FIONA – Documentation

A Tool to Analyze Interacting Open Nets
FIONA, version 3.0-unreleased, October 21, 2008
`http://service-technology.org/fiona`
`fiona@service-technology.org`

Peter Massuthe and Daniela Weinberg
Humboldt–Universität zu Berlin
Institut für Informatik
Unter den Linden 6
10099 Berlin, Germany
`{massuthe,weinberg}@informatik.hu-berlin.de`

Abstract

This documentation is for Fiona, version 3.0-unreleased, a tool designed to check behavioral correctness of a service and to analyze the interaction of services in service oriented architectures, for instance, last updated on October 21, 2008. As a formal model for services Fiona uses *open nets*, a special class of Petri nets. Fiona implements very efficient data structures and algorithms, which have partly been adapted from the Petri net model-checker LoLA. Fiona has been proven to be applicable in practice by service designers, service publishers, and service brokers.

In this documentation we describe the analysis scenarios for which Fiona can be used. Further, we give a detailed view on how Fiona is invoked and we discuss the structure of the input file formats Fiona accepts.

Copyright

Fiona is maintained by Peter Massuthe, Daniela Weinberg, and Karsten Wolf (coordinator, `karsten.wolf@informatik.uni-rostock.de`).

Contents

1 Introduction

This documentation presents Fiona, version 3.0-unreleased, a tool to analyze the interaction of services. Its features cover the check of controllability and the construction of an operating guideline of a service as well as other derived notions. *Controllability* [1, 2] is a minimal correctness criterion of a service stating the existence of a behaviorally compatible partner for the service. An *operating guideline* (OG) [3] of a service is an operational characterization of *all* behaviorally compatible partners of this service. As a formal model for services Fiona uses *open nets* [4, 3], a special class of Petri nets that extend classical Petri nets by an interface for communication with other open nets.

The development of Fiona started in 2006 as a reimplementation of a tool called Wombat (available at `http://www.informatik.hu-berlin.de/top/wombat`). Wombat was designed to constructively decide controllability of an open net (called workflow module in Wombat). The main reasons for a reimplementation were (1) a completely new theoretical foundation for deciding controllability, (2) a desired focus on efficiency rather than mere effectiveness, (3) the newly developed concept of an operating guideline, and, thus, (4) the need for a separation of computing compatible interactions from reduction rules to ensure efficiency of the algorithms.

Currently, Fiona is developed distributedly by the groups of Wolfgang Reisig (Humboldt-Universität zu Berlin, Germany) and Karsten Wolf (University of Rostock, Germany). It is maintained by Daniela Weinberg, Peter Massuthe, and Karsten Wolf (coordinator).

Fiona is released as free software under the terms of the GNU General Public License. It is written in C++ and its distribution is based on the GNU autotools, which provide the possibility to run Fiona on most operating systems. It compiles on MS Windows® (with Cygwin), Unix (Solaris), Linux, and Mac® OS.

The functionality of Fiona comprises (among others) the following main analysis scenarios:

Controllability. Controllability of an open net N is decided by synthesizing a partner of N (as an automaton called *interaction graph* (IG) [2]). If no partner can be synthesized (i.e. the IG is empty), then N is un-controllable.

Operating guideline. An operating guideline (OG) [3] is a finite characterization of *all* behaviorally compatible partners by annotating a single partner (as automaton) with Boolean formulae in order to derive all other partners.

Matching. Given an open net M and an operating guideline OG_N of an open net N, Fiona decides whether M is behaviorally compatible to N by matching M with OG_N. Matching is more efficient than composing the two nets and model checking the composition (as proposed by other approaches, like the *public view* approach).

Partner synthesis. Given an open net N, Fiona computes a behaviorally compatible partner open net M (if possible). The synthesis can be triggered to construct a small M (with respect to communication) or a partner M which exhaustively communicates with N.

Fiona is a stand-alone tool, designed to be used as a background service of existing service modeling tools. Therefore, Fiona has no graphical interface — the analysis task as well as the input file(s) are given to Fiona via command line options. Fiona then computes and reports the result and, if needed, generates the output file(s).

2 Context

Fiona can be used within the new computing paradigms of service-oriented computing (SOC) and service-oriented architectures (SOA), as well as other areas of intra- and interorganizational business process modeling and analysis. Therein, a *service* represents a self-contained software unit that offers an encapsulated functionality via a well-defined interface. Services are used as building blocks to implement complex, highly dynamic, and flexible business processes. SOAs introduce a *service broker* to organize the challenges of *service discovery*, i.e. the publishing and management of available services and the introduction of procedures to enable a client to find and use such a service.

Controllability is a minimal requirement for the correctness of a service and is particularly relevant for service designers. Operating guidelines are suited to support service discovery and can be used to decide *substitutability of services* [5] or to generate *adapters* for incompatible services [6]. Thus, Fiona is intended to be used by service designers, by service providers, and by service brokers.

3 Download and Installation

3.1 How to Get Fiona

You can download the latest stable release of Fiona at

`http://service-technology.org/files/fiona`.

A nightly build of the most current (possibly unstable) version of Fiona is available at

`http://www.informatik.uni-rostock.de/~nl/cc`.

The Fiona project is hosted by `Gna.org`. Fiona's Subversion (SVN) repository can be checked out through anonymous access over the SVN protocol (TCP

3690), or over http. Depending on your network configuration you may prefer to use one or the other.

For checking out using the SVN protocol (TCP 3690), type

```
> svn co svn://svn.gna.org/svn/service-tech/trunk/fiona
```

or, if you prefer to use http, type

```
> svn co http://svn.gna.org/svn/service-tech/trunk/fiona
```

The distribution of FIONA was created using the GNU autotools. Detailed information on how to build and install FIONA can be found in the file "`INSTALL`" in the root directory of FIONA. If you are familiar with the setup and installation procedure of any GNU tool or you already have a binary of FIONA you may skip the rest of this section.

3.2 How to Build FIONA

The steps of how to build FIONA depend on whether you downloaded a tar-ball containing the current version of FIONA or whether you checked out the SVN repository.

3.2.1 Preparing the SVN Respository

FIONA was created using the GNU autotools. So, in order to install FIONA you need to do the following steps. Successively type

```
    > aclocal
    > autoconf
    > autoheader
    > automake -a -c
```

Alternatively, you can also type

```
    > autoreconf -i
```

which will do the steps mentioned above for you.

3.2.2 Preparing the tar-ball Distribution

After successful download of FIONA, version 3.0-unreleased, change into your download directory and type

```
    > tar xfz fiona-3.0-unreleased.tar.gz
    > cd fiona-3.0-unreleased
```

3.2.3 Building Fiona

Once you have finished with the steps described above, you will find a file called `configure` in your Fiona directory. Run the configure shell script by entering

```
> ./configure
```

which attempts to guess correct values for various system dependent variables used during compilation and creates a "`Makefile`" in each directory of the package. Then, to build the sources, enter

```
> make
```

After compilation, a binary "`src/fiona`"[1] is generated. If you experience any compiler warnings, don't worry: Fiona contains some generated or third party code that is not under our control. To install the binary, type

```
> make install
```

You might need superuser permissions to do so.

If you wish to fully delete Fiona from your system, type

```
> make uninstall
```

Afterwards you just have to delete the directory of your Fiona distribution/ installation. Then there should be no trace leading to Fiona on your system anymore.

The setup and installation procedure can be customized by running `./configure` with several command-line options. Type

```
> ./configure --help
```

for more information.

3.3 Software Dependencies

In order to compile Fiona correctly and to use the whole functionality of Fiona please make sure that the following software is installed on your machine.

- CygWin (`http://cygwin.com/`) (for Windows users only)
- in case you checked out our SVN repository, you will need
  - autoconf (`http://www.gnu.org/software/autoconf/`) and
  - automake (`http://www.gnu.org/software/automake/`)
  - Flex (`http://flex.sourceforge.net/`)
  - Bison (`http://www.gnu.org/software/bison/`)
- GCC (`http://gcc.gnu.org/`)

[1]Under Windows the binary is called "`fiona.exe`"

– Graphviz dot (`http://www.graphviz.org/`) to automatically create PNG files
– petrify (`http://www.lsi.upc.edu/~jordicf/petrify/`) — only needed for partner synthesis
– CUDD (`http://vlsi.colorado.edu/~fabio/CUDD/`) for creating Binary Decision Diagrams (BDD) (the CUDD package is already part of the FIONA distribution)

3.4 Testing Your Compilation

The distribution of FIONA provides an extensive testsuite. In the directory called "`tests`" you can find script files (`*.sh`) which define a set of tests that correspond to certain functionalities of FIONA. The input files of each such test script are located in a directory within the `tests` directory that has the same name as the script.

In order to test whether your FIONA compilation works well on your system there are special make targets. The command

```
> make check
```

initiates FIONA to perform a number of self-tests: the test scripts (`<test>.sh`) are successively invoked. Type

```
> make check-<test>
```

to invoke a single test script `<test>.sh` in the `tests` directory. For instance, `make check-samples` calls the script `samples.sh`.

3.5 Further Make Targets

The generated Makefiles of FIONA provide different make targets. To build a FIONA Windows executable which is independent of CygWin, type

```
> make win32
```

To build a 64 bit executable of FIONA, type

```
> make 64bit
```

In case you use the FIONA distribution from our SVN repository and you update the files rather frequently, then it is advisable to clean up your current compilation before compiling it again. FIONA provides a special make target that removes all generated files from your system. Type

```
> make clean
```

4 Running Example

In the following sections we will describe the analysis scenarios of Fiona with the help of a small example. Fig. 1 shows the open net model of an online shop. For a more detailed introduction into open nets, please refer to [4, 3].

Initially, the shop waits for a customer to log in (transition ?login). Then, the shop decides internally whether the customer is new to the shop or if the customer has used the shop before. That decision is reported to the customer (transitions !new or !known). So, the customer receives either message. If the shop did not recognize the customer, the shop expects the customer to accept the terms of payment of the shop (modeled by sending a message terms — transition ?terms) and to send the order (modeled by sending a message order — transition ?order). Afterwards, the shop sends the delivery notice to the customer (transition !deliver). If, however, the customer is known to the shop, it will first expect the customer to send out his order (transition ?order) and then, it will send a delivery message to the customer (transition !deliver). The online shop reaches its final state after the delivery notice has been sent out (place p9 is marked).

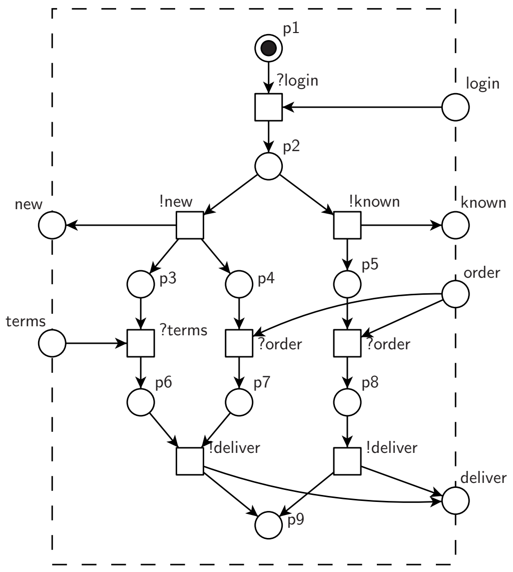

Figure 1: Online Shop as an Open Net

You can find the example online shop in file `docShop.owfn` in the documentation directory of your Fiona distribution.

5 Input and Output File Formats

Fiona supports different types of input files — open net files and operating guideline files. Here, it ignores every file extension and checks the type of an input file by its content. So, an open net file or an operating guideline file may contain any file extension, for example `.net` or `.whatever` is possible.

5.1 Open Nets

The open net file format (`.owfn`) has been adapted from the LoLA [7] file format. It is fairly easy to read and to comprehend. So, it is possible to model an open net manually. Additionally, there exists a compiler BPEL2oWFN (available at `http://service-technology.org/bpel2owfn`) that implements a feature-complete open net semantics [8] for WS-BPEL and automatically generates an open net file for a given WS-BPEL process.

The Fiona distribution contains two directories, `nets` and `tests`, where you can find many example open nets. The open net files have the extension `owfn` (open workflow net).The name of the open nets which are the output of Fiona and BPEL2oWFN will always end with `.owfn`.

In the remainder of this documentation we will always refer to such an open net file by calling it *owfn-file*.

The two main differences between an open net and a Petri net are (1) the set of places is split up into *internal*, *input*, and *output* places, and (2) there exists a set of final markings.

Therefore, the open net file consists of four parts:

1. Places
2. Initial Marking
3. Final Markings, and
4. Transitions

We will describe each part in the following sections by successively specifying the online shop of Sect. 4.

You may comment the contents of the file. A comment is enclosed with curly brackets, i.e. `{ comment }`. However, you may not use `{$ ... $}`. This notion is used for debug purposes.

5.1.1 Places

Each open net file starts with the definition of its places. As mentioned above, the set of places is divided into three (pairwise disjoint) parts: internal places (keyword `PLACE`), input places (`INPUT`), and output places (`OUTPUT`).

The online shop consists of 15 places: places p1, ..., p9 are internal places, places login, terms, and order are input places, and places new, known, and deliver are output places. We specify this as follows.

```
PLACE
  p1, p2, p3, p4, p5, p6, p7, p8, p9;
INPUT
  login, terms, order;
OUTPUT
  new, known, delivery;
```

The elements of a set of places are separated by comma. Each list ends with a semicolon. The place names are arbitrary and do not conform to any scheme. An empty set of input places (output places analogously) can either be specified by

```
INPUT
  ;
```

or by simply omitting the keyword `INPUT` and the following `;` completely.

5.1.2 Initial Marking

The `PLACES` section is followed by the `INITIALMARKING` section. Every place which is not listed in the `INITIALMARKING` section is implicitly assumed to have zero tokens initially. If you want to create an initial state in which some places are marked, you have to list exactly those places in the `INITIALMARKING` section. Make sure that you specify the correct number of tokens of each place.

Initially, the online shop waits for a customer to log in. So, the initial marking of the shop is one token on place p1.

```
INITIALMARKING
  p1: 1;
```

You may omit the explicit *one token*. If you do not specify how many tokens there shall be, then FIONA implicitly assumes the marking to be one token only. So, the following specification has the same meaning as the one stated above.

```
INITIALMARKING
  p1;
```

Suppose, we would have specified the initial marking to be two tokens on place `p1`. Then you would specify this by

```
INITIALMARKING
  p1: 2;
```

or

```
INITIALMARKING
  p1, p1;
```

which has the same meaning.

5.1.3 Final Markings

The third section lets you specify one or more final markings. There are two possibilities to define final markings which are described in the following: either `FINALMARKING` or `FINALCONDITION`.

Final markings. The final marking is specified in the same manner as the initial marking (see Sect. 5.1.2). The final marking of the online shop is one token on place `p9`. All other places of the shop have to contain no token. You can specify this as follows.

```
FINALMARKING
  p9 : 1;
```

You may also specify a set of final markings. Suppose, you want the shop to have three final markings: (1) a token on `p9`, (2) a token on `p6` and `p7`, and (3) a token on `p8`. You can specify this by

```
FINALMARKING
  p9;  p6, p7;  p8;
```

If your net should not contain any final marking, you have to omit the section `FINALMARKING` continuing with the transitions section (see Sect. 5.1.4).

In contrast, specifying

```
FINALMARKING
  ;
```

characterizes exactly one final marking which is the unique marking where every place of your net contains no token.

Final conditions. You can also use a powerful, implicit characterization of final markings by specifying a FINALCONDITION instead of FINALMARKING.

```
FINALCONDITION
  condition;
```

Then, every marking that *satisfies* the condition is accepted as a final marking. A condition has the following structure:

```
Condition := Place COMP Token | Condition OP Condition |
             Condition OP OTHERS-EMPTY | '(' Condition ')'
COMP := '=' | '>' | '<' | '<=' | '>='
OP := 'AND' | 'OR'
OTHERS_EMPTY := 'ALL_OTHER_PLACES_EMPTY' |
                'ALL_OTHER_INTERNAL_PLACES_EMPTY' |
                'ALL_OTHER_EXTERNAL_PLACES_EMPTY'
```

So, a condition is either

- a declaration of how many tokens a place has to hold (p1 = 2), or
- a logical connection of two conditions using AND or OR, or
- a logical connection of a condition and a OTHERS_EMPTY predicate using AND or OR, or
- bracketing a condition.

For instance, the condition

```
FINALCONDITION
  p9 = 1;
```

accepts each marking with exactly one token on place p9 and arbitrarily many tokens on the other places as a final marking. The condition

```
FINALCONDITION
  p9 = 1 AND p1 = 0 AND p2 = 0 AND p3 = 0 AND
  p4 = 0 AND p5 = 0 AND p6 = 0 AND p7 = 0 AND
  p8 = 0 AND login = 0 AND terms = 0 AND
  order = 0 AND new = 0 AND known = 0 AND
  delivery = 0;
```

for the example online shop exactly corresponds to

```
FINALMARKING
  p9 : 1;
```

The special predicates in OTHERS_EMPTY can be used to avoid the lengthy definitions of AND p1 = 0 AND p2 = 0 AND ... (as done in the example final condition above) by requiring emptiness of a set of places. Such predicates can only be used in combination with a ”normal” condition and the specified set depends on the places named in the condition.

For instance, the final condition

```
FINALCONDITION
  (p9 = 1 AND ALL_OTHER_PLACES_EMPTY) OR
  (p8 = 1 AND ALL_OTHER_PLACES_EMPTY)
```

accepts final markings in which either (a) p9 contains 1 token and all other places of the shop contain zero tokens, or (b) p8 contains 1 token and all other places contain zero tokens. That is, the first ALL_OTHER_PLACES_EMPTY predicate specifies emptiness of the places p1, ..., p8, whereas the second one specifies emptiness of p1, ..., p7, and p9.

5.1.4 Transitions

The last part of an open net file is an enumeration of the transitions of the net.

Suppose you would like to define transition ?login, which consumes one token from place p1 and one token from input place login and produces a token on place p2. Then you would specify it as follows.

```
TRANSITION t1       {?login}
  CONSUME p1, login;
  PRODUCE p2;
```

By careful by naming each transition of the net. The names have to be unique. That is why we chose to name that transition t1 with ?login as a comment only. The net might contain another ?login transition.

The definition of the set of places from which the transition consumes tokens and to which it produces tokens is again specified as stated in Sect. 5.1.2.

5.1.5 Open Net Specification of the Online Shop

The first part of the open net specification of the online shop looks like this.

```
{ "online shop" of fiona documentation }

PLACE INTERNAL
  p1, p2, p3, p4, p5, p6, p7, p8, p9;
INPUT
  login, order, terms;
OUTPUT
  new, known, deliver;

INITIALMARKING
  p1;

FINALMARKING
  p9;

TRANSITION t1  { ?login }
CONSUME
  p1, login;
PRODUCE
  p2;

TRANSITION t2  { !new customer }
CONSUME
  p2;
PRODUCE
  p3, p4, new;

...
```

5.2 Operating Guidelines

Fiona is able to read a textual representation of an operating guideline stored in a special file format (og-file with extension `.og`). Within an og-file the OG is stored as an annotated graph: every node is annotated with a formula and a color. There exists a distinguished initial node and all nodes are connected by labeled edges (`TRANSITIONS`).

Figure 2 shows the operating guideline of the online shop. The following specification describes the first part of the og-file of the shop. You can find the text file (`docShop.owfn.og`) in the documentation directory of your Fiona distribution. Or, you can easily generate it by having Fiona calculate the operating guideline of the shop.

```
NODES
    0 : (!login + !order) : blue,
   17 : !login : blue,
   18 : (?new * (?deliver + ?known)) : blue,
   19 : !terms : blue,
         ...
   21 : final : blue : finalnode,
         ...
INITIALNODE
    0;

TRANSITIONS
    0 -> 17 : !order,
    0 -> 38 : !login,
   17 -> 18 : !login,
   18 -> 19 : ?new,
   18 -> 28 : ?deliver,
         ...
```

As you take a closer look at the specification you can see that node `21` has an additional attribute – `finalnode`. So, besides annotating a node with its boolean annotation and its color, you can also declare a node to be a final node of the operating guideline. In the graphics such a node will always be depicted with two surrounding lines.

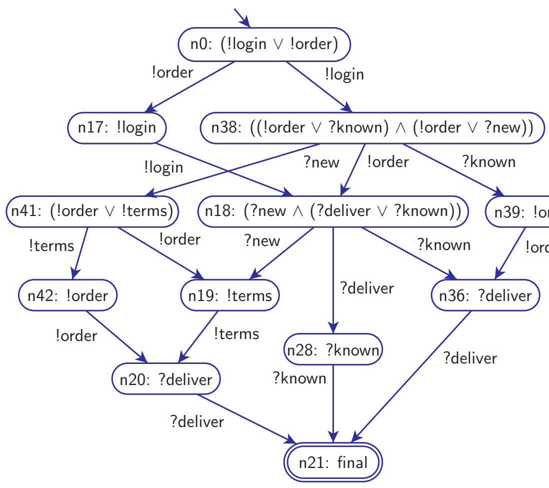

Figure 2: Operating Guideline of Online Shop of Sect. 4.

5.3 Adapter Rules

One of the analysis scenarios of Fiona is to generate an adapter which functions as a mediator between different nets (see Sect. 6.2). In order to build such an adapter it is necessary for the algorithm to know which messages can be mapped onto each other. We encode this mapping in a special *adapter rules file* [6], commonly extended with `.ar`.

Consider the online shop as described in Sect. 4 again. Suppose that there exists another net which models a potential customer of the shop. The composition of the two nets, however, does not behave well. As you take a closer look at the services, you notice what the problem is — the customer net is only able to receive a message that is called `delivery_notice` as opposed to the message that the online shop sends out. That message is called `delivery`. You would now specify this fact as an adapter rule as follows.

```
delivery_notice -> delivery;
```

Given the two nets and the adapter rule file Fiona will now be able to construct an adapter for the two nets.

Specifying

```
-> A ;
```

states that the adapter may generate message `A`. And

```
B -> ;
```

specifies that message `B` may be deleted by the adapter. The following rules duplicate message `C` and split message `D` into three parts – `E`, `F`, and `G`

```
C -> C, C;
D -> E, F, G;
```

6 How to Use Fiona

Fiona can be used in a broad range of service analysis scenarios. Each scenario is invoked by a certain option and Fiona's output and behavior can then be modified by various other options. Type

```
> fiona --help
```

to get all available options you can use to control Fiona. By typing

```
> fiona --version
```

Fiona will print out the complete version information.

In the following Sect. 6.1 we describe the main analysis scenarios for which Fiona can be used. In Sect. 6.2 we briefly describe other analysis scenarios and in Sect. 6.4 we show the options that can be used to trigger the analysis tasks and/or to modify the output of Fiona.

6.1 Main Analysis Scenarios

Suppose the online shop of Sect. 4 is stored in file `docShop.owfn` in directory `doc`.

Controllability. You would now like to check if the online shop is controllable or not. So, you type

```
> fiona -t ig doc/docShop.owfn
```

to have Fiona check controllability. Fiona decides controllability by constructing an IG. After computing the IG, Fiona reports whether the net is controllable or not. Furthermore, you will find a graphic file showing the IG in directory `doc`. The graphics is generated by GraphViz Dot (available at `http://www.graphviz.org`).

Depending on the size of the given open net, it may take some time for Fiona to build up the IG. So, it is advisable to have Fiona build up a reduced interaction graph [2]. In general, the resulting graph is significantly smaller than the complete IG and thus it takes less time to compute it and it consumes less memory as it is constructed. Type

```
> fiona -t ig -r doc/docShop.owfn
```

to calculate the reduced IG. There are different reduction techniques that can be applied to the IG. Fiona provides the possibility to apply two of those reduction techniques independently. The parameter `-p cre` refers to the reduction technique *combine receiving events* and the parameter `-p rbs` refers to *receiving before sending.*

```
> fiona -t ig -r -p cre doc/docShop.owfn

> fiona -t ig -r -p rbs doc/docShop.owfn
```

Operating guideline. You would like to construct the operating guideline of the online shop. You can do this by typing

```
> fiona -t og doc/docShop.owfn
```

FIONA reports whether N is controllable or not and again you will find a graphic file showing the OG in directory `doc`. Since FIONA just constructed an OG, it stores the information of the OG textually in a file called `docShop.owfn.og`. You can also find this file in directory `doc`.

You can also invoke FIONA to generate an OG as BDDs. Type

```
> fiona -t og -b 0 doc/docShop.owfn
```

if the generated BDDs shall be reordered by CUDD_REORDER_SAME.

Matching. Consider the following scenario. You would like to check whether your shop is behaviorally compatible to another shop `otherShop.owfn` from which you have given its corresponding og-file `otherShop.owfn.og` only. Use the following command for invoking the matching algorithm.

```
> fiona -t match doc/docShop.owfn ogs/otherShop.owfn.og
```

FIONA will report whether the net matches with the given OG or not. In case the matching fails, FIONA reports where the failure manifests in.

Partner synthesis. Once you have modeled your service as an open net, you may be interested in the possible partner services that can properly interact with your service. You can use FIONA to automatically construct two different types of partners: a (very) small one or a partner that exhaustively communicates with the net.

```
> fiona -t smallpartner doc/docShop.owfn

> fiona -t mostpermissivepartner doc/docShop.owfn
```

The partner open net is then modeled as an open net and you will find the corresponding file `docShop-partner.owfn` in directory `doc`. FIONA generates the open net with the help of the tool Petrify (available at `http://www.lsi.upc.es/~jordicf/petrify/distrib/home.html`).

6.2 Further Analysis Scenarios

In this section we describe further analysis scenarios.

Public View. Suppose you would like to publish your online shop. You do not want to publish any private information about which decision is taken and when. Then you would publish a public view of your shop which hides any details of implementation. Invoke FIONA as follows to generate such a public view.

```
> fiona -t pv doc/docShop.owfn
```

Afterwards, you will find a file `docShop.pv.owfn` that specifies the public view of the shop in directory `doc`. Further, FIONA generates a graphics that shows the open net of the public view.

Substitutability. We distinguish different notions of substitutability [5]: *accordance* and *equivalence*. Suppose you have modeled a new online shop and stored it in file `myNets/myShop.owfn`. Now you would like to check whether the online shop of Sect. 4 can be substituted by your online shop. Here, you have two options. You use FIONA to check whether the set of compatible partners of your online shop is a subset of the set of partners of the documentation online shop. To check this property, type

```
> fiona -t simulation myNets/myShop.owfn doc/docShop.owfn
```

Typically, FIONA expects og-files as its input. If owfn-files are given, FIONA will first generate an operating guideline of both nets and will then run the algorithm. Please be careful by invoking FIONA as the order of the arguments is important. FIONA will print out whether your shop has at least the partners of the documentation shop or not.

The second option is to check whether your online shop characterizes the same set of partners as the documentation online shop. Invoke FIONA as follows to check equivalence.

```
> fiona -t equivalence myNets/myShop.owfn doc/docShop.owfn
```

Here, the order of the given nets does not matter. But again, FIONA expects og-files as its input. If owfn-files are given, FIONA will first generate an operating guideline of both nets to run the algorithm. Finally, FIONA will state whether both nets specify the same partners or not.

Adapter generation. Suppose, in addition to the online shop you have given the model of a customer as an owfn-file. You know that both nets are *not* behaviorally compatible. Further, you have specified an adapter rule file `rules.ar` (see Sect. 5.3) which defines certain message mappings. Then, you can use FIONA to synthesize a partner that functions as an adapter for both nets. If such a partner exists, it mediates between both nets. Thus, the composition of all three nets is well-behaving by construction. For more details see [6]. Invoke FIONA as follows.

```
> fiona -t adapter doc/docShop.owfn myNets/customer.owfn
-a rules.ar
```

By invoking FIONA with

```
> fiona -t smalladapter doc/docShop.owfn myNets/customer.owfn
-a rules.ar
```

a rather small adapter will be generated by using certain reduction rules. In either case a file called `docShop_customer_result.owfn` will be generated which models the adapter of the two nets with respect to the adapter rules given.

6.3 Preprocessing & Conversions

We will now describe the options that either preprocess or convert a given input file.

Invoke Fiona by typing

```
> fiona -t <parameter> FILES
```

Again, depending on the `parameter` the variable `FILES` stands for an open net, an operating guideline or a set of open nets or operating guidelines. The `parameter` may be any of the following.

`<parameter>`	description	`FILES`
`minimizeOG`	minimizes a given OG	og-file
`reduce`	structurally reduce a given open net (choose reduction level using parameter `r1` ... `r5`)	owfn-file
`normalize`	normalize all given open nets	owfn-file
`isacyclic`	check a given OG for cycles	og-file
`count`	count the number of services that are characterized by a given OG	og-file
`checkfalsenodes`	look for nodes that violate their own annotation in a given OG	og-file
`removefalsenodes`	remove all nodes that violate their own annotation in a given OG	og-file
`png`	generate png file of given open net	owfn-file

6.4 More Options

Fiona provides different options to modify the output, to speed up the computation or to control the analysis tasks, for instance. In the following we will briefly describe every option.

Set message maximum. Set the maximum number of same messages per state to `<level>` (in [3] this value is called *message bound*).

Option: `--messagemaximum = <level>` or `-m <level>`

If no message maximum is set manually, the default value `1` is assumed.

Reduce IG. Generate a reduced IG by applying reduction techniques while the graph is calculated. By default, the techniques *combine receiving events* and *receiving before sending* are applied. This can be changed by specifying the desired techniques as parameters to option `-p`. If any technique is given, only the specified reduction techniques are used.

Option: `--reduceIG` or `-r`

Reduce Node states. This reduction technique stores less states in each node of the IG/OG and during the calculation of the graphs. So, in general it reduces memory. However, it might increase the computation time.

Option: `--reduce-nodes` or `-R`

Show additional information. There are different display options for the graphics of operating guidelines and interaction graphs.

Option: `--show = <parameter>` or `-s <parameter>`

Parameters:

`<parameter>`	description
`allnodes`	show all nodes of the graph
`blue`	show blue nodes only (default)
`rednodes`	show blue and red nodes (no empty node and no black nodes)
`empty`	show empty node
`allstates`	show all calculated states per node
`deadlocks`	show all but transient states

BDD Construction. An operating guideline can be represented with BDDs. Here, the BDD is created after the operating guideline has been calculated. The argument `<reordering>` specifies the type of reodering that is used.

Option: `--BDD = <reordering>` or `-b <reordering>`

`<reordering>`	type
0	CUDD_REORDER_SAME
1	CUDD_REORDER_NONE
2	CUDD_REORDER_RANDOM
3	CUDD_REORDER_RANDOM_PIVOT
4	CUDD_REORDER_SIFT
5	CUDD_REORDER_SIFT_CONVERGE
6	CUDD_REORDER_SYMM_SIFT

7	CUDD_REORDER_SYMM_SIFT_CONV
8	CUDD_REORDER_WINDOW2
9	CUDD_REORDER_WINDOW3
10	CUDD_REORDER_WINDOW4
11	CUDD_REORDER_WINDOW2_CONV
12	CUDD_REORDER_WINDOW3_CONV
13	CUDD_REORDER_WINDOW4_CONV
14	CUDD_REORDER_GROUP_SIFT
15	CUDD_REORDER_GROUP_SIFT_CONV
16	CUDD_REORDER_ANNEALING
17	CUDD_REORDER_GENETIC
18	CUDD_REORDER_LINEAR
19	CUDD_REORDER_LINEAR_CONVERGE
20	CUDD_REORDER_LAZY_SIFT
21	CUDD_REORDER_EXACT

On the Fly BDD Construction. An operating guideline can also be calculated as BDDs on the fly.

Option: `--OnTheFly = <reordering>` or `-B <reordering>`

Parameters: See *BDD Construction* for possible parameters.

Output prefix. Sets a prefix string to all output files.

Option: `--output = <filename prefix>` or `-o <filename prefix>`

No output. No output will be generated at all.

Option: `--no-output` or `-Q`

Additional parameters. Further modification of the execution.

Option: `--parameter = <parameter>` or `-p <parameter>`

Parameters:

`<parameter>`	description
`no-png`	does not create a PNG file
`no-dot`	does not create dot related output
`r1 - r5`	set reduction level in mode `"-t reduce"`
`cre`	use the “combine receiving events” – reduction technique to reduce the IG
`rbs`	use the “receiving before sending”– reduction technique to reduce the IG

Debug level. The default compilation of FIONA provides the possibility to get complete debug information.

Option: `--debug = <level>` or `-d <level>`

Here the parameter `level` may be any of the following values, depending on the degree of detailed information you wish FIONA to output.

`<level>`	description
1	show nodes and dfs information
2	show analysis information (i.e. colors)
3	show information on events and states
4	yet to be defined ;)
5	show detailed information on everything

The debug compilation of FIONA is not optimized to reach a high-performance. If you would like your FIONA compilation to work better with respect to time, then you should configure it as follows:

```
> ./configure --disable-assert
```

Running configure with the mentioned parameter results in the definition of a preprocessor directive called `NDEBUG` in file `src/fiona.h`.

7 Some last Words

We hope, that you will have fun using FIONA and that our tool actually helps you to reach your goals. If you should, however, discover unusual behavior of FIONA, please do not hesitate to send us an email (`fiona@service-technology.org`). Describe the problem you have encountered briefly, state the exact call that led to the buggy behavior and please also add the output of the command

```
> fiona --bug
```

to your email. That call lets FIONA state some facts about your local compilation. If possible, it would be very helpful if you could attach your input files as well.

Thanks for using FIONA.

References

[1] Schmidt, K.: Controllability of Open Workflow Nets. In: EMISA 2005. LNI, Bonner Köllen Verlag (2005) 236–249

[2] Weinberg, D.: Efficient controllability analysis of open nets. In: WS-FM 2008. LNCS, Springer-Verlag (2008) accepted.

[3] Lohmann, N., Massuthe, P., Wolf, K.: Operating Guidelines for Finite-State Services. In: ICATPN 2007. Volume 4546 of LNCS., Springer-Verlag (2007) 321–341

[4] Massuthe, P., Reisig, W., Schmidt, K.: An Operating Guideline Approach to the SOA. AMCT **1**(3) (2005) 35–43

[5] Stahl, C., Massuthe, P., Bretschneider, J.: Deciding Substitutability of Services with Operating Guidelines. Technical Report 222, Humboldt-Universität zu Berlin, Germany (2008) accepted for a journal.

[6] Gierds, C., Mooij, A.J., Wolf, K.: Specifying and generating behavioral service adapter based on transformation rules. Technical Report CS-02-08, Universität Rostock, Germany (2008) submitted to a conference.

[7] Schmidt, K.: LoLA: A Low Level Analyser. In: ICATPN 2000. Volume 1825 of LNCS., Springer-Verlag (2000) 465–474

[8] Lohmann, N.: A feature-complete Petri net semantics for WS-BPEL 2.0. In: WS-FM 2007. Volume 4937 of LNCS., Brisbane, Australia, Springer-Verlag (2008) 77–91