MIO Workbench: A Tool for Compositional Design with Modal Input/Output Interfaces[⋆]

Sebastian S. Bauer[1], Philip Mayer[1], and Axel Legay[2]

[1] Ludwig-Maximilians-Universität München, Germany
[2] INRIA/IRISA Rennes, France

Abstract. Modal Input/Output interfaces (MIOs) is a new specification theory for systems communicating via inputs and outputs. The approach combines the advantages of both modal automata and interface automata, two dominant specification theories for component-based design. This paper presents the MIO Workbench that is the first complete implementation of the MIO theory.

1 Context

Evolution of computer science technology has permitted the development of large size systems that facilitate (if not govern) our daily life. Such systems, which have to interact with uncertain environments, are much too complex to be developed by a single team or unit. Rather, the current trend in software engineering suggests that huge systems shall result from the assembly of several subsystems called components, each of them being developed by a dedicated team. This component-based design view has the advantage of not only reducing complexity but also hiding code information/secrets of individual participants.

While this view offers flexibility in the design, there is still the need that all the participants agree on what the interface of each component shall be. Such an interface precises the behaviors expected from the component as well as the environment in where it can be used. According to state of practice, interfaces are typically described using Word/Excel text documents or modeling languages such as UML/XML. A series of recent works, now widely accepted by industrials [5,15], instead recommend relying most possibly on mathematically sound formalisms, thus best reducing ambiguities.

Existing interface models [13,9,4,2,8] are generally nothing more than transition systems whose transitions are equipped with labels on which a dedicated semantic can be built. Many of those powerful theories have recently been unified through the *Modal Input/Output interface* (MIO) theory. Like modal automata [8], the formalism allows to finitely model a possibly infinite set of systems (aka implementations) by distinguishing those behaviors (transitions) that *must* always be implemented from those that *may* not be implemented. Moreover, similarly to *I/O automata* [10], MIOs communicate via *input* and *output* actions and propose the optimistic structural composition from interface automata [2].

[⋆] This work has been partially sponsored by the EU project ASCENS, 257414.

This means that, contrary to I/O automata, MIOs are not input-enabled and a state in a composed system, where one component can perform an output that cannot be caught by a receiving component, is declared as an error state. However, reaching an error state does not necessarily mean that the two components cannot work properly together. Instead, two components can be composed if there exists an environment in where no such communication errors occur. As a summary, the MIO theory is equipped with *structural (optimistic and pessimistic) composition together with compatibility*, *refinement* (that allows to compare two sets of implementations), *satisfiability* (that allows to check whether an implementation matches the requirements of the specification), and a *logical composition* that allows to combine requirements represented by interfaces. In addition, there is a *quotient* operator that allows to synthesize missing interface requirements in a large size design. Finally the theory also permit independent implementability [3].

It is clear that interface theories can be used to facilitate the development of real-life applications. But, surprisingly, only a few tools exist for specific theories [6,1]. This paper introduces the MIO Workbench toolset, that is the very first complete implementation of the MIO theory. The paper presents the tool architecture and describes the basic creation and analysis facilities the MIO Workbench provides. Details about theory and tool can be found at [12].

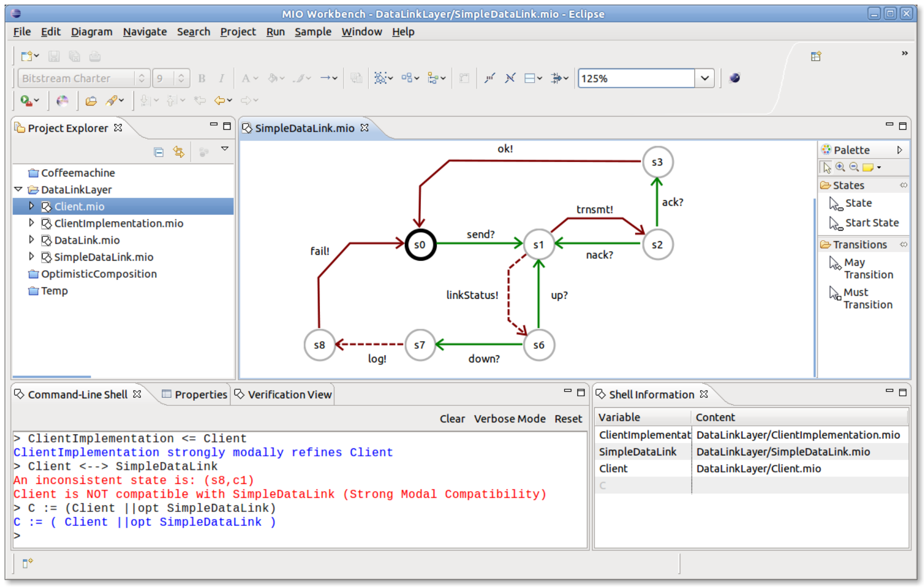
2 Tool Architecture

The MIO Workbench is implemented in Java as a series of Eclipse plug-ins [7] which makes it easily extendable in case new operations are added to the MIO theory. We have used the Eclipse Modeling Framework (EMF) to define a metamodel for MIOs. This allowed us to generate code for creation and access of (objects representing) MIOs, thus code maintenance is much easier and flexible, particularly for later changes and extensions of the domain model. The architecture of the tool is briefly described hereafter. For details, see [11,12].

The graphical user interface of the tool consists of an *editor* for drawing MIOs, a *verification view* to execute operations on MIOs (composition, refinement, . . .), and finally, a *command-line shell* which is a powerful interface of computing complex tasks (e.g., combining operations and checks). An overview of the standard perspective of the tool can be seen in Fig. 1.

The MIO Editor displays MIOs as a graph in the classical way by using nodes as states and edges as transitions. May and must transitions are drawn with dashed and solid arrows, respectively. Furthermore, each transition is equipped with an internal action (black arrow), input action (suffixed with *?*, green arrow), or output action (suffixed with *!*, red arrow). The MIO editor offers all the usual operations such as adding new states and transitions, moving them around, changing labels, types, and manual layouting.

The MIO Workbench Verification View provides a way to visually execute individual operations and depict the results graphically. Two MIOs can be placed



```
> ClientImplementation <= Client
ClientImplementation strongly modally refines Client
> Client <--> SimpleDataLink
An inconsistent state is: (s8,c1)
Client is NOT compatible with SimpleDataLink (Strong Modal Compatibility)
> C := (Client ||opt SimpleDataLink)
C := ( Client ||opt SimpleDataLink )
>
```

Variable	Content
ClientImplementat	DataLinkLayer/ClientImplementation.mio
SimpleDataLink	DataLinkLayer/SimpleDataLink.mio
Client	DataLinkLayer/Client.mio
C	

Fig. 1. MIO Workbench and its editor and views.

on the left hand and right hand side, by dragging .mio files from the project explorer and dropping them on one side of the verification view; then all the MIO operations are available from the middle panel. The output of performing a refinement or compatibility check is a refinement relation or the matching states side-by-side between the two input MIOs in the positive case, or in the negative case the view graphically displays, side-by-side, the path which led to an erroneous state or the transition possible in one automaton, but not in the other. The output of composition, conjunction and quotient is a MIO which can be saved and reused.

The MIO Workbench Shell is a shell-like interpreter that facilitates combination of operations. MIOs from the project explorer can be made available in the shell by drag-and-drop. All the operations can be executed by entering simple commands. For instance, if `S` and `T` are variables, then the command `S <= T` performs a refinement check. To construct new MIOs, we can compose two MIOs `S1` and `S2` and store the result in a new variable by executing `C := (S1 || S2)`. The main advantage is the possibility of performing complex verification tasks like `(S && T && U) <= (A -- B)` (where `&&` is conjunction and `--` is quotient). The complete input grammar as well as a tutorial can be found at [12].

3 Experiments and Future work

As of yet, interface theories are not used on a large scale in industry. However, together with industrials, we believe that such theories can greatly facilitate the development of real-life applications, and that the major stumbling block to their deployment lies in the unavailability of tools, to which the MIO Workbench is a remedy. We have already applied the MIO Workbench and the implemented interface theories in the context of the EU project Sensoria [14], where we have worked with industry partners to model and verify service-based architectures (based on Web services) from the domains of Finance, Automotive, and Education. Using the UML profile UML4SOA [16,11] for modeling the system, we were able to automatically translate the models into MIOs and perform a rigorous analysis with the MIO Workbench, like compatibility and refinement checks, which was later re-annotated to the UML model. This analysis has proven helpful in finding the more subtle problems in the UML models.

For future work, we plan to integrate our recent work on extensions of modal interfaces to include data by an efficient BDD-based implementation. Also, code generation transforming MIOs to correct implementations is of interest.

References

1. Adler, B.T., de Alfaro, L., da Silva, L.D., Faella, M., Legay, A., Raman, V., Roy, P.: Ticc: A Tool for Interface Compatibility and Composition. In: CAV 2006. LNCS, vol. 4144, pp. 59–62. Springer (2006)
2. de Alfaro, L., Henzinger, T.A.: Interface Theories for Component-Based Design. In: EMSOFT 2001. LNCS, vol. 2211, pp. 148–165. Springer (2001)
3. de Alfaro, L., Henzinger, T.A.: Interface-based Design. In: Engineering Theories of Software-intensive Systems. NATO, vol. 195, pp. 83–104. Springer (2005)
4. Bauer, S.S., Mayer, P., Schroeder, A., Hennicker, R.: On weak modal compatibility, refinement, and the MIO Workbench. In: TACAS 2010. LNCS, vol. 6015, pp. 175–189. Springer (2010)
5. COMBEST, `http://www.combest.eu/home/`
6. David, A., Larsen, K.G., Legay, A., Nyman, U., Wasowski, A.: ECDAR: An Environment for Compositional Design and Analysis of Real Time Systems. In: ATVA 2010. LNCS, vol. 6252, pp. 365–370. Springer (2010)
7. Eclipse: `http://www.eclipse.org/`
8. Hüttel, H., Larsen, K.G.: The Use of Static Constructs in A Modal Process Logic. In: Logic at Botik 1989. LNCS, vol. 363, pp. 163–180. Springer (1989)
9. Larsen, K.G., Nyman, U., Wasowski, A.: Modal I/O Automata for Interface and Product Line Theories. In: ESOP 2007. LNCS, vol. 4421, pp. 64–79. Springer (2007)
10. Lynch, N.A., Tuttle, M.R.: An introduction to input/output automata. CWI Quarterly 2, 219–246 (1989)
11. Mayer, P.: MDD4SOA: Model-Driven Development for Service-Oriented Architectures. Ph.D. thesis, LMU München (December 2010)
12. MIO Workbench: `http://www.miowb.net/`
13. Raclet, J.B., Badouel, E., Benveniste, A., Caillaud, B., Legay, A., Passerone, R.: Modal interfaces: unifying interface automata and modal specifications. In: EMSOFT 2009. pp. 87–96. ACM (2009)
14. Sensoria: `http://www.sensoria-ist.eu/`
15. SPEEDS, `http://www.speeds.eu.com/`
16. UML4SOA – A profile for modeling service behaviour in UML: `http://www.uml4soa.eu/`