Contract-based Modeling and Verification of Timed Safety Requirements for System Design in SysML

Iulia Dragomir

Université Toulouse III Paul Sabatier

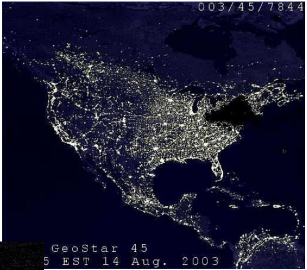
December 3rd, 2014

Defense committee:

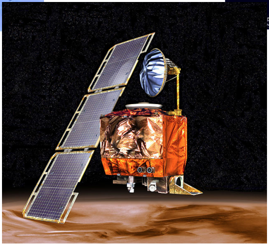
Béatrice BÉRARD	Université Pierre et Marie Curie	Reviewer
Jean-Paul BODEVEIX	Université Toulouse III Paul Sabatier	Examiner
Susanne GRAF	CNRS-VERIMAG	Examiner
Thomas LAMBOLAIS	École des Mines d'Alès	Examiner
Alexander KNAPP	Universität Augsburg	Reviewer
Iulian OBER	Université Toulouse II Jean Jaurès	Adviser
Christian PERCEBOIS	Université Toulouse III Paul Sabatier	Adviser

Safety-critical systems are not always error-free

(Ariane 5 flight 501, 1996)



(Northeast blackout, 2003)



(Mars Climate Orbiter, 1999)

Challenges in system design

Key factors:

- Consideration and best handling of systems growing size and complexity
- System's correctness with respect to the specified requirements
- Efficiency with reduced effort and costs

⇝ Compositional component-based design driven by requirements

Offers support for:

- Manageable systems by decomposition
- Incremental design by successive refinement
- Independent implementation of sub-systems (components)
- Sub-systems reusability

Challenges in system design

Key factors:

- Consideration and best handling of systems growing size and complexity
- System's correctness with respect to the specified requirements
- Efficiency with reduced effort and costs

⇝ Compositional component-based design driven by requirements

Offers support for:

- Manageable systems by decomposition
- Incremental design by successive refinement
- Independent implementation of sub-systems (components)
- Sub-systems reusability

Challenges in system design

Key factors:

- Consideration and best handling of systems growing size and complexity
- System's correctness with respect to the specified requirements
- Efficiency with reduced effort and costs

$\rightsquigarrow$ Compositional component-based design driven by requirements

Offers support for:

- Manageable systems by decomposition
- Incremental design by successive refinement
- Independent implementation of sub-systems (components)
- Sub-systems reusability

Compositional system design in industrial practice

Large, reactive, timed, asynchronous system specifications in UML/SysML:

- Rich graphical semi-formal language
  ⇒ ambiguous or unspecified operational semantics
  ⇒ different interpretations of the design which may result in erroneous implementations
- The correctness of semi-formal designs must be ensured by model-checking
  ⇒ subject to the state space explosion problem

... such as the ATV Solar Generation System: 4-level architecture, 95 running objects and 62 possible hardware failures

Compositional system design in industrial practice

Large, reactive, timed, asynchronous system specifications in UML/SysML:

- Rich graphical semi-formal language
  ⇒ ambiguous or unspecified operational semantics
  ⇒ different interpretations of the design which may result in erroneous implementations
- The correctness of semi-formal designs must be ensured by model-checking
  ⇒ subject to the state space explosion problem

... such as the ATV Solar Generation System: 4-level architecture, 95 running objects and 62 possible hardware failures

Compositional system design in industrial practice

Large, reactive, timed, asynchronous system specifications in $\mathrm{UML/SysML}$:

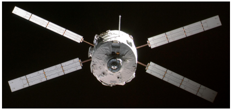
- Rich graphical semi-formal language
  ⇒ ambiguous or unspecified operational semantics
  ⇒ different interpretations of the design which may result in erroneous implementations
- The correctness of semi-formal designs must be ensured by model-checking
  ⇒ subject to the state space explosion problem

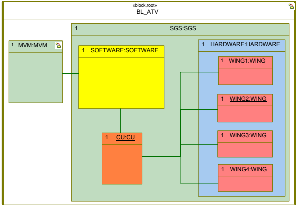
... such as the ATV Solar Generation System: 4-level architecture, 95 running objects and 62 possible hardware failures

Compositional system design in industrial practice

Large, reactive, timed, asynchronous system specifications in UML/SysML:

- Rich graphical semi-formal language
  ⇒ ambiguous or unspecified operational semantics
  ⇒ different interpretations of the design which may result in erroneous implementations
- The correctness of semi-formal designs must be ensured by model-checking
  ⇒ subject to the state space explosion problem





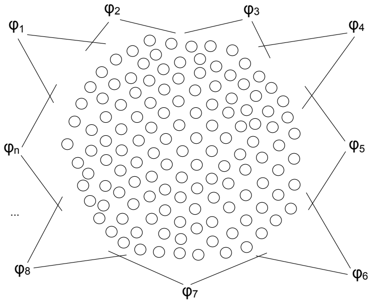
... such as the ATV Solar Generation System: 4-level architecture, 95 running objects and 62 possible hardware failures

Requirement-driven component-based design dilemma

Let S be a component-based system and $\varphi_1, \cdots, \varphi_n$ a set of requirements.

How to achieve correct compositional design when:

- a requirement is in general satisfied by the collaboration of a set of components and
- a component is involved in the satisfaction of several requirements?

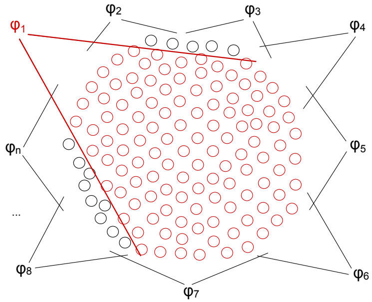


Requirement-driven component-based design dilemma

Let S be a component-based system and $\varphi_1, \cdots, \varphi_n$ a set of requirements.

How to achieve correct compositional design when:

- a requirement is in general satisfied by the collaboration of a set of components and
- a component is involved in the satisfaction of several requirements?

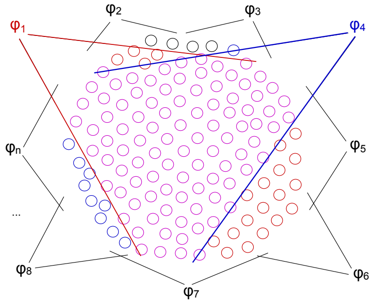


Requirement-driven component-based design dilemma

Let S be a component-based system and $\varphi_1, \cdots, \varphi_n$ a set of requirements.

How to achieve correct compositional design when:

- a requirement is in general satisfied by the collaboration of a set of components and
- a component is involved in the satisfaction of several requirements?



Contracts

Idea: Abstractly specify how a component is involved in the satisfaction of a requirement φ.

A contract $\mathcal{C}$:

- Is a partial and abstract specification modeling how a component behaves under some assumptions.
- Formally, $\mathcal{C} = (A, G)$ where:
  - the *assumption* A is an abstract description of the environment (if the component behaves according to G)
  - the *guarantee* G is an abstract description of the component (if the environment behaves according to A).

Advantages:

- Requirement-driven iterative design
- Substitutivity and reuse of components
- Independent implementability

Contracts

Idea: Abstractly specify how a component is involved in the satisfaction of a requirement φ.

A contract $\mathcal{C}$:

- Is a partial and abstract specification modeling how a component behaves under some assumptions.
- Formally, $\mathcal{C} = (A, G)$ where:
  - the *assumption* A is an abstract description of the environment (if the component behaves according to G)
  - the *guarantee* G is an abstract description of the component (if the environment behaves according to A).

Advantages:

- Requirement-driven iterative design
- Substitutivity and reuse of components
- Independent implementability

Contracts

Idea: Abstractly specify how a component is involved in the satisfaction of a requirement φ.

A contract $\mathcal{C}$:

- Is a partial and abstract specification modeling how a component behaves under some assumptions.
- Formally, $\mathcal{C} = (A, G)$ where:
  - the *assumption* A is an abstract description of the environment (if the component behaves according to G)
  - the *guarantee* G is an abstract description of the component (if the environment behaves according to A).

Advantages:

- Requirement-driven iterative design
- Substitutivity and reuse of components
- Independent implementability

Contracts

Idea: Abstractly specify how a component is involved in the satisfaction of a requirement φ.

A contract $\mathcal{C}$:

- Is a partial and abstract specification modeling how a component behaves under some assumptions.
- Formally, $\mathcal{C} = (A, G)$ where:
  - the *assumption* A is an abstract description of the environment (if the component behaves according to G)
  - the *guarantee* G is an abstract description of the component (if the environment behaves according to A).

Advantages:

- Requirement-driven iterative design
- Substitutivity and reuse of components
- Independent implementability

Goal

Provide a compositional design and verification method with contracts for the correct development of systems in SysML with respect to timed safety requirements.

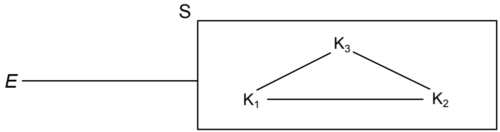
Outline

Outline

A methodology for contract-based reasoning

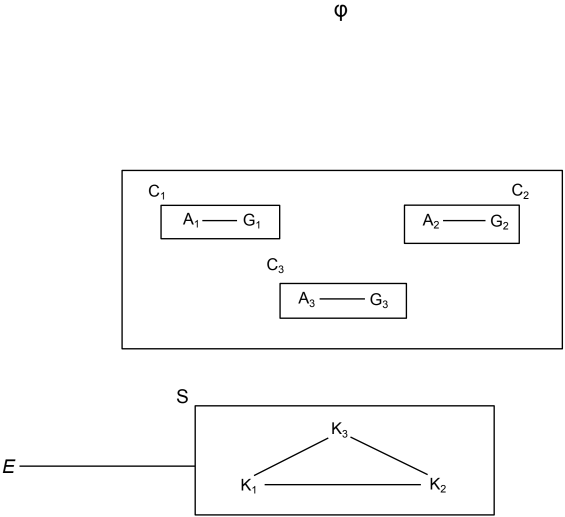
(S Quinton, S Graf: *A framework for contract-based reasoning: Motivation and application*. FLACOS 2008)

φ



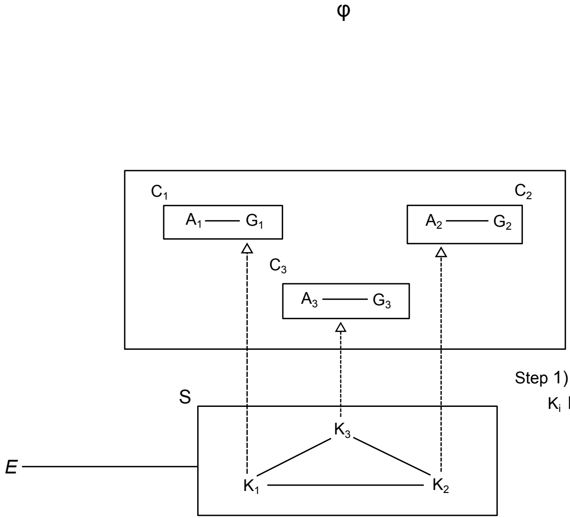
A methodology for contract-based reasoning

(S Quinton, S Graf: *A framework for contract-based reasoning: Motivation and application*. FLACOS 2008)



A methodology for contract-based reasoning

(S Quinton, S Graf: *A framework for contract-based reasoning: Motivation and application*. FLACOS 2008)

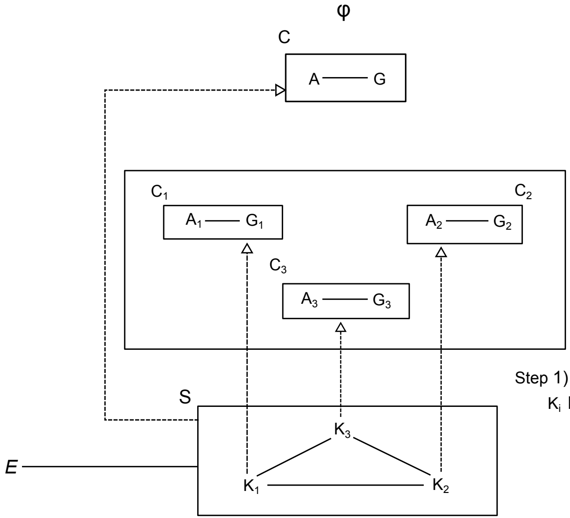


Step 1) **Satisfaction** :

$K_i \models C_i, \forall i$

A methodology for contract-based reasoning

(S Quinton, S Graf: *A framework for contract-based reasoning: Motivation and application*. FLACOS 2008)

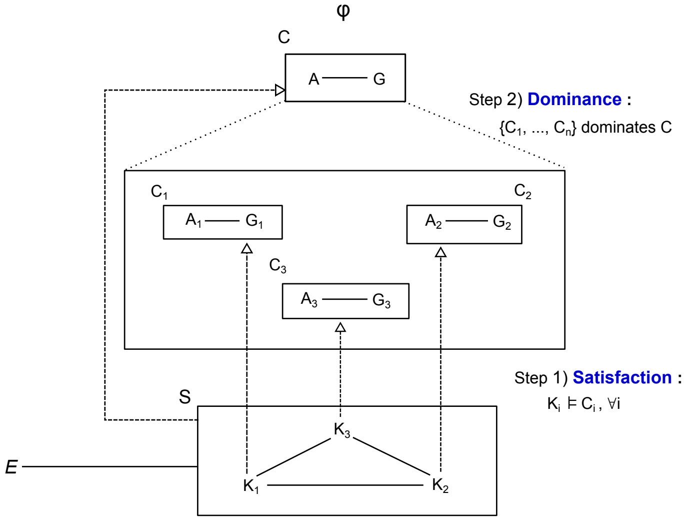


Step 1) **Satisfaction :**

$K_i \models C_i, \forall i$

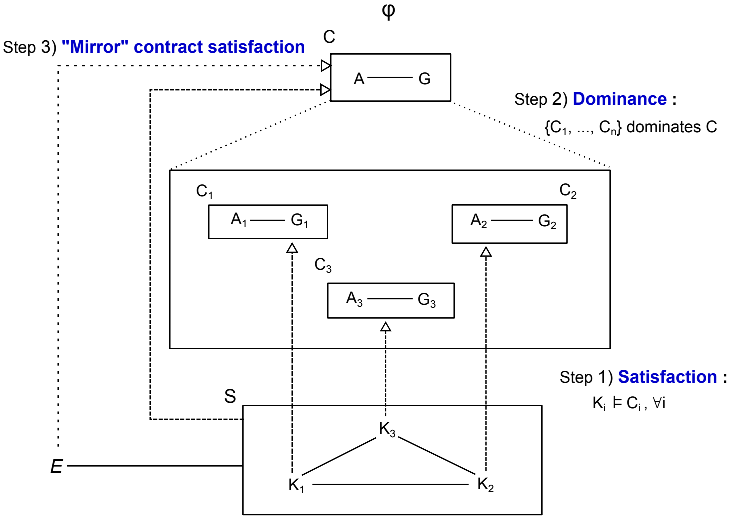
A methodology for contract-based reasoning

(S Quinton, S Graf: *A framework for contract-based reasoning: Motivation and application*. FLACOS 2008)



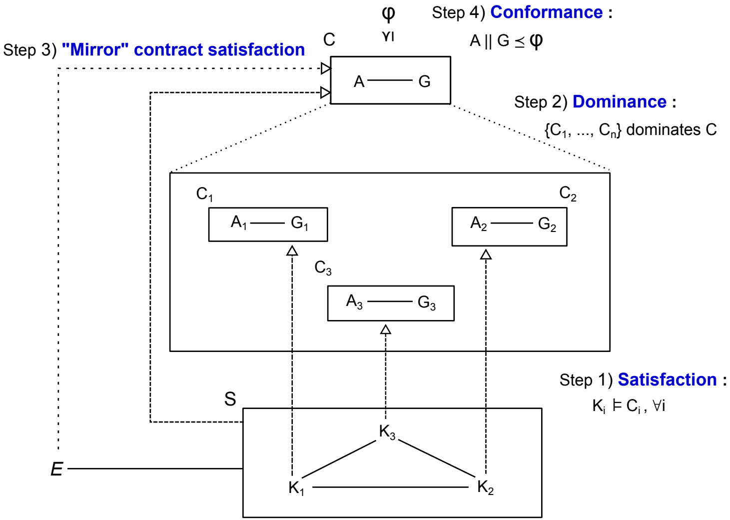
A methodology for contract-based reasoning

(S Quinton, S Graf: *A framework for contract-based reasoning: Motivation and application*. FLACOS 2008)

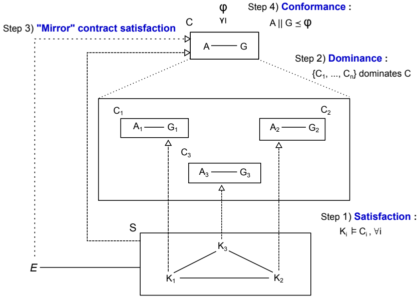


A methodology for contract-based reasoning

(S Quinton, S Graf: *A framework for contract-based reasoning: Motivation and application*. FLACOS 2008)



A meta-theory of contracts



Parameters to be instantiated:

- formal model of components
- conformance relation ($\preceq$)
- satisfaction relation ($\models$)

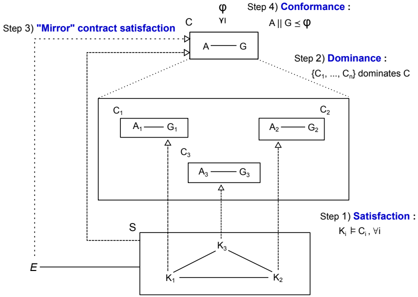
Prerequisites concerning parameters:

- compositionality of $\models$
- soundness of circular reasoning

The meta-theory provides the sufficient conditions for dominance:

- $G_1 \parallel G_2 \parallel G_3 \models (A, G)$ and
- $A \parallel G_2 \parallel G_3 \models (G_1, A_1)$ and
- $G_1 \parallel A \parallel G_3 \models (G_2, A_2)$ and
- $G_1 \parallel G_2 \parallel A \models (G_3, A_3)$

A meta-theory of contracts



Parameters to be instantiated:

- formal model of components
- conformance relation ($\preceq$)
- satisfaction relation ($\models$)

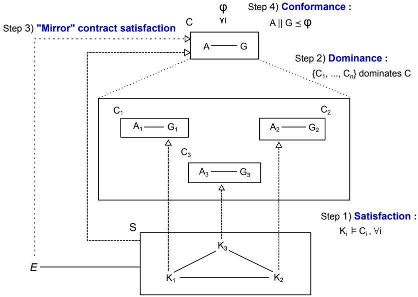
Prerequisites concerning parameters:

- compositionality of $\models$
- soundness of circular reasoning

The meta-theory provides the sufficient conditions for dominance:

- $G_1 \parallel G_2 \parallel G_3 \models (A, G)$ and
- $A \parallel G_2 \parallel G_3 \models (G_1, A_1)$ and
- $G_1 \parallel A \parallel G_3 \models (G_2, A_2)$ and
- $G_1 \parallel G_2 \parallel A \models (G_3, A_3)$

A meta-theory of contracts



Parameters to be instantiated:

- formal model of components
- conformance relation ($\preceq$)
- satisfaction relation ($\models$)

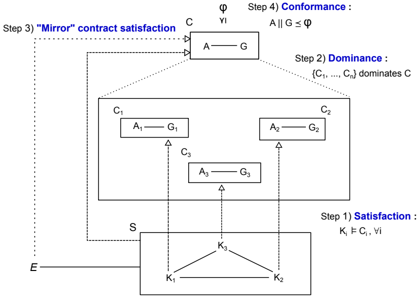
Prerequisites concerning parameters:

- compositionality of $\models$
- soundness of circular reasoning

The meta-theory provides the sufficient conditions for dominance:

- $G_1 \parallel G_2 \parallel G_3 \models (A, G)$ and
- $A \parallel G_2 \parallel G_3 \models (G_1, A_1)$ and
- $G_1 \parallel A \parallel G_3 \models (G_2, A_2)$ and
- $G_1 \parallel G_2 \parallel A \models (G_3, A_3)$

A meta-theory of contracts



Parameters to be instantiated:

- formal model of components
- conformance relation ($\preceq$)
- satisfaction relation ($\models$)

Prerequisites concerning parameters:

- compositionality of $\models$
- soundness of circular reasoning

The meta-theory provides the sufficient conditions for dominance:

- $G_1 \parallel G_2 \parallel G_3 \models (A, G)$ and
- $A \parallel G_2 \parallel G_3 \models (G_1, A_1)$ and
- $G_1 \parallel A \parallel G_3 \models (G_2, A_2)$ and
- $G_1 \parallel G_2 \parallel A \models (G_3, A_3)$

Contributions

Providing a contract-based theory by instancing the meta-theory defined by S Quinton et al. (2008) for SysML components.

Theoretical contributions:

1. Defining the syntax of the contract-related notions in SysML
2. Formalizing the semantics of the SysML component language with a variant of Timed Input/Output Automata
3. Defining a sound contract framework for Timed Input/Output Automata and timed safety properties
4. Providing a model-checking method for verifying proof obligations

Contributions

Providing a contract-based theory by instancing the meta-theory defined by S Quinton et al. (2008) for SysML components.

Practical contributions:

1. Defining and formalizing with OCL a set of well-formedness rules for ensuring the syntax compliance to the meta-theory (using Topcased[a])
2. Implementing the SysML to Timed Input/Output Automata formalization in the IFx2 Toolset[b]
3. Applying the approach on the ATV SGS industrial-scale system design

[a] http://polarsys.org/
[b] http://www.irit.fr/ifx/

Outline

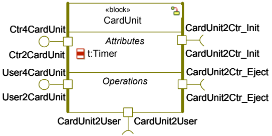
Outline

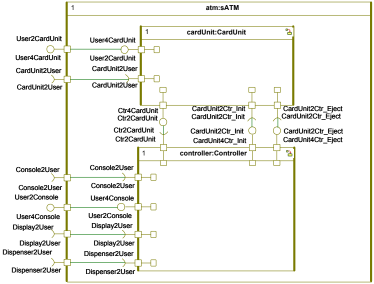
A SysML subset for modeling hierarchical systems

- Structure
  - SysML Block Definition Diagrams & Internal Block Diagrams
  - Blocks with properties and state machines, interconnection elements and relationships
  - Interfaces and signals
- Discrete behavior
  - State machines
  - Asynchronous communication through signals

A SysML subset for modeling hierarchical systems

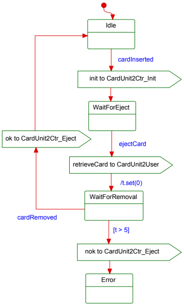
- Structure
  - SysML Block Definition Diagrams & Internal Block Diagrams
  - Blocks with properties and state machines, interconnection elements and relationships
  - Interfaces and signals
- Discrete behavior
  - State machines
  - Asynchronous communication through signals





A SysML subset for modeling hierarchical systems

- Structure
  - SysML Block Definition Diagrams & Internal Block Diagrams
  - Blocks with properties and state machines, interconnection elements and relationships
  - Interfaces and signals
- Discrete behavior
  - State machines
  - Asynchronous communication through signals

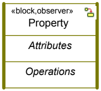


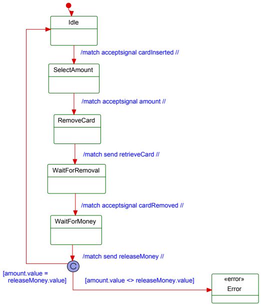
Real-time and safety properties: the OMEGA profile

- Real time
  - Continuous time model
  - Clocks specified by the type Timer
    $\Rightarrow$ allows to model time guards
  - Transition urgency (from Timed Automata with urgency)
- Observers
  - Formalizes a safety property
  - Consists of an object monitoring the system's events and gives verdicts about the requirement satisfaction

Real-time and safety properties: the OMEGA profile

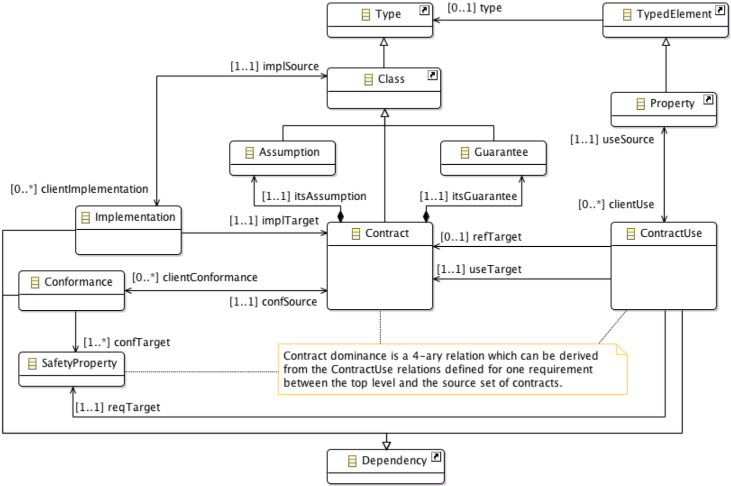
- Real time
  - Continuous time model
  - Clocks specified by the type Timer
    $\Rightarrow$ allows to model time guards
  - Transition urgency (from Timed Automata with urgency)
- Observers
  - Formalizes a safety property
  - Consists of an object monitoring the system's events and gives verdicts about the requirement satisfaction





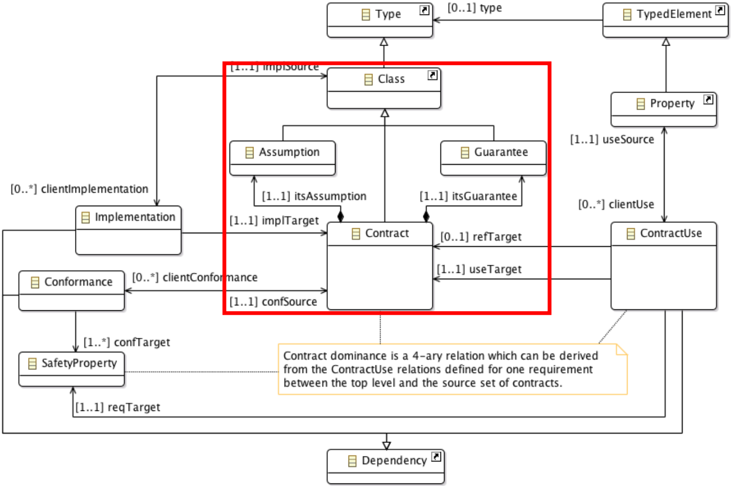
A domain meta-model for contracts

Extending UML meta-model



A domain meta-model for contracts

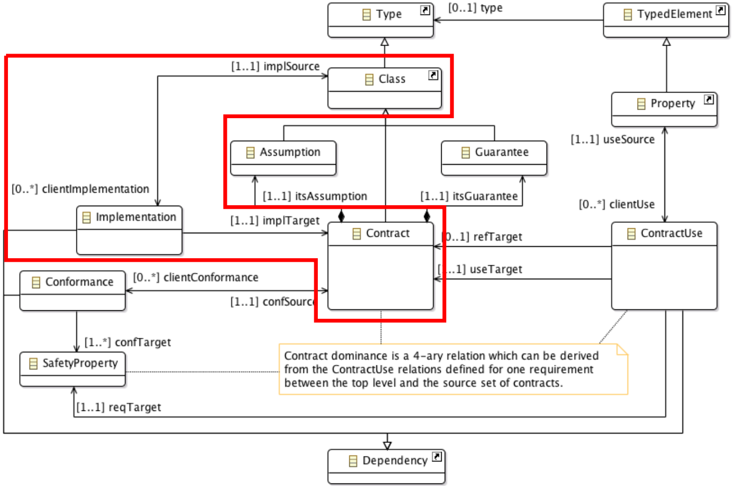
Extending UML meta-model



A contract is a closed composite structure formed of one assumption and one guarantee.

A domain meta-model for contracts

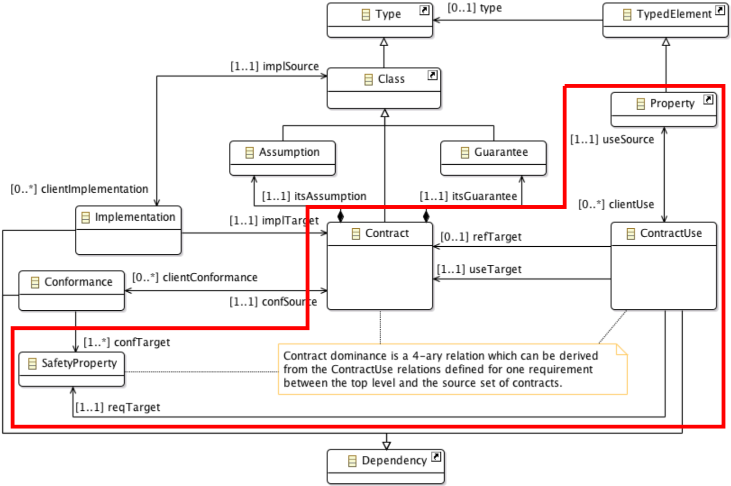
Extending UML meta-model



A class implements 0..* contracts.
(Ports of G must partially match those of class.)

A domain meta-model for contracts

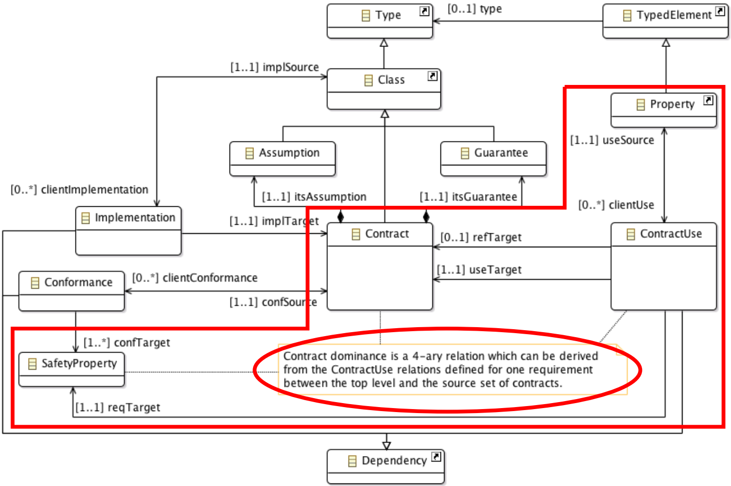
Extending UML meta-model



A contract is *used* in the context of a *part* of a larger structure. (The part's type must *implement* the contract in this case.)

A domain meta-model for contracts

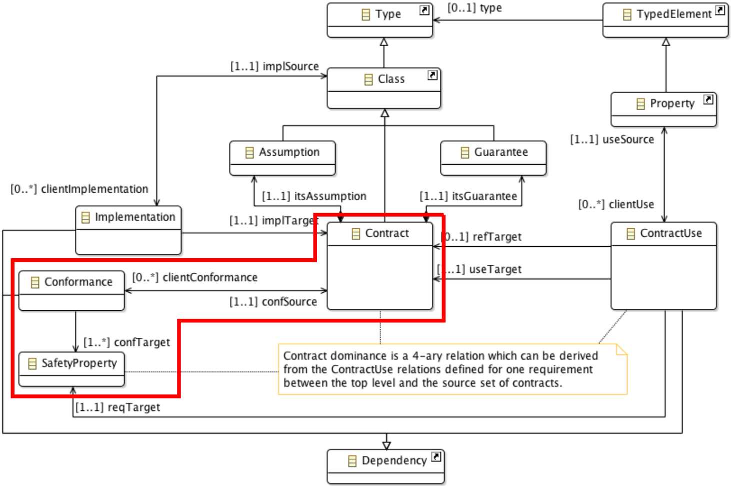
Extending UML meta-model



A set of contracts for the parts of a larger structure can refine (dominate) the contract of the structure.
(Ports of G must partially match those of refining contracts' Gs.)

A domain meta-model for contracts

Extending UML meta-model



A contracts conforms to (satisfies) a safety property.

Outline

Timed Input/Output Automaton

(a variant of D Kaynar, N Lynch, R Segala, F Vaandrager: *The Theory of Timed I/O Automata*. Morgan&Claypool Publ., 2010)

The semantics of a UML/SysML state machine can be formalized by a *timed input/output automaton*.

Definition

Timed input/output automaton $\mathcal{A} = (X, Clk, Q, \theta, I, O, V, H, D, \mathcal{T})$.

Differences wrt D Kaynar et al. (2010) due to the SysML semantics:

- modeling of visible actions besides *inputs*, *outputs* and *internals*,
- *trajectories* restricted to the linear function with slope 1 for clocks and constant for discrete variables.

Behavior:

- Execution: sequence of trajectories and actions.
- Trace: sequence of time-passage lengths and external actions (from $I \cup O \cup V$).

Timed Input/Output Automaton

(a variant of D Kaynar, N Lynch, R Segala, F Vaandrager: *The Theory of Timed I/O Automata*. Morgan&Claypool Publ., 2010)

The semantics of a $\mathrm{UML/SysML}$ state machine can be formalized by a *timed input/output automaton*.

Definition

Timed input/output automaton $\mathcal{A} = (X, Clk, Q, \theta, I, O, V, H, D, \mathcal{T})$.

Differences wrt D Kaynar et al. (2010) due to the SysML semantics:

- modeling of visible actions besides *inputs*, *outputs* and *internals*,
- *trajectories* restricted to the linear function with slope 1 for clocks and constant for discrete variables.

Behavior:

- Execution: sequence of trajectories and actions.
- Trace: sequence of time-passage lengths and external actions (from $I \cup O \cup V$).

Timed Input/Output Automaton

(a variant of D Kaynar, N Lynch, R Segala, F Vaandrager: *The Theory of Timed I/O Automata*. Morgan&Claypool Publ., 2010)

The semantics of a UML/SysML state machine can be formalized by a *timed input/output automaton*.

Definition

Timed input/output automaton $\mathcal{A} = (X, Clk, Q, \theta, I, O, V, H, D, \mathcal{T})$.

Differences wrt D Kaynar et al. (2010) due to the SysML semantics:

- modeling of visible actions besides *inputs*, *outputs* and *internals*,
- *trajectories* restricted to the linear function with slope 1 for clocks and constant for discrete variables.

Behavior:

- Execution: sequence of trajectories and actions.
- Trace: sequence of time-passage lengths and external actions (from $I \cup O \cup V$).

Timed Input/Output Automaton

(a variant of D Kaynar, N Lynch, R Segala, F Vaandrager: *The Theory of Timed I/O Automata*. Morgan&Claypool Publ., 2010)

The semantics of a UML/SysML state machine can be formalized by a *timed input/output automaton*.

Definition

Timed input/output automaton $\mathcal{A} = (X, Clk, Q, \theta, I, O, V, H, D, \mathcal{T})$.

Differences wrt D Kaynar et al. (2010) due to the SysML semantics:

- modeling of visible actions besides *inputs*, *outputs* and *internals*,
- *trajectories* restricted to the linear function with slope 1 for clocks and constant for discrete variables.

Behavior:

- Execution: sequence of trajectories and actions.
- Trace: sequence of time-passage lengths and external actions (from $I \cup O \cup V$).

TIOA parallel composition

- Composition compatibility: $\mathcal{A}_i \parallel \mathcal{A}_j$ defined iff
  $X_i \cap X_j = Clk_i \cap Clk_j = O_i \cap O_j = I_i \cap I_j = H_i \cap A_j = V_i \cap A_j = \emptyset$, for $i \neq j$
- Synchronization on common I/O actions, interleaving of other actions
- Difference wrt D Kaynar et al. (2010): an input of $\mathcal{A}_i$ synchronized with an output of $\mathcal{A}_j$, $i \neq j$, becomes a visible action in $\mathcal{A}_i \parallel \mathcal{A}_j$
  $\Rightarrow$ closer to the semantics of SysML signals
  $\Rightarrow$ broadcasts are forbidden

Theorem

The parallel composition operator is commutative and associative.

TIOA parallel composition

- Composition compatibility: $\mathcal{A}_i \parallel \mathcal{A}_j$ defined iff $X_i \cap X_j = Clk_i \cap Clk_j = O_i \cap O_j = I_i \cap I_j = H_i \cap A_j = V_i \cap A_j = \emptyset$, for $i \neq j$
- Synchronization on common I/O actions, interleaving of other actions
- Difference wrt D Kaynar et al. (2010): an input of $\mathcal{A}_i$ synchronized with an output of $\mathcal{A}_j$, $i \neq j$, becomes a visible action in $\mathcal{A}_i \parallel \mathcal{A}_j$
  $\Rightarrow$ closer to the semantics of SysML signals
  $\Rightarrow$ broadcasts are forbidden

Theorem

The parallel composition operator is commutative and associative.

TIOA parallel composition

- Composition compatibility: $\mathcal{A}_i \parallel \mathcal{A}_j$ defined iff $X_i \cap X_j = Clk_i \cap Clk_j = O_i \cap O_j = I_i \cap I_j = H_i \cap A_j = V_i \cap A_j = \emptyset$, for $i \neq j$
- Synchronization on common I/O actions, interleaving of other actions
- Difference wrt D Kaynar et al. (2010): an input of $\mathcal{A}_i$ synchronized with an output of $\mathcal{A}_j$, $i \neq j$, becomes a visible action in $\mathcal{A}_i \parallel \mathcal{A}_j$
  $\Rightarrow$ closer to the semantics of SysML signals
  $\Rightarrow$ broadcasts are forbidden

Theorem

The parallel composition operator is commutative and associative.

Conformance relation

Comparable components: $I_i \cup O_i \cup V_i = I_j \cup O_j \cup V_j$, $i \neq j$

Definition

Let K_1 and K_2 be two comparable components. $K_1 \preceq K_2$ if $traces_{K_1} \subseteq traces_{K_2}$.

Theorem

Conformance is a preorder relation.

Theorem (Composability)

Let K_1 and K_2 be two comparable components with $K_1 \preceq K_2$ and E a component compatible with both K_1 and K_2. Then $K_1 \parallel E \preceq K_2 \parallel E$.

(Straightforward extensions of results from D Kaynar et al. (2010).)

Conformance relation

Comparable components: $I_i \cup O_i \cup V_i = I_j \cup O_j \cup V_j$, $i \neq j$

Definition

Let K_1 and K_2 be two comparable components. $K_1 \preceq K_2$ if $traces_{K_1} \subseteq traces_{K_2}$.

Theorem

Conformance is a preorder relation.

Theorem (Composability)

Let K_1 and K_2 be two comparable components with $K_1 \preceq K_2$ and E a component compatible with both K_1 and K_2. Then $K_1 \parallel E \preceq K_2 \parallel E$.

(Straightforward extensions of results from D Kaynar et al. (2010).)

Conformance relation

Comparable components: $I_i \cup O_i \cup V_i = I_j \cup O_j \cup V_j$, $i \neq j$

Definition

Let K_1 and K_2 be two comparable components. $K_1 \preceq K_2$ if $traces_{K_1} \subseteq traces_{K_2}$.

Theorem

Conformance is a preorder relation.

Theorem (Composability)

Let K_1 and K_2 be two comparable components with $K_1 \preceq K_2$ and E a component compatible with both K_1 and K_2. Then $K_1 \parallel E \preceq K_2 \parallel E$.

(Straightforward extensions of results from D Kaynar et al. (2010).)

Mapping SysML to TIOA

- Limited to the identified SysML subset for modeling hierarchical component-based systems
- Similar with related transformations
- An atomic component K_i is a TIOA $\mathcal{A}_{K_i}$
  - features $\rightsquigarrow$ internal variables
  - two predefined internal variables *location* and *queue* (for asynchronous communication)
  - state machine transitions $\rightsquigarrow$ sets of TIOA transitions
  - triggers $\rightsquigarrow$ internal actions
  - ...
- A composed component K is the TIOA obtained by applying the parallel composition on the corresponding TIOA components
- An observer O is a TIOA $\mathcal{A}_O$ with only visible actions

Outline

Formal contract

Some terminology :

- *Component* K: a timed input/output automaton.
- *Signature* of K: $I \cup O \cup V$.
- Closed component: $I = O = \emptyset$.
- Open component: a component that it is not closed.
- Environment E for K: a timed input/output automaton compatible with K such that $I_E \subseteq O_K$ and $O_E \subseteq I_K$.

Definition

A *contract* $\mathcal{C}$ for a component K is a pair (A, G) of TIOA such that $I_A = O_G$ and $O_A = I_G$ (i.e. the composition is a closed system), and $I_G \subseteq I_K$, $O_G \subseteq O_K$ and $V_G \subseteq V_K$ (i.e. the signature of K is a refinement of that of G).

Formal contract

Some terminology :

- *Component* K: a timed input/output automaton.
- *Signature* of K: $I \cup O \cup V$.
- Closed component: $I = O = \emptyset$.
- Open component: a component that it is not closed.
- Environment E for K: a timed input/output automaton compatible with K such that $I_E \subseteq O_K$ and $O_E \subseteq I_K$.

Definition

A *contract* $\mathcal{C}$ for a component K is a pair (A, G) of TIOA such that $I_A = O_G$ and $O_A = I_G$ (i.e. the composition is a closed system), and $I_G \subseteq I_K$, $O_G \subseteq O_K$ and $V_G \subseteq V_K$ (i.e. the signature of K is a refinement of that of G).

Refinement under context relation

Definition

Let K_1 and K_2 be two components such that $I_{K_2} \subseteq I_{K_1} \cup V_{K_1}$, $O_{K_2} \subseteq O_{K_1} \cup V_{K_1}$ and $V_{K_2} \subseteq V_{K_1}$. Let E be an environment for K_1 compatible with both K_1 and K_2. We say that K_1 *refines* K_2 *in the context of* E, denoted $K_1 \sqsubseteq_E K_2$, if

$$K_1 \parallel E \parallel E' \preceq K_2 \parallel E \parallel K' \parallel E'$$

where K' and E' are *chaotic components* defined such that both members of the conformance relation are *comparable* and *closed*.

Definition

$K \models \mathcal{C} = (A, G) \Leftrightarrow K \sqsubseteq_A G$

Refinement under context relation

Definition

Let K_1 and K_2 be two components such that $I_{K_2} \subseteq I_{K_1} \cup V_{K_1}$, $O_{K_2} \subseteq O_{K_1} \cup V_{K_1}$ and $V_{K_2} \subseteq V_{K_1}$. Let E be an environment for K_1 compatible with both K_1 and K_2. We say that *K_1 refines K_2 in the context of E*, denoted $K_1 \sqsubseteq_E K_2$, if

$$K_1 \parallel E \parallel E' \preceq K_2 \parallel E \parallel K' \parallel E'$$

where K' and E' are *chaotic components* defined such that both members of the conformance relation are *comparable* and *closed*.

Definition

$K \models \mathcal{C} = (A, G) \Leftrightarrow K \sqsubseteq_A G$

Refinement under context relation

Definition

Let K_1 and K_2 be two components such that $I_{K_2} \subseteq I_{K_1} \cup V_{K_1}$, $O_{K_2} \subseteq O_{K_1} \cup V_{K_1}$ and $V_{K_2} \subseteq V_{K_1}$. Let E be an environment for K_1 compatible with both K_1 and K_2. We say that *K_1 refines K_2 in the context of E*, denoted $K_1 \sqsubseteq_E K_2$, if

$$K_1 \parallel E \parallel E' \preceq K_2 \parallel E \parallel K' \parallel E'$$

where K' and E' are *chaotic components* defined such that both members of the conformance relation are *comparable* and *closed*.

Definition

$K \models \mathcal{C} = (A, G) \Leftrightarrow K \sqsubseteq_A G$

Properties of refinement under context

Theorem

Given a set $\mathcal{K}$ of comparable components and a fixed environment E for that interface, the refinement under context relation $\sqsubseteq_E$ is a preorder over $\mathcal{K}$.

Proposition

Let K_1, K_2 and K_3 be three components not necessarily comparable and E an environment such that $K_1 \sqsubseteq_E K_2$ and $K_2 \sqsubseteq_E K_3$. Then $K_1 \sqsubseteq_E K_3$.

Theorem (Compositionality)

Let K_1 and K_2 be two components and E an environment compatible with both K_1 and K_2 such that $E = E_1 \parallel E_2$.

$$K_1 \sqsubseteq_{E_1 \parallel E_2} K_2 \Leftrightarrow K_1 \parallel E_1 \sqsubseteq_{E_2} K_2 \parallel E_1$$

Properties of refinement under context

Theorem

Given a set $\mathcal{K}$ of comparable components and a fixed environment E for that interface, the refinement under context relation $\sqsubseteq_E$ is a preorder over $\mathcal{K}$.

Proposition

Let K_1, K_2 and K_3 be three components not necessarily comparable and E an environment such that $K_1 \sqsubseteq_E K_2$ and $K_2 \sqsubseteq_E K_3$. Then $K_1 \sqsubseteq_E K_3$.

Theorem (Compositionality)

Let K_1 and K_2 be two components and E an environment compatible with both K_1 and K_2 such that $E = E_1 \parallel E_2$.

$$K_1 \sqsubseteq_{E_1 \parallel E_2} K_2 \Leftrightarrow K_1 \parallel E_1 \sqsubseteq_{E_2} K_2 \parallel E_1$$

Properties of refinement under context

Theorem

Given a set $\mathcal{K}$ of comparable components and a fixed environment E for that interface, the refinement under context relation $\sqsubseteq_E$ is a preorder over $\mathcal{K}$.

Proposition

Let K_1, K_2 and K_3 be three components not necessarily comparable and E an environment such that $K_1 \sqsubseteq_E K_2$ and $K_2 \sqsubseteq_E K_3$. Then $K_1 \sqsubseteq_E K_3$.

Theorem (Compositionality)

Let K_1 and K_2 be two components and E an environment compatible with both K_1 and K_2 such that $E = E_1 \parallel E_2$.

$$K_1 \sqsubseteq_{E_1 \parallel E_2} K_2 \Leftrightarrow K_1 \parallel E_1 \sqsubseteq_{E_2} K_2 \parallel E_1$$

Correctness of circular reasoning

Central result, required for applying the meta-theory:

Theorem (Circular reasoning)

Let K be a component, E its environment and $\mathcal{C} = (A, G)$ the contract for K such that K and G are compatible with each of E and A. If the following conditions hold:

1. $traces_G$ *is closed under limits*
2. $traces_G$ *is closed under time-extension*
3. $K \sqsubseteq_A G$
4. $E \sqsubseteq_G A$

then $K \sqsubseteq_E G$.

Contract dominance

Follows directly from the correctness of circular reasoning:

Theorem (Sufficient condition for dominance)

$\{\mathcal{C}_i\}_{i=1}^n$ dominates $\mathcal{C}$ if, $\forall i$, $traces_{G_i}$ and $traces_G$ are closed under limits and under time-extension and

$$\begin{cases} G_1 \parallel \dots \parallel G_n \sqsubseteq_A G \\ A \parallel G_1 \parallel \dots \parallel G_{i-1} \parallel G_{i+1} \parallel \dots \parallel G_n \sqsubseteq_{G_i} A_i, \ \forall i \end{cases}$$

(Result presented in S Quiton et al. and adapted to our notation and prerequisites.)

$\Rightarrow$ lists the proof obligations that need to be discharged when applying the verification methodology

Outline

Automated verification of proof obligations

Timed trace inclusion is undecidable.

⇒ Verify the generated proof obligations by model-checking.

1. Transform the deterministic safety property into a timed property automaton (i.e. observer) by negating the formalized property.
2. Compute the observer composition $\bowtie$ between the component under study and the timed property automaton.
3. Apply reachability analysis for the error state π.

Automated verification of proof obligations

Timed trace inclusion is undecidable.

$\Rightarrow$ Verify the generated proof obligations by model-checking.

1. Transform the deterministic safety property into a timed property automaton (i.e. observer) by negating the formalized property.
2. Compute the observer composition $\bowtie$ between the component under study and the timed property automaton.
3. Apply reachability analysis for the error state π.

Outline

Evaluation of our approach: expressiveness of contracts

1. The component playing the role of the guarantee is a safety property.
   - the automaton is non-Zeno
   - all internal transitions are eager
   - all outputs/visible actions are lazy

   ⇒ the execution of an output/visible action cannot be guaranteed
   … yet, if an output/visible action is executed, the execution can be performed before or after a deadline

2. The safety property must be deterministic for applying automatic verification.
   ⇒ assumptions must satisfy this condition for automatically proving contract dominance

Evaluation of our approach: expressiveness of contracts

1. The component playing the role of the guarantee is a safety property.
   - the automaton is non-Zeno
   - all internal transitions are eager
   - all outputs/visible actions are lazy

   ⇒ the execution of an output/visible action cannot be guaranteed
   … yet, if an output/visible action is executed, the execution can be performed before or after a deadline

2. The safety property must be deterministic for applying automatic verification.
   ⇒ assumptions must satisfy this condition for automatically proving contract dominance

Evaluation of our approach: expressiveness of contracts

1. The component playing the role of the guarantee is a safety property.
    - the automaton is non-Zeno
    - all internal transitions are eager
    - all outputs/visible actions are lazy

   ⇒ the execution of an output/visible action cannot be guaranteed
   ... yet, if an output/visible action is executed, the execution can be performed before or after a deadline

2. The safety property must be deterministic for applying automatic verification.
   ⇒ assumptions must satisfy this condition for automatically proving contract dominance

Evaluation of our approach: expressiveness of contracts

1. The component playing the role of the guarantee is a safety property.
    - the automaton is non-Zeno
    - all internal transitions are eager
    - all outputs/visible actions are lazy

   $\Rightarrow$ the execution of an output/visible action cannot be guaranteed
   ... yet, if an output/visible action is executed, the execution can be performed before or after a deadline

2. The safety property must be deterministic for applying automatic verification.
   $\Rightarrow$ assumptions must satisfy this condition for automatically proving contract dominance

Related work

Contracts in UML/SysML:

- Provided/required interfaces for typing components: T Weiss et al. (UML 2001), ...
- OCL (pre,post) conditions for operations: M Kriegger et al. (GPCE 2010), P André et al. (IDM 2011, WCSI 2010), E Cariou et al. (ECMFA 2011), ...
- State machine on connectors for describing component compatibility: R Payne et al. (2011)

Formal contract-based theories:

- Specification theories defining quotient, logical conjunction, ...
- Defined for:
  - TIOA (ECDAR): S Bauer et al. (FASE 2012, FACS 2012), A David et al. (STTT 2012), ...
  - timed logical specifications (i.e. sets of traces): C Chilton et al. (FACS 2012, FORMATS 2012, Sci. Comp. Prog. 2014)
- Differences wrt:
  - the modeling of contracts
  - the reasoning method with contracts

Related work

Contracts in UML/SysML:

- Provided/required interfaces for typing components: T Weiss et al. (UML 2001), ...
- OCL (pre,post) conditions for operations: M Kriegger et al. (GPCE 2010), P André et al. (IDM 2011, WCSI 2010), E Cariou et al. (ECMFA 2011), ...
- State machine on connectors for describing component compatibility: R Payne et al. (2011)

Formal contract-based theories:

- Specification theories defining quotient, logical conjunction, ...
- Defined for:
  - TIOA (ECDAR): S Bauer et al. (FASE 2012, FACS 2012), A David et al. (STTT 2012), ...
  - timed logical specifications (i.e. sets of traces): C Chilton et al. (FACS 2012, FORMATS 2012, Sci. Comp. Prog. 2014)
- Differences wrt:
  - the modeling of contracts
  - the reasoning method with contracts

Outline

Outline

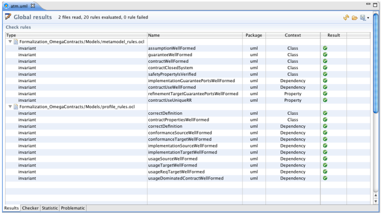
OCL formalization of profile's set of well-formedness rules

Well-formedness rules tackling:

- the meta-model's conformance to the meta-theory
- a model's conformance to the meta-model
- signature refinement of contracts for requirement-driven design

⇝ formalized with OCL in order to automatically verify the static typing of a model extended with contracts (with Topcased[1])

- 20 rules
- 480 lines of code



Type	Name	Package	Context	Result
Formalization_OmegaContracts/Models/metamodel_rules.ocl				
invariant	assumptionWellFormed	uml	Class	✓
invariant	guaranteeWellFormed	uml	Class	✓
invariant	contractWellFormed	uml	Class	✓
invariant	contractClosedSystem	uml	Class	✓
invariant	safetyPropertyIsVerified	uml	Class	✓
invariant	implementationGuaranteePortsWellFormed	uml	Dependency	✓
invariant	contractUseWellFormed	uml	Dependency	✓
invariant	refinementTargetGuaranteePortsWellFormed	uml	Property	✓
invariant	contractUseUniqueRR	uml	Property	✓
Formalization_OmegaContracts/Models/profile_rules.ocl				
invariant	correctDefinition	uml	Class	✓
invariant	contractPropertiesWellFormed	uml	Class	✓
invariant	correctDefinition	uml	Dependency	✓
invariant	conformanceSourceWellFormed	uml	Dependency	✓
invariant	conformanceTargetWellFormed	uml	Dependency	✓
invariant	implementationSourceWellFormed	uml	Dependency	✓
invariant	implementationTargetWellFormed	uml	Dependency	✓
invariant	usageSourceWellFormed	uml	Dependency	✓
invariant	usageTargetWellFormed	uml	Dependency	✓
invariant	usageReqTargetWellFormed	uml	Dependency	✓
invariant	usageDominatedContractWellFormed	uml	Dependency	✓

[1] http://polarsys.org/

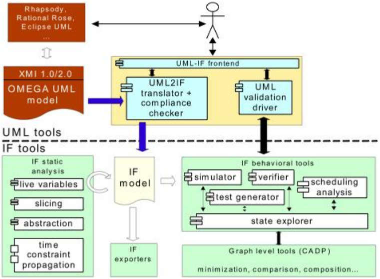
Outline

Implementation in the IFx2 toolset[2]

Goal: Early model validation and debugging

Functionalities:

- Simulation
- Static analysis: dead code/variable elimination, ...
- Model-checking: observers, state graph minimization, ...



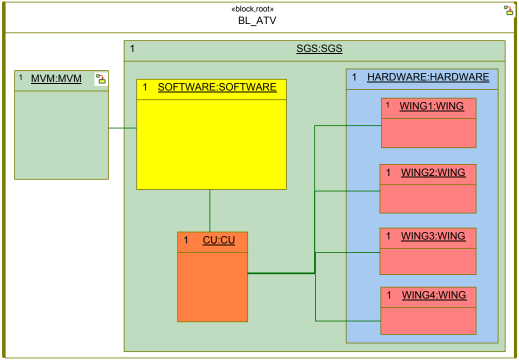
Implementation details:

- Proprietary compiler which takes as input an (OMEGA) SysML model (in XMI 2.0 format) and produces the TIOA model (in the IF language)
- Adapted for IBM Rhapsody and Papyrus
- Several compiling options available: uml/sysml, rhapsody/papyrus, rhplang, eager, ...
- Used technologies: Java, Eclipse UML 2.3, Eclipse EMF
- $\sim$ 13000 lines of code

[2] www.irit.fr/ifx/

Outline

The Solar Generation System (SGS)

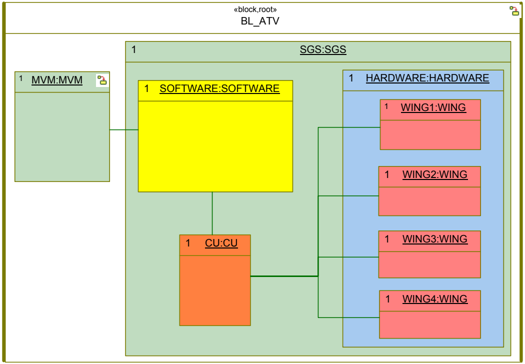


Reverse engineered from the actual system by *Airbus Defence and Space*:

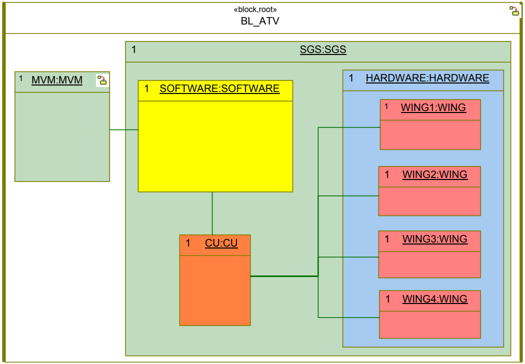
- 4-level architecture, 20 blocks and 95 block instances
- 661 port and 504 connector instances
- 1-fault tolerant
- 62 possible hardware failures

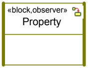
Requirement φ: At the end of the deployment sequence, all 4 WINGs are deployed.

Formalizing the requirement



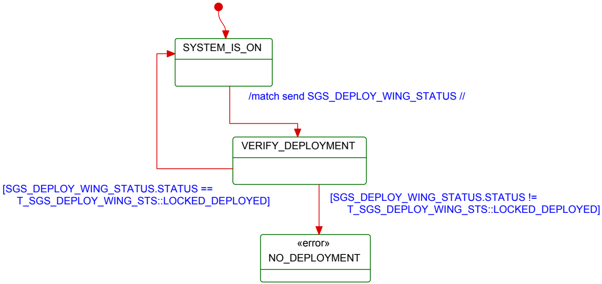
Formalizing the requirement





Formalizing the requirement

Behavior of *Property* (φ)

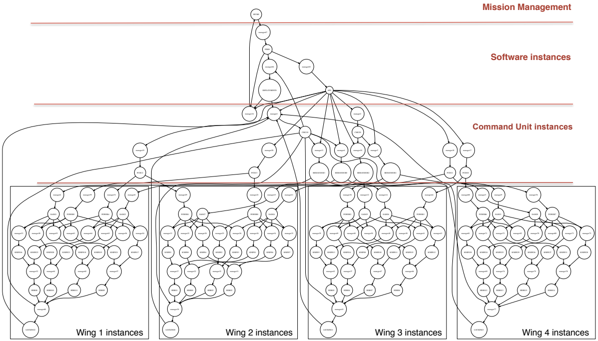


Preliminary V&V

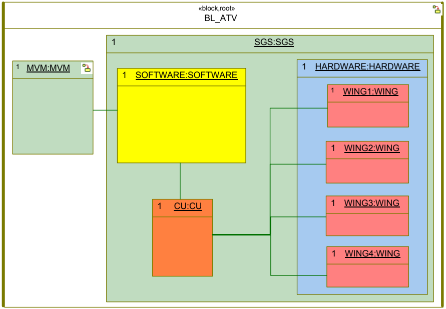
- Simulation
  - Scenario length: 2400 steps and one minute execution
  - Discovered modeling errors: unexpected message receptions
- Model-checking
  - Subject to the combinatorial state space explosion problem

Preliminary V&V

- Simulation
  - Scenario length: 2400 steps and one minute execution
  - Discovered modeling errors: unexpected message receptions
- Model-checking
  - Subject to the combinatorial state space explosion problem



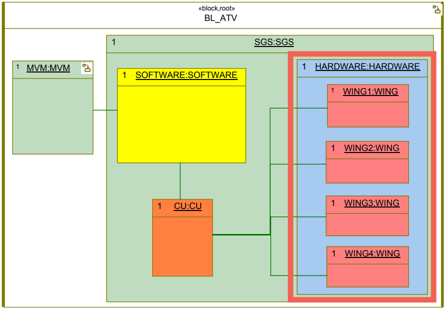
Applying contract-based reasoning



φ: At the end of the deployment sequence, all 4 WINGs are deployed.

- The system S is given by HARDWARE which contains the WINGs.
- The environment E is given by MVM $\|$ SOFTWARE $\|$ CU.

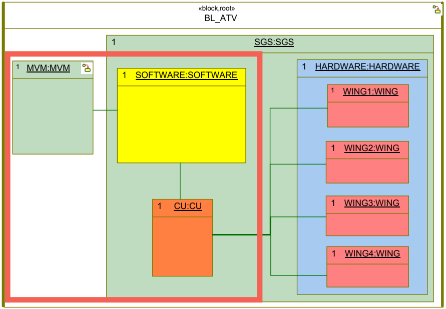
Applying contract-based reasoning



φ: At the end of the deployment sequence, all 4 WINGs are deployed.

- The system S is given by HARDWARE which contains the WINGs.
- The environment E is given by MVM || SOFTWARE || CU.

Applying contract-based reasoning

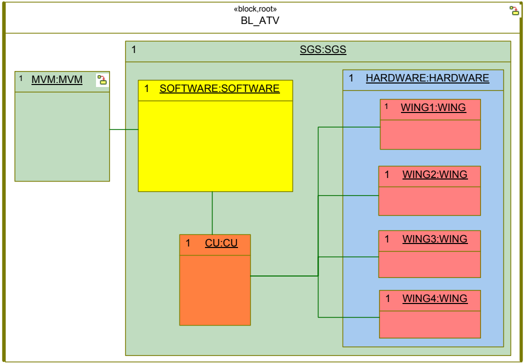


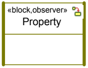
φ: At the end of the deployment sequence, all 4 WINGs are deployed.

- The system S is given by HARDWARE which contains the WINGs.
- The environment E is given by MVM $\parallel$ SOFTWARE $\parallel$ CU.

Individual contracts for wings

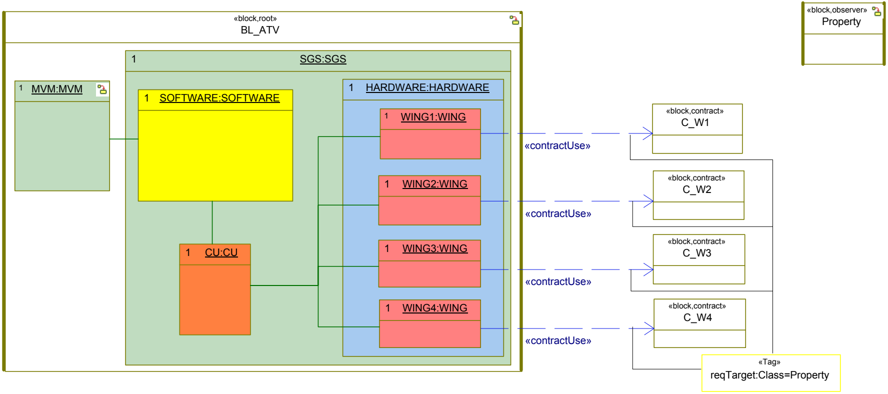
Modeling contract satisfaction





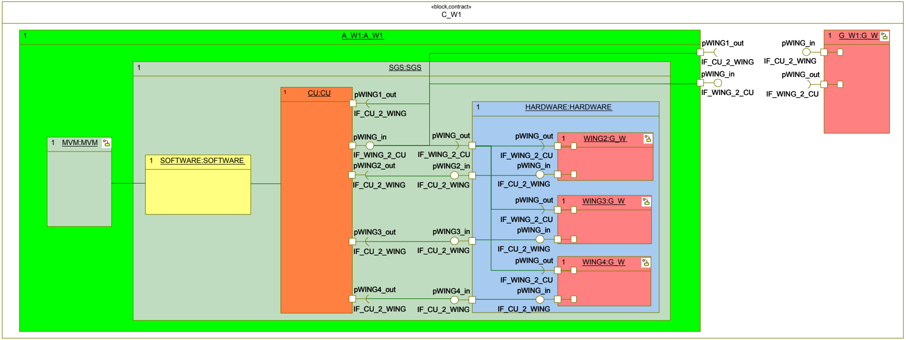
Individual contracts for wings

Modeling contract satisfaction



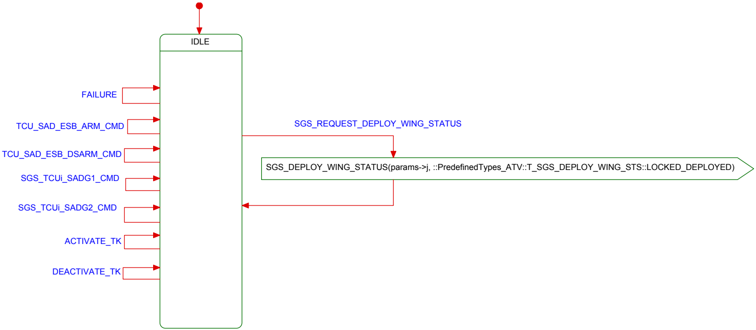
Individual contracts for wings

Contract architecture



Individual contracts for wings

Behavior of the guarantee

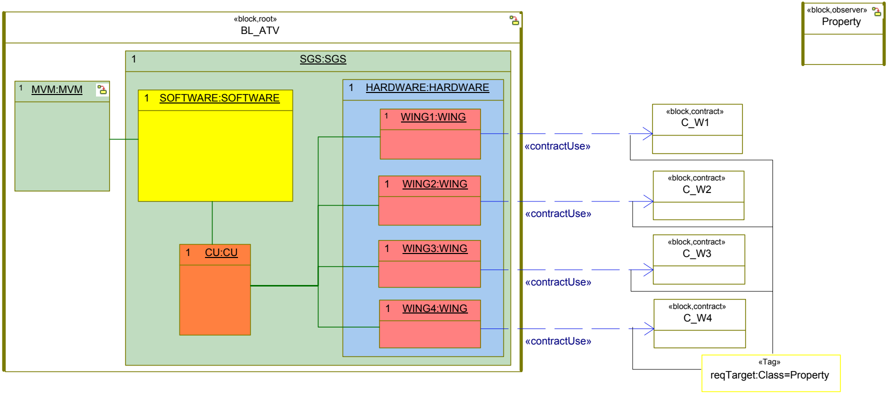


Contract satisfaction

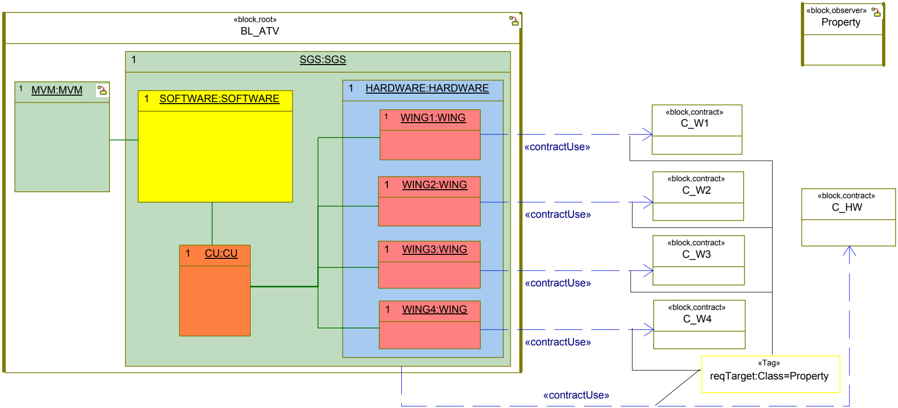
Generated proof obligations

- $WING1 \sqsubseteq_{MVM \| SOFTWARE \| CU \| WING2 \| WING3 \| WING4} G\_W1$
  where $WINGi$, $i \in 2..4$, is of type G_W
- $WING2 \sqsubseteq_{MVM \| SOFTWARE \| CU \| WING1 \| WING3 \| WING4} G\_W2$
- $WING3 \sqsubseteq_{MVM \| SOFTWARE \| CU \| WING1 \| WING2 \| WING4} G\_W3$
- $WING4 \sqsubseteq_{MVM \| SOFTWARE \| CU \| WING1 \| WING2 \| WING3} G\_W4$

Top contract for the system

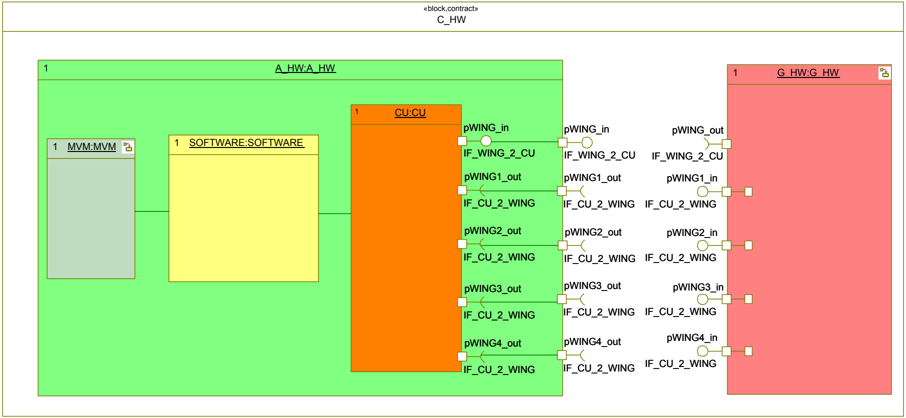


Top contract for the system



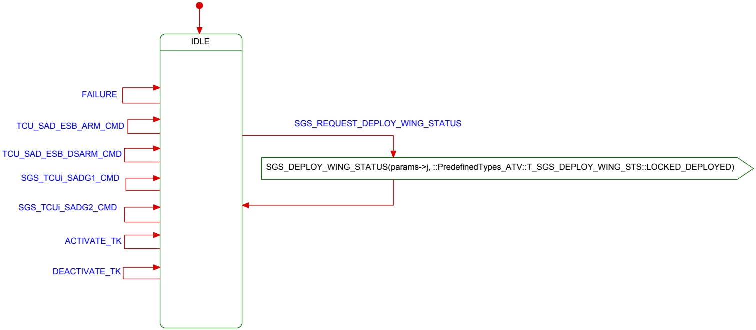
Top contract for the system

Contract architecture



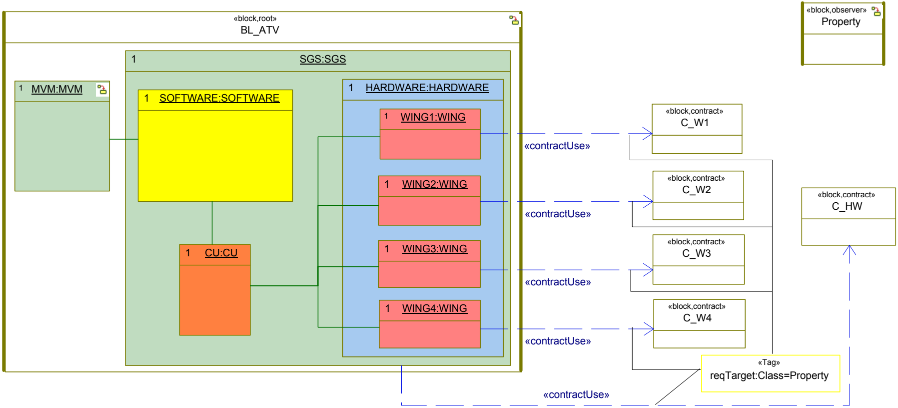
Top contract for the system

Behavior of the guarantee

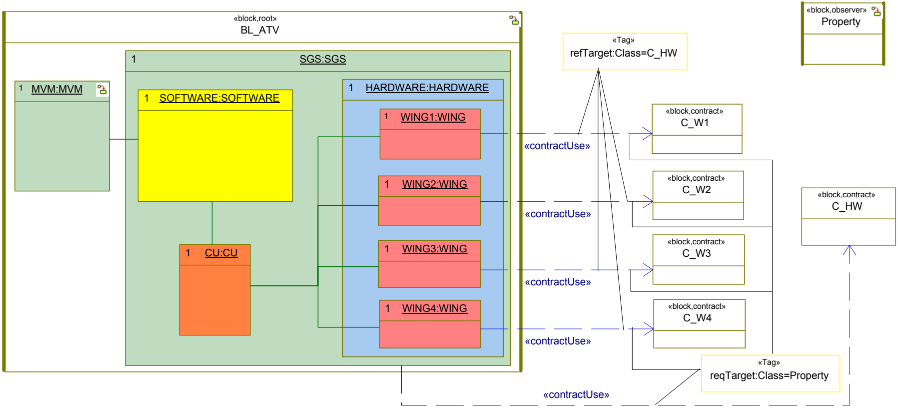


Dominance

Modeling dominance



Modeling dominance

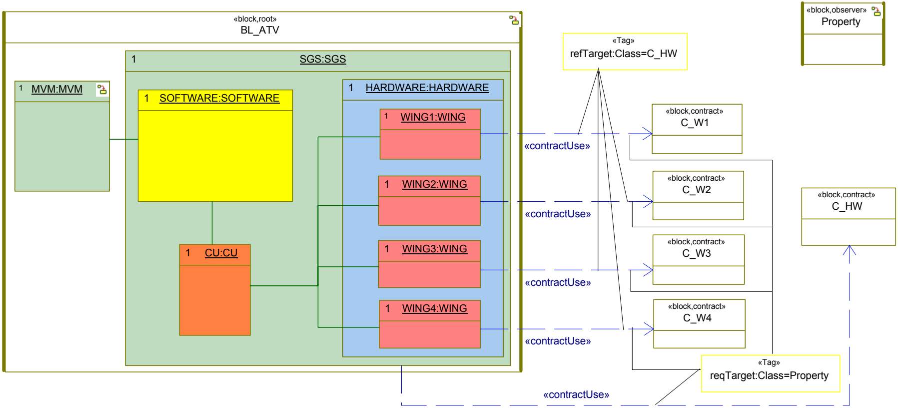


Dominance

Generated proof obligations

1. $G\_W1 \parallel G\_W2 \parallel G\_W3 \parallel G\_W4 \sqsubseteq_{MVM \parallel SOFTWARE \parallel CU} G\_HW$
2. $(MVM \parallel SOFTWARE \parallel CU) \parallel G\_W2 \parallel G\_W3 \parallel G\_W4 \sqsubseteq_{G\_W1}$ $MVM \parallel SOFTWARE \parallel CU \parallel G\_W2 \parallel G\_W3 \parallel G\_W4$
3. $(MVM \parallel SOFTWARE \parallel CU) \parallel G\_W1 \parallel G\_W3 \parallel G\_W4 \sqsubseteq_{G\_W2}$ $MVM \parallel SOFTWARE \parallel CU \parallel G\_W1 \parallel G\_W3 \parallel G\_W4$
4. $(MVM \parallel SOFTWARE \parallel CU) \parallel G\_W1 \parallel G\_W2 \parallel G\_W4 \sqsubseteq_{G\_W3}$ $MVM \parallel SOFTWARE \parallel CU \parallel G\_W1 \parallel G\_W2 \parallel G\_W4$
5. $(MVM \parallel SOFTWARE \parallel CU) \parallel G\_W1 \parallel G\_W2 \parallel G\_W3 \sqsubseteq_{G\_W4}$ $MVM \parallel SOFTWARE \parallel CU \parallel G\_W1 \parallel G\_W2 \parallel G\_W3$

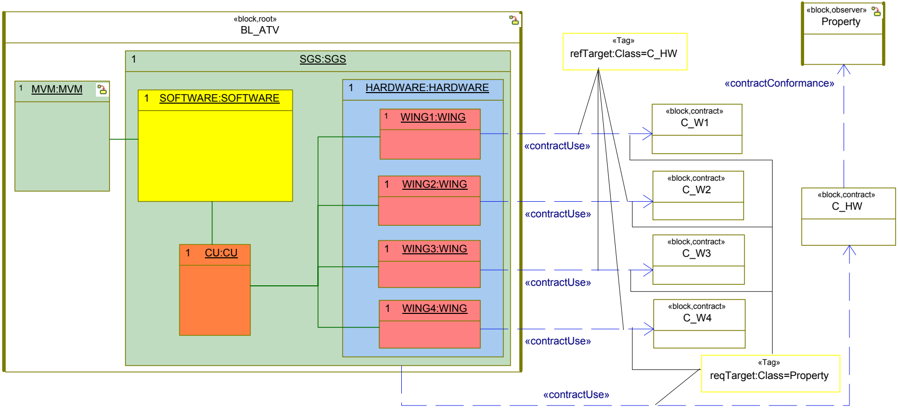
Top “mirror” contract satisfaction



⇒ generates the following proof obligation:

$$MVM \parallel SOFTWARE \parallel CU \sqsubseteq_{G\_HW} MVM \parallel SOFTWARE \parallel CU$$

Conformance



⇒ generates the following proof obligation:

$$MVM \parallel SOFTWARE \parallel CU \parallel G\_HW \preceq Property$$

Verification results

1. Contract satisfaction:
   - detected error: deployment deactivation in case of failure of disabled components
   - after correction:

	Average verification time (s)			
Type of induced failure	Wing 1	Wing 2	Wing 3	Wing 4
Thermal knife	13993	6869	18842	11412
Hold-down and release system	12672	6516	16578	9980
Solar array driving group	11527	5432	13548	6807

2. Dominance:
   - for G_HW: < 1 second
   - for assumptions: trivial
3. Top "mirror" contract satisfaction: trivial
4. Conformance < 1 second

Outline

Conclusion

Overview

Behavioral contract framework for the compositional design and verification of system models in UML/SysML with respect to timed safety requirements

Features of the developed approach:

- scalability in order to tackle the design and verification of very large systems,
- reusability for both contracts and components.

Behavioral contract framework for the compositional design and verification of system models in UML/SysML with respect to timed safety requirements

Features of the developed approach:

- scalability in order to tackle the design and verification of very large systems,
- reusability for both contracts and components.

1. An extension for modeling contracts in UML/SysML amenable to compositional verification
2. Ensured compliance with the methodology for reasoning with contracts
3. Formalization of the UML/SysML component and contract language with a variant of Timed Input/Output Automata
4. A partial implementation in the IFx2 verification tool
5. A contract theory for Timed Input/Output Automata supporting the verification of general safety properties
6. Automated model-checking for verification of contract satisfaction and deterministic safety properties
7. Experimental evidence that previously intractable models can be tamed

1. An extension for modeling contracts in UML/SysML amenable to compositional verification
2. Ensured compliance with the methodology for reasoning with contracts
3. Formalization of the UML/SysML component and contract language with a variant of Timed Input/Output Automata
4. A partial implementation in the IFx2 verification tool
5. A contract theory for Timed Input/Output Automata supporting the verification of general safety properties
6. Automated model-checking for verification of contract satisfaction and deterministic safety properties
7. Experimental evidence that previously intractable models can be tamed

Conclusion

Contributions

1. An extension for modeling contracts in UML/SysML amenable to compositional verification
2. Ensured compliance with the methodology for reasoning with contracts
3. Formalization of the UML/SysML component and contract language with a variant of Timed Input/Output Automata
4. A partial implementation in the IFx2 verification tool
5. A contract theory for Timed Input/Output Automata supporting the verification of general safety properties
6. Automated model-checking for verification of contract satisfaction and deterministic safety properties
7. Experimental evidence that previously intractable models can be tamed

Conclusion

Contributions

1. An extension for modeling contracts in UML/SysML amenable to compositional verification
2. Ensured compliance with the methodology for reasoning with contracts
3. Formalization of the UML/SysML component and contract language with a variant of Timed Input/Output Automata
4. A partial implementation in the IFx2 verification tool
5. A contract theory for Timed Input/Output Automata supporting the verification of general safety properties
6. Automated model-checking for verification of contract satisfaction and deterministic safety properties
7. Experimental evidence that previously intractable models can be tamed

1. I Dragomir, I Ober, C Percebois: *Safety Contracts for Timed Reactive Components in SysML*. SOFSEM 2014
2. I Dragomir, I Ober, C Percebois: *Integrating Verifiable Assume/Guarantee Contracts in UML/SysML*. ACES-MB 2013
3. I Dragomir, I Ober, C Percebois: *Safety contracts for reactive timed systems (extended abstract)*. GDR GPL 2013
4. I Dragomir, I Ober, D Lesens: *A Case Study in Formal System Engineering with SysML*. ICECCS 2012
5. E Conquet, F-X Dormoy, I Dragomir, S Graf, D Lesens, P Nienaltowski, I Ober: *Formal Model Driven Engineering for Space Onboard Software*. ERTS2 2012
6. Il Ober, Iu Ober, I Dragomir, E A Aboussoror: *UML/SysML semantic tunings*. ISSE 2011
7. E Conquet, F-X Dormoy, I Dragomir, A Le Guennec, D Lesens, P Nienaltowski, I Ober: *Modèles système, modèles logiciel et modèles de code dans les applications spatiales*. Génie logiciel 2011
8. I Ober, I Dragomir: *Unambiguous UML composite structures: the OMEGA2 experience*. SOFSEM 2011
9. I Dragomir, I Ober: *Well-formedness and typing rules for UML Composite Structures*. CoRR 2010
10. I Ober, I Dragomir: *OMEGA2: A new version of the profile and the tools*. UML&AADL 2010

Perspectives

Short-term perspectives:

1. Extend the contract framework in order to encompass other types of requirements, i.e. progress, etc.
2. Automate all the verification steps, provide automate assistance for building the proof tree

Long-term perspectives:

1. Provide methods or methodological guidelines for deriving intermediate contracts from the properties one is trying to prove
2. Automatically generate assumptions and guarantees
3. Perform error diagnostics on contracts both locally and globally in the proof tree and bridge the gap to the semi-formal model

Thank you!

Perspectives

Short-term perspectives:

1. Extend the contract framework in order to encompass other types of requirements, i.e. progress, etc.
2. Automate all the verification steps, provide automate assistance for building the proof tree

Long-term perspectives:

1. Provide methods or methodological guidelines for deriving intermediate contracts from the properties one is trying to prove
2. Automatically generate assumptions and guarantees
3. Perform error diagnostics on contracts both locally and globally in the proof tree and bridge the gap to the semi-formal model

Thank you!

Perspectives

Short-term perspectives:

1. Extend the contract framework in order to encompass other types of requirements, i.e. progress, etc.
2. Automate all the verification steps, provide automate assistance for building the proof tree

Long-term perspectives:

1. Provide methods or methodological guidelines for deriving intermediate contracts from the properties one is trying to prove
2. Automatically generate assumptions and guarantees
3. Perform error diagnostics on contracts both locally and globally in the proof tree and bridge the gap to the semi-formal model

Thank you!