Contract-based design of safety-critical software components

Stefano Tonetta

Fondazione Bruno Kessler (FBK)
Embedded Systems Unit

International Workshop on Critical Software Component Reusability and Certification across Domains (CSC 2013)
ICSR13 workshop
June 18 2013
Credits

■ Work developed within the projects:
  - ARTEMIS project *Safety Certification of Software-Intensive Systems with Reusable Components* (**SafeCer**)
  - ESA project *Functional requirements and verification techniques for the software reference architecture* (**FoReVer**) with Intecs and Thales Alenia Space

■ Joint work with Alessandro Cimatti and Michele Dorigatti
Embedded systems

- Embedded with software to deliver intelligent:
  - Transportation
  - Communication
  - Automation

- Across domains:
  - Railways
  - Avionics
  - Automotive
  - Space
  - Health

- Key properties and challenges:
  - Physical environment
  - Real-time constraints
  - Interaction of components
  - Decomposition of services
  - Safety requirements
SafeCer compositional approach

- SafeCer: ARTEMIS project (2011-2015)
- SafeCer adopts a compositional approach
  - to enable reuse of qualified components;
  - to decrease the cost of the system certification.
- “Compositional” means that properties of the system are deduced from the properties of its components.
- Certification uses verification results as evidences of system safety.
- Consequently, the project adopts:
  - A component-based modeling approach,
  - Compositional verification techniques.
Contract-based approach

- Stepwise refinement
- Compositional verification
- Reuse of components
OCRA tool support

- **OCRA** = Othello Contract Refinement Analysis
- Contracts’ assertions specified in Othello
  - Language for embedded systems properties.
- Built on top of NuSMV3 for infinite-state model checking.
- Integrated with CHESS (by Intecs) for SysML and UML modeling.
- One of the few tools supporting contract-based design for embedded systems.

**Main features:**
- Rich component interfaces to specify:
  - Input/output ports
  - Data/Event ports.
  - Including real-time and safety aspects.
- Contracts in temporal logics.
- Temporal formulas used to characterize set of traces over the ports of components.
Discrete and hybrid traces and temporal properties

\[ G(x > 0 \rightarrow F(y < 0)) \]

\[ G(\text{speed} > \text{limit} \rightarrow F(\text{warning})) \]

In Othello:
“always speed > limit implies in the future warning”
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.

**OCRA example:**

COMPONENT system
INTERFACE
INPUT PORT Pedal_Pos1: event;
INPUT PORT Pedal_Pos2: event;
OUTPUT PORT Brake_Line: event;
A component has an internal structure.

**Architecture view:**
- Subcomponents
- Inter-connections
- Delegations

**State-machine view:**
- Internal state
- Internal transitions
- Language over the ports

**OCRA example:**

```
REFINEMENT
SUB bscu: BSCU;
SUB hydr: Hydraulic;
DEFINE bscu.Pedal_Pos1 := Pedal_Pos1;
DEFINE bscu.Pedal_Pos2 := Pedal_Pos2;
DEFINE Brake_Line := hydr.Brake_Line;
DEFINE hydr.CMD_AS := bscu.CMD_AS;
DEFINE hydr.Valid := bscu.Valid;
```
Component implementation

- $I_S$: input ports of component $S$
- $O_S$: output ports of $S$
- $V_S = I_S \cup O_S$: all ports of $S$
- $Tr(X)$ traces over $X \subseteq V_S$ (sequence of assignments to $X$)
- State machine $Imp$ implementation of $S$ iff $L(Imp) \subseteq Tr(V_S)$
- $M$ can be associated with $\mu_{Imp}: Tr(I_S) \rightarrow 2^{Tr(O_S)}$ such that $\mu_{Imp}(\sigma_i) = \{\sigma_o \mid \sigma_i \times \sigma_o \in L(Imp)\}$
  - Empty set corresponds to rejected input trace
Component environment

- State machine $Env$ environment of $S$ iff 
  \[ L(Env) \subseteq Tr(I_S) \]

- Compatibility of implementation with environment (e.g., for reuse):
  - For any reachable state of $Imp \times Env$, for any input transition of $Env$, there exists a matching transition of $Imp$. 
Trace-based contracts

- Contracts used to characterize the correctness of component implementations and environments.
- Othello assertions used to represent sets of traces
  - \( \phi(V) \) assertion over variables \( V \)
  - \( \langle\langle \phi \rangle\rangle \subseteq Tr(V) \) semantics of \( \phi \)
- A contract of component \( S \) is a pair \( \langle A, G \rangle \) of assertions over \( V_S \)
  - \( A \) is the assumption,
  - \( G \) is the guarantee.
- \( Env \) is a correct environment iff \( L(Env) \subseteq \langle\langle A \rangle\rangle \)
- \( Imp \) is a correct implementation iff \( L(Imp) \cap \langle\langle A \rangle\rangle \subseteq \langle\langle G \rangle\rangle \)

Assumption Behaviors Guarantee

\[
\text{Assumption} \quad \text{Behaviors} \quad \text{Guarantee}
\]

\[
A \\
\downarrow\\
\text{Input} \\
\downarrow\\
\text{Component} \\
\downarrow\\
\text{Output} \\
\downarrow\\
\text{Imp} \\
\downarrow\\
\text{G} \\
\downarrow\\
\]

**OCRA example:**

CONTRACT brake_time
assume:
always (Pedal_Pos1 iff Pedal_Pos2)
guarantee:
always ( (Pedal_Pos1 or Pedal_Pos2) implies (time_until(Brake_Line) <= 10) );
Trace-based contract refinement

- Set of contracts \( \{C_i\} \) refines \( C \) (\( \{C_i\} \preceq C \)) iff for all correct implementations \( \text{Imp}_i \) of \( C_i \) and correct environment \( \text{Env} \) of \( C \):
  1. The composition of \( \{\text{Imp}_i\} \) is a correct implementation of \( C \).
  2. For all \( k \), the composition of \( \text{Env} \) and \( \{\text{Imp}_i\}_{i \neq k} \) 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 $C_1=<\alpha_1, \beta_1>$, ..., $C_n=<\alpha_n, \beta_n>$, $C=<\alpha, \beta>$
- Proof obligations for $\{C_i\} \preceq C$:

\[
(\neg \alpha_1 \lor \beta_1) \land ... \land (\neg \alpha_n \lor \beta_n) \rightarrow (\neg \alpha \lor \beta) \\
(\alpha \land \bigwedge_{2 \leq j \leq n, j \neq i} (\neg \alpha_j \lor \beta_j)) \rightarrow \alpha_i \\
... \\
(\alpha \land \bigwedge_{1 \leq j \leq n-1} (\neg \alpha_j \lor \beta_j)) \rightarrow \alpha_n
\]

- **Theorem:** $\{C_i\}$ refines $C$ iff proof obligations are valid
Case studies

- **Academic case studies such as a Wheel Braking System.**
  - Also used to show efficiency of compositional approach

- **Case studies developed in FoReVer (ESA project with Intecs and Thales Alenia Space France)**
  - FDIR of EagleEye satellite
  - GNC of GlobalStar2 satellite

- **Case studies under development in the projects SafeCer such as:**
  - Control system of train’s doors
  - Distance measuring equipment of airplanes
Reuse of SRA components in EagleEye FDIR design

<table>
<thead>
<tr>
<th>Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Device_SRA_PseudoComponent</td>
<td>+ bus_delay : Real</td>
</tr>
<tr>
<td></td>
<td>+ measurement : Measurement</td>
</tr>
<tr>
<td></td>
<td>+ contractProperty : contractProperty</td>
</tr>
<tr>
<td>SW-indent</td>
<td>+ fdir : thermal</td>
</tr>
<tr>
<td></td>
<td>+ fdir : SRAComponents</td>
</tr>
<tr>
<td></td>
<td>+ fdir_speed_critical_value : fdir_speed_critical_value</td>
</tr>
<tr>
<td></td>
<td>+ fdir_power_critical_value : fdir_power_critical_value</td>
</tr>
<tr>
<td></td>
<td>+ fdir_temperature_critical_value : fdir_temperature_critical_value</td>
</tr>
<tr>
<td>SWSub_FDIR_critical_value_impl</td>
<td>+ critical_value : critical_value</td>
</tr>
<tr>
<td></td>
<td>+ critical_threshold_min : Real</td>
</tr>
<tr>
<td></td>
<td>+ critical_threshold_max : Real</td>
</tr>
<tr>
<td></td>
<td>+ monitoring_check_enable : monitoringType</td>
</tr>
<tr>
<td></td>
<td>+ value_type : valueType</td>
</tr>
<tr>
<td></td>
<td>+ monitoring : monitoring</td>
</tr>
<tr>
<td>SWRecovery_impl</td>
<td>+ critical_value : critical_value</td>
</tr>
<tr>
<td></td>
<td>+ monitoring_frequency : Real</td>
</tr>
<tr>
<td></td>
<td>+ monitoring : Monitoring</td>
</tr>
<tr>
<td></td>
<td>+ contractProperty : contractProperty</td>
</tr>
<tr>
<td></td>
<td>+ monitoring : Monitoring</td>
</tr>
<tr>
<td></td>
<td>+ critical_threshold_min : Real</td>
</tr>
<tr>
<td></td>
<td>+ monitoring_frequency : Real</td>
</tr>
<tr>
<td></td>
<td>+ critical_threshold_max : Real</td>
</tr>
<tr>
<td></td>
<td>+ expected_value : Real</td>
</tr>
<tr>
<td></td>
<td>+ delta_min : Real</td>
</tr>
<tr>
<td></td>
<td>+ delta_max : Real</td>
</tr>
<tr>
<td></td>
<td>+ value_type : valueType</td>
</tr>
<tr>
<td></td>
<td>+ monitoring_check_enable : monitoringType</td>
</tr>
</tbody>
</table>
Conclusions

- **Contract-based design:**
  - Powerful for:
    1. Step-wise refinement,
    2. Compositional verification,
    3. Reuse of components.
  - Related to works on:
    • Assume-guarantee reasoning,
    • Interface theory.

- **OCRA**
  - One of the few tools supporting contracts for embedded systems.
  - Based on Othello language to specify embedded-system properties.

- **Future works on:**
  - Contract-based fault-tree analysis,
  - Contract-based FDIR,
  - Contract-based testing,
  - Contract-based derivation of schedulability problems,
  - Synthesis of contracts,
  - Synthesis of implementations.
Thank you!