**CHASE Version 1.0: What Is General versus What is Specific for the Electric Power System?**

**Pierluigi Nuzzo and Michele Lora**

We provide below a list of the main components of CHASE 1.0 and comment on their general versus hard-coded parts:

(from directory include/)

* Architecture: Contains prevalently hard-coded definition of components and topology for an aircraft electric power system (EPS). To generalize we need to implement:
  + Mechanisms (data structures, methods, …) to instantiate an architecture as a labelled graph, where labels specify the domain specific “types” and “attributes” of the architecture nodes
  + Mechanisms to specify libraries of components, categorized into “types” (classes of components in the library), represented by labels in the graph
  + A mechanism to generate an architecture (labelled graph) from inputs (e.g., XML files) including the library and a description of the architecture.

The input architecture specification should define the “types” of the architecture (library) and the instantiation and interconnection of the typed nodes. This portion of Chase should be able to handle this kind of specification in the future.

* Behaviors: Generic, include mechanisms to specify logic formulas (will be extended to different logics) and timing (the timing mechanism is under development). The specification of logic formulas is based on their abstract syntax tree representation.
* Chase2TuLiP: Independent of the application example, generates the input files in TuLiP format (can be extended to other backend tools). Mostly devoted to translate logic formulas specified in Behaviors into their TuLiP representation.
* Contracts: Generic, provides the basic infrastructure to represent assume-guarantees contracts and eventually the possibility to extend the representation for different interface theories.
* Manipulations: Generic, implements the basic Visitor design pattern mechanism to visit the abstract syntax tree of the problem representation.
* Patterns2Chase:
  + EPSPatterns:
    - EPSPatterns: defines the “low-level” patterns; these patterns are similar to temporal logic and therefore could be made generic; such a parser can be replaced, for instance, by an XML file parser.
    - HighLevelEPSPatterns: handles patterns that reasons on connections, paths, and types; some of them are specific, e.g., the notion of “live path” uses attributes of the specific EPS types, but can be generalized once generic libraries and types are available.
    - TimedEPSPatterns: as above (handle expressions with real time intervals, ongoing work).
  + TemporalLogicsParser: Generic. It implements a simple parser to manage LTL formulas expressed using the SPIN syntax for LTL and build the abstract syntax trees representing the formulas using the classes in the Behaviors/Logics namespace.
  + TextualParsing: Recognize the text of the patterns; specific to EPS
* Specification:
  + TextBasedPattern: provides the virtual class to represent patterns. It defines methods to match patterns and extract logic formulas and pattern instance parameters
* Utilities: Generic. It provides support classes and methods to manage the extraction of information from the architecture graph, and to manage time (e.g., Greater Common Divider for generating counters).

Other files in src/

* chase.cc: receives a text files with a list of requirements using low-level patterns and perform their consistency analysis
* epstool.cc: is the “main” used for the EPS demo
* Patterns2Chase.cc: used to debug pattern recognition and printout of LTL A/G contracts

Misc:

* Third party directory contains the POCO libraries (<http://pocoproject.org/>) source code. The API provided by POCO was intended to be used to create a XML-based exchange representation for the internal representation of Chase. The functionality is planned; however, it is not on high-priority.
* Chase requires third party components:
  + Boost C++ libraries, used to implement the SPIN syntax parser for LTL formula. The API is released under the Boost Software License: <http://www.boost.org/users/license.html>
  + TuLiP, used as back-end tool for the controller synthesis. It is released under the 3-clauses Berkeley Software Distribution License (<https://opensource.org/licenses/BSD-3-Clause)>
* What could be the license for CHASE, once we release it open source? Try the 3-clause BSD license…

Copyright (c) 2015-2016 by the University of California, Berkeley and the University of Verona, Italy.   
All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

3. Neither the name of the copyright holder(s) nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDERS OR THE CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.