \le n}(\alpha_j \rightarrow \beta_j)\big) \rightarrow (\alpha \rightarrow \beta)\Big)$
  - $\gamma\Big(\big(\bigwedge_{2 \le j \le n}(\alpha_j \rightarrow \beta_j)\big) \rightarrow (\alpha \rightarrow \alpha_1)\Big)$
  - …
  - $\gamma\Big(\big(\bigwedge_{1 \le j \le n, j \ne i}(\alpha_j \rightarrow \beta_j)\big) \rightarrow (\alpha \rightarrow \alpha_i)\Big)$
  - …
  - $\gamma\Big(\big(\bigwedge_{1 \le j \le n-1}(\alpha_j \rightarrow \beta_j)\big) \rightarrow (\alpha \rightarrow \alpha_n)\Big)$

- Theorem: $\{C_i\} \preccurlyeq_\gamma C$ iff the proof obligations are valid.

Hierarchical Safety Assessment



Integration with testing



OCRA vs. AADL

- Same component-based approach
- Same separation of architecture and implementations
- OCRA
  - Has a formal semantics
  - Both discrete-time and hybrid semantics
  - Both synchronous and asynchronous composition
  - Expressions in connections (e.g., in:= outp1 or outp2)
  - Built-in data types
- AADL
  - Event data ports
  - Richer language for design
  - Extensible with property sets and annexes
  - Behavioral and error annexes
- SLIM
  - Variant of AADL with formal semantics
  - Asynchronous composition
  - Built-in data types
  - Error models
  - Now supports AADL-compliant models

AADL property set for OCRA Contracts

- COMPASS, tool developed within ESA projects in collaboration with RWTH (Aachen University)
- integrated with OCRA for contract-based design of a subset of AADL models
  - First developed in the FP7 D-MILS project
  - Extended in the ESA CATSY project
- Automatic translation of AADL into OCRA
- Mapping back of results

CATSY Example

```
system Camera
features
  image: in data port PhysicalImage;
  take_picture: in event port;
  put_picture: out event data port DigitalPicture;
properties
  SLIMpropset::Contracts => ([Name => "take_picture"; Assumption => "true";
  Guarantee => "always (take_picture implies time_until(put_picture)<=5) and
                 always (put_picture implies data(put_picture)=picture(image))";]);
end Camera;

data Position
end Position;

data PhysicalImage
end PhysicalImage;

data DigitalPicture
end DigitalPicture;


data Attitude
end Attitude;

properties
  SLIMpropset::Constants => "
  picture: function PhysicalImage -> DigitalPicture;";
```

CSSP Example

```
system Camera
features
  image: in data port PhysicalImage;
  take_picture: in event port;
  put_picture: out event data port DigitalPicture;
  properties
  CSSP::Reaction => reference(put_picture) applies to take_picture;
  CSSP::ReactionMaxDelay => 5sec applies to take_picture;
  CSSP::Function => "picture(image)" applies to put_picture;
  SLIMpropset::Contracts => ([Name => "take_picture"; Assumption => "true";
  Guarantee => "ReactionProperty(take_picture) and FunctionProperty(put_picture)";]);
end Camera;

data Position
end Position;

data PhysicalImage
end PhysicalImage;

data DigitalPicture
end DigitalPicture;


data Attitude
end Attitude;

properties
  SLIMpropset::Constants => "
  picture: function PhysicalImage -> DigitalPicture; ";
```

Past and Current Projects

- SafeCer (Apr. 2011 – Mar. 2015)
  - ARTEMIS project on Safety Certification of Software-Intensive Systems with Reusable Components.
  - First development of OCRA
  - Used by Thales Comm., TTTECH, CAF, …
- FoReVer (Jan. 2012 – Mar. 2013)
  - ESA study on Functional Requirements and Verification Techniques for the Software Reference Architecture.
- D-MILS (Nov. 2012 – Oct. 2015)
  - FP7 project on Distributed MILS for Dependable Information and Communication Infrastructures.
- Collaboration with Boeing (2014 – 2015)
- CATSY (Dec. 2014 – Apr. 2016)
  - ESA study on Catalogue of System and Software Properties
- AMASS (Apr. 2016 – Mar. 2019)
  - ECSEL project on architecture-driven assurance and more.

Some Case Studies

- Redundant Sensors (developed with Thales in FoReVer)
- AUTOSAR E2E Protection example (developed with Quviq in SafeCer)
- ARP4761 WBS (taken from literature)
- AIR6110 WBS (developed with Boeing)
  - 30 component types for 169 instances
  - max depth of 6 levels
  - 149 contracts

Conclusions

- OCRA: tool to support contract-based design of system architecture
- Contracts specified in linear-time temporal logic
- Support for discrete-time, infinite-state, continuous time
- Main supported analysis:
  – Refinement verification
  – Validation of specification
  – Fault-tree generation
- Integrated in COMPASS for analysis of AADL models