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

⇝ 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 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
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?
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 $\varphi$.

A contract $C$:

- Is a partial and abstract specification modeling how a component behaves under some assumptions.
- Formally, $C = (A, G)$ 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 $\varphi$.

A contract $C$:

- Is a partial and abstract specification modeling how a component behaves under some assumptions.
- Formally, $C = (A, G)$ 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 \( \varphi \).

A contract \( C \):

- Is a partial and abstract specification modeling how a component behaves under some assumptions.
- Formally, \( C = (A, G) \) 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
Idea: Abstractly specify how a component is involved in the satisfaction of a requirement $\varphi$.

A contract $C$:

- Is a partial and abstract specification modeling how a component behaves under some assumptions.
- Formally, $C = (A, G)$ 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
Provide a compositional design and verification method with contracts for the correct development of systems in SysML with respect to timed safety requirements.
Outline

1. Context and Problematics
2. A Method for Reasoning with Contracts
3. Theoretical Contributions
4. Practical Contributions
5. Conclusion and Perspectives
Outline

1. Context and Problematics
2. A Method for Reasoning with Contracts
3. Theoretical Contributions
4. Practical Contributions
5. Conclusion and Perspectives
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)

\[ \varphi \]

\[ \begin{align*}
C_1 & \quad \rightarrow \quad G_1 \\
A_1 & \quad \rightarrow \quad G_1 \\
A_2 & \quad \rightarrow \quad G_2 \\
A_3 & \quad \rightarrow \quad G_3
\end{align*} \]

Step 1) **Satisfaction**: 
\[ K_i \models C_i \quad \forall i \]
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)

Step 1) Satisfaction:

Step 2) Dominance:

{C_1, ..., C_n} dominates C

K_i ⊧ C_i, ∀i

E

K_3

K_2

K_1

A_1 → G_1

A_2 → G_2

A_3 → G_3

A → G

C

C_1

C_2

C_3
A methodology for contract-based reasoning

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

Step 1) **Satisfaction**: 

\[ K_i \models C_i, \forall i \]

Step 2) **Dominance**: 

\[ \{C_1, \ldots, C_n\} \text{ dominates } C \]

Step 3) **"Mirror" contract satisfaction**: 

\[ \phi \]

\[ \varphi \]

\[ A \xrightarrow{G} C \]

\[ C_1\xrightarrow{A_1G_1} \]

\[ C_2\xrightarrow{A_2G_2} \]

\[ C_3\xrightarrow{A_3G_3} \]

\[ E \]

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

Step 1) **Satisfaction**:

Step 2) **Dominance**:

Step 3) "Mirror" contract satisfaction

Step 4) **Conformance**:

\[ \phi \preceq \mathcal{E} \]

\[ A \parallel \mathcal{G} \leq \phi \]

\{C_1, \ldots, C_n\} dominates C

K_i \models C_i, \forall i
A meta-theory of contracts

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
- $G_1 \parallel G_2 \parallel A \models (G_3, A_3)$

Parameters to be instantiated:
- formal model of components
- conformance relation ($\preceq$)
- satisfaction relation ($\models$)
A meta-theory of contracts

Prerequisites concerning parameters:
- compositionality of \(\models\)
- soundness of circular reasoning

The meta-theory provides the sufficient conditions for dominance:
- \(G_1 \parallel G_2 \parallel G_3 \models (A, G)\)
- \(A \parallel G_2 \parallel G_3 \models (G_1, A_1)\)
- \(G_1 \parallel A \parallel G_3 \models (G_2, A_2)\)
- \(G_1 \parallel G_2 \parallel A \models (G_3, A_3)\)

Parameters to be instantiated:
- formal model of components
- conformance relation \((\preceq)\)
- satisfaction relation \((\models)\)
A meta-theory of contracts

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
- $G_1 \parallel G_2 \parallel A \models (G_3, A_3)$

Parameters to be instantiated:
- formal model of components
- conformance relation ($\succeq$)
- satisfaction relation ($\models$)
A meta-theory of contracts

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
- $G_1 \parallel G_2 \parallel A \models (G_3, A_3)$

Parameters to be instantiated:
- formal model of components
- conformance relation $(\preceq)$
- satisfaction relation $(\models)$
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\textsuperscript{a})
2. Implementing the SysML to Timed Input/Output Automata formalization in the IFx2 Toolset\textsuperscript{b}
3. Applying the approach on the ATV SGS industrial-scale system design

\textsuperscript{a}http://polarsys.org/
\textsuperscript{b}http://www.irit.fr/ifx/
Outline

1. Context and Problematics

2. A Method for Reasoning with Contracts

3. Theoretical Contributions
   - Integrating Contracts in SysML
   - Modeling Components: a Timed Input/Output Automata Flavour
   - A Formal Contract Theory for TIOA
   - Automated Verification with Model-Checking
   - Evaluation and Related Work

4. Practical Contributions

5. Conclusion and Perspectives
Outline

1 Context and Problematics

2 A Method for Reasoning with Contracts

3 Theoretical Contributions
   - Integrating Contracts in SysML
   - Modeling Components: a Timed Input/Output Automata Flavour
   - A Formal Contract Theory for TIOA
   - Automated Verification with Model-Checking
   - Evaluation and Related Work

4 Practical Contributions

5 Conclusion and Perspectives
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`
    ⇒ 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

Contract dominance is a 4-ary relation which can be derived from the ContractUse relations defined for one requirement between the top level and the source set of contracts.
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 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

Contract dominance is a 4-ary relation which can be derived from the ContractUse relations defined for one requirement between the top level and the source set of contracts.

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

1. Context and Problematics

2. A Method for Reasoning with Contracts

3. Theoretical Contributions
   - Integrating Contracts in SysML
   - **Modeling Components: a Timed Input/Output Automata Flavour**
   - A Formal Contract Theory for TIOA
   - Automated Verification with Model-Checking
   - Evaluation and Related Work

4. Practical Contributions

5. Conclusion and Perspectives
Timed Input/Output Automaton

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

**Definition**

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

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

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

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

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


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

Definition

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

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

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

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

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:** $A_i \parallel A_j$ defined iff
  
  $$X_i \cap X_j = Clk_i \cap Clk_j = O_i \cap O_j = I_i \cap I_j = H_i \cap A_j = V_i \cap A_j = \emptyset, \text{ for } i \neq j$$

- Synchronization on common I/O actions, interleaving of other actions
- Difference wrt D Kaynar et al. (2010): an input of $A_i$ synchronized with an output of $A_j$, $i \neq j$, becomes a visible action in $A_i \parallel A_j$
  
  ⇒ closer to the semantics of SysML signals
  
  ⇒ broadcasts are forbidden

**Theorem**

*The parallel composition operator is commutative and associative.*
TIOA parallel composition

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

**Theorem**

The parallel composition operator is commutative and associative.
TIOA parallel composition

- Composition compatibility: $\mathcal{A}_i \parallel \mathcal{A}_j$ defined iff
  $X_i \cap X_j = \text{Clk}_i \cap \text{Clk}_j = O_i \cap O_j = I_i \cap I_j = H_i \cap A_j = V_i \cap A_j = \emptyset$, 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 \( \text{traces}_{K_1} \subseteq \text{traces}_{K_2} \).

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 \( \text{traces}_{K_1} \subseteq \text{traces}_{K_2} \).

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 \( \text{traces}_{K_1} \subseteq \text{traces}_{K_2} \).

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 $A_{K_i}$
  - features $\mapsto$ internal variables
  - two predefined internal variables location and queue (for asynchronous communication)
  - state machine transitions $\mapsto$ sets of TIOA transitions
  - triggers $\mapsto$ 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 $A_O$ with only visible actions
Outline

1. Context and Problematics

2. A Method for Reasoning with Contracts

3. Theoretical Contributions
   - Integrating Contracts in SysML
   - Modeling Components: a Timed Input/Output Automata Flavour
   - A Formal Contract Theory for TIOA
   - Automated Verification with Model-Checking
   - Evaluation and Related Work

4. Practical Contributions

5. Conclusion and Perspectives
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** $C$ 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 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** $C$ 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

$K \models C = (A, G) \iff K \sqsubseteq_A G$
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

$K \models C = (A, G) \iff K \sqsubseteq_A G$
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

$K \models C = (A, G) \iff K \sqsubseteq_A G$
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 $E = E_1 \parallel E_2$.

\[ K_1 \sqsubseteq_{E_1 \parallel E_2} K_2 \iff K_1 \parallel E_1 \sqsubseteq_{E_2} K_2 \parallel E_1 \]
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 $E = E_1 \parallel E_2$.

$$K_1 \sqsubseteq_{E_1 \parallel E_2} K_2 \iff K_1 \parallel E_1 \sqsubseteq_{E_2} K_2 \parallel E_1$$
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 $E = E_1 \parallel E_2$.

$$K_1 \sqsubseteq_{E_1 \parallel E_2} K_2 \iff K_1 \parallel E_1 \sqsubseteq_{E_2} K_2 \parallel E_1$$
Correctness of circular reasoning

Central result, required for applying the meta-theory:

**Theorem (Circular reasoning)**

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

1. $\text{traces}_G$ is closed under limits
2. $\text{traces}_G$ 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)**

\[ \{C_i\}_{i=1}^n \text{ dominates } C \text{ if, } \forall i, \text{ traces}_{G_i} \text{ and traces}_G \text{ are closed under limits and under time-extension and} \]

\[
\begin{align*}
\{ G_1 \parallel ... \parallel G_n \subseteq_A G \\
A \parallel G_1 \parallel ... \parallel G_{i-1} \parallel G_{i+1} \parallel ... \parallel G_n \subseteq_{G_i} A_i, \forall i
\end{align*}
\]

(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

1. Context and Problematics

2. A Method for Reasoning with Contracts

3. Theoretical Contributions
   - Integrating Contracts in SysML
   - Modeling Components: a Timed Input/Output Automata Flavour
   - A Formal Contract Theory for TIOA
   - Automated Verification with Model-Checking
   - Evaluation and Related Work

4. Practical Contributions

5. Conclusion and Perspectives
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 \( ⊿◁ \) between the component under study and the timed property automaton.
3. Apply **reachability analysis** for the error state \( π \).
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 ◁ between the component under study and the timed property automaton.

3. Apply **reachability analysis** for the error state $\pi$. 
Outline

1. Context and Problematics

2. A Method for Reasoning with Contracts

3. Theoretical Contributions
   - Integrating Contracts in SysML
   - Modeling Components: a Timed Input/Output Automata Flavour
   - A Formal Contract Theory for TIOA
   - Automated Verification with Model-Checking
   - Evaluation and Related Work

4. Practical Contributions

5. Conclusion and Perspectives
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
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
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

The safety property must be deterministic for applying automatic verification.

⇒ assumptions must satisfy this condition for automatically proving contract dominance
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

The safety property must be deterministic for applying automatic verification.
⇒ 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

1. Context and Problematics

2. A Method for Reasoning with Contracts

3. Theoretical Contributions

4. Practical Contributions
   - Profile Well-Formedness Rules
   - Implementation of the SysML to TIOA Transformation
   - The ATV SGS Case Study

5. Conclusion and Perspectives
Outline

1. Context and Problematics
2. A Method for Reasoning with Contracts
3. Theoretical Contributions
4. Practical Contributions
   - Profile Well-Formedness Rules
   - Implementation of the SysML to TIOA Transformation
   - The ATV SGS Case Study
5. Conclusion and Perspectives
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

\(^1\)http://polarsys.org/
Outline

1. Context and Problematics
2. A Method for Reasoning with Contracts
3. Theoretical Contributions
4. Practical Contributions
   - Profile Well-Formedness Rules
   - Implementation of the SysML to TIOA Transformation
   - The ATV SGS Case Study
5. Conclusion and Perspectives
Implementation in the IFx2 toolset\textsuperscript{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

\textsuperscript{2}www.irit.fr/ifx/
Outline

1. Context and Problematics

2. A Method for Reasoning with Contracts

3. Theoretical Contributions

4. Practical Contributions
   - Profile Well-Formedness Rules
   - Implementation of the SysML to TIOA Transformation
   - The ATV SGS Case Study

5. Conclusion and Perspectives
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 $\varphi$: At the end of the deployment sequence, all 4 WINGs are deployed.
Formalizing the requirement
Formalizing the requirement
Formalizing the requirement

Behavior of Property ($\varphi$)
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 $\parallel$ SOFTWARE $\parallel$ CU.
Applying contract-based reasoning

\[ \varphi: \text{At the end of the deployment sequence, all 4 WINGs are deployed.} \]

- The system \( S \) is given by HARDWARE which contains the WINGs.
- The environment \( E \) is given by MVM \( \parallel \) SOFTWARE \( \parallel \) CU.
Applying contract-based reasoning

\( \varphi \): 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

Statechart Diagram

- IDLE
- FAILURE
- TCU_SAD_ESB_ARM_CMD
- TCU_SAD_ESB_DSARM_CMD
- SGS_TCUI_SADG1_CMD
- SGS_TCUI_SADG2_CMD
- ACTIVATE_TK
- DEACTIVATE_TK
- SGS_REQUEST_DEPLOY_WING_STATUS

SGS_DEPLOY_WING_STATUS(params->), ::PredefinedTypes_ATV::T_SGS_DEPLOY_WING_STS::LOCKED_DEPLOYED)
Contract satisfaction
Generated proof obligations

\[ \text{WING}_1 \sqsubseteq \text{MV}_M \parallel \text{SOFTWARE} \parallel \text{CU} \parallel \text{WING}_2 \parallel \text{WING}_3 \parallel \text{WING}_4 \ G_W^1 \]
where \( \text{WING}_i, \ i \in 2..4 \), is of type \( G_W \)

\[ \text{WING}_2 \sqsubseteq \text{MV}_M \parallel \text{SOFTWARE} \parallel \text{CU} \parallel \text{WING}_1 \parallel \text{WING}_3 \parallel \text{WING}_4 \ G_W^2 \]

\[ \text{WING}_3 \sqsubseteq \text{MV}_M \parallel \text{SOFTWARE} \parallel \text{CU} \parallel \text{WING}_1 \parallel \text{WING}_2 \parallel \text{WING}_4 \ G_W^3 \]

\[ \text{WING}_4 \sqsubseteq \text{MV}_M \parallel \text{SOFTWARE} \parallel \text{CU} \parallel \text{WING}_1 \parallel \text{WING}_2 \parallel \text{WING}_3 \ G_W^4 \]
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

Statechart Diagram
Dominance
Modeling dominance
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:

\[ MVM \parallel SOFTWARE \parallel CU \sqsubseteq_{G\_HW} MVM \parallel SOFTWARE \parallel CU \]
⇒ generates the following proof obligation:

\[ MVM \parallel SOFTWARE \parallel CU \parallel G\_HW \preceq Property \]
1. **Contract satisfaction:**
   - detected error: deployment deactivation in case of failure of disabled components
   - after correction:

<table>
<thead>
<tr>
<th>Type of induced failure</th>
<th>Wing 1</th>
<th>Wing 2</th>
<th>Wing 3</th>
<th>Wing 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Thermal knife</td>
<td>13993</td>
<td>6869</td>
<td>18842</td>
<td>11412</td>
</tr>
<tr>
<td>Hold-down and release system</td>
<td>12672</td>
<td>6516</td>
<td>16578</td>
<td>9980</td>
</tr>
<tr>
<td>Solar array driving group</td>
<td>11527</td>
<td>5432</td>
<td>13548</td>
<td>6807</td>
</tr>
</tbody>
</table>

2. **Dominance:**
   - for $G_{HW}$: $< 1$ second
   - for assumptions: trivial

3. **Top “mirror” contract satisfaction:** trivial

4. **Conformance** $< 1$ second
Outline

1. Context and Problematics
2. A Method for Reasoning with Contracts
3. Theoretical Contributions
4. Practical Contributions
5. Conclusion and Perspectives
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.
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
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
Conclusion

Publications


3. I Dragomir, I Ober, C Percebois: *Safety contracts for reactive timed systems (extended abstract)*. GDR GPL 2013


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!