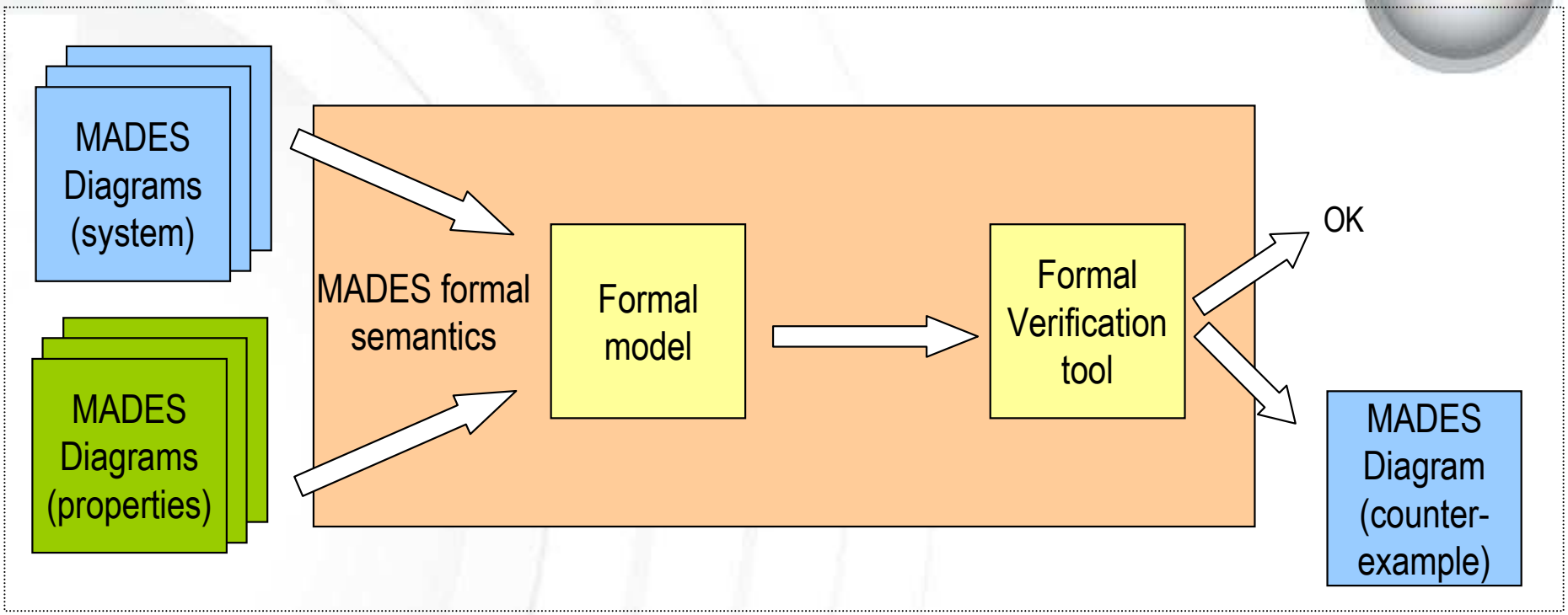
UML-based Formal Verification of Embedded Systems

L. Baresi, A. Morzenti, A. Motta, **M. Rossi**

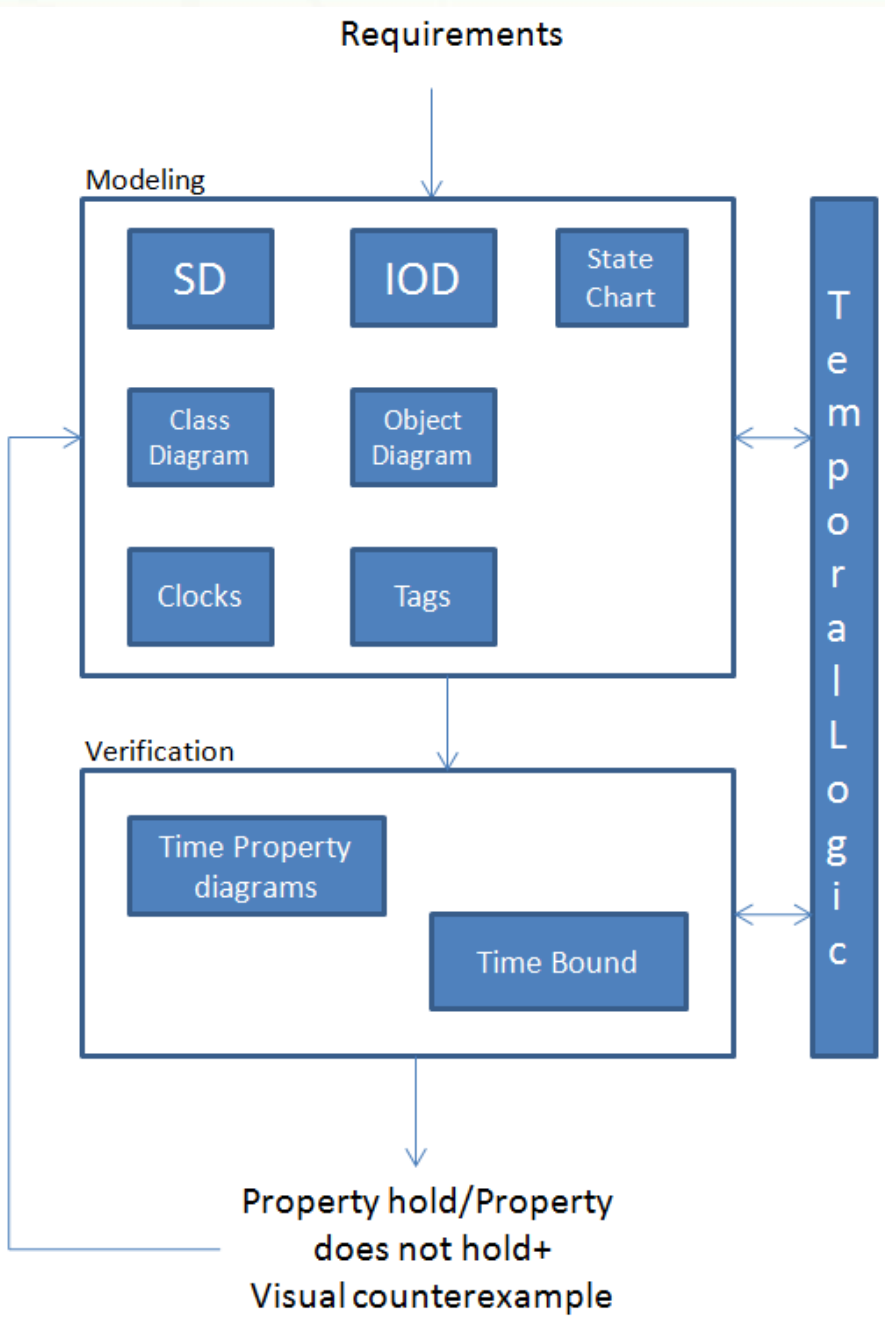
DEEP-SE Group,
Dipartimento di Elettronica e Informazione
Politecnico di Milano

Approach Overview and Principles



- Keep the UML notation as standard as possible
- Hide the formal details from the user

A More Detailed Overview



Formal Ingredients: TRIO

- TRIO is a first-order linear temporal logic
  - with a metric notion of time
  - the time domain can be discrete or dense
  - here we focus on a discrete time domain

- The TRIO specification of a system consists of a set of TRIO formulae
  - The formulae state how items are constrained and how they vary over time

TRIO temporal operators

Operator	Definition
$\mathrm{Past}(F,t)$	$t \geq 0 \wedge \mathrm{Dist}(F,-t)$
$\mathrm{Futr}(F,t)$	$t \geq 0 \wedge \mathrm{Dist}(F,t)$
$\mathrm{Alw}(F)$	$\forall d : \mathrm{Dist}(F,d)$
$\mathrm{AlwP}(F)$	$\forall d > 0 : \mathrm{Past}(F,d)$
$\mathrm{AlwF}(F)$	$\forall d > 0 : \mathrm{Futr}(F,d)$
$\mathrm{SomF}(F)$	$\exists d > 0 : \mathrm{Futr}(F,d)$
$\mathrm{SomP}(F)$	$\exists d > 0 : \mathrm{Past}(F,d)$
$\mathrm{Lasted}(F,t)$	$\forall d \in (0,t] : \mathrm{Past}(F,d)$
$\mathrm{Lasts}(F,t)$	$\forall d \in (0,t] : \mathrm{Futr}(F,d)$
$\mathrm{WithinP}(F,t)$	$\exists d \in (0,t] : \mathrm{Past}(F,d)$
$\mathrm{WithinF}(F,t)$	$\exists d \in (0,t] : \mathrm{Futr}(F,d)$
$\mathrm{Since}(F,G)$	$\exists d > 0 : \mathrm{Lasted}(F,d) \wedge \mathrm{Past}(G,d)$
$\mathrm{Until}(F,G)$	$\exists d > 0 : \mathrm{Lasts}(F,d) \wedge \mathrm{Futr}(G,d)$

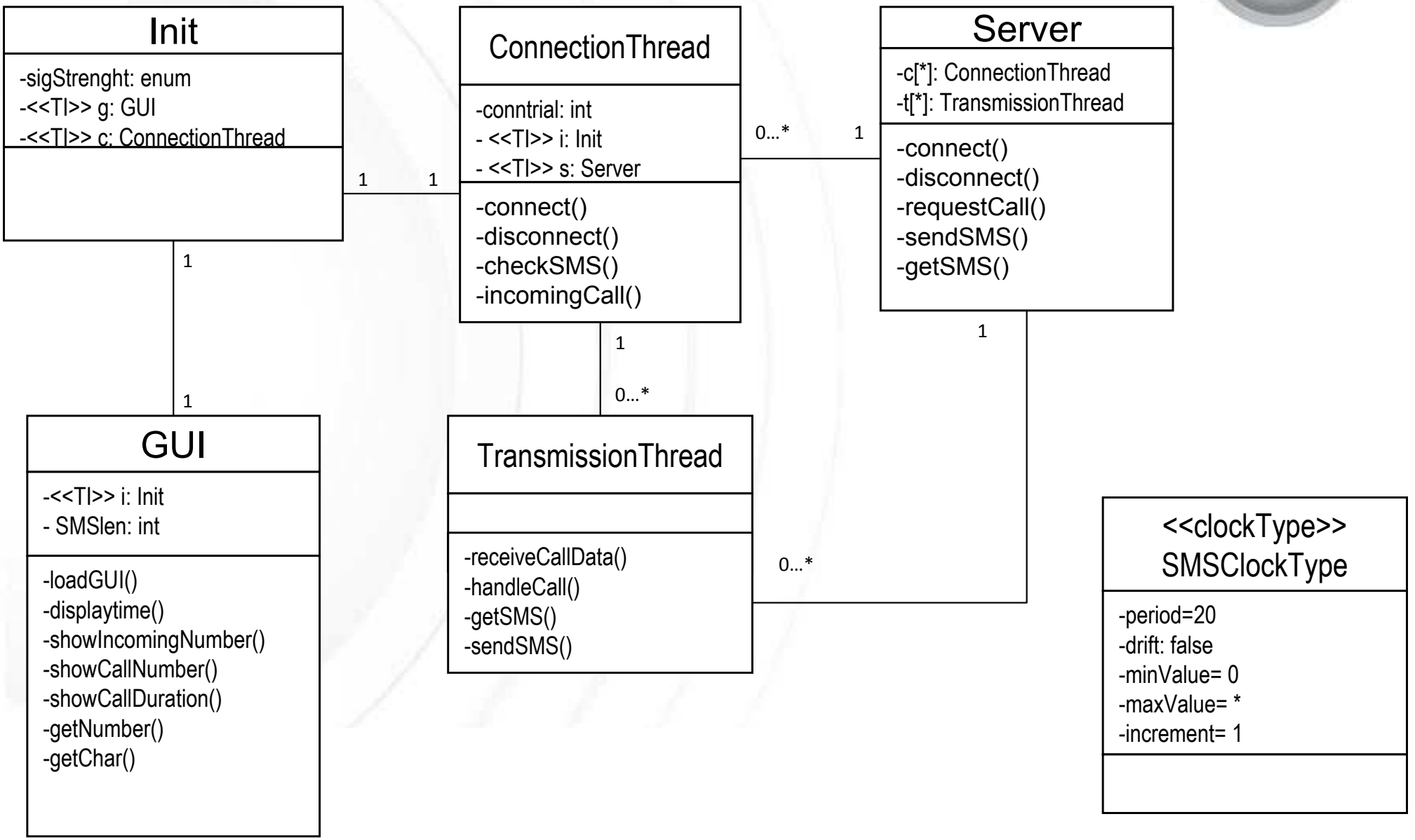
Formal Ingredients: Zot

- A bounded satisfiability/model checker that supports verification of discrete-time TRIO models
  - it checks whether stated properties hold for the system being analyzed
  - If a property does not hold, Zot produces a counterexample that violates it
- Zot is plugin-based
  - every plugin defines an encoding of a formal language (e.g., TRIO) into the input language of a verification engine
  - two kinds of verification engines: SAT solvers (DIMACS input) and SMT solvers (SMT-LIB input)

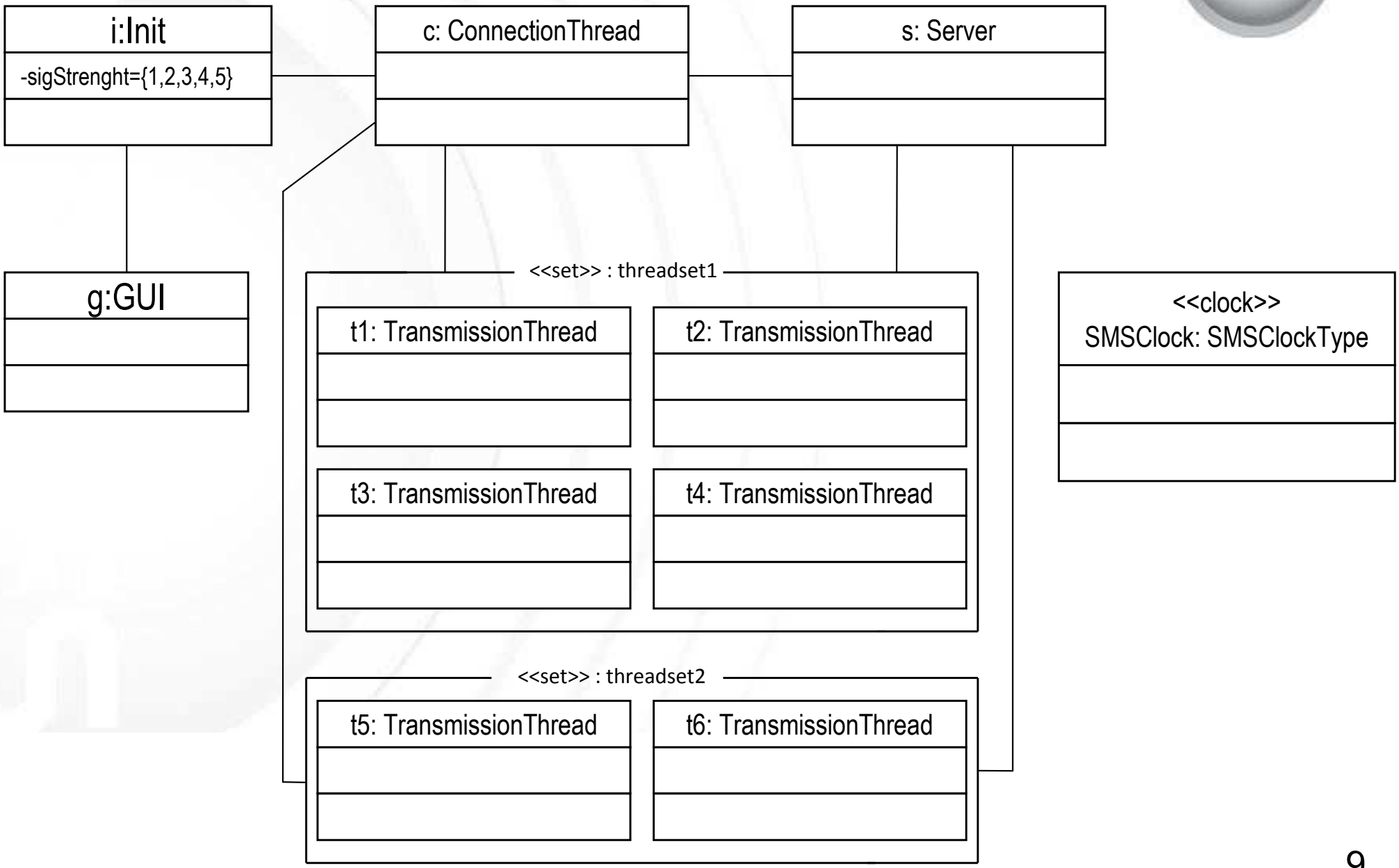
Example System

The telephone system should provide the following features:
At the startup the system should connect to the remote server and initialize the graphical user interface (GUI). If the telephone is not correctly connected to the server the GUI will not be shown. The connection is attempted 3 times with a timeout of 10 seconds.
When the startup is finished the system is ready to receive incoming calls and SMSs. Incoming calls may arrive at any instant. Incoming SMS are checked on the server every 20 seconds by the telephone itself. If no reply is received by the server within 10 seconds the attempt is not repeated.
If the telephone is idle (e.g. it is not performing any call, or any SMS composition) and the user presses a number, the number itself is shown on the screen and the telephone waits for the remaining digits until the *green* button is pressed. If the red button is pressed the system aborts the operation.
If the telephone is idle and the user presses the *ok* button, then a textual interface is shown to compose the SMS. When the *ok* button is pressed again the GUI waits for the telephone number and when the *ok* button is pressed the SMS is sent. SMSs are sent with tokens of 160 characters. The transmission time follows this formula: trTime=length(SMS)/sigStrength*10 sec, where sigStrength may be {1,2,3,4,5}.

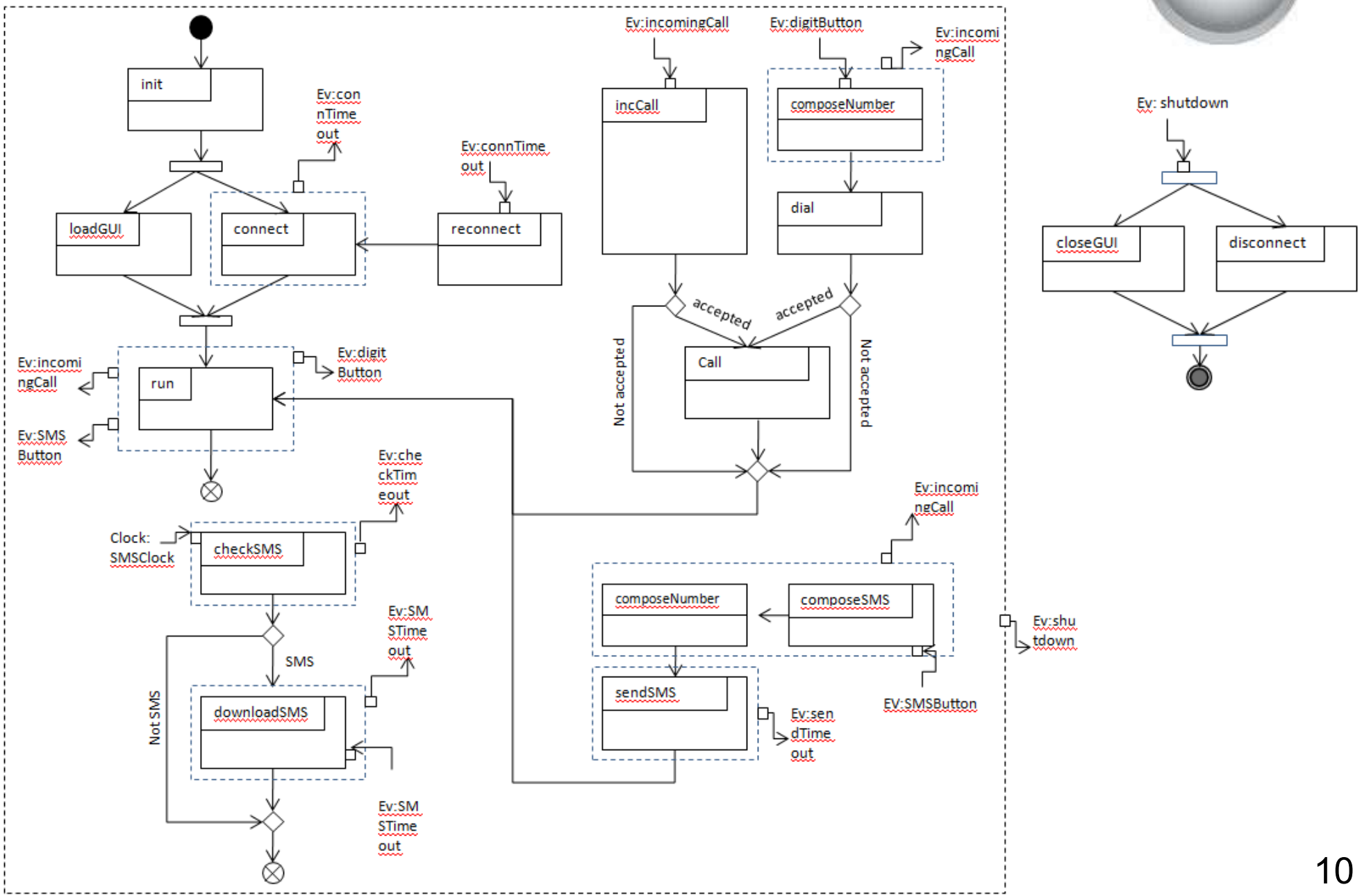
Example of Class diagram



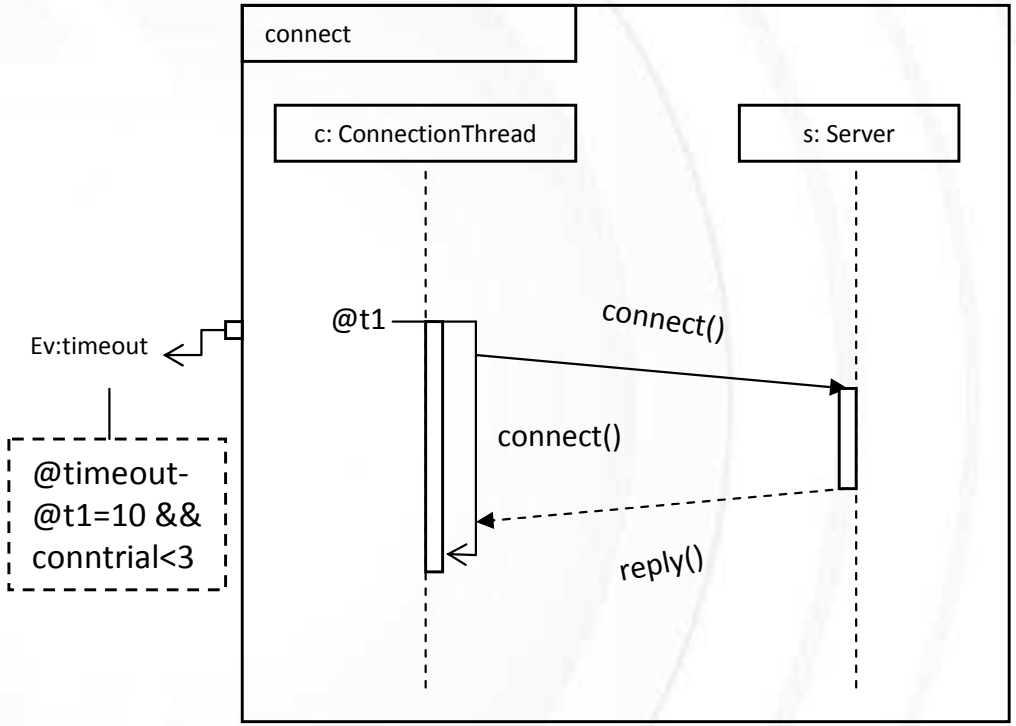
Example of Object diagram

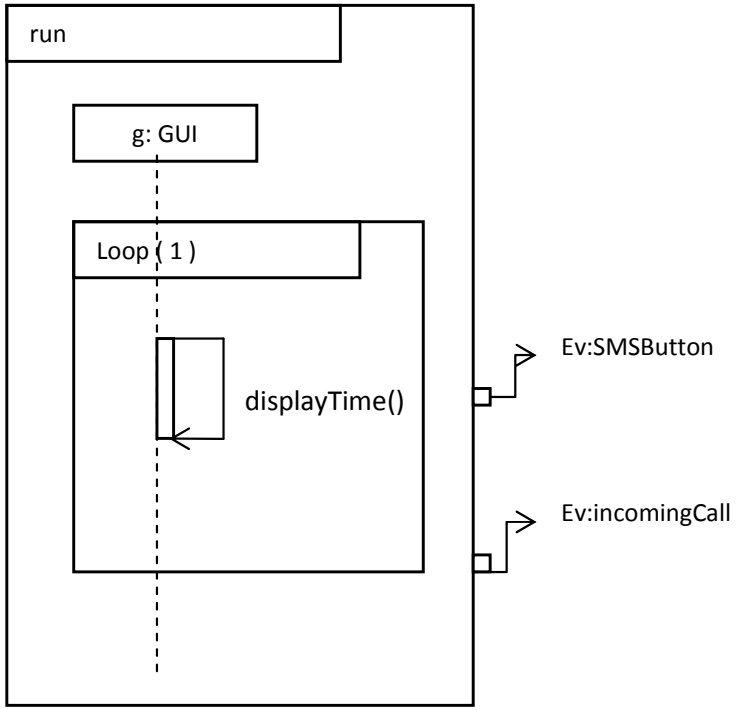


Example of Interaction Overview Diagram



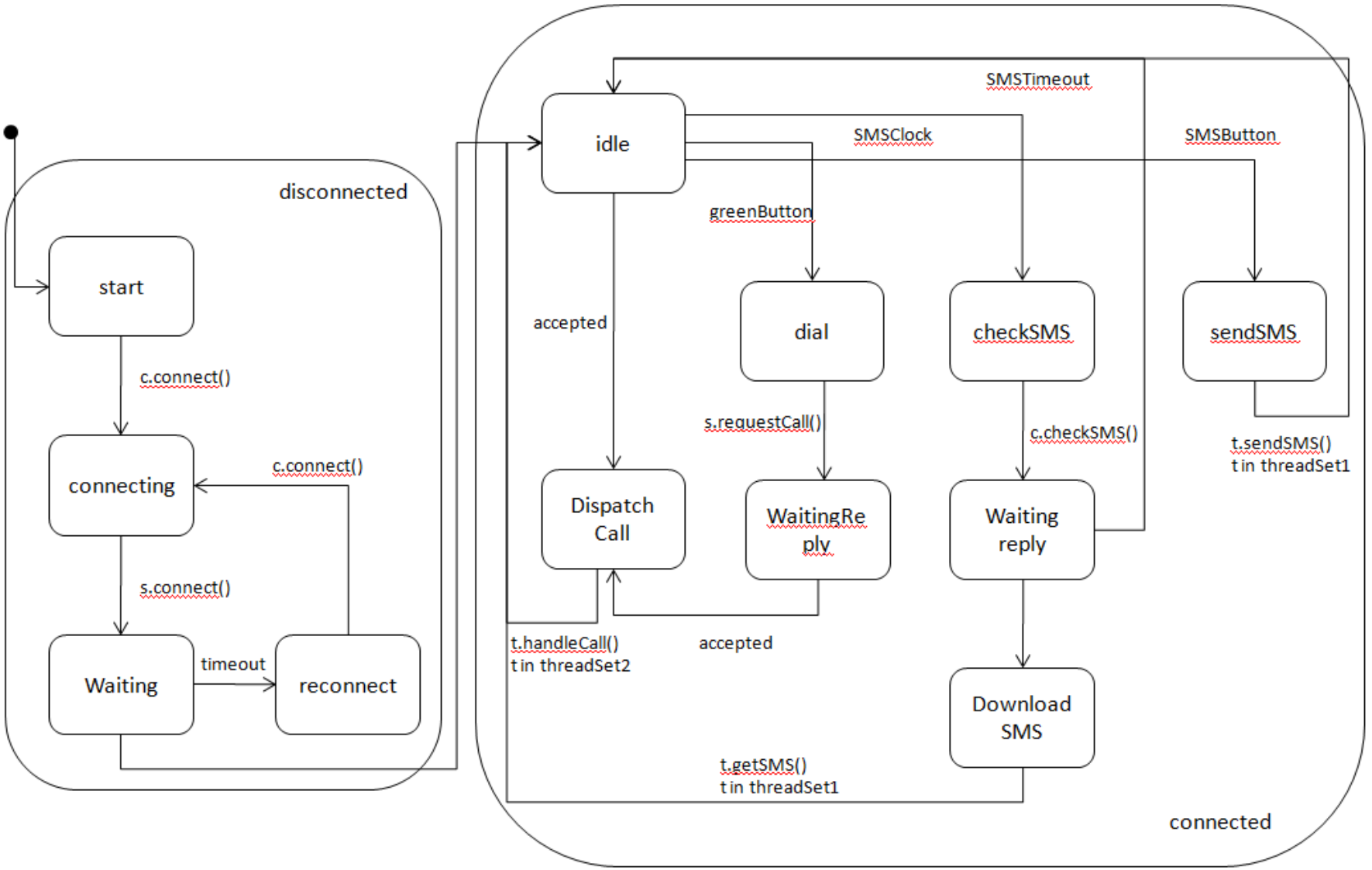
Examples of Sequence Diagrams



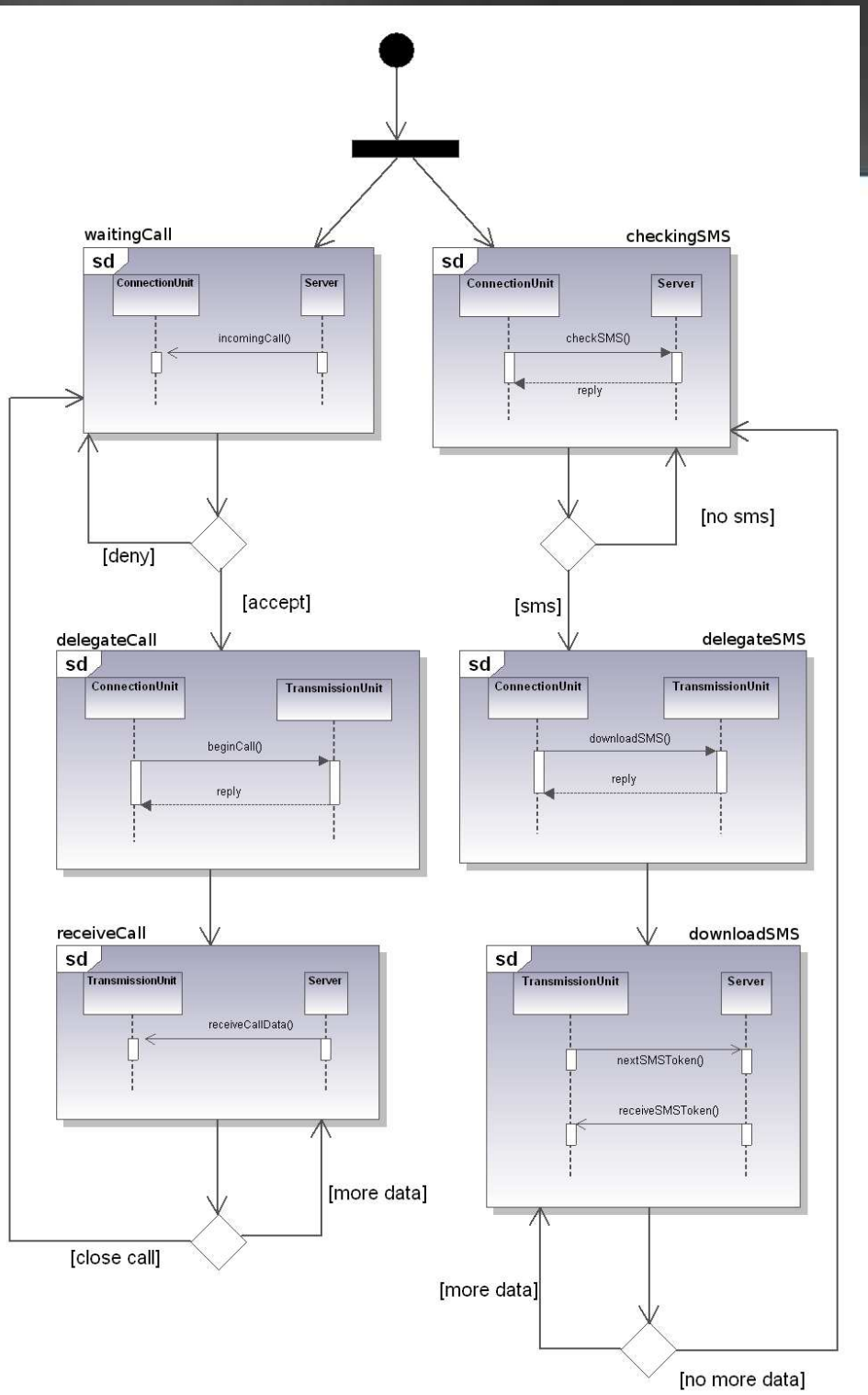


Example of State diagram

ConnectionThread

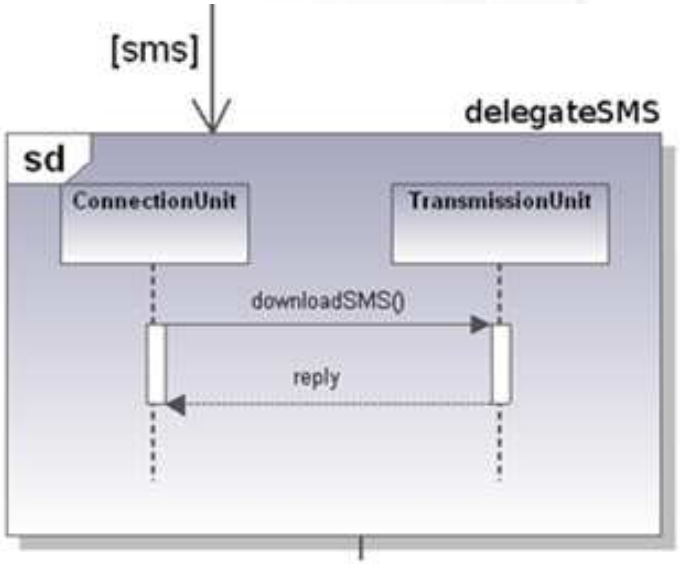


A simpler example



From syntax to semantics

- To carry out formal verification, the syntax of the MADES notation is being given a formal semantics
  - the semantics is based on the TRIO temporal logic

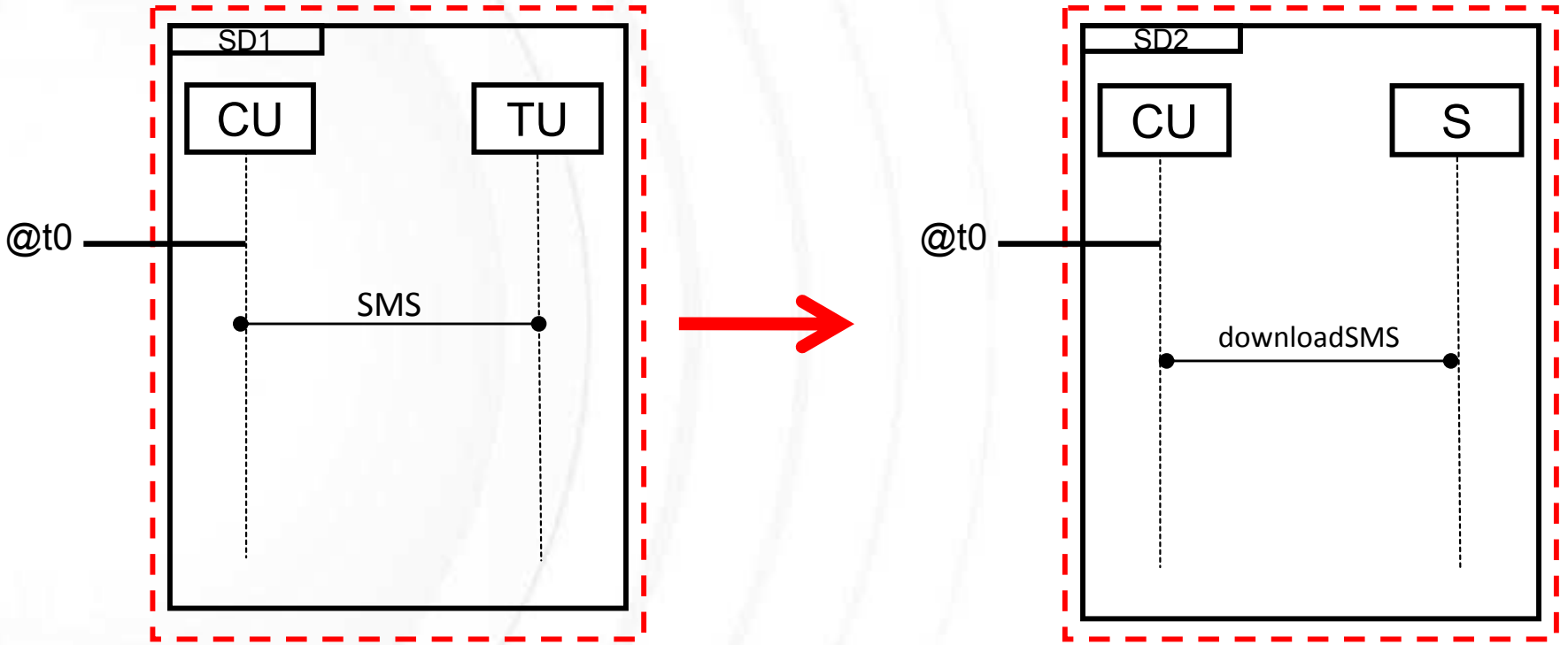


$$delegateSMSSTART \Leftrightarrow downloadSMS$$

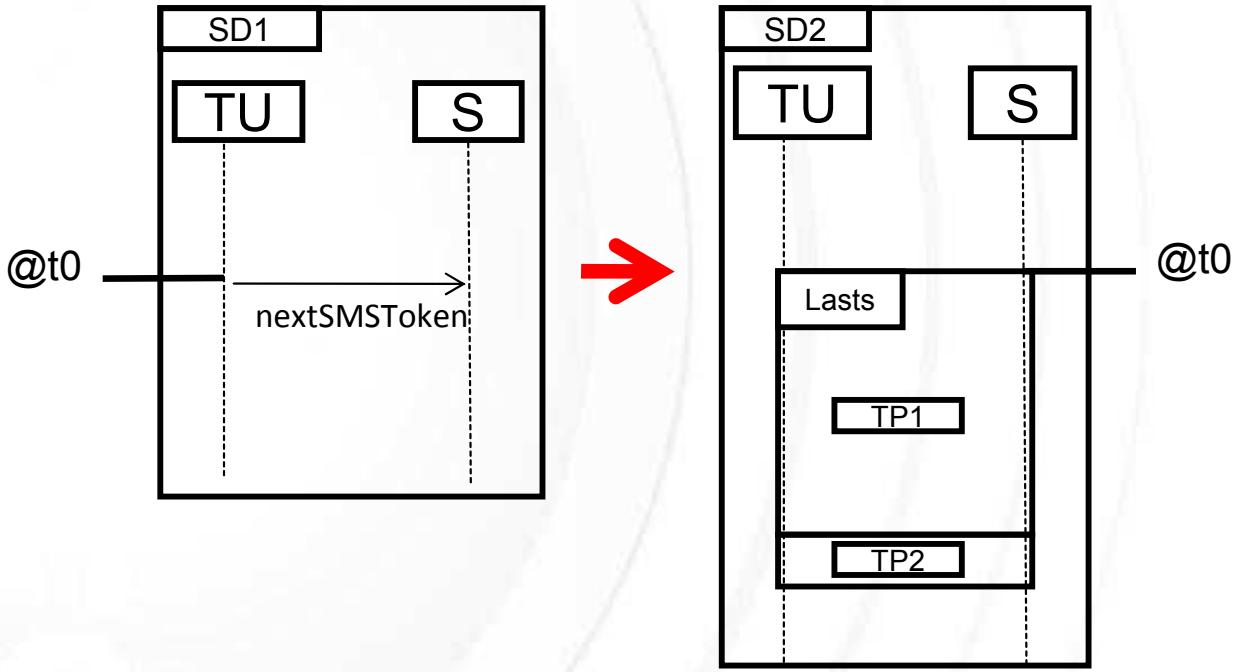
$$delegateSMSEND \Leftrightarrow reply3$$

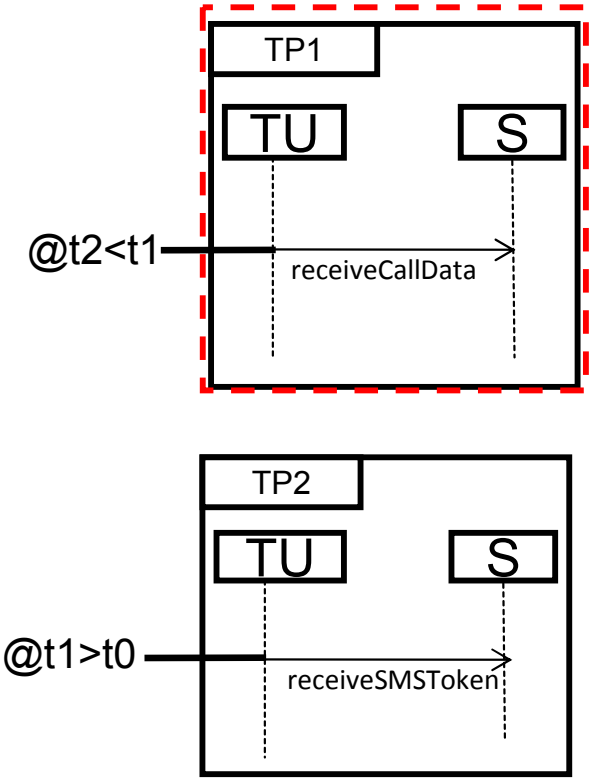
$$delegateSMS \Leftrightarrow delegateSMSSTART \vee \mathrm{Since}(\neg delegateSMSEND, delegateSMSSTART)$$

From Temporal Properties to Temporal Logic



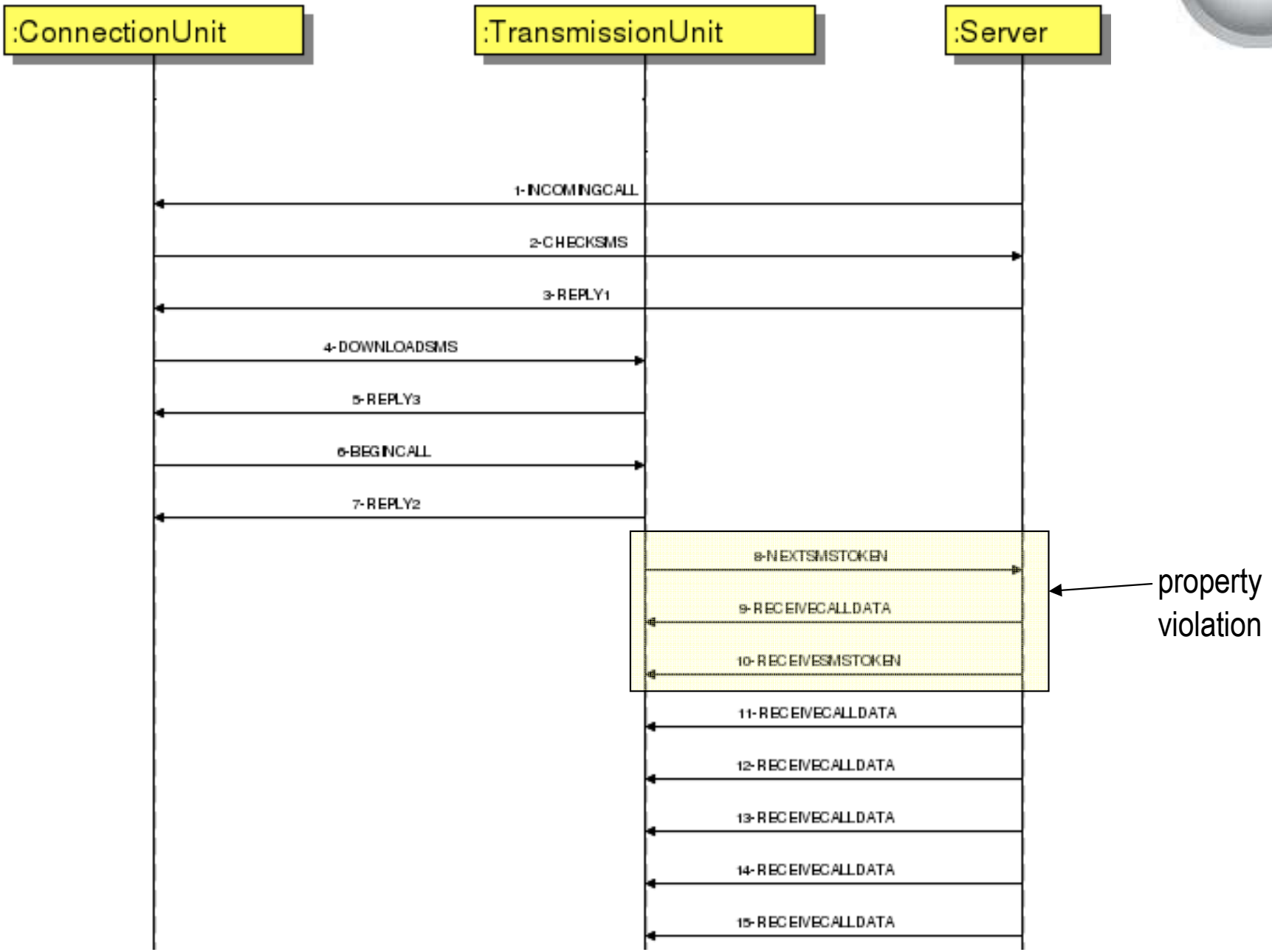
$$\neg \mathrm{SomF}(SMS) \Rightarrow \neg \mathrm{SomF}(downloadSMS)$$





$$nextSMSToken \Rightarrow \text{Until}(\neg receiveCallData, receiveSMSToken)$$

Counterexample



Conclusions and Future Works

- This is a first step towards a technique to model and verify embedded systems using an intuitive UML-based notation
- We have a preliminary formal semantics of part of the notations
  - based on metric temporal logic
  - supported by an automated tool, to verify temporal properties of modeled systems
- The semantics will be completed in the upcoming months
- Information provided by the user through tags in the UML diagrams will be used to speed up the verification phase