Common Thomas Network Analysis Interface

Specification v1.0

Adam Streck

14.10.2012

Table of Contents

1 Introduction

This document provides specification of an interface purposed for user-friendly manipulation with both current and future tools for parameter identification in Thomas networks. Main purpose of the whole mainframe is to allow for reverse engineering of models of regulatory networks. We want to provide maximal capabilities for a user-guided process of restriction of the parametrization space by evaluating diverse properties of the members of the space. The user is expected to collaborate on specification of such properties and to decide which values should be considered unsatisfactory. The tool is expected to execute analysis as demanded by user and to provide user with results in the form of so-called marking or classification i.e. association of each of the parametrizations with a result of the analysis conducted.

The specification is not required to be followed in detail nor to be taken literally and should serve as a mere set of suggestion.

The document expects the reader to have basic understanding of Thomas formalism and the tools specified within the document.

1.1 Purpose of this document

This document should serve as a development guideline for further design of the Common Thomas Network Analysis Interface (work-title). This document does not describe the design of the application itself, neither it specifies methods or technologies that should be employed.

1.2 Scope of the project

Main purpose of the interface is to provide uniform and easy-to-use access to applications already existing or in development. The interface should be usable either on publicly accessible server or in an offline mode on a local computer and therefore there should be no assumptions about

- the hardware platform,
- the operating system,
- the file system and the access rights out of the scope of the folder hosting the interface itself.

The interface should provide best-effort functionality on any platform.

The interface should be strictly sever-based. No code should be executed on the client side and no data should be stored either.

The interface should include as little functionality as possible, relying only on underlying applications. For each data manipulation task the interface should call an appropriate tool. We propose such a modular design to ease independent development of underlying tools and to help with possible further development by breaking the potential monolithicity.

1.3 Definitions

In this section we define, for clarity, some terms associated with the Thomas formalism and also the names of the tools and methods referenced throughout the document.

1.3.1 Terms

Context: A subset of regulators a component.

Interaction graph: A directed labelled multigraph (V,E) (also called regulatory network) with:

1. V being a set of labelled vertices. Each vertex is a pair (v,m) with v being its name and $m\in\mathbb{N},m\neq 0$ being its maximal activation level. We use the term component for a vertex.
2. E being a set of labelled edges i.e. triples $((v,m),(v',m'),t):t\in\mathbb{N},0\leqslant t\leqslant m$ with v being the regulator, v' being the regulated and z being the threshold. We use the term regulation for an edge.

Parameter: For a component (v,m) we use the term (logical) parameter for a pair formed from a context of v and its activation value $n:0\leqslant n\leqslant m$.

Parametrization: A set of parameters of a regulatory network. A parametrized graph is called model.

Regulation: An asymmetric interaction between two components.

Regulator: A component of a regulation influencing its effect.

1.3.2 Names

CAPCHA: A web element devised to block usage of the interface by bots.

Classifier: An adapter for the TomClass application. Each adapter creates a bilateral connection between the TomClass application and a third party tool e.g. NuSMV.

CytoscapeWeb: Publicly available web based implementation of the Cytoscape tool.

Interface: The application specified in this document.

Parsybone: A compiled tool for LTL model checking of Thomas networks

TomClass: An application for ranking of parametrizations in Thomas network.

1.4 Overview of the document

First, we describe general appearance of the interface itself. Independent parts of the interface are given a swift description as well. Further we provide coarse specification of sub-functions the interface should provide and architecture of their interoperability.

For each sub-function available in the interface we use the term widget – in this context we use the term to point out that with respect to the user interface, these sub-functions should appear and behave equally.

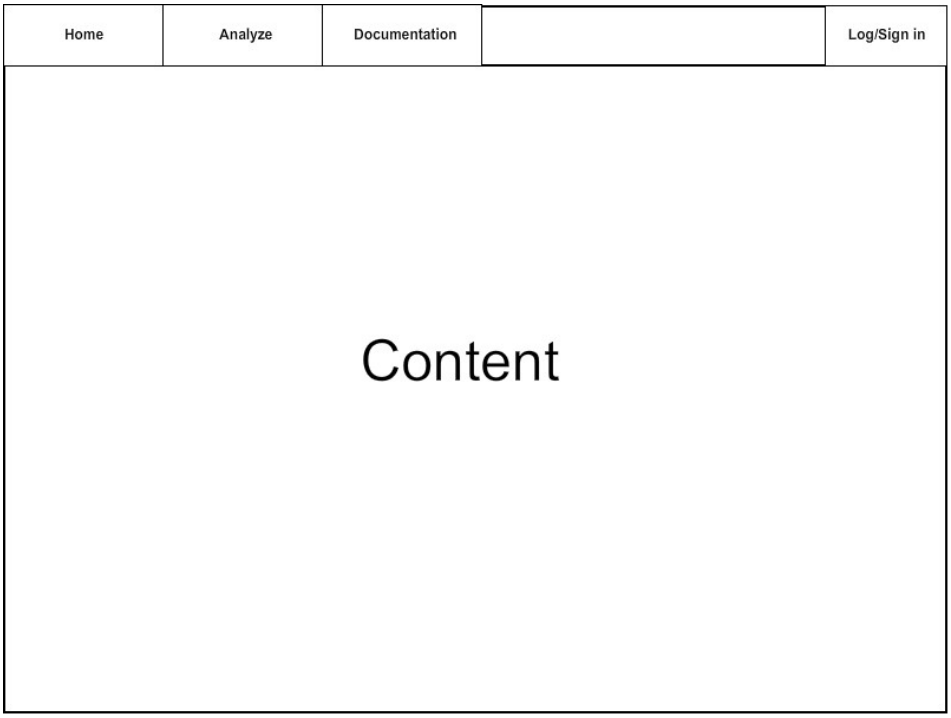
In the second part we give more detailed description of some architectural concepts. Finally, we provide few suggestions for possible ways of implementing the functionality.

Note that parts given in grey font are not to be included in the current version of the tool, however their additional implementation should be anticipated and therefore the current architecture should ease their possible addition.

2 General system description

The system is based on a set of server-side applications that are executed on demand. This set should be accessible through a single web-page which connects to the applications using internal adapters. This page should provide all the required functionality and content of the underlying tools through a possibly extensible set of widgets.

Given that the interface is supposed to be publicly available, this page should be accompanied by other web pages e.g. page for a user account management. We present a sketch of possible structure:



2.1 Overview of pages

This section contains short description of individual pages that are accessible through individual HTTP links.

2.1.1 Home

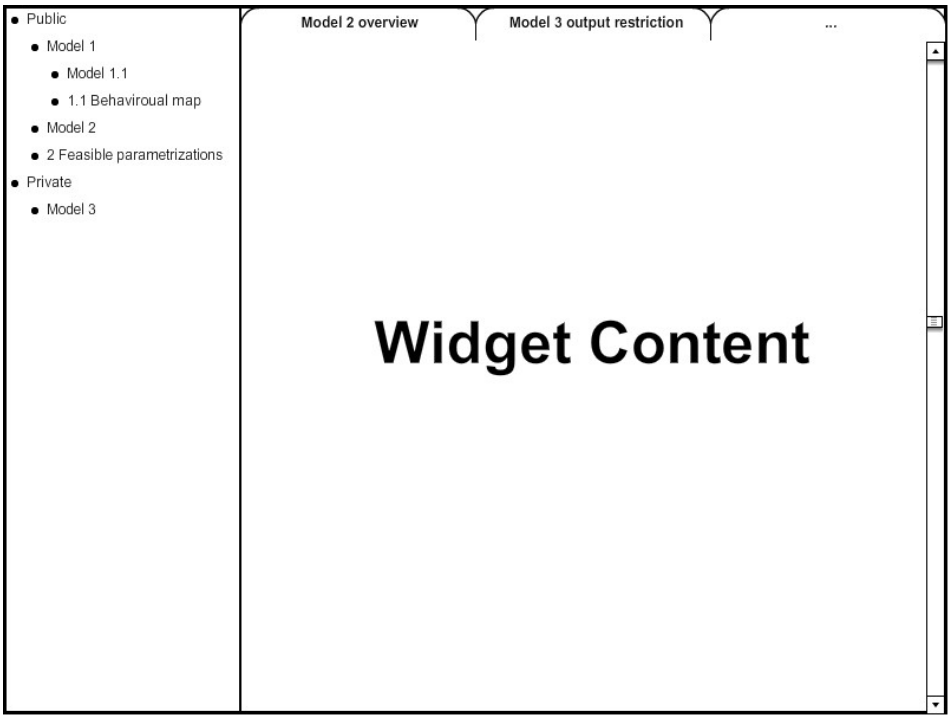
This page should display short description of the interface itself, short list of references and basic legal information. Also a small site map with links to other pages should be included.

The home button might be replaced with actual name / logotype of the tool.

2.1.2 Analyse

The analysis interface should be accessible wholly in the analysis page. For this purpose the page itself must provide capabilities for content access.

We suggest to divide the page into two panes. The left pane should host a file tree and provide instant accessibility for any file. The right pane is intended for displaying of file contents. As the content of each file should be pre-processed before displaying, the right pane should actually contain a widget for utilization of the content. Also a series of tabs for listing between open files should be present. For clarity we give a sketch of a possible layout of the analysis page:



2.1.3 Documentation

This page might be actually divided in several sub-pages if necessary, however we do expect documentation to be present usually in the form of PDF files and therefore this page should merely provide access to such files and their short description for easy navigation.

2.1.4 Log/Sign in

Access policies are necessary for administration of resources and therefore basic account management must be included in the interface. This page should contain only options to log in, using a name and a password, or to sign in. Should the user choose to sign in, content of this page should be replaced with another page containing the registration form. After logging in the Log/Sign in button should be replaced with label containing current user's name and a smaller button for logout underneath the name label.

2.1.5 Registration form

This form may be accessible only from the login page. This form should ask for basic account information e.g. user name, password, e-mail for verification and occupation/institute. The form should be also accompanied with a CAPCHA.

We propose registration to be done on individual basis manually i.e. after validation using the CAPCHA the current content of the registration form should be send to a designated address, checked by an administrator of the web and if correct, the user should be registered in the tool and noticed via e-mail. Automation of the process may be conducted later.

2.2 Basic system logic

The whole system is purposed for reverse engineering of an interaction graph of a regulatory network. Ideologically, this is process composed of three consecutive steps:

1. Regulatory network description.
2. Models analysis.
3. Results properties extraction.

These steps may influence each other and in the end should provide as exact and vast knowledge of the network as possible.

2.2.1 Regulatory network description

In the first step, the user is required to put in as much information about the structure of the interaction graph as possible – unknown behaviour gives rise to the set of parametrizations, corresponding to all possibilities of the behaviour the model can express.

There should be multiple levels of components the regulatory network can contain based on their effect on parametrization space and also some visual clues telling the user to which level the model currently belongs. These levels are:

1. Number of components and their value range. Defines transition graph.
2. Edges and their thresholds. Defines regulatory contexts.
3. Local parametrizations. Restricts the parametrization space.

The third step may seem to be more relative to procedures of classification. However it is important to note that classification processes, e.g. model checking, have to consider the whole initial space and therefore are a mere marking of the space, the step three actually greatly restricts the parametrization space even before it is enumerated and for this reason we consider it to be a part of model definition rather than analysis.

2.2.2 Model analysis

In this step, the user adds some new information i.e. time series and allows the models to be ranked. This step allows to distinguish between models validated by the model checking process which are in the basic approach considered equal. In this step the user works with the database of models only.

2.2.3 Results properties extraction

In this step the user obtains data from the ranked models and uses them to obtain additional knowledge that may be employed for restrictions in steps one and two. This step produces usually files that are build from the database as a highly-specific output and are not used by other parts of the system.

3 Inner systems description

In this section we describe what functionalities the server should provide.

3.1 Account system overview

Functionality should be mainly available only for registered and logged users as usage of the analysis tool is likely to consume servers resources. Demo files may be available within the analysis tool for unregistered users to give them some ideas about the functionality.

3.1.1 File access

Each file can have different policies based on the status of the user, which may be:

1. Owner of the file.
2. Member of one of the groups specified in the file.
3. Other user.

For each of the files the user can have one of the following permissions:

1. Change (write) the file.
2. Read the file.
3. Use the file in tools (execute).

In the basic system model the owner of the file should have all the privileges and while the others are given none. To allow access for other user the owner must directly change the file privileges and also the owner should be the only one who can do so.

For simplicity only one user should be allowed to write the file at the time. If the file is being edited, other users should be restricted from access, even if the request is made by the owner. Restriction may be either enforced fully or the user may be given right to access the file in the read-only mode. Further development may allow for more firm access privileges.

Execution of the file, meaning the process of producing some sort of result using a tool, produces a file or files that belong to the user that started the execution whoever might have been the owner of the file.

3.1.2 Resource access

To avoid congestion of the server we must endorse rather strict resource access. Each user should have at most 10GB of accessible data storage and be allowed to have at most two processes running while logged in.

When logged in the user is not allowed to actually to overcome 5GB limit. The remaining 5GB should be allowed only for storage of temporary files produced by analysis. For simplicity, this may mean that user is allowed to have 5GB of files and as soon as some of his running processes causes the limit to be overcome, he is restricted from starting another sort of analysis, unless the size of the file is reduced so that the whole directory of the user is below 5GB again. If the user logs out these files should remain in memory.

Restriction on two processes means that apart from the thread used for actual communication with the user, the user can also start two tools/programs on the server. After that the user can not start a new computation until one of the two processes ends.

The values proposed here should be considered to be the default ones, however there should be a way how to change them by administrator on an account basis e.g. for offline usage the user should be allowed to set his resources to unlimited.

3.2 Widgets overview

The functionality itself should be available through a set of widgets, each being purposed for different sort of file. Here we give basic description of the widgets and their associated files.

Models should be stored in a database table as well. The content should be dependent on the specification level – each level should have an independent table, therefore different table should be present for:

1. Nodes and their range.
2. Interactions and their thresholds.
3. Local parametrizations.

This is because some model creating / ranking tools may be compatible only with some of those

levels. Notice that these may be a content of the single file.

Since currently the only way how to employ an interaction graph is the Parsybone tool, implementation of an SQL representation can be postponed and Parsybone internal format can be used for models storage.

3.2.1 Interaction graph widget

This is the entry point for regulatory network analysis, which holds specification of the model itself and allows for its editing.

The file should contain the name of the regulatory network and a short description given by the author. Second part of this widget should provide a way how to edit the model, either as a text file or using forms or using visual editor, probably based on the CytoscapeWeb tool.

From this widget the graph is then used for parametrizations enumeration.

Saved changes in the graph should create a copy of the graph and not rewrite the original graph file. In the average case we expect the model to be associated with some result files – changing the graph may render some of those files invalid.

3.2.2 Parsybone widget

As for now, Parsybone is the only tool capable of creating the database from the model. The widget takes a model and possibly a formula and produces database with parametrizations for the current model together with possible ranking.

The widget should be able to pass the execution arguments to the Parsybone tool.

3.2.3 Formula widget

Logical formulas are used for classification of models. A formula can be specified using:

- Büchi automaton
- Time series
- CTL

Note that formulas are always linked with the Level 1 or higher of a model and therefore the user should be lead to using only propositions that are valid for the model.

For the process of defining a formula we again propose to employ a text file or web forms or the CytoscapeWeb .

3.2.4 TomClass Parametrizations Restriction widget

Additional restrictions on parametrization space are allowed using formulas of the answer set programming system. This parametrizations however test each parametrization independently and therefore it is sensible to employ them only as a classifier. The widget takes a model and requires the user to specify constrains on the parametrization space e.g. mutual exclusion of two regulatory effects of different regulators.

This widget takes a parametrization database file and produces a new one with required restrictions given as marking of parametrizations.

3.2.5 TomClass NuSMV widget

A visual interface for the NuSMV based analysis. The tool should allow the user to pick a relevant CTL formula and possibly to specify execution parameters.

This is also a database-processing file.

3.2.6 Parameter View widget

This widget takes a file with parametrizations and to display its results. Main purpose is to allow the user to view the results directly and possibly to pick from them manually.

Since a database may be quite big, the widget should manipulate with it in an intelligent manner – displaying only a part of the database at a time and search capability are some of the possible functionalities the widget could have.

3.2.7 Parameter Filter widget

This widget takes a database file and allows to define a set restrictions on the database e.g. highest cost value the parametrization can have. This is also a widget for database processing and as such it should create a new parametrization file.

In further development inclusion of a form for user-created SQL queries should be added.

3.2.8 Behaviour Map widget

This widget should visualise a behaviour map for parametrizations picked in the Parameter View widget (possibly all of them). This view should incorporate CytoscapeWeb for manipulation with the map as layouting is not expected when the map is produced. If such incorporation is not possible, the map should be presented in the form of a picture, but in this case the map must be sorted beforehand.

3.2.9 Parametrization Properties widget

This widget will be used for inclusion of a tool that is not yet in development. Purpose of the tool is to find or test more complex properties within the set – e.g. mutual independence of local parametrizations.

3.3 File tree overview

The file tree pane should contain all the files the user has access to. Each model file should give a single node in the tree and be accompanied by all its associate files. Since we can lay model is a partial ordering, the structure of the tree should be generated automatically based on a partial order given by model inclusion.

There are many possibilities of how the analysis files can be attached – they may be present freely, listed under their respective node or clustered by their type etc.

Opening a model / analysis file should create a new tab in the content pane presenting data from the file using the appropriate widget.

3.3.1 Inclusion system

Newly created model should be automatically inserted in the tree structure. Its position is defined by its specification level as proposed in Section 3.2.1. A node should be sorted as an ancestor of a node with higher level of specification if they match on the common values e.g. two node boolean model without regulations should become an ancestor of two node boolean model with regulations specified.

Note that to achieve such functionality, some testing is necessary, since components may be ordered differently etc. We propose to create an independent utility for process of sorting and not to include it in the interface itself, since capability for usage of the utility out of the scope of the interface may be useful.

We can also partially order files on the same level of specification. If models A and B share the specification level, A should be considered ancestor of B if and only if:

1. They are both on Level 1 and the transition system of B is a strict subset of the transition system of A.
2. They are both on Level 2, they are equal on Level 1 and the set of interactions of A is a strict subset of the set of interactions of B.
3. They are both on Level 3, they are equal on Level 2 and the set of local parametrizations of A is a strict subset of the set of local parametrizations of B.

3.3.2 Comparison system

One of the main purposes of the interface is capability for easy comparison of parametrizations and their respective properties. For this it should be possible to share classifiers between the models for which these classifications are common, e.g. LTL formula for all the models with the same structure.

Also it should be possible to merge tables. This merge should create an over-approximation (full outer join in the terms of SQL) with a new classification specifying the origin.

Design of such features should be chosen based on the technologies used for implementation of the interface.

4 Design proposals

In this section we suggest a design approach to some parts of the interface. Content of this section should be treated only as an advices, though.

4.1 File system

To keep a track of files of a single user, the file tree may be actually sorted in a tree of directories. Also, it is necessary to store somehow access rights to the files. This can be done using a single file present in each directory. This way it is also easy to check if the data storage quota has not been exceeded.

To keep track of public files, a single file with their locations may be used.

To avoid data loss, each model should be auto-saved as frequently as possible.

It is also important to note that some widgets use the same input file type e.g. model file and on the other hand some take more than one input file type. We actually propose for each widget to have a single associated file type which saves all the data given by the user.

4.2 Widgets access

This document does not give exact specification of how a user can start underlying tools and create new files from the interface as we expect that this should be chosen by the creator based on the technology used, we would however like to propose some options.

New files are usually created as an output of a tool. For this task, the user must certainly be able to start the tool, which is however somewhat based on the nature of the tool. It is up to discussion whether the widgets should be accessible without a file to work with, which behaviour we actually suggest to omit – even the Model widget should be started, probably from the file tree, with an empty file unsorted in the file tree at the moment. Widgets that process files, like TomClass might be accessible from the file icon in the file tree. Other possibility which is to start a widget from a widget sharing the same sort of file e.g. to start Behaviour Map widget from Parameter View widget.

4.3 Comparison system

Ability to compare results from analyses is one of the main features the tool should provide. To

do such a task efficiently, the user should know which files contain comparable results and should be able to pick them in a simple manner. Also the result of their merge/comparison should be readable to say at least.

File tree with its files might be sufficient for the first two tasks, as well as the currently proposed structure of widgets might be just enough for the third one, however it is important to keep this properties in mind when designing widget access and files storage.

4.4 General proposals

- There should be easy enough interface for files manipulation. User should be able to create new files, rename files, duplicate files and save current files easily
- Also some sort of marking of files (a star for important ones etc.) may be designed.
- Even though components in models are expected to have labelling, i.e. name of the gene, the interface should see them as uniform graphs to allow for simple inclusion, comparison etc. Also we propose to define ordering on components – this may ease sorting as well.
- Some of the widgets actually change the size of the parametrization space instead of just marking – to pass positions of those that remained we propose to use an uniform ordering on parametrizations and pass their numbers in form of bitmasks.