Declare: A Tool Suite for Declarative Workflow Modeling and Enactment

Michael Westergaard[⋆] and Fabrizio M. Maggi[⋆⋆]

Department of Mathematics and Computer Science,
Eindhoven University of Technology, The Netherlands
{m.westergaard,f.m.maggi}@tue.nl

Abstract. Declare adheres to the declarative workflow modeling paradigm, where, instead of modeling allowed behavior and explicit choices, users model disallowed behavior. This makes it easier to model loosely structured processes. Without appropriate precautions, however, users are less supported in choosing which actions lead to the desired end-result. The goal of Declare is to ensure flexibility when modeling loosely structured processes and, at the same time, to provide support for decision making during the execution.
Declare consists of a Designer component for creating and verifying models, a framework which acts as a server for enacting models, and a Worklist, which allows users to see and pick tasks to perform. Declare can act as a client of the operational support service in ProM, which makes it possible to guide users towards desired goals, to obtain statistics about an executed process, and to monitor a running case. In addition to this, a ProM plug-in has been also implemented that allows for the discovery of Declare models from logs.

Procedural languages enforce explicitly specifying all possible orderings of tasks in a process model. This undoubtedly provides support during process execution. However, in environments with a lot of variability, processes are often loosely structured. These processes are hard to describe using a procedural paradigm as the large number of allowed executions makes it difficult to include them all in a process model. In the declarative workflow paradigm [4], we do not model allowed behavior, but rather disallowed behavior. Instead of explicitly specifying all possible interactions among process tasks, declarative models allow describing a process through a set of constraints that must be satisfied throughout the process execution. Therefore, contrarily to procedural approaches that produce "closed" models, i.e., all what is not explicitly specified

[⋆] This research is supported by the Technology Foundation STW, applied science division of NWO and the technology program of the Dutch Ministry of Economic Affairs.

[⋆⋆] This research has been carried out as a part of the Poseidon project at Thales under the responsibility of the Embedded Systems Institute (ESI). This project is partially supported by the Dutch Ministry of Economic Affairs under the BSIK program.

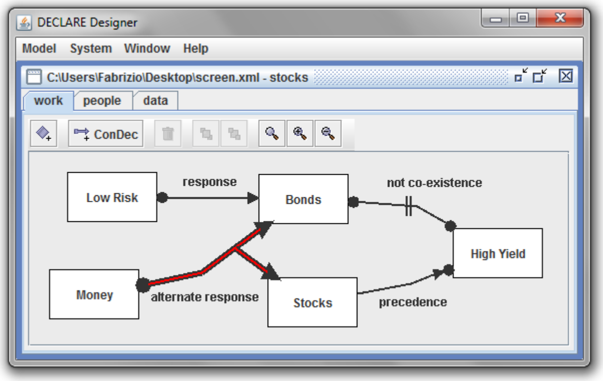


Fig. 1. Screenshot from the Declare Designer.

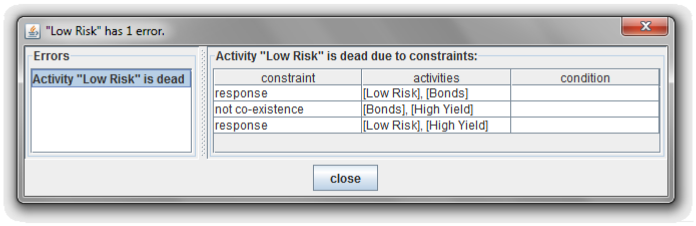
is forbidden, declarative models are "open" and tend to offer more possibilities for execution.

Declare [1] is based on the declarative paradigm and supports flexibility. In addition, it also supports decision making by providing users with recommendations during the process execution. In this way, users can balance flexibility and support and decide the right trade-off between them. The version of Declare used here improves on previous versions by increasing the speed of analysis by a factor of up to 500.000, making it viable to construct and enact models of real-life size [6]. Furthermore, Declare is now modular, which allows programmers to construct tools relying or communicating with parts of the Declare tool suite.

In the screenshot in Fig. 1, we see the Declare Designer with a loaded model consisting of a number of tasks and constraints. Here, tasks are shown as rectangles (e.g., Money) and constraints as arcs between them (e.g., response between Low Risk and Bonds).

The model in Fig. 1 models a simple stock investment strategy and contains tasks related to that, such as getting Money, purchasing Stocks or Bonds, and a desire for Low Risk and High Yield. The constraints restrict the allowed behavior. The constraint alternate response from Money to Bonds and Stocks reflects that if you get money, you should invest it (in Bonds or Stocks). The precedence constraint between Stocks and High Yield models that only if you invest in stocks can you subsequently get high yield (but there is no guarantee). The not co-existence between Bonds and High Yield states that you can never get high yield if you invest in bonds. Finally, the response constraint between Low Risk and Bonds states that if you want low risk, you must invest in bonds.

Declare supports constructing such models using a graphical editor, the Declare Designer. The Designer can also verify that a model is consistent. Suppose, for instance, we add to the model in Fig. 1 a constraint saying that if we want Low Risk we must subsequently obtain High Yield. Declare can then detect that in the resulting model Low Risk can never be executed (as we subsequently have to


"Low Risk" has 1 error.

Errors

Activity "Low Risk" is dead

Activity "Low Risk" is dead due to constraints:

constraint	activities	condition
response	[Low Risk], [Bonds]	
not co-existence	[Bonds], [High Yield]	
response	[Low Risk], [High Yield]	

Fig. 2. Screenshot from the Declare Designer showing a verification error.

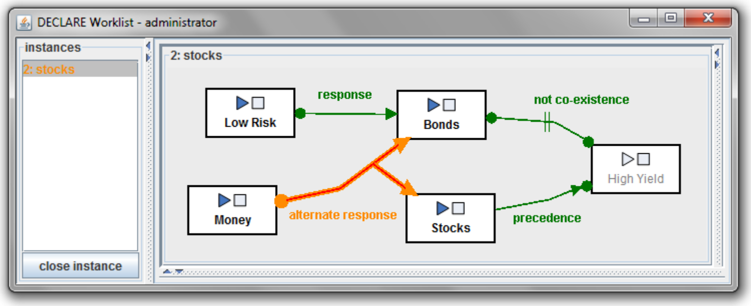


Fig. 3. Screenshot from the Declare Worklist showing an execution of our model.

execute both Bonds and High Yield, which is not allowed by the not co-existence constraint). Declare is also able to pinpoint the exact reason of the inconsistency as shown in Fig. 2. This is possible as Declare has a formal semantics, realized by associating with each constraint an LTL formula, which can be used to translate a Declare model to a finite automaton for analysis.

We can also use that automaton to enact the model, ensuring that we never execute events that would violate constraints. Enactment is done using the Declare Framework as the backend server and the Declare Worklist as the user client. When enacting a Declare model, one or more constraints may be temporarily violated. This is reported to the user, who can take appropriate actions. For instance, in Fig. 3, we see an execution where we have just obtained money. The constraint alternate response is temporarily violated, shown using orange color instead of green (used for satisfied constraints). To satisfy the temporarily violated constraint, the user can execute Bonds or Stocks. Note also that all tasks except High Yield can be executed. High Yield is disabled because its execution would violate the constraint precedence.

Due to the declarative nature of the Declare models, it is not always easy to foresee the effects of choices during the execution of a process. To guide users in the decision-making task, it is possible to integrate Declare with the operational support service in ProM [5]. An operational support provider can give, for instance, recommendations to increase the chances of reaching a desired goal. In the example in Fig. 1, a user may want High Yield, but inadvertently

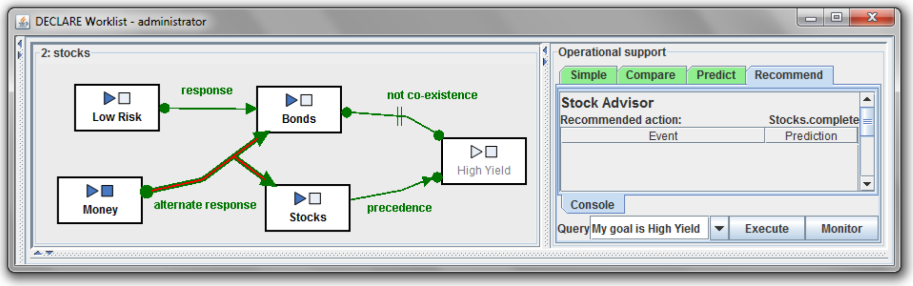


Fig. 4. Screenshot from the Declare Worklist showing information from the operational support service in ProM.

execute Low Risk, even though this permanently disables High Yield (as Bonds must subsequently be executed, which conflicts with High Yield). In this case, to achieve the desired goal, an advisor could suggest buying stocks. The user can choose to adhere to the advise or to ignore it. In Fig. 4, we see a simple example of the operational support client in Declare. The Stock Advisor provider suggests executing the task Stocks in order to later achieve the goal of high yield. The operational support service makes also it possible to obtain statistics about a running instance (for instance, to get the average time spent for each task) and to compare this with similar previous executions.

Using the operational support service, it is also possible to monitor a running process w.r.t. a given Declare model and get health information about it. This is completely orthogonal to the Declare Worklist application. In this case, the Monitor only observes the behavior of a process which executes independently and detect possible violations. In Fig. 5, we see the Declare Monitor in action. Here, 4 instances of our example process are being executed (as shown by instances Smith, Westergaard, Brown, and Maggi). The Monitor provides at-a-glance information about the health of all instances, by adding the text Warning in red after an instance if it has violated constraints. In the example in Fig. 5, the instance Brown violates the precedence constraint after executing High Yield. The status of each constraint is updated after each executed event (displayed on the horizontal axis). We additionally get a measure of the overall health of the instance at the top. We see that violating the precedence constraint lowers the health to 0.69. Furthermore, the Diagnostics panel provides users with additional information about violated constraints.

As well as manually creating a Declare model using the Designer, users can also discover models from logs using the Declare Miner [1]. This can be useful, for instance, for monitoring processes based on historical data.

Declare is an advanced prototype, which supports models of real-life size. The Declare Miner and the Monitor have been used in the Poseidon project [1] to monitor sea vessels. In particular, in [2,3], we show how it is possible to construct

[1] `http://www.esi.nl/short/poseidon/`

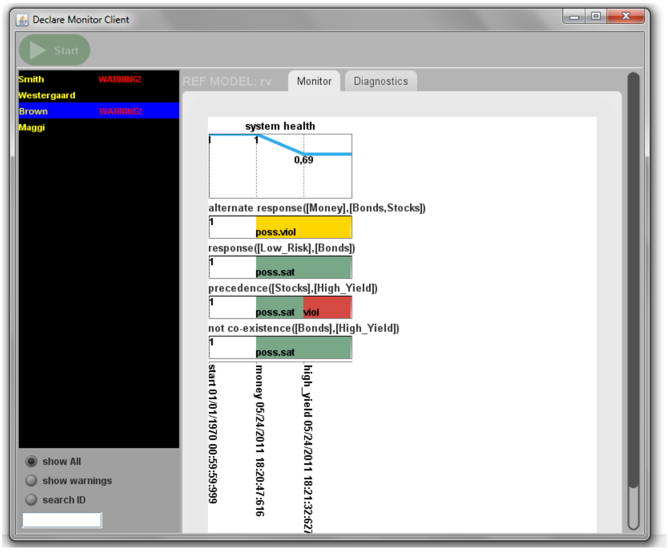

Declare Monitor Client
Start
Smith WARNING!
Westergaard
Brown WARNING!
Maggi
show All
show warnings
search ID
REF MODEL: rv
Monitor
Diagnostics
system health
1
0,69
alternate response([Money],[Bonds,Stocks])
poss.viol
response([Low_Risk],[Bonds])
poss.sat
precedence([Stocks],[High_Yield])
poss.sat
viol
not co-existence([Bonds],[High_Yield])
poss.sat
start 01/01/1970 00:59:59:999
money 05/24/2011 18:20:47:616
high_yield 05/24/2011 18:21:32:627

Fig. 5. Screenshot from the Declare Monitor.

Declare models from real-life historical data and monitor live data w.r.t. to the mined model. We consider Declare stable enough for evaluating declarative approaches. Declare is open source (licensed under the GNU Public License). Binaries and source code can be obtained from its homepage at `declare.sf.net`.

References

1. Declare webpage. Online: `declare.sf.net`.
2. F.M. Maggi, M. Montali, M. Westergaard, and W.M.P. van der Aalst. Monitoring Business Constraints with Linear Temporal Logic: An Approach Based on Colored Automata. In *Proc. of BPM*, LNCS. Springer-Verlag, 2011.
3. F.M. Maggi, A.J. Mooij, and W.M.P. van der Aalst. User-Guided Discovery of Declarative Process Models. In *2011 IEEE Symposium on Computational Intelligence and Data Mining*, 2011.
4. M. Pesic, H. Schonenberg, and W.M.P. van der Aalst. DECLARE: Full Support for Loosely-Structured Processes. In *Proc. of EDOC'07*, page 287, 2007.
5. Process mining webpage. Online: `processmining.org`.
6. M. Westergaard. Better Algorithms for Analyzing and Enacting Declarative Workflow Languages Using LTL. In *Proc. of BPM*, LNCS. Springer-Verlag, 2011.