BBF RFC 55: Standard Biological Part Automatic Modeling Database Language (MoDeL)

Zhen Wang, Chen Liao, Hao Jiang, Xiaomo Yao, Kun Jiang

27 October 2010

1. Purpose

This BioBricks Foundation Request for Comments (BBF RFC) describes the Standard Biological Part Automatic Modeling Database Language (MoDeL). MoDeL provides a language and syntax standard for automatic modeling databases used by synthetic biology software. Meanwhile, MoDeL allows detailed description of biological complex, and presents the concept of Chain-Node Model.

2. Relation to other BBF RFCs

BBF RFC 55 does not update or replace any earlier BBF RFC.

3. Copyright Notice

4. Table of Contents

5. Introduction

MoDeL (Standard Biological Part Automatic **Mo**deling **D**atabas**e** **L**anguage) is a database representation format for synthetic biology automatic modeling. This project of MoDeL aims at providing a language and syntax standard for automatic modeling databases used in synthetic biology models on a number of topics, including cell signaling pathways, metabolic pathways, biochemical reactions, gene regulations, and so on.

MoDeL is developed as a machine-readable format in XML fashion, programming languages with XML library API (such as libxml2 for C++) are strongly recommended for reading MoDeL database. Examples of MoDeL databases can be downloaded from http://2010.igem.org/Team:USTC_Software

MoDeL is an attempt to define a universal database markup language for synthetic biology automatic modeling. It supports descriptions of a wide variety of different parts, including biobrick, compound, compartment, substituent, and so on. It can also describe the species constructed by these parts, and reactions happen with the species, such as binding of species, dilution of species, degradation of proteins, amplification of cells, multi-compartment reactions, and so on.

Many traditional methods fail to consider structure of species when trying to markup reactions with only reactants, modifiers, products and kinetic laws. The best part of MoDeL is that it can preserve detailed structure in species and reactions using **Chain-Node Model**. Based on the model, MoDeL is able to markup almost all kinds of reactions in a detailed and clear way.

5.1. iGEM2010

MoDeL is also the core of the iGaME project of USTC_Software in iGEM 2010. To learn more about our team and project, please visit: http://2010.igem.org/Team:USTC_Software

6. Features

6.1. Chain-Node Model

In most manual modeling, complex is treated as a whole entity in reactions. Plenty of synthetic biology automatic modeling software treats complex the same way. However, there is a big difference between manual and automatic modeling, that is: Model constructors always know which detail of complex needs to be ignored or preserved, but software does not. So in most software, detailed structure has to be specified by modelers, otherwise will be ignored. In most cases, this requires plenty of manual operations and thus making the software not so automatic. Knowing this, MoDeL chooses to preserve detailed structure in complex and present **Chain-Node Model**, which view complex as a construction of its parts throughout the modeling process. With

chain and nodes, describing large biological complex with a great number of basic units and binding sites becomes possible. It also offers great flexibility in modeling. Detailed concepts are explained below.

6.1.1. The Abstract Concept of Chain

When it comes to biobrick assembly, DNA of plasmid is often considered as an abstract straight line where biobricks can be placed on and connected together. It actually acts as the basic frame of biobrick assembly. Inspired by this, we present the abstract concept of chain, which also acts as the basic frame of part assembly. **Chain is an abstract straight line where different parts can be placed on and connected together. Part** is the basic unit of chain, and **chain** can have as many parts as possible.

To go beyond DNA, and to unify the data structure of Species, we extend the concept of chain in MoDeL to include all the parts we can describe, including RNA, protein, and even compartment, compound, substituent, etc. Parts of DNA, RNA and protein, are all originated from DNA, and they are considered as **"Sequence"**. Sequence can be connected to other parts of sequence on a chain. On the contrary, parts of compartment and compound are considered as **"Non-sequence"**. Parts of non-sequence cannot be connected to any other part on a chain, which means, they can only exist on a One-Part-Chain. Substituent is used as a substitution of any other part, so it can be either sequence or non-sequence.

It is important here to mention that the Sequence/Non-sequence division is only a suggestion of modeling. It is not a constraint to Chain-Node model. Actually, we choose to maximally preserve the flexibility and leave all the freedom of constructing a species to users. Users are allowed to construct **ANY** species they can using Chain-Node Model. For example, one can create a strange species with compartment, compound, and DNA parts on the same chain, which makes little sense in biology. So, it is really up to the users to decide how to use Chain-Node Model in modeling.

6.1.2. The Abstract Concept of Node

Node is used to characterize binding structure in biological complex. **Node is an abstract concept of binding site or site that can be bound.** Links always exist between a binding site and the sites it binds together, indicating the relationship between them. We use trees to represent the binding structure in complex. The parent node represents the binding site, and the child node represents the bound sites. For example the binding structure of TetR dimer is considered as a TetR2 node with two TetR nodes as its children. There is one exception though, the **ROOT** node. ROOT node is used as the top node in a tree. It can only have one child and no parent. The function of ROOT is to indicate that its child is the last binding site in this tree, and it binds to nothing any more.

In the reactions of promoter repression and activation, the promoter is repressed or activated after binding with repressor or inducer. With its properties changed, it is not the original promoter any more. This is an example where binding changes the properties of node. Since we do not store all the possible property alterations within one part, the solution is to replace the old part with a new

part that stores the correct properties.
Available nodes can be both a part on a chain or a node of binding site, which leaves a lot of possibilities in constructing complicated binding structure. Again, there are no restrictions, but parts of compartments are not suggested to appear in a binding structure.

6.1.3. The Basic Assumption

The basic assumption of Chain-Node Model is that a part **ALWAYS** carries its properties wherever it is placed in an instance of Chain-Node Model. This means no matter how complicated the complex is, you can always keep track of the location and properties of every single part. By letting complex inherits its parts' properties, the assumption saves modelers from rewriting the structure and function of each new complex. However, we **should** address that the assumption is not suitable for all circumstances. For example, under this assumption, a fusion protein will carry the properties of the fused proteins, which is not true in many cases. A fusion protein may not remain its old properties and even have new properties that none of the original parts have. In this case, our suggestion is to define the fusion protein as a new part. So it is suggested that users choose different strategies of describing the same thing in Chain-Node Model base on different situations, instead of modeling everything in the same way.

6.1.4. Examples

Here are a few examples of species described in Chain-Node Model:

Example of a species: E.coli cell

In MoDeL, the cell of E.coli is described as a species with only one part of E.coli (see Figure). Since the concept of part has been extended greatly, living cells can be described in the same way other species. Compartment belongs to "Non-sequence", so it is suggested that E.coli only exists as a one-part-species.

Figure 6.1. The Chain-Node Model of an E.coli cell

Example of a species: pLacI*:LacI4 and its tree structure of binding.

Here is another example of template species with more complicated structure (Figure 6.2). In biology, LacI tetramer (or LacI4) can bind with promoter pLacI as a repressor. When bound with LacI4, pLacI is repressed and its transcription rate will decrease significantly. In order to express this alteration, as mentioned above, we use another part of pLacI* with decreased transcription t

instead of pLacI in the species.

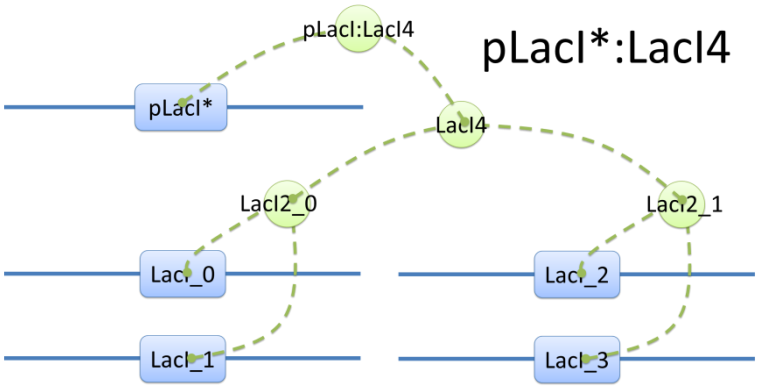


Figure 6.2 Species of pLacI*:LacI4

The structure of binding sites is shown in Figure below. The graph indicates that LacI monomers first bind together to form LacI dimer, then LacI dimers forms LacI tetramer, and finally LacI4 binds with pLacI* to form pLacI*:LacI4.

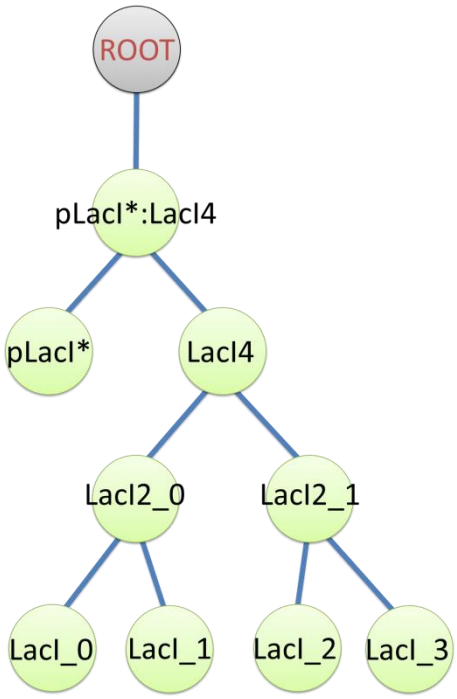


Figure 6.3 Tree structure of binding in pLacI*:LacI4

6.2. Modeling with Templates

Manual modeling of biological system is widely used in synthetic biology. However, it requires an overall understanding of the biological network and large amount of data, making it difficult even for professionals. Actually, the data provided manually are redundant because different reactions may occur with a same mechanism. Based on this idea, we introduce the concept of **Template**, which is the minimal data set for describing biological reaction networks, as a feasible way to implement automatic modeling. The term **"Automatic"** does not mean modeling without providing any information. Instead, we aim at providing minimal amount of data. Similar to C++ programming

language, templates allow generic description of species of a certain structural pattern or reactions of a certain reaction mechanism.

6.2.1. Substituent and Template

There are two kinds of templates: **Template Species** and **Template Reactions**. They are designed to represent a family of species with similar structure and a family of reactions with similar mechanism respectively. In order to realize this idea, we add the container of **Substituent** into MoDeL. There are currently six objects in this container, ANY, ANYUB, ONE, ONEUB, NZ and NZUB. Though defined differently, they actually serve the same purpose: representing other parts. Let's take ANY as an example (see Figure 6.4). ANY, represents parts on ONE chain and the number of represented parts can range from zero to infinity. For example, a DNA species with structure ANY - pTetR - ANY represents any DNA with part pTetR.

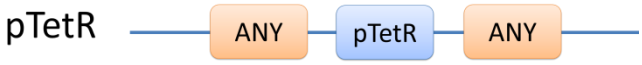


Figure 6.4

A reaction with template species as its reactants, modifiers or products is a **Template Reaction.** A template reaction describes the core structure of reactant that causes the reaction to happen and provides a specification for generating reactions with the same mechanism. This could be understood more clearly by dividing parts in a species into functional groups -- a reaction template describes the interaction mechanism of these functional groups. For example, pTetR promoter is deactivated in presence of TetR dimer. The template species are pTetR DNA template and TetR dimer protein template (see Figure 6.5) and the functional groups are pTetR promoter and TetR dimer. All the parts of ANY **should** also be counted as a functional group, though they don't have functions in this reaction. Any pair of species which partially contain pTetR DNA and TetR protein dimer respectively would bind according to the description of this reaction template. Modeling with templates allows users to define species and reactions only once for one certain family without rewriting them again in database.

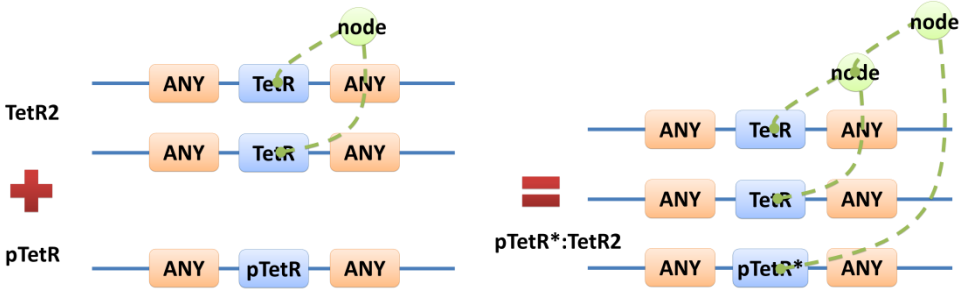


Figure 6.5

We are still looking for more substituents. We are planning to have more detailed substituents that can replace certain number of parts on certain number of chains. Even the arrangement of parts and chains can be specified. We believe it will make our template modeling more powerful.

7. Standard Biological Part Automatic Modeling Database Language

MoDeL aims at providing support language support for automatic modeling. We will discuss a few relevant topics in the following sections.

7.1. Overview of MoDeL

The following is an example of a simple network of biochemical reactions that can be automatically constructed using MoDeL:

pTetR + TetR2 = pTetR*:TetR2

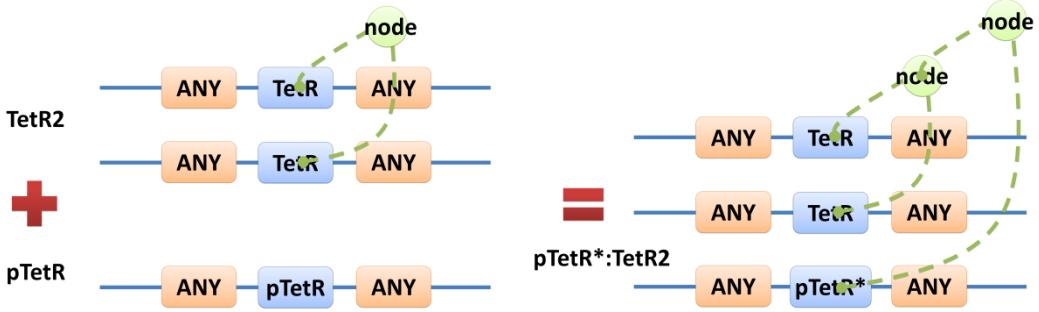


Figure 7.1

MoDeL supports automatic construction of such network with only the input of its initial conditions and environmental parameters. In order to do this, obviously the data of parts, species, reactions, compartment, units and so on is needed. Besides, relationships between these components are also needed, so we can link to a reaction when we find a species. Data of relationship is stored as references and exists in most MoDeL components.

To define all the needed components in a systematic way, MoDeL organizes them in five component containers at the top level:

- **System**: This container contains definitions of units, functions, rules and global parameters. These are all basic elements for other components of MoDeL, and many are mathematical concepts without direct synthetic biology meanings. The name of the container, System, separates this container from others which contains components of biological significance, and indicates this container includes the basic parameters of the whole system.
- **Compartment**: This container contains definitions of compartments. Compartment here is considered as a holder of substance with certain volume, so it only stores related data such as units and volume. It also stores a list of compartments that are permitted to be placed inside. For compartments like E.coli that can multiply, a corresponding compartment with the same id will be created as a component of compartment in Part.

- **Part**: This container contains definitions of parts. Part includes the basic units that can be inserted in an abstract chain. The container of Part has five components which are: compartment, biobrick, compound, plasmidBackbone, and substituent. Biobrick contains biobricks from PartsRegistry together with other similar components created by users. Compartment contains cells and other microscopic organisms that can multiply. compound contains small molecules like IPTG and other compounds. PlasmidBackbone contains plasmid backbones come from PartsRegistry or defined by users. Substituent contains substitutes for template modeling.
- **Species**: This container contains definitions of various species defined by Chain-Node Model. Each component stores the cnModel structure of chains and trees. Related reactions are also stored. It is suggested that the constructor of a database stores template species instead of real species in this container. Only in this way can he make use of the power feature of template modeling of MoDeL.
- **Reactions**: This container contains definitions of various reactions. Each component has its own compartments, reactants, modifiers, products, substituent transfers, and kinetic law. This forms a detailed description of reactions. Again, constructor of a database is suggested to store template reactions instead of real reactions .

The organizational structure of components in MoDeL is shown in the following figure.

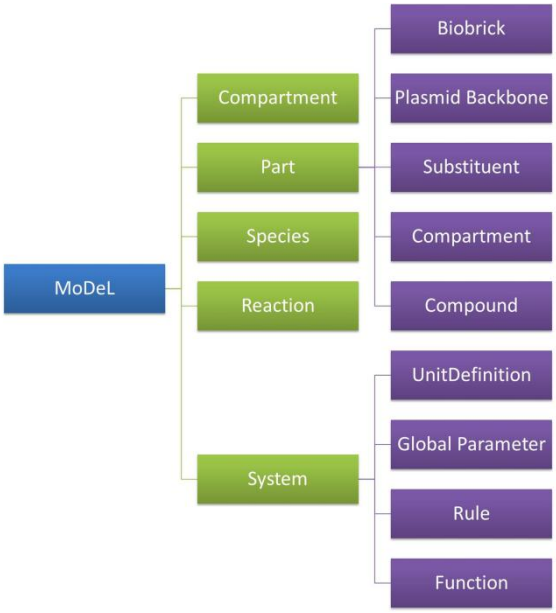


Figure 7.2

As mentioned above, reference relationships are stored in most components. The following figure shows the relationship between different components, the pointer pointing to a component indicates referencing to that component.

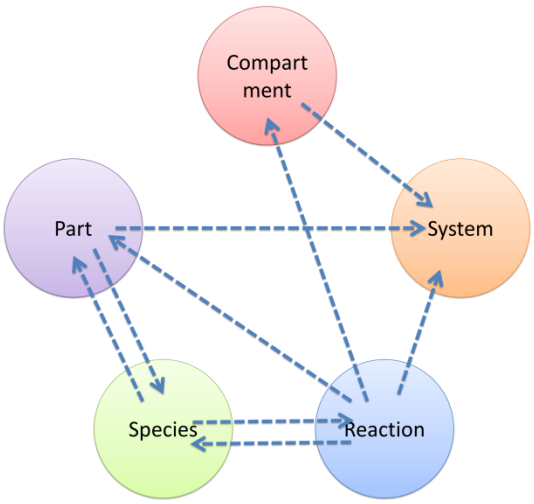


Figure 7.3

It is worth noting that MoDeL does not attempt to include all information about each component. In PartsRegistry, a long list of data in XML format is provided including part author, status, sequence, and even group access information. Such data are redundant for our database since they are not necessary elements needed to construct reaction network. Only the useful data is retained.

7.2. Suggestions for Programmers

The project of MoDeL aims at providing a language and syntax standard for automatic modeling databases. Usually, we assume such databases are used by automatic modeling software. In this way, hopefully, MoDeL can set biologists free from inputting mass of data and constructing biological reaction networks manually.

There is actually no right or wrong way for software to implement automatic modeling. However, since MoDeL is new, providing the workflow of iGaME may be helpful. Steps taken in iGAME to complete the automatic modeling process are listed in the following:

1. Users construct biological systems using standard parts, and setting up environmental parameters.
2. iGaME kernel program starts to construct reaction network automatically and output a model of reaction network encoded in SBML[1].
3. Numerical analysis will be performed after reading the SBML file as input. Dynamic curves of species (relations of concentration depending on time) in this network are given as output.

The **algorithm flow chart** of iGaME within step 2 is like following:

1. Setup initial conditions and environmental parameters read from the input file.
2. Species produced are inserted into a list in the order of time. For each species, we search the database to find template species which could be matched to this species in structural pattern. If not found, kernel will go for the next species.
3. For each template species we found, we continue to search template reactions containing this template species as a reactant or modifier or both. Only forward or reverse reaction is handled,

never for both.

4. For each template reaction found, we search in the species list for possible species which could be instances of templates of other reactants and modifiers. If not found, kernel goes for the next reaction.
5. For each possible combination of reactants and modifiers, if the reaction parameter conditions are satisfied, we generate the structure of products based on the transfer table and structural templates of products. And then add this new reaction to the reaction list.
6. Go to step 2 until the end of the species list.

We believe this is not the only way of realizing automatic modeling. Programmers are encouraged to use any method they can think of to do so within the frame of MoDeL.

7.3. Relationship with SBML

SBML[1](**The Systems Biology Markup Language**) is a machine-readable language in XML fashion. It's oriented towards describing systems where biological entities are involved in, and modified by, processes that occur over time. As a widely used model exchange standard, SBML is chosen to be the output of our auto-modeling module.

Generally speaking, MoDeL needs not to be conformed to any constraints set by SBML. However, since SBML is a widely used standard, it is better for MoDeL to share common concepts and components with SBML as many as possible. For example, MoDeL still uses rules, function definitions and unit definitions in SBML. By doing so, it will be easier for software tools to change formats between MoDeL and SBML.

8. Preliminary definitions and principles

This section covers certain concepts and constructs that are used repeatedly in the rest of MoDeL.

8.1. Parameter Management

There are three types of components related to parameter definition. The objects of **GlobalParameter** are defined in the container of **System**. The components of GlobalParameter represent global parameters that are allowed to be changed by user for modeling. The objects of **ConditionalParameter** are defined in the components in Part container. They represent properties of the components whose values depend on conditions, such as compartments in which the part locates. All conditional parameters are required to be constant and could not be changed during simulation process of software. The **LocalParameter** element is designed for parameters defined in kinetic laws of reactions, which are required to be constant, too.
The syntaxes for referencing parameters defined elsewhere are different for different parameter

types. Global parameters could be used by directly referencing its id. Local parameters are not allowed to be referenced externally. For referencing conditional parameters, users **must** use an external reference, which is defined in the element of **ReferencedParameter** in the component of Reaction. We will give details in the section of Reaction.

8.2. The id and name attributes on MoDeL components

As will become apparent below, most objects in MoDeL include two common attributes: **id** and **name**. These attributes are not defined on some objects, but where they do appear, the common rules of usage described below apply.

8.2.1. The id attribute

The **id** attribute is mandatory on most objects in MoDeL. It is used to identify a component within the MoDeL. Other MoDeL objects can refer to the component by using this identifier. Identifier conforms to no syntax, yet the data type **must** be text string.

A database can contain a large number of objects. This leads to a problem in deciding the scope of an identifier: in what contexts does a given identifier X represent the same thing? The solution we provide is to divide the collection of components into three categories: global, the case of compartments, and local. The identifier of global components **must** be unique across the set of all such identifiers in the MoDeL, so that a global identifier X represents the same thing wherever it appears in a given context.
The identifiers of compartment belongs to the set of global identifiers, however, it is not unique in the global set since there are actually two compartment containers in MoDeL, one is located at the top level, the other is inside the container of Part. Two objects with one from each of the two containers may have a same identifier, for example E_coli. When handling compartment in iGaME, we combine data from the two objects and treat them as one whole concept of compartment in biology. When one object is missing, we consider the whole compartment do not have that part of data. So in the sense of biology concept, compartment still have a unique id. We admit this is an inconsistency of MoDeL, and may cause confusion in software developing. The detailed description of this problem is in the section of compartment. In summary, the main reason is that we have to break the properties of compartment into two parts, so we need two types of object. Luckily, this problem does not cause any trouble in iGaME, because both id and the path of container are required when doing cross-referencing.
The identifier of local components is placed in a separate identifier namespace, depending on the context. For example, individual rate expressions are separate namespaces, same local identifier X can represent different quantities in different rate expressions.

The detailed scoping rule in MoDeL is here:

- **Biobrick, Compound, PlasmidBackbone, Substituent, Species, Reaction, GlobalParameter, UnitDefinition, Rule, FunctionDefinition** are all global components, the identifier of these **must** be unique across the set of all such identifiers in the MoDeL. This means, for example, that a biobrick and a species cannot both have the same identifier.
- Each of the component from the Compartment container at the top level is allowed and recommended to have the same id as **ONLY ONE** component of compartment in Part, but **must not** be the same as any other components in iGaME. Having the same id means the two components are of the same concept of compartment in biology.
- Each instance of component in container of Part establishes a separate private local space for ConditionalParameter, within the definition of a Part of any type, the identifier of each ConditionalParameter **must** be unique. Each instance of Reaction establishes a separate private local space for LocalParameter, within the definition of a Reaction, the identifier of each LocalParameter **must** be unique.

Hopefully, the guidelines described here will provide a clean idea for users, and a clean transition path to future versions of MoDeL, which will hopefully resolve the inconsistency of current version.

8.2.2. The name attribute

Different from the id attribute, the **name** attribute is optional and is not intended to be used for cross-referencing purposes within a database. Its purpose is to provide a human-readable label for the component. The data type of name is string, and the length is recommended to be limited within a few words, longer description **should** be placed in **shortDesc** instead.
The recommended practice for handling name is as follows. If a software tool using the database has the capability for displaying the content of name attributes, it **should** display the content to the user as a component's label instead of id. If the content of name cannot be displayed or if the content of name is left empty, the attribute of id **should** be displayed.

8.3. The interface element

Most objects of MoDeL include the common element of **interface**. Just as the name implies, interface was designed for the User Interface part of iGaME to provide useful supplemental information and external links of part, species, reactions, etc. The modeling core of iGaME will always ignore this element when reading data from database.
The interface element contains two subelements: **url** and **shortDesc**. Both are optional. The content of url is the address of a webpage which contains detailed information about this object, usually a Wikipedia or PartsRegistry link. The content of shortDesc is a short human-readable sentence that describes the meaning and function of the object in brief.
The recommended practice for handling interface is similar as the attribute of name. If a software tool using the database has the capability for displaying information within interface element, it **should** display the information to the user together with name. If the information in interface

cannot be displayed or if the content of interface is left empty, the value of name attribute **should** be displayed.
The element of interface always has the same meaning and usage wherever it appears in a database, so we summary it in this paragraph and organize it in front of the detailed description of each MoDeL components. In the following document the details of interface will not be mentioned again.

8.4. The math element

MoDeL borrows the **math** element of SBML L2V4[1] to define functions and other mathematical expressions. The only difference is that the namespace of XML **must not** be written in a math element. In this version of MoDeL, only the components of **Reaction, FunctionDefinition and Rule** can have the element of math. Details will be discussed in relevant sections. The content of math element is represented using **MathML 2.0**[2](W3C, 2000b). For more details, please refer to SBML L2V4 (ref) Section 3.4.

9. MoDeL components

MoDeL organizes its components in a tree structure, and they primarily fall into five categories: System, Compartment, Part, Species, and Reaction, we call them containers. In this section, we define each of the containers of MoDeL, and the components defined in them. We also illustrate the use of SBML components by giving partial model definitions in XML.

9.1. System

There are five components defined in the container of System. They are **listOfFunctionDefinitions, listOfUnitDefinitions, listOfRules and listOfGlobalParameters,** and they define a list of **FunctionDefinitions, UnitDefinitions, Rules, GlobalParameters** respectively. Components in System container all have a global scope, and can be referenced by other components directly. For these five components, MoDeL partly borrows the syntax of corresponding components in SBML, since the concept of the components is similar in both languages.

9.1.1. FunctionDefinition

The definition of **FunctionDefinition** in MoDeL is similar to that in SBML. The **FunctionDefinition** object associates an identifier with a function definition. This identifier can then be used as the function called in other MoDeL components.

The id and name attributes

The **id and name** attributes both have the type of string, and operate in the manner described in Section 4.2. Since the scope of **FunctionDefinition** is global, other components can directly refer to the function definition using the value of its id attribute.

The math element

The **math** element of **FunctionDefinition** defines a function with arguments, and is the only place where a lambda element can be used. Syntax to define a math element is the similar to that used in SBML, the only difference is that there **must not**. Followings are some restrictions:

- The math element is a container for MathML that defined the function. The content of this element can only be a MathML lambda or a MathML semantics element containing a lambda element.
- Content of the math element cannot contain references to variables other than the variables declared to the lambda itself.

Example

Following gives an example of FunctionDefinition that defines a piecewise function with four arguments t, ts, te, s. The function returns zero when t is less than ts, or s when t is greater than te. When t is between ts and te, it returns a linear function. Notice that in the example, XML namespace is not written according to the restriction of math element described in Section 4.4.

```
<MoDeL>
    ...
    <functionDefinition id="square_wave">
        <math>
            <lambda>
                <bvar><ci> t </ci></bvar>
                <bvar><ci> ts </ci></bvar>
                <bvar><ci> te </ci></bvar>
                <bvar><ci> s </ci></bvar>
                <piecewise>
                    <piece>
                        <cn type="integer"> 0 </cn>
                        <apply>
                            <lt/>
                            <ci>t</ci>
                            <ci>ts</ci>
```

```
                    </apply>
                </piece>
                <piece>
                    <apply>
                        <times/>
                        <apply>
                            <minus/>
                            <ci>t</ci>
                            <ci>ts</ci>
                        </apply>
                        <apply>
                            <divide/>
                            <ci>s</ci>
                            <apply>
                                <minus/>
                                <ci>te</ci>
                                <ci>ts</ci>
                            </apply>
                        </apply>
                    </apply>
                    <apply>
                        <and/>
                        <apply>
                            <geq/>
                            <ci>t</ci>
                            <ci>ts</ci>
                        </apply>
                        <apply>
                            <leq/>
                            <ci>t</ci>
                            <ci>te</ci>
                        </apply>
                    </apply>
                </piece>
                <piece>
                    <ci>s</ci>
                    <apply>
                        <gt/>
                        <ci>t</ci>
                        <ci>te</ci>
                    </apply>
                </piece>
```

```
                </piecewise>
            </lambda>
        </math>
    </functionDefinition>
    ...
</MoDeL>
```

9.1.2. UnitDefinition

MoDeL borrows the unit system from SBML L2V4. The component of **UnitDefinition** of MoDeL is the same as that of SBML. We summarize the important information and list it here.

The id and name attribute

The required elements **id and name** are both have the type string. The id element is used to give the defined unit a unique identifier by which other components of MoDeL can refer to it. The name element is intended to be used for giving the **UnitDefinition** a human-readable name.
UnitDefinition follows the listed restrictions, same as SBML:

- The id of a UnitDefinition **must not** contain a value listed in Table 1 on Page 37 in SBML L2V4 Document[1] since they are reserved base unit names that are prevented from redefinition.
- There is a set of reserved identifiers for the predefined units in SBML; these identifiers are "substance", "volume", "area", "length", and "time". Redefinition of these units will have effect on database-wide default units for the corresponding quantities (Refer to Section 4.4.3 in SBML L2V4 Document[1] for more details).

The ListOfUnits element

The **listOfUnits** element defines a composite unit by separately defining each of its subunit using unit element. The composite unit is the product of all its subunits.
Each unit element has four children elements to define it: kind, exponent, scale, and multiplier. Same as SBML unit system, a unit in MoDeL is defined by following rule:

$$\{u\} = (multiplier \cdot 10^{scale}\{u_b\})^{exponent}$$

where u_b is the reserved base unit in unit system. Please refer to section 4.4.2 of SBML L2V4 Document[1] for more details about SBML unit system.

Example

The following skeleton of a unit definition illustrates an example use of UnitDefinition:

```
<MoDeL>
    ...
    <unitDefinition id="litre_per_mole_per_second">
        <listOfUnits>
            <unit kind="litre" exponent="1"/>
            <unit kind="mole" exponent="-1"/>
            <unit kind="second" exponent="-1"/>
        </listOfUnits>
    </unitDefinition>
    ...
</MoDeL>
```

9.1.3. Rule

Same as SBML, **Rules** in MoDeL provide additional ways to define the values of variables, their relationships, and the dynamical behaviors of those variables. Rules are separated into three subclasses: AssignmentRule, RateRule, AlgebraicRule. The three subclasses are based on the following three different possible functional forms:

$Algebraic$	$left - hand\ side\ is\ zero$	$0 = f(W)$
$Assignment$	$left - hand\ side\ is\ a\ scalar$	$x = f(V)$
$Rate$	$left - hand\ side\ is\ a\ rate - of - change$	$dx/dt = f(W)$

To read the detailed descriptions and restrictions of the three subclasses, please refer to SBML L2V4 Document[1] Section 4.11.

The variable element

Different from SBML, variable of Rule is written as an element instead of attribute. The **variable element** can refer to the identifier of a species, compartment, or global parameter object (but not a reaction or a conditional parameter or a local parameter) in a database, indicating certain property of the object acts as variable x in the above functions (ref function). Only RateRule and AssignmentRule can have this element, AlgebraicRule does not need this element due to its functional form.

The math element

The component of **Rule** has a required element called **math**, containing a MathML expression defining the mathematical formula of the rule. Mathematical expressions defined in Rule follow the restrictions in Section 4.4. More details about math and units of the formula can be found in Section 4.11.2, 4.11.3 and 4.11.4 in SBML L2V4 Document[1]. Here we list one important principle on the units of rule:

- In AssignmentRule, the overall units of the formula in math element **should** be the same as the units of the related variable of this AssignmentRule, while in a RateRule, the overall units **should** be x/time, where x is the units of related variable of RateRule.

Example

The following example of AssignmentRule defines the GlobalParameter of t to be equal to the simulation time of the system (please refer to SBML L2V4 Document[1] Section 3.4.7). In this way the variable of t can be used explicitly representing time.

```
<MoDeL>
    ...
    <assignmentRule>
        <variable> t </variable>
        <math>
            <csymbol encoding="text" definitionURL="http://
www.sbml.org/sbml/symbols/time">
                time
            </csymbol>
        </math>
    </assignmentRule>
    ...
</MoDeL>
```

9.1.4. GlobalParameter

As is mentioned in Section 4.1, in the parameter family, only **GlobalParameter** have a global scope and can be directly referred to by its identifier. So **GlobalParameter** is used to define parameters that will be used in the whole database. Meanwhile, **GlobalParameter** is also the only one in the parameter family that can be changed by a **Rule**.

The id and name elements

For some historical reasons in the development process of MoDeL, the **id and name** of GlobalParameter is stored as elements instead of attributes. However, their meanings and usages are the same, and follow the principles in Section 4.2. The id element is required, and name is optional. The data type of both **must** be string.

The units, value and constant elements

All of the three elements are optional. The **units** element associates the value of the global parameter with specified units. The units **must** be chosen from pre-defined units or user-defined units in ListOfUnitDefinitions(Section 5.1.2). The **value** element determines the value of global parameter, and the default value is set to zero. When it is left empty, the value **should** be determined by a rule or some other external resources. The boolean element of **constant** indicates whether a global parameter can vary. The element's default value is "true". A value of "false" indicates the **GlobalParameter**'s value can be changed by rules and that the value is actually intended to be the initial value of the global parameter.

Example

The following is an example object of GlobalParameter. It defines the variable of t that can be used explicitly as the simulation time (please refer to SBML L2V4 Document[1] Section 3.4.7) in a database.

```
<MoDeL>
    <system>
        <listOfGlobalParameters>
            <globalParameter>
                <id> t </id>
                <units> second </units>
                <constant> false </constant>
            </globalParameter>
        </listOfGlobalParameters>
    </system>
</MoDeL>
```

9.2. Compartment

In biology, the description of a living cell includes its growing, death, size, inner compartments, naturally existed species associated with related reactions, and even more. Modeling of every detail

is next to impossible. MoDeL chooses to describe only the growing, death, size and permitted inner compartments of a compartment. MoDeL places the properties of size and permitted inner compartment in the compartment component in Compartment container, and the properties of growing and death in the compartment component in Part container.
We have to separate the properties because we choose to support synthetic biology model encoded with SBML. In SBML, a **compartment** only represents a bounded space in which species are located. In order to describe growing and death, we have to add a **species** of that compartment. Of course, the corresponding data in a database can still be stored in one place. However, we tried and found it can cause big problem in the consistency of MoDeL. Finally, we choose to separate the concept of compartment into two components with a same identifier, indicating they describe the same thing. A compartment is not always required to have both of the two components. An artificial one may not have its own correspondence in Part if it does not reproduce. However, if it does, users **must** create a species in correspondence to the compartment in Part and link it to a reproduction reaction.
The component in Compartment container still represents a bounded space in which species are located, as SBML does. So in this container, the compartments of living cells, for example E.coli, have no difference with artificial compartments such as flask. In summary, the compartments in the container of Compartment do not necessarily have to correspond to actual structures inside or outside of a biological cell. The compartments in the Part container act just as other parts do, we will discuss it later in the section of Part.
The worry about distinguishing the two types of compartment components is not necessary, for we will separate them by container when referencing.
From here on in this section, when we mention "**compartment**" we mean the component of compartment in Compartment container.

9.2.1. The id and name attributes

As most components of MoDeL, the compartment have a mandatory attribute of **id**. It also has an optional attribute of **name**. The data types of both attributes are string. The id of compartment, though not unique across the set of global identifier, has its own constraints. Details are described in Section 8.2.1.
Since MoDeL is aimed to support synthetic biology model encoded with SBML, we inherit the semantics that id of a compartment indicates its size.

9.2.2. The spatialDimensions attribute

Same as SBML, a compartment has an optional attribute of **spatialDimensions**, whose value **must** be a positive integer indicating the number of spatial dimensions possessed by the compartment. The possible values in SBML are "3" (a three-dimension volume), "2" (a two-dimension area), "1" (a one-dimension curve), and "0" (for a point). To simplify MoDeL, we choose to only support the dimension of "3", and the default value is set to "3".

9.2.3. The size attribute

Each compartment has a required attribute, **size**, representing the initial total size of the compartment. This element could not be allowed to leave undefined since it will make other quantities whose definitions are based on that of compartment size ambiguous. The value of size may be changed by a Rule. In which situation, the value of size is actually the initial value of size.

9.2.4. The units attribute

The units associated with the compartment's size value may be set using the optional attribute units. The value of the units attribute **must** be one of the base units defined in Table 1 on Page 37 in SBML L2V4 Document[1], or the predefined unit identifiers "volume", "area", "length", or "dimensionless", or a new unit defined by a unit definition.

9.2.5. The constant attribute

A **Compartment** also has an optional boolean attribute called **constant** that indicates whether the compartment's size stays constant or can be varied. The default value is false when the attribute is left empty, and if it is written explicitly, the value **must** be false too. This is required by the automatic modeling algorithm of iGaME. Currently when dealing with density in a multi-compartment system, iGaME will sum up the volumes of all compartments of the same type in the same area. iGaME do this by adding a rule to the compartment automatically. So in most situations the size **must not** be constant. It is recommended that users just don't write this attribute so it will use its default.

9.2.6. The interface element

The required element of **interface** contains optional URL and short description that provide additional information to the compartment. The usage of interface follows the principle in Section 8.3.

9.2.7. The listOfInnerCompartmentPermitted element

This required element defines a list compartments that are allowed to be placed inside current compartment. Though compartment here represents only a bounded space, it is recommended that users write this element according to biological facts. For example, the compartment of E.coli **should** not have the compartment of mitochondrion listed as permitted inner compartment. If the element is left empty, it means no compartment is allowed to be placed inside.

9.2.8. Example

The following skeleton of a compartment definition illustrates an example use of compartment component:

```
<MoDeL>
    <compartment id="E_coli" name="Escherichia coli" spatialDimensions="3" size="1E-14" units="litre">
        <interface>
            <url>http://en.wikipedia.org/wiki/Escherichia_coli</url>
            <shortDesc>Escherichia coli is a Gram negative rod-shaped bacterium that is commonly found in the lower intestine of warm-blooded organisms.</shortDesc>
        </interface>
        <listOfInnerCompartmentPermitted>
            <compartmentReference>ribosome</compartmentReference>
        </listOfInnerCompartmentPermitted>
    </compartment>
</MoDeL>
```

9.3. Part

BioBrick standard biological parts are DNA sequences of defined structure and function. They share a common interface and are designed to be composed and incorporated into living cells such as E. coli to construct new biological systems. The desired functions could be cell death, biosynthesis, cell-cell signaling, quorum sensing, and so on. With unified BioBrick prefix and suffix and standard assembly, BioBrick parts becomes standard and interchangeable. We try to have standard abstract biological parts in MoDeL, and we also try to go beyond DNA and describe all of the parts we can describe in a unified way. To realize this, we construct the container of **Part**, standardize the properties of all the parts we have, and categorize them into five types of components:

- **Biobrick**: Standard biological parts of PartsRegistry or other resources.
- **PlasmidBackbone**: Vector that propagates the Biobrick part.
- **Compound**: Compounds in chemistry like IPTG and ATC.
- **Compartment**: Living cells and organelles.
- **Substituent**: Species with unknown but constrained structure.

All of the above categories share the same elements and attributes in MoDeL, so we can discuss them together here.

9.3.1. The id and name attributes

All components in Part container have the common mandatory attribute of **id** and the optional

attribute of **name**. Id, as is described in Section 8.2.1, is used as the identifier of component in cross-referencing, and name is provided as a human-readable text. Both of the attributes are of the type string.

9.3.2. The registryName attribute

The optional attribute of **registryName** is designed to store the name of BBa_... according to PartsRegistry. With this information, users are able to view the registryName or choose the part they want by searching its name in PartsRegistry. Since only the components of biobrick and plasmidBackbone have a registry name, registryName of other components in Part **should** be left empty.

9.3.3. The originalConformation attribute

The optional attribute of **originalConformation** is designed to indicate the relationship between the original component of a biobrick and the component of the same biobrick in binding state. The component in binding state **should** have this attribute with the value of the identifier of the original component. In this way they are linked together. Components that are not in a binding state **should** not have this attribute.
This will be better explained by giving an example. When protein of TetR2 binds with promoter of pTet, pTet is repressed and the transcription rate of which will decrease significantly. Since MoDeL does not store all the possible attributes alterations within one component of part, the solution is to replace the old component with a new component that stores the correct properties. In this case, we create a new component with the id R0040_ptetR_bs1 to represent the bound component, and we have the original component with the id of R0040_ptetR. The component of R0040_ptetR_bs1 **should** have an originalConformation attribute, whose content is R0040_ptetR. In this way, the two components are connected. This is required by certain algorithm in iGaME.

9.3.4. The interface element

Like most components of MoDeL, components in **Part** have a required element of interface, which contains two optional elements of **url** and **shortDesc**. The usage and restrictions of this element follows the descriptions in Section 8.3

9.3.5. The listOfConditionalParameters element

The **listOfConditionalParameters** contains a list of conditional parameters that locally define the properties of component under different circumstances. In this version of MoDeL, only the component of biobrick has conditional parameters, and we consider only compartments can have effect on the value of parameter. The effect is defined by adding a parameterValue element

associated with each compartment. When parameterValue is left empty, the default value is got from the optional commonValue element. Besides compartment conditions, more conditions may be added to determinate values of parameters in future version.
Different with global parameters, parameters defined here are local and not allowed to be changed during simulation process. Based on this, we eliminate element of constant since it is always "true".

The id, name and units attributes

Being a member of the parameter family, the **id, name and units** attributes of conditional parameter is similar to those of global parameters described in Section 9.1.4. The only difference is the scope of their identifiers. The scope of conditional parameter is local so we can place them in different components with a same identifier. Optional name attribute is still used to provide human readable text describing the parameter, and units can only be chosen from pre-defined units or the set of units in listOfUnitDefinitions.

The commonValue attribute

When parameterValue is left empty, the default value could be obtained from the optional **commonValue** element, which is used to indicate the common value under any circumstances. Though this attribute can be used any way with the definition, we recommend users to use it according to biological facts.
For example, the conditional parameter of forwardStartCodonEfficiency in biobrick can have the integer value of 0 or 1, knowing that the existence of start codon on a DNA won't shift due to different compartments, we save the value of current biobrick as a commonValue. When database constructor is not sure about the common value or only have the knowledge of values under a few circumstances, it is not recommended to write a commonValue. Instead, store the value as parameterValue and left the commonValue undefined is recommended.
In the situation where the parameterValue and commonValue are both left undefined, the value of a conditionalParameter will be set as zero by default.

The parameterValue element

The element of **parameterValue** is used to relate a value with a certain circumstance. There **may** be one or more parameterValues in a conditionalValue element. This element has a required attribute of compartment, which stores the identifier of related compartment. The data type of parameterValue is **must** be the same as the parent element of conditionalParameter.

Example

The following skeleton of a conditionalParameter definition illustrates an example use of conditionalParameter component:

```
<MoDeL>
    <part>
        <biobrick id="pcIlam189_bs1" originalConformation="pcIlam189">
            ...
            <listOfConditionalParameters>
                <conditionalParameter           id="forwardPromoterEfficiency"
units="per_second">
                    <parameterValue
compartment="E_coli">0.000</parameterValue>
                </conditionalParameter>
            </listOfConditionalParameters>
            ...
        </biobrick>
    </part>
</MoDeL>
```

9.3.6. The listOfReferencedSpecies element

The **listOfReferencedSpecies** element is an auxiliary element to help construct reaction network more easily. Each species defined in listOfReferencedSpecies contains the part as the basic unit on some chain. We use an optional element referencedSpecies to indicate the species identifier and partType to denote the type of current part in the species. The value of partType **must** be from(ref) However, the element is not necessary and may be eliminated in the future.

When to use this attribute might be a little complicated. The situations can be divided into following categories:

- When the referencedSpecies is a template species with substituent parts (ST) together with other non-substituent parts (NST), the part of STs **must not** have the element of referencedSpecies, while the NSTs **must** have one.
- When the referencedSpecies is a template species with only ST parts (such species are used to handle universal reactions happen to all substance or substance of a certain kind), all possible related parts **should** have an element of referencedSpecies. By “possible related” we mean a part that can possibly be consisted in an actual species that can have the reaction of this template species. For example the biobrick of TetR will exist in both RNA and protein form, and will have the reaction of degradation of both forward or reverse RNA and forward or reverse protein, so the part of TetR will have four elements of referencedSpecies indicating the relationships with certain template species. The example of TetR will be shown below.

There is still one more point need to be mentioned. The modeling of DNA in MoDeL is different

from other sequence of RNA and protein, since DNA is considered as a double-strand structure thus having both forward and reverse directions. Actually, it makes no difference reversing a DNA molecule as a whole, since there are no preferable directions. However, when it comes to modeling, we have to manually assign a preferable direction that determines which end of the DNA strand is called LEFT, and which RIGHT. In the current version of iGaME, the program will choose one by certain algorithm automatically. The consequence of this is the partType of a DNA part may be switching between ForwardDNA and ReverseDNA. Our solution is to create a duplicate species with DNA in the opposite direction, and have relevant parts store a referencedSpecies of this species.
Of course we have noticed that it is not an easy task to write this attribute correctly, we are looking forward for a better way of handling it. For more details about the algorithm, please refer to Algorithm on USTC_Software wiki (ref).

Example

The following is an example of referencedSpecies element used in biobrick of placI. The species referened are listed below:
fw_placI185_dna: The template DNA species with a forward placI.
rev_placI185_dna: The template DNA species with a reverse placI.
protein_sc: The template species of any protein. This is used to handle degradation of protein.
rna_sc: The template species of any RNA. This is used to handle degradation of RNA.
any_species: The template of all species. This is used to handle dilution.

```
<MoDeL>
    <part>
        <biobrick id="placI185" name="lacI-reg promoter (BBa_R0010)">
        ...
            <listOfReferencedSpecies>
                <referencedSpecies
partType="ForwardDNA">fw_placI185_dna</referencedSpecies>
                <referencedSpecies
partType="ReverseDNA">rev_placI185_dna</referencedSpecies>
                <referencedSpecies
partType="ForwardProtein">protein_sc</referencedSpecies>
                <referencedSpecies
partType="ReverseProtein">protein_sc</referencedSpecies>
                <referencedSpecies
partType="ForwardRNA">rna_sc</referencedSpecies>
                <referencedSpecies
partType="ReverseRNA">rna_sc</referencedSpecies>
                <referencedSpecies>any_species</referencedSpecies>
            </listOfReferencedSpecies>
        </biobrick>
    </part>
```

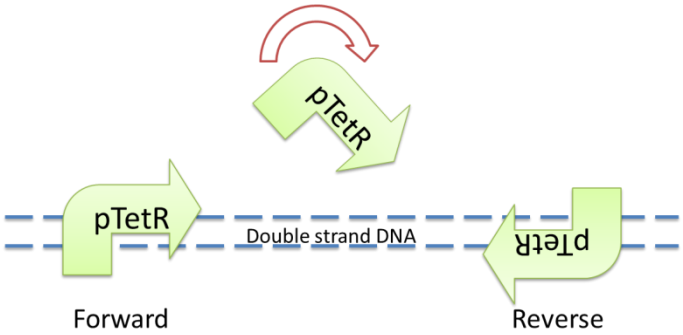
```
</MoDeL>
```

9.3.7. Biobrick

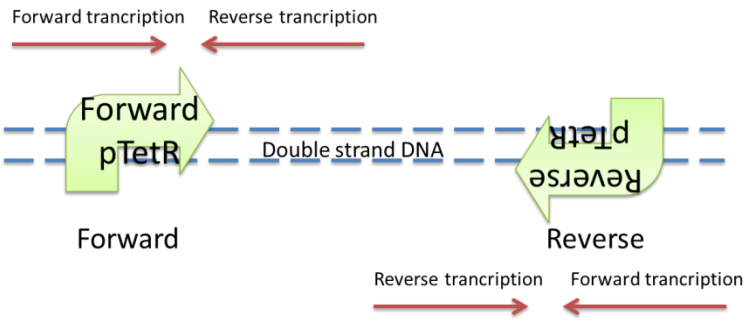
Following PartsRegistry, we borrow the term of **Biobrick** to represent standard parts of sequence. DNAs, RNAs, and proteins do not make a difference here and an identifier can actually represent its DNA part, RNA part after transcription, protein part after translation simultaneously. It is needed to be mentioned that the components of biobrick is not limited to the biobrick parts in PartsRegistry. Any standard part of sequence can be a biobrick.

The component of biobrick has ten conditionalParameters representing its properties. They are listed below. We **must** emphasize that the ten parameters are suitable for all biobricks, and there are not essential differences between different biobricks. Actually, by altering the conditionalParameters, you can turn a promoter into a terminator.

- **forwardPromoterEfficiency and reversePromoterEfficiency** represents the efficiency of starting a transcription reaction by a part when placed in forward or reverse direction, respectively. By "**reverse**" we mean rotating the part for 180 degrees on the same surface and then insert it in a chain, which is shown in the Figure below. The concept can also be understood comparing BBa_K093008 and BBa_R0011 and other similar pairs of parts in PartsRegistry.



- **forwardTerminatorEfficiency and reverseTerminatorEfficiency** represents the efficiency of terminating a transcription reaction by a part when placed in forward or reverse direction, respectively. The "reverse" here have the same meaning as above.
- **forwardRbsEfficiency and reverseRbsEfficiency** represents efficiency of starting a translation reaction by a part when the part is transcripted forwardly or reversely. The "reverse" here means the transcriptional direction according to the part's direction. Following Figure shows this:



The six parameters above can have any value of type double between 0.0 and 1.0.
The data types following four parameters are all integer type, and can have value of 0 or 1.

- **forwardStartCodonEfficiency and reverseStartCodonEfficiency** represents if there is a start codon on the part when transcripted forwardly and reversely, respectively. The value of 0 represents FALSE, and the 1 represents TRUE.
- **forwardStopCodonEfficiency and reverseStopCodonEfficiency** represents if there is a stop codon on the part when transcripted forwardly and reversely, respectively. The value of 0 represents FALSE, and the 1 represents TRUE.

Example

The following example shows biobrick definition within MoDeLs:

```
<MoDeL>
    <part>
        <biobrick id="ptetR187" name="tetR-reg promoter (BBa_R0040)">
            <interface>

    <url>http://partsregistry.org/cgi/xml/part.cgi?part=BBa_R0040</url>
                <shortDesc>promoter (tetR repressor regulated)</shortDesc>
            </interface>
            <listOfConditionalParameters>
                <conditionalParameter           id="forwardPromoterEfficiency"
units="per_second">
                    <parameterValue compartment="E_coli">0.5</parameterValue>
                </conditionalParameter>
            </listOfConditionalParameters>
            <listOfReferencedSpecies>
                <referencedSpecies
partType="ForwardDNA">fw_ptetR187_dna</referencedSpecies>
                <referencedSpecies
partType="ReverseDNA">rev_ptetR187_dna</referencedSpecies>
                <referencedSpecies
partType="ForwardProtein">protein_sc</referencedSpecies>
                <referencedSpecies
partType="ReverseProtein">protein_sc</referencedSpecies>
                <referencedSpecies
partType="ForwardRNA">rna_sc</referencedSpecies>
                <referencedSpecies
partType="ReverseRNA">rna_sc</referencedSpecies>
                <referencedSpecies>any_species</referencedSpecies>
            </listOfReferencedSpecies>
        </biobrick>
```

```
    </part>
</MoDeL>
```

9.3.8. Compartment

Since the identifier of **compartment** defined in Compartment container represents its size when it appears in a MathML expression, we need to define a correspond species to indicate its concentration when it appears in MathML formula. To indicate the relation with its compartment definition, both components' identifiers are required to be same. There are no specified conditional parameters for compartment. The behaviors of its growth and death are described in relevant reactions.

Example

The following example shows Compartment definition in Part container within MoDeLs:

```
<MoDeL>
    <part>
        <compartment id="E_coli" name="Escherichia coli">
            <interface>
                <url>http://en.wikipedia.org/wiki/Escherichia_coli</url>
                <shortDesc>a Gram negative rod-shaped bacterium that is commonly
found in the lower intestine of warm-blooded organisms (endotherms)</shortDesc>
            </interface>
            <listOfReferencedSpecies>
                <referencedSpecies
partType="Compartment">E_coli_cell</referencedSpecies>
                <referencedSpecies>any_species</referencedSpecies>
            </listOfReferencedSpecies>
        </compartment>
    </part>
</MoDeL>
```

9.3.9. Compound

Compound represents compounds like IPTG and ATC that appears frequently in biological system. Same as Compartment, there are no specified conditional parameters for Compound.

Example

The following example shows Compound definition within MoDeLs:

```
<MoDeL>
    <part>
        <compound id="IPTG" name="IPTG">
            <interface>

    <url>http://en.wikipedia.org/wiki/Isopropyl_%CE%B2-D-1-thiogalactopyranos
ide</url>
                <shortDesc>Isopropyl-beta-D-thiogalactopyranoside, abbreviated
IPTG, is a molecular biology reagent</shortDesc>
            </interface>
            <listOfReferencedSpecies>
                <referencedSpecies
partType="Compound">IPTG_compound</referencedSpecies>
                <referencedSpecies
partType="Compound">IPTGlacI153_4</referencedSpecies>
                <referencedSpecies>any_species</referencedSpecies>
            </listOfReferencedSpecies>
        </compound>
    </part>
</MoDeL>
```

9.3.10. PlasmidBackbone

The plasmid backbone propagates the assembled parts. It is defined as the sequence beginning with the biobrick suffix, including the replication origin and antibiotic resistance marker, and ending with the Biobrick prefix.

PlasmidBackbone has two more optional attributes. One is **origin**, indicating the genetic element responsible for the replication of plasmids during cell growth and division. Another is resistance, which allows the cell to grow even in the presence of a particular antibiotic, making it possible to select cells that contain desired plasmid.

Example

```
<MoDeL>
    <part>
        <biobrick id="pSB1A3" name="pSB1A3" origin="pMB1" resistance="A">
            <interface>
```

```
    <url>http://partsregistry.org/cgi/xml/part.cgi?part=pSB1A3</url>
                <shortDesc>High copy BioBrick assembly plasmid</shortDesc>
            </interface>
            <listOfReferencedSpecies>
                <referencedSpecies
partType="ForwardDNA">tm_pSB1A3_fw</referencedSpecies>
                <referencedSpecies
partType="ReverseDNA">tm_pSB1A3_rev</referencedSpecies>
                <referencedSpecies
partType="ForwardRNA">rna_sc</referencedSpecies>
                <referencedSpecies
partType="ReverseRNA">rna_sc</referencedSpecies>
                <referencedSpecies
partType="ForwardProtein">protein_sc</referencedSpecies>
                <referencedSpecies
partType="ReverseProtein">protein_sc</referencedSpecies>
                <referencedSpecies>any_species</referencedSpecies>
            </listOfReferencedSpecies>
        </biobrick>
    </part>
</MoDeL>
```

9.3.11. Substituent

Components of **substituent** are special ones that are designed to enable template modeling in MoDeL. For a better understanding of template modeling, please refer to Section 6.2.
Current version of MoDeL only contains six substituents, they are:

1. **ANY**: Represents unknown parts of any amount on one chain.
2. **ANYUB**: Represents unknown parts which are unbound of any amount on one chain.
3. **ONE**: Represents one unknown part on one chain.
4. **ONEUB**: Represents one unknown part that is unbound on one chain.
5. **NZ**: Represents **one or more** unknown parts on one chain.
6. **NZUB**: Represents **one or more** unbound unknown parts on one chain.

There are no conditional parameters specified for substituent.
The identifiers of substituents are all global. Moreover, they are reserved as keywords in this version of MoDeL. We admit that this requirement is due to the drawbacks of iGaME. Solutions to this problem are planned in the next version.

Example

The following example shows Substituent definition within MoDeLs:

```
<MoDeL>
    <part>
        <substituent id="ANY" name="substituent of ANY">
            <interface>
                <url>http://2010.igem.org/Team:USTC_Software</url>
                <shortDesc>monochain substituent of anything</shortDesc>
            </interface>
            <listOfConditionalParameters>
            </listOfConditionalParameters>
            <listOfReferencedSpecies>
                <referencedSpecies>anyspecies</referencedSpecies>
            </listOfReferencedSpecies>
        </substituent>
    </part>
</MoDeL>
```

9.4. Species

A **species** refers to a pool of reacting entities that take part in reactions and are located in a specified compartment. Since Chain-Node description of species has been introduced in Section 6.1, here we are going to specify its structure by defining its chains and trees in the element of **cnModel**.

9.4.1. The id and name attributes

As with other components in MoDeL, Species has a mandatory attribute, **id**, used to give the species an identifier, the identifier conforms to no syntax, yet **must** be a text string. The species also have an optional **name** attribute, of type string. The two attributes **must** be used as described in Section 8.2

9.4.2. The interface element

The required element of **interface** is used to provide additional information as described in Section 8.3. If the software supports the display of this element, it **should** present the information inside to users on its interface.

9.4.3. The cnModel element

A preview of the structure of cnModel is given below:

```
<cnModel>
    <listOfChains>
        <chain>
            <listOfParts>
                <part>
                ...
                </part>
            </listOfParts>
        </chain>
    </listOfChains>
    <listOfTrees>
        <tree>
            <listOfNodes>
                <node>
                ...
                </node>
            </listOfNodes>
        </tree>
    </listOfTrees>
</cnModel>
```

The listOfChains element

The **listOfChains** element is composed of one or more chain element that describes the concept of chain. There is no ordering of chains in a Chain-Node Model species, all chains are equivalent. So the order of chain elements in a listOfChain element is trivial, and switching them will not make any difference to the species.

Each chain element is constructed in a multilevel structure, each level is required. From top to bottom, the elements are chain, listOfParts and part, respectively. Just as chains, there **must** be one or more part elements in the listOfParts. Unlike chain, order of part elements does make sense. In this version of MoDeL, we consider the order of elements represents the actual order of parts placed on a chain, and we have made the restriction that the part of plasmidBackbone **must** be the first part on a chain if present. Within part lie four required elements of partReference, partLabel, partType and partCategory.

The partReference and partLabel elements

The element of **partReference** indicates the reference of a part. The content of partReference **must** be the identifier of existing part in database. The element of partLabel is used to assign a unique label to each part within the species, uniqueness across species is not required.

The partType element

The element of **partType** marks the form of each part on a chain. Currently in MoDeL, we have eleven partTypes. They are, **ForwardDNA, ReverseDNA, ForwardRNA, ReverseRNA, ForwardProtein, ReverseProtein, DNA, RNA, Protein, Compartment and Compound.**
For a DNA part, there are three available partType which are ForwardDNA, ReverseDNA and DNA, indicating a DNA part in forward direction, DNA in reverse direction and DNA without specified direction respectively. The meanings of forward and reverse are the same as those of forwardPromoterEfficiency and reversePromoterEfficiency described in Section 9.3.7 错误!未找到引用源。.
For a RNA part, there are also three available partType: ForwardRNA, ReverseRNA and RNA. The partType of RNA indicates the direction is not specified. Forward and reverse here means the direction a part is transcripted, same as the forwardRBSEfficiency and reverseRBSEfficiency in Section 9.3.7 错误!未找到引用源。.
For a protein part, it can have the partType of ForwardProtein, ReverseProtein and Protein. Meanings of forward and reverse are the same as RNA part. The partType of Protein indicates the direction is not specified.
For a compartment part, the partType is always Compartment.
For a compound part, the partType is always Compound.

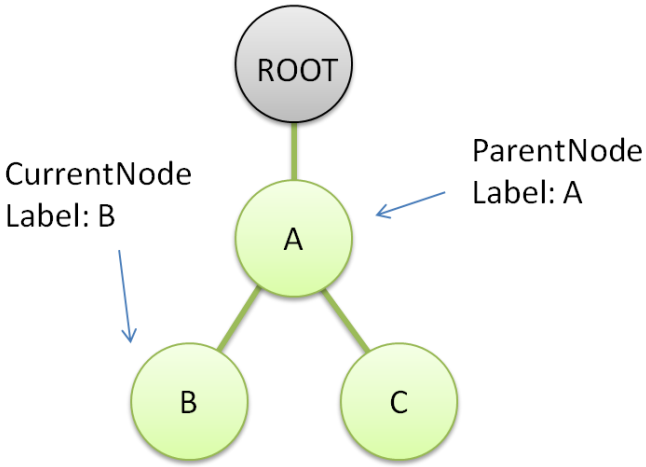
The partCategory element

The element of **partCategory** marks the category of a part. Knowing there are five categories in the container of Part, the available values are biobrick, compound, compartment, plasmidBackbone, substituent. The rules to apply them are obvious.
This element is required by the algorithm of iGaME. Having this element will make the search of parts more quickly and precisely, however, partCategory can be eliminated in the future.

The listOfTrees element

The **listOfTrees** element defines the binding structure in a complex using Chain-Node Model. There can be tree elements of any amount (including zero) in the listOfTrees. The structure of tree element is organized in three levels: tree, listOfNodes, node.

As described in Section 6.1.2, binding of nodes forms a tree. In order to describe the tree structure, there are two elements: currentNodeLabel and parentNodeLabel in an element of node. They store the label of a node itself and the label of parent node respectively. Users **should** pay attention to the uniqueness of labels within one species. The Figure shown below will gives a better understanding.



Each node except ROOT is required to have a node element to specify itself and its parent. And is shall be mentioned that the word "**ROOT**" is reserved as another keyword of MoDeL.

9.4.4. The listOfReferencedReactions element

Each species has a required element, **listOfReferencedReactions**, used to record the list of reactions which contain this species pattern as reactants or modifiers, or products if the reaction is reversiable. Each referecedReaction element stores the identifier of corresponding reaction, and has an attribute of speciesType indicating the role of current species in that reaction. Avaliable speciesTypes are product, modifier and product.

When does a reaction have to be referenced? The answer is as follows:

- When the reaction is reversible, all species in the reaction **must** have this reaction referenced.
- When the reaction is not reversible, only its reactants and modifiers are required to have this element.

Similar to listOfReferencedSpecies, listOfReferencedReactions is designed for the convenience of searching database to construct reaction network. However, it is possible to eliminate this element in the future version of MoDeL.

9.4.5. Example

The following example defines the species of pTet*:TetR2. The asterisk is added to indicate the difference between parts in bound and unbound states. This is a species with three chains: two protein chains and one DNA chain.

```
<MoDeL>
    <species id="fw_ptetR187_tetR154_2" name="forward ptetR187:tetR154_2">
```

```
        <interface>
            <url>http://2010.igem.org/Team:USTC_Software</url>
            <shortDesc>binding product of forward ptetR187 and tetR
dimer</shortDesc>
        </interface>
        <cnModel>
            <listOfChains>
                <chain>
                    <listOfParts>
                        <part>
                            <partReference>ANY</partReference>
                            <partLabel>X0</partLabel>
                            <partType>DNA</partType>
                            <partCategory>substituent</partCategory>
                        </part>
                        <part>
                            <partReference>ptetR187_bs1</partReference>
                            <partLabel>ptetR187_bs1_0</partLabel>
                            <partType>ForwardDNA</partType>
                            <partCategory>biobrick</partCategory>
                        </part>
                        <part>
                            <partReference>ANY</partReference>
                            <partLabel>X1</partLabel>
                            <partType>DNA</partType>
                            <partCategory>substituent</partCategory>
                        </part>
                    </listOfParts>
                </chain>
                <chain>
                    <listOfParts>
                        <part>
                            <partReference>ANY</partReference>
                            <partLabel>X2</partLabel>
                            <partType>Protein</partType>
                            <partCategory>substituent</partCategory>
                        </part>
                        <part>
                            <partReference>tetR154</partReference>
                            <partLabel>tetR_0</partLabel>
                            <partType>ForwardProtein</partType>
                            <partCategory>biobrick</partCategory>
```

```
                    </part>
                    <part>
                        <partReference>ANY</partReference>
                        <partLabel>X3</partLabel>
                        <partType>Protein</partType>
                        <partCategory>substituent</partCategory>
                    </part>
                </listOfParts>
            </chain>
            <chain>
                <listOfParts>
                    <part>
                        <partReference>ANY</partReference>
                        <partLabel>X4</partLabel>
                        <partType>Protein</partType>
                        <partCategory>substituent</partCategory>
                    </part>
                    <part>
                        <partReference>tetR154</partReference>
                        <partLabel>tetR_1</partLabel>
                        <partType>ForwardProtein</partType>
                        <partCategory>biobrick</partCategory>
                    </part>
                    <part>
                        <partReference>ANY</partReference>
                        <partLabel>X5</partLabel>
                        <partType>Protein</partType>
                        <partCategory>substituent</partCategory>
                    </part>
                </listOfParts>
            </chain>
        </listOfChains>
        <listOfTrees>
            <tree>
                <listOfNodes>
                    <node>
                        <currentNodeLabel>tetR_0</currentNodeLabel>
                        <parentNodeLabel>tetR2</parentNodeLabel>
                    </node>
                    <node>
                        <currentNodeLabel>tetR_1</currentNodeLabel>
                        <parentNodeLabel>tetR2</parentNodeLabel>
```

```
                        </node>
                        <node>
                            <currentNodeLabel>tetR2</currentNodeLabel>
                            <parentNodeLabel>ptetRtetR2</parentNodeLabel>
                        </node>
                        <node>

    <currentNodeLabel>ptetR187_bs1_0</currentNodeLabel>
                            <parentNodeLabel>ptetRtetR2</parentNodeLabel>
                        </node>
                        <node>
                            <currentNodeLabel>ptetRtetR2</currentNodeLabel>
                            <parentNodeLabel>ROOT</parentNodeLabel>
                        </node>
                    </listOfNodes>
                </tree>
            </listOfTrees>
        </cnModel>
        <listOfReferencedReactions>
            <referencedReaction
speciesType="product">bid_fw_ptetR187_tetR154_2</referencedReaction>
        </listOfReferencedReactions>
    </species>
</MoDeL>
```

9.5. Reaction

A **reaction** represents any transformation, transport or binding process, typically a chemical reaction that can change the quantity of one or more species. MoDeL characterizes a reaction mainly the same as SBML does. However, with species described in Chain-Node Model, we have to record the transfer of substituent. Otherwise it is impossible to track the source and destination of a substituent. In the following sections, we will emphasize on the new elements in MoDeL.

9.5.1. The id and name attributes

As with other components in MoDeL, Reaction has a required element, **id**, used to give the reaction an identifier. The identifier **must** be a unique text string. MoDeL also has a **name** element with type string. It is not required to be unique.

9.5.2. The reversible attribute

The boolean element **reversible** indicates whether the reaction is reversible. The default is "true".
A reaction is reversible means it can proceed in either the forward or the reverse direction. Although the reversibility of a reaction can sometimes be deduced by inspecting forwardKinecticLaw and reverseKineticLaw, this flag supports the ability to perform some kinds of model analyses in the absence of performing a time-course simulation of the model.
Moreover, both forwardKineticLaw and reverseKineticLaw are optional and allowed to be left undefined. In which case, a flag to indicate the reversibility is quite useful.

9.5.3. The fast attribute

The optional boolean element **fast**, with its default value, "false", indicates whether the reaction is operating on a time scale significantly faster than the other reactions. Compared to slow reactions, fast ones are considered to be instaneous. Further research reveals that it may lead to different results if software tools ignore the value of the fast. Most prevailing software tools provide support of fast reaction using a pseudo steady-state approximation for the set of fast reactions when constructing the system of equations for the model. Hence, this flag is important over the course of simulation, and it **must** be careful to make the separation of time scales between the fast reactions and other processes.

9.5.4. The interface element

The element of **interface** contains two optional elements of url and shortDesc. They **must** be used as described in Section **8.3**.

9.5.5. The listOfCompartments element

Reaction has an element, **listOfCompartments**, used to define compartments in which the reaction happens. MoDeL views compartments in a nested structure, the inner compartment is considered as child, and the outer one parent. In this way, the structure of compartments can be described in a tree form. This way of describing compartments is very useful since some reactions may only happen under certain structure of compartments.
Each compartment element has three child elements:

- **compartmentReference**: Available compartment identifier defined in the compartment container of database.
- **currentCompartmentLabel**: a unique label in this reaction for current compartment.
- **parentCompartmentLabel**: a unique label in this reaction label of compartment outside of current compartment.

The description of tree structure of compartments is similar to that of species, a node of ROOT is

also required here to denote the parent of the outermost compartment, indicating there's no more compartments outside.

9.5.6. The listOfReactants, listOfModifiers and listOfProducts elements

The species participating as reactants and products in a reaction are declared using element **listOfReactants, and listOfProducts**. Sometimes a species appears in the kinetic rate formula of a reaction but is itself neither created nor destroyed in that reaction (for example, because it acts as a catalyst or inhibitor). Same as SBML, in MoDeL all such species are simply called **modifiers** without regard to the detailed role of those species in the model.
Each reactant/modifier/product has the same child elements:

- **speciesReference**: Reference of species identifier defined in Species Container.
- **speciesLabel**: A unique label of species in this reaction.
- **compartmentLabel**: The label of compartment in which this species locates in.

The element of compartmentLabel also has an attribute of "itself". As described in Section 9.2, we have to add a species of compartment to record its concentration, later we find it necessary to know which compartment is used as which species. This is done by writing the label of compartment in the attribute of itself in the corresponding species. Without this attribute, the template reactions happen between a compartment and a species may get a wrong match.

9.5.7. The listOfSubstituentTransfers element

Based on the design principle of species, each template species may contain several substituent parts. Since the actual parts represented by substituent are unknown, there **must** be a way to record the transfer of a substituent. Only in this way can we construct a product from reactants and modifiers. Therefore, each substituent transfer relation is constructed with two elements: from and to.

- The **From** element has an attribute of speciesLabel, which contains the label of species where the substituent comes from. Label of reactants and modifiers **should** be used here. The content of **From** is the partLabel of the substituent.
- The **To** element has the same structure as From, only the content and attribute are the labels of destination species and destination substituent.

Here is another suggestion for writing a reasonable substituentTransfer element. The two partTypes of the two substituents in a transfer **should** be the same. This is because transfers actually don't change the type of parts.

9.5.8. The kineticLaw element

The element of **kineticLaw** is divided into subelements of forwardKinecticLaw and reverseKineticLaw. They describe the kinetic law of reaction in forward and reverse direction

respectively. We borrow the KineticLaw element defined in SBML and make little modification to use in MoDeL. Both of forward and reverse kinetic laws defined in MoDeL have four elements, math, listOfLocalParameters, listOfReferencedParameters, and listOfConstriants.

The math element

Same as SBML, **forwardKineticLaw** and **reverseKineticLaw** both have a **math** element for holding a MathML formula defining the rate of the reaction. It is important to know the rate is written in substance/time units instead of substance/volume/time units in order to handle multi-compartment reactions. Refer to SBML Document L2V4[1] for more details about math element.

There **must** be some reason why MoDeL separates the definitions of forwardKineticLaw and reverseKineticLaw. The reason is complicated. Since each reaction in MoDeL represents not just one reaction, but a reaction of a specific pattern. Each set of reactants and modifiers which match species structure in database can have this reaction. This will leads to a complex problem: The real number of products may be less than that defined in Reaction because different products may be connected by substituent to form a new complex. It is actually an intermolecular reaction. However, current version of MoDeL does not support intermolecular reactions, and we assume different substructure in one complex would not react with each other. Consequently, only forward reaction is allowed, and that is why we separate the kinetic law definitions of forward and reverse reaction.

Identifiers of five objects are allowed to appear in math element: global parameters defined in System container, speciesLabel and compartmentLabel defined in Reaction container, conditional parameters defined in Part container, and local parameters defined within kinetic law definition in Reaction container. In math element, the identifier of a species represents its concentration. The identifier of a compartment represents its size, and the identifier of a parameter represents the value of the parameter.

The listOfLocalParameters element

The element **listOfLocalParameters** defines local parameters in Reaction. The structure of this element is the same as global parameter. However, the scope of identifier is local and the value is required to be constant.

The listOfReferencedParameters element

The element **listOfReferencedParameters** can contain a list of one or more external parameter references, which are defined in Part container. Each referencedParameterve element has a required subelement of id, which can be used in a math element representing its value.

To refer an external parameter defined in Part container, one have to specify the required

subelement of extRefSource together with two attributes:

- **extRefSource**: content of this element is the identifier of referred external parameter.
- **partLabel**: contains the partLabel of the part which includes the referenced parameter.
- **speciesLabel**: contains the speciesLabel of the species which includes the part.

The listOfConstraints element

This element contains a list of **constraints** on forward of reverse reaction. If any one of the constraints is not satisfied, the reaction in that direction won't happen.
Each constraint element has two required subelements. One is formula, which stores the string form of a mathematical expression. The formula is used as the judgment condition of the constraint, and the return value of this formula is required to be boolean. The other is listOfVariables, which contains a list of all the identifiers of global parameters needed in the formula.

9.5.9. Example

The following example defines the binding reaction of the promoter of ptetR and the protein of TetR dimer.

```
<MoDeL>
    <reaction id="bid_fw_ptetR187_tetR154_2" name="binding forward-ptetR:tetR2"
reversible="true" fast="false">
        <interface>
            <url>http://2010.igem.org/Team:USTC_Software</url>
            <short_desc>binding of forward ptetR187 DNA and tetR
dimer</short_desc>
        </interface>
        <listOfCompartments>
            <compartment>
                <compartmentReference>Flask</compartmentReference>
                <currentCompartmentLabel>Flask</currentCompartmentLabel>
                <parentCompartmentLabel>ROOT</parentCompartmentLabel>
            </compartment>
            <compartment>
                <compartmentReference>E_coli</compartmentReference>
                <currentCompartmentLabel>E_coli</currentCompartmentLabel>
                <parentCompartmentLabel>Flask</parentCompartmentLabel>
            </compartment>
        </listOfCompartments>
        <listOfReactants>
            <reactant>
                <speciesReference>fw_ptetR187_dna</speciesReference>
```

```
            <speciesLabel>ptetR187_dna</speciesLabel>
            <compartmentLabel>E_coli</compartmentLabel>
        </reactant>
        <reactant>
            <speciesReference>tetR154_dimer</speciesReference>
            <speciesLabel>tetR154_dimer</speciesLabel>
            <compartmentLabel>E_coli</compartmentLabel>
        </reactant>
    </listOfReactants>
    <listOfModifiers>
    </listOfModifiers>
    <listOfProducts>
        <product>
            <speciesReference>fw_ptetR187_tetR154_2</speciesReference>
            <speciesLabel>ptetR187tetR154_2</speciesLabel>
            <compartmentLabel>E_coli</compartmentLabel>
        </product>
    </listOfProducts>
    <listOfSubstituentTransfers>
        <substituentTransfer>
            <from speciesLabel="ptetR187_dna">X0</from>
            <to speciesLabel="ptetR187tetR154_2">X0</to>
        </substituentTransfer>
        <substituentTransfer>
            <from speciesLabel="ptetR187_dna">X1</from>
            <to speciesLabel="ptetR187tetR154_2">X1</to>
        </substituentTransfer>
        <substituentTransfer>
            <from speciesLabel="tetR154_dimer">X0</from>
            <to speciesLabel="ptetR187tetR154_2">X2</to>
        </substituentTransfer>
        <substituentTransfer>
            <from speciesLabel="tetR154_dimer">X1</from>
            <to speciesLabel="ptetR187tetR154_2">X3</to>
        </substituentTransfer>
        <substituentTransfer>
            <from speciesLabel="tetR154_dimer">X2</from>
            <to speciesLabel="ptetR187tetR154_2">X4</to>
        </substituentTransfer>
        <substituentTransfer>
            <from speciesLabel="tetR154_dimer">X3</from>
            <to speciesLabel="ptetR187tetR154_2">X5</to>
```

```
            </substituentTransfer>
        </listOfSubstituentTransfers>
        <kineticLaw>
            <forwardKineticLaw>
                <math>
                  <apply>
                    <times/>
                    <ci> kon </ci>
                    <ci> ptetR187_dna </ci>
                    <ci> tetR154_dimer </ci>
                    <ci> E_coli </ci>
                  </apply>
                </math>
                <listOfLocalParameters>
                    <localParameter>
                        <id>kon</id>
                        <value>1.00E+6</value>
                        <units>litre_per_mole_per_second</units>
                        <constant>true</constant>
                    </localParameter>
                </listOfLocalParameters>
                <listOfReferencedParameters>
                </listOfReferencedParameters>
                <listOfConstraints>
                </listOfConstraints>
            </forwardKineticLaw>
            <reverseKineticLaw>
                <math>
                  <apply>
                    <times/>
                    <ci> koff </ci>
                    <ci> ptetR187tetR154_2 </ci>
                    <ci> E_coli </ci>
                  </apply>
                </math>
                <listOfLocalParameters>
                    <localParameter>
                        <id>koff</id>
                        <value>1.00E-5</value>
                        <units>per_second</units>
                        <constant>true</constant>
                    </localParameter>
```

```
                </listOfLocalParameters>
                <listOfReferencedParameters>
                </listOfReferencedParameters>
                <listOfConstraints>
                </listOfConstraints>
            </reverseKineticLaw>
        </kineticLaw>
    </reaction>
</MoDeL>
```

10. Table of detailed MoDeL structure

In this section we present a detailed table of MoDeL structure. Indent in table indicates the level of an element, {} in identifier means an element with one or more subelements, {} in datatype are used to contain available values.

/System/listOfFunctionDefinitions/functionDefinition				
		optional	datatype	default
attribute	id		DBId	
attribute	name	optional	string	
element	math{}			
element	lambda			

/System/listOfRules/rule				
		optional	datatype	default
element	assignmentRule/rateRule{}	optional		
attribute	variable		DBId	
element	math			
element	algebraicRule{}	optional		
element	math			

/System/listOfUnitDefinitions/unitDefinition				
		optional	datatype	default
attribute	id		DBId	
attribute	name	optional	string	
element	listOfUnits{}			
element	unit			
attribute	kind		DBId	
attribute	exponent	optional	int	
attribute	scale	optional	int	
attribute	multiplier	optional	double	

/System/listOfGlobalParameters/globalParameter				
		optional	datatype	default
element	id		DBId	
element	name	optional	string	
element	units	optional	DBId	
element	value	optional	double	0.00
element	constant	optional	boolean	true

/Compartment				
		optional	datatype	default
attribute	id		DBId{same as /Part/compartment}	
attribute	name	optional	string	
attribute	spatialDimensions	optional	int{3}	3
attribute	size		double	0.00
attribute	units	optional	DBId	
attribute	constant	optional	boolean{false}	false
element	interface{}			
element	url	optional	string	
element	shortDesc	optional	string	
element	listOfInnerCompartmentPermitted{}			
element	compartmentReference	optional	DBId	

/Part/biobrick				
		optional	datatype	default
attribute	id		DBId	
attribute	name	optional	string	
attribute	registryName	optional	regName	
attribute	originalConformation	optional	DBId	
element	interface{}			
element	url	optional	string	
element	shortDesc	optional	string	
element	listOfConditionalParameters{}			
element	conditionalParameter{}	optional		
attribute	id		DBId{forwardPromoterEfficiency,reversePromoterEfficiency,forwardRbsEfficiency,reverseRbsEfficiency,forwardTerminatorEfficiency,reverseTerminatorEfficiency}	
attribute	name	optional	string	
attribute	units	optional	DBId	
attribute	commonValue	optional	double	0.00
element	parameterValue	optional	double	
attribute	compartment		DBId	
element	conditionalParameter{}	optional		
attribute	id		DBId{forwardStartCodonEfficiency,reverseStartCodonEfficiency,forwardStopCodonEfficiency,reverseStopCodonEfficiency}	
attribute	name	optional	string	
attribute	units	optional	DBId	
attribute	commonValue	optional	int{0,1}	0
element	parameterValue	optional	int{0,1}	
attribute	compartment		DBId	
element	listOfReferencedSpecies{}			
element	referencedSpecies	optional	DBId	
attribute	partType	optional	string{ForwardDNA,ReverseDNA,ForwardRNA,ReverseRNA,ForwardProtein,ReverseProtein,Compound,Compartment,DNA,RNA,Protein}	

/Part/plasmidBackbone				
		optional	datatype	default
attribute	id		DBId	
attribute	name	optional	string	
attribute	registryName	optional	regName	
attribute	originalConformation	optional	DBId	
attribute	origin	optional	string	
attribute	resistance	optional	string	
element	interface{}			
element	url	optional	string	
element	shortDesc	optional	string	
element	listOfConditionalParameters{}			
element	listOfReferencedSpecies{}			
element	referencedSpecies	optional	DBId	
attribute	partType	optional	string{ForwardDNA,ReverseDNA,ForwardRNA,ReverseRNA,ForwardProtein,ReverseProtein,Compound,Compartment,DNA,RNA,Protein}	

/Part/compartment				
		optional	datatype	default
attribute	id		DBId{same as /Compartment}	
attribute	name	optional	string	
attribute	registryName	optional	regName	
attribute	originalConformation	optional	DBId	
element	interface{}			
element	url	optional	string	
element	shortDesc	optional	string	
element	listOfConditionalParameters{}			
element	listOfReferencedSpecies{}			
element	referencedSpecies	optional	DBId	
attribute	partType	optional	string{ForwardDNA,ReverseDNA,ForwardRNA,ReverseRNA,ForwardProtein,ReverseProtein,Compound,Compartment,DNA,RNA,Protein}	

/Part/compound				
		optional	datatype	default
attribute	id		DBId	
attribute	name	optional	string	
attribute	registryName	optional	regName	
attribute	originalConformation	optional	DBId	
element	interface{}			
element	url	optional	string	
element	shortDesc	optional	string	
element	listOfConditionalParameters{}			
element	listOfReferencedSpecies{}			
element	referencedSpecies	optional	DBId	
attribute	partType	optional	string{ForwardDNA,ReverseDNA,ForwardRNA,ReverseRNA,ForwardProtein,ReverseProtein,Compound,Compartment,DNA,RNA,Protein}	

/Part/substituent				
		optional	datatype	default
attribute	id		DBId{ANY,ANYUB,NZ,NZUB,ONE,ONEUB}	
attribute	name	optional	string	
attribute	registryName	optional	regName	
attribute	originalConformation	optional	DBId	
element	interface{}			
element	url	optional	string	
element	shortDesc	optional	string	
element	listOfConditionalParameters{}			
element	listOfReferencedSpecies{}			
element	referencedSpecies	optional	DBId	
attribute	partType	optional	string{ForwardDNA,ReverseDNA,ForwardRNA,ReverseRNA,ForwardProtein,ReverseProtein,Compound,Compartment,DNA,RNA,Protein}	

/Species				
		optional	datatype	default
attribute	id		DBId	
attribute	name	optional	string	
element	interface{}			
element	url	optional	string	
element	shortDesc	optional	string	
element	cnModel{}			
element	listOfChains{}			
element	chain{}	optional		
element	listOfParts{}			
element	part{}			
element	partReference		DBId	
element	partLabel		string	
element	partType		string{ForwardDNA,ReverseDNA,ForwardRNA,ReverseRNA,ForwardProtein,ReverseProtein,Compound,Compartment,DNA,RNA,Protein}	
element	partCategory		string{biobrick,compound,compartment,plasmidBackbonesubstituent}	
element	listOfTrees{}			
element	tree{}	optional		
element	listOfNodes{}			
element	node{}			
element	currentNodeLabel		string{partLabel}	
element	parentNodeLabel		string{partLabel}	
element	listOfReferencedReactions{}			
element	referencedReaction	optional	DBId	
attribute	speciesType		string{reactant,modifier,product}	

/Reaction				
		optional	datatype	default
attribute	id		DBId	
attribute	name	optional	string	
attribute	reversible	optional	boolean	true
attribute	fast	optional	boolean	false
element	interface{}			
element	url	optional	string	
element	shortDesc	optional	string	
element	listOfCompartments{}			
element	compartment{}			
element	compartmentRefernce		DBId	
element	currentCompartmentLabel		string	
element	parentCompartmentLabel		string	
element	listOfReactants\Modifiers\Products{}			
element	reactant\modifier\product{}	optional		
element	speciesReference		DBId	
element	speciesLabel		string	
element	compartmentLabel		string{currentCompartmentLabel}	
attribute	itself	optional	string{currentCompartmentLabel}	
element	listOfSubstituentTransfers{}			
element	substituentTransfer{}	optional		
element	from{}		string{partLabel}	
attribute	speciesLabel		string{speciesLabel}	
element	to{}		string{partLabel}	
attribute	speciesLabel		string{speciesLabel}	
element	kineticLaw{}			
element	forward/reverseKineticLaw{}			
element	math	optional	MathML2.0	
element	listOfLocalParameters{]			
element	localParameter{}	optional		
element	id		string{used in math}	
element	name		string	
element	value		double	0.00
element	units		DBId	
element	constant		boolean	true
element	listOfReferencedParameters{}			
element	referencedParameter{}	optional		
element	id		string{used in math}	
element	extRefSource			
attribute	speciesLabel		string{speciesLabel}	
attribute	partLabel		string{partLabel}	
element	listOfConstraints{}			
element	constraint{}	optional		
element	listOfVariables{}			
element	variable		DBId{GlobalParameter}	
element	formula		string{math form}	

11. A complete example of database construction

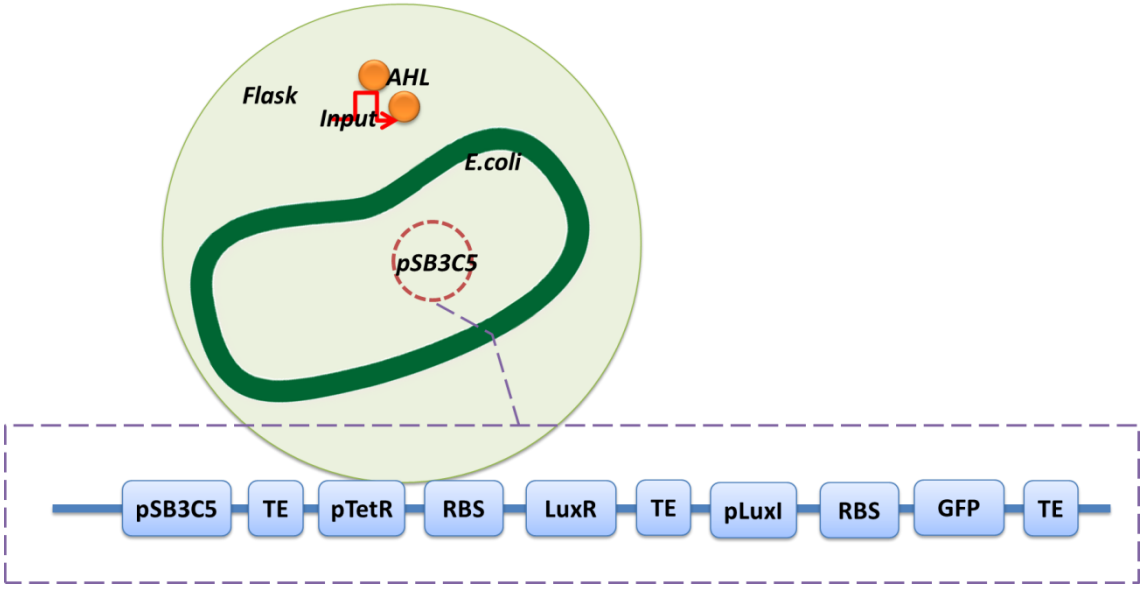
In this section, we will present a detailed instruction on constructing a database using the language of MoDeL. We choose a concrete and classical example that we have successfully modeled with our software (see the second Demo on http://2010.igem.org/Team:USTC_Software). This example includes the diffusion of AHL, the propagation of E.coli, the activation of pluxI by AHL:LuxR dimer, and so on.

The construction of such a system is shown in the following figure. The outermost compartment is a Flask, and it has a sub-compartment of E.coli, which is a species simultaneously since it can propagate. The compound of AHL is placed in the flask outside of E.coli, and a rule of input is attached to it indicating the time and amount of adding AHL.

Within E.coli is the user-constructed plasmid and the backbone is pSB3C5. As is mentioned in the

document, a backbone always comes first on a chain. The next part on the chain is TE the terminator, and there are two more on the chain. All have 100% forward and reverse terminator efficiency. This **should** be required especially for the first and last terminator on the chain, so that the transcription starts from promoter won't go through plasmid backbone. Then come the parts of pTetR, RBS, LuxR, pLuxI, another RBS, GFP, and TE. They are drawn in a unified form intentionally to show that all parts are equivalent in MoDeL.

We **should** mention here that in this simple example we do not take environmental conditions into account, and the parameters we use here may not be experimentally correct.



Experiment of this system is done by Haseloff Lab, MIT (http://partregistry.org/Part:BBa_F2620:Transfer_function), and our simulation result is in good agreement with the experiment result.

The automatic modeling database of this system clearly needs to store all the reaction, species, parts, inputs, functions, etc. It is important not to miss any element. The complete list is given below:

Compartment	Species	Reaction
Flask	AHL_compound	tm_deg_rna_ecoli
E.coli	AHL_LuxR	tm_deg_protein_ecoli
	AHL2_LuxR2	tm_dil_ecoli
	AHL2_LuxR2_fw_pLuxI	repro_ecoli_cell
	AHL2_LuxR2_rev_pLuxI	repli_tm_pSB3C5_fw
	LuxR_prot	repli_tm_pSB3C5_rev
	GFP_prot	repre_tm_pSB3C5_fw
	E_coli_cell	repre_tm_pSB3C5_rev
	fw_pLuxI_dna	deg_AHL_ecoli
	rev_pLuxI_dna	diffusion_AHL
	protein_sc	bid_AHL_LuxR
	rna_sc	dmr_AHL2_LuxR2
	any_species	bid_AHL2_luxR2_fw_pluxI
		bid_AHL2_luxR2_rev_pluxI

Part				
compartment	compound	substituent	biobrick	plasmidBackbone
E.coli	AHL	ANY	TE	pSB3C5
		ANYUB	pTetR	
			RBS	
			LuxR	
			GFP	
			pLuxI	

System			
functionDefinition	rule	unitDefinition	globalParameter
piecewise function	t = simulation time	all used units	t

It is important to know the database just provide the necessary elements, actual compartment structure, initial concentrations, input functions, and so on are defined by users in the input file. An input file will be read by iGaME and provide missing information of the system. This **should** not be described in this document about MoDeL, but the input file of this system is given below for convenience.

```
<MoDeL>
    <dbInterface>
        <input>
            <listOfParameters>
                <parameter>
                    <id> ts </id>
                    <value> 10000 </value>
                    <units> second </units>
                    <constant> true </constant>
                </parameter>
                <parameter>
                    <id> te </id>
                    <value> 10001 </value>
                    <units> second </units>
                    <constant> true </constant>
                </parameter>
                <parameter>
                    <id> s </id>
                    <value> 1E-9 </value>
                    <units> mole_per_litre </units>
                    <constant> true </constant>
                </parameter>
```

```
            </listOfParameters>
            <listOfRules>
                <assignmentRule>
                    <variable>sPecIes1</variable>
                    <math>step_func (t, ts, te, s)</math>
                </assignmentRule>
            </listOfRules>
            <listOfRules>
            </listOfRules>
            <listOfCompartments>
                <compartment db="Flask">
                    <id>Flask</id>
                    <size>0.40</size>
                    <outside>ROOT</outside>
                    <constant>true</constant>
                </compartment>
                <compartment db="E_coli">
                    <id>E_coli</id>
                    <size>7E-16</size>
                    <outside>Flask</outside>
                    <constant>true</constant>
                </compartment>
            </listOfCompartments>
            <listOfSpecies>
                <species db="E_coli_cell">
                    <id>sPecIes0</id>
                    <compartment itself="E_coli">Flask</compartment>
                    <initialConcentration>4.15135E-24</initialConcentration>
                </species>
                <species db="AHL_compound">
                    <id>sPecIes1</id>
                    <compartment>Flask</compartment>
                    <initialConcentration> 0 </initialConcentration>
                    <constant> false </constant>
                    <boundaryCondition> true </boundaryCondition>
                </species>
                <species>
                    <id>sPecIes2</id>
                    <compartment>E_coli</compartment>
                    <initialConcentration>2.37E-9</initialConcentration>
                    <cnModel>
                        <listOfChains>
```

```
                    <chain>
                        <listOfParts>
                            <part>
                                <partReference>pSB3C5</partReference>
                                <partLabel>pSB3C5_0</partLabel>
                                <partType>ForwardDNA</partType>
                                <partCategory>biobrick</partCategory>
                            </part>
                            <part>
                                <partReference>TE_X_100</partReference>
                                <partLabel>TE_X_100_0</partLabel>
                                <partType>ForwardDNA</partType>
                                <partCategory>biobrick</partCategory>
                            </part>
                            <part>
                                <partReference>Promoter1</partReference>
                                <partLabel>Promoter1_0</partLabel>
                                <partType>ForwardDNA</partType>
                                <partCategory>biobrick</partCategory>
                            </part>
                            <part>
                                <partReference>RBS147</partReference>
                                <partLabel>RBS147_0</partLabel>
                                <partType>ForwardDNA</partType>
                                <partCategory>biobrick</partCategory>
                            </part>
                            <part>
                                <partReference>C0062_luxR</partReference>
                                <partLabel>C0062_luxR_0</partLabel>
                                <partType>ForwardDNA</partType>
                                <partCategory>biobrick</partCategory>
                            </part>
                            <part>
                                <partReference>TE_X_100</partReference>
                                <partLabel>TE_X_100_1</partLabel>
                                <partType>ForwardDNA</partType>
                                <partCategory>biobrick</partCategory>
                            </part>
                            <part>
                                <partReference>R0062_pluxI</partReference>
                                <partLabel>R0062_pluxI_0</partLabel>
                                <partType>ForwardDNA</partType>
```

```
                                        <partCategory>biobrick</partCategory>
                                    </part>
                                    <part>
                                        <partReference>RBS147</partReference>
                                        <partLabel>RBS147_1</partLabel>
                                        <partType>ForwardDNA</partType>
                                        <partCategory>biobrick</partCategory>
                                    </part>
                                    <part>
                                        <partReference>E0040_GFP</partReference>
                                        <partLabel>E0040_GFP_0</partLabel>
                                        <partType>ForwardDNA</partType>
                                        <partCategory>biobrick</partCategory>
                                    </part>
                                    <part>
                                        <partReference>TE_X_100</partReference>
                                        <partLabel>TE_X_100_2</partLabel>
                                        <partType>ForwardDNA</partType>
                                        <partCategory>biobrick</partCategory>
                                    </part>
                                </listOfParts>
                            </chain>
                        </listOfChains>
                    </cnModel>
                </species>
            </listOfSpecies>
        </input>
    </dbInterface>
</MoDeL>
```

With the knowledge of MoDeL components, it **should** be difficult to understand this input file.

We **should** explain some of the template species included. protein_sc, rna_sc are used to handle degradation of any unbound protein and any unbound RNA. We consider only these two kinds of species have degradation because DNA doesn't degrade and complex first unbind into its each composition. Following is the example protein_sc, and rna_sc is similar to it.

```
<MoDeL>
    <species id="Protein_sc" name="single Protein chain">
        <interface>
            <url>http://2010.igem.org/Team:USTC_Software</url>
            <shortDesc>species template for any unbinded Protein
chain</shortDesc>
        </interface>
        <cnModel>
```

```
            <listOfChains>
                <chain>
                    <listOfParts>
                        <part>
                            <partReference>ANYUB</partReference>
                            <partLabel>ANYUB</partLabel>
                            <partType>Protein</partType>
                            <partCategory>substituent</partCategory>
                        </part>
                    </listOfParts>
                </chain>
            </listOfChains>
            <listOfTrees>
            </listOfTrees>
        </cnModel>
        <listOfReferencedReactions>
            <referencedReaction
speciesType="reactant">tm_deg_protein_ecoli</referencedReaction>
        </listOfReferencedReactions>
    </species>
</MoDeL>
```

Another special template species is any_species, which is used to handle dilution caused by cell reproduction. Any_species is written as follows.

```
<MoDeL>
    <species id="any_species" name="any species">
        <interface>
            <url>http://2010.igem.org/Team:USTC_Software</url>
            <shortDesc>species template for anything</shortDesc>
        </interface>
        <cnModel>
            <listOfChains>
                <chain>
                    <listOfParts>
                        <part>
                            <partReference>ANY</partReference>
                            <partLabel>ANY</partLabel>
                            <partCategory>substituent</partCategory>
                        </part>
                    </listOfParts>
                </chain>
            </listOfChains>
            <listOfTrees>
```

```
            </listOfTrees>
        </cnModel>
        <listOfReferencedReactions>
            <referencedReaction speciesType="reactant">tm_dil_ecoli</referencedReaction>
            <referencedReaction
speciesType="reactant">tm_dil_ecoli_2</referencedReaction>
            <referencedReaction speciesType="reactant">tm_dil_chemo</referencedReaction>
        </listOfReferencedReactions>
    </species>
</MoDeL>
```

In the following we will discuss the reactions in the system. Notice that since the reaction rate is written in substance/time units, a compartment volume is always multiplied in the rate expression. First, let's have a look at the template reaction of dilution.

The tm_dil_ecoli reaction has any_species as reactant, and E_coli_cell as modifier for the growth rate will appear in rate expression. The rate of dilution is considered to be directly proportional to the concentration of any_species and the growth rate of E_coli_cell. Knowing this, the template reaction is written as follows.

```
<MoDeL>
    <reaction id="tm_dil_ecoli" name="dilution of species in E.coli" reversible="false"
fast="false">
        <interface>
            <url>http://2010.igem.org/Team:USTC_Software</url>
            <short_desc>dilution of species due to reproduction of E.coli</short_desc>
        </interface>
        <listOfCompartments>
            <compartment>
                <compartmentReference>Flask</compartmentReference>
                <currentCompartmentLabel>Flask0</currentCompartmentLabel>
                <parentCompartmentLabel>ROOT</parentCompartmentLabel>
            </compartment>
            <compartment>
                <compartmentReference>E_coli</compartmentReference>
                <currentCompartmentLabel>E_coli0</currentCompartmentLabel>
                <parentCompartmentLabel>Flask0</parentCompartmentLabel>
            </compartment>
        </listOfCompartments>
        <listOfReactants>
            <reactant>
                <speciesReference>any_species</speciesReference>
                <speciesLabel>any_species</speciesLabel>
                <compartmentLabel>E_coli0</compartmentLabel>
            </reactant>
```

```
        </listOfReactants>
        <listOfModifiers>
            <modifier>
                <speciesReference>E_coli_cell</speciesReference>
                <speciesLabel>E_coli_cell</speciesLabel>
                <compartmentLabel itself="E_coli0">Flask0</compartmentLabel>
            </modifier>
        </listOfModifiers>
        <listOfProducts>
        </listOfProducts>
        <listOfSubstituentTransfers>
        </listOfSubstituentTransfers>
        <kineticLaw>
            <forwardKineticLaw>
                <math>
                  <apply>
                    <times/>
                    <ci> kgr </ci>
                    <apply>
                      <minus/>
                      <cn type="integer"> 1 </cn>
                      <apply>
                        <divide/>
                        <ci> E_coli_cell </ci>
                        <ci> maxc </ci>
                      </apply>
                    </apply>
                    <ci> any_species </ci>
                    <ci> E_coli0 </ci>
                  </apply>
                </math>
                <listOfLocalParameters>
                    <localParameter>
                        <id>kgr</id>
                        <value>1.92E-4</value>
                        <units>per_second</units>
                        <constant>true</constant>
                    </localParameter>
                    <localParameter>
                        <id>maxc</id>
                        <value>1.66E-12</value>
                        <units>mole_per_litre</units>
```

```
                        <constant>true</constant>
                    </localParameter>
                </listOfLocalParameters>
            </forwardKineticLaw>
            <reverseKineticLaw>
            </reverseKineticLaw>
        </kineticLaw>
    </reaction>
</MoDeL>
```

There are several template degradation reactions, tm_deg_protein_ecoli, tm_deg_rna_ecoli, deg_AHL_ecoli, deg_AHL_flask. We decide to handle all degradations of RNA and protein in a unified way since it's impossible to obtain the rate for each species. However the situation of compound is not so complicated and we can still write single reactions for each compound. The degradation rate is considered to be proportional to the concentration of the species. Tm_deg_protein_ecoli is given below.

```
<MoDeL>
    <reaction id="tm_deg_protein_ecoli" name="degradation of proteins in E.coli"
reversible="false" fast="false">
        <interface>
            <url>http://2010.igem.org/Team:USTC_Software</url>
            <short_desc>degradation of proteins in E_coli</short_desc>
        </interface>
        <listOfCompartments>
            <compartment>
                <compartmentReference>E_coli</compartmentReference>
                <currentCompartmentLabel>E_coli</currentCompartmentLabel>
                <parentCompartmentLabel>ROOT</parentCompartmentLabel>
            </compartment>
        </listOfCompartments>
        <listOfReactants>
            <reactant>
                <speciesReference>protein_sc</speciesReference>
                <speciesLabel>anyub_forwardprotein</speciesLabel>
                <compartmentLabel>E_coli</compartmentLabel>
            </reactant>
        </listOfReactants>
        <listOfModifiers>
        </listOfModifiers>
        <listOfProducts>
        </listOfProducts>
        <listOfSubstituentTransfers>
        </listOfSubstituentTransfers>
```

```
        <kineticLaw>
            <forwardKineticLaw>
                <math>
                  <apply>
                    <times/>
                    <ci> kd </ci>
                    <ci> anyub_forwardprotein </ci>
                    <ci> E_coli </ci>
                  </apply>
                </math>
                <listOfLocalParameters>
                    <localParameter>
                        <id>kd</id>
                        <value>2.3E-3</value>
                        <units>per_second</units>
                        <constant>true</constant>
                    </localParameter>
                </listOfLocalParameters>
                <listOfReferencedParameters>
                </listOfReferencedParameters>
                <listOfConstraints>
                </listOfConstraints>
            </forwardKineticLaw>
            <reverseKineticLaw>
            </reverseKineticLaw>
        </kineticLaw>
    </reaction>
</MoDeL>
```

Next is the diffusion of AHL between E.coli and Flask. Since this is a multi-compartment reaction, the rate may seem confusing. We sum up the volume of all E.coli and multiply the volume to forward and reverse kinetic law in order to solve the problem of volume conversion. The reaction is written below.

```
<MoDeL>
    <reaction id="diffusion_AHL" name="AHL diffusion" reversible="true"
fast="false">
        <interface>
            <url>http://2010.igem.org/Team:USTC_Software</url>
            <short_desc>transport of AHL to Ecoli</short_desc>
        </interface>
        <listOfCompartments>
            <compartment>
                <compartmentReference>Flask</compartmentReference>
```

```
            <currentCompartmentLabel>Flask</currentCompartmentLabel>
            <parentCompartmentLabel>ROOT</parentCompartmentLabel>
        </compartment>
        <compartment>
            <compartmentReference>E_coli</compartmentReference>
            <currentCompartmentLabel>E_coli</currentCompartmentLabel>
            <parentCompartmentLabel>Flask</parentCompartmentLabel>
        </compartment>
    </listOfCompartments>
    <listOfReactants>
        <reactant>
            <speciesReference>AHL_compound</speciesReference>
            <speciesLabel>AHL_in</speciesLabel>
            <compartmentLabel>E_coli</compartmentLabel>
        </reactant>
    </listOfReactants>
    <listOfModifiers>
    </listOfModifiers>
    <listOfProducts>
        <product>
            <speciesReference>AHL_compound</speciesReference>
            <speciesLabel>AHL_out</speciesLabel>
            <compartmentLabel>Flask</compartmentLabel>
        </product>
    </listOfProducts>
    <listOfSubstituentTransfers>
    </listOfSubstituentTransfers>
    <kineticLaw>
        <forwardKineticLaw>
            <math>
              <apply>
                <times/>
                <ci> kon </ci>
                <ci> AHL_in </ci>
                <ci> E_coli </ci>
              </apply>
            </math>
            <listOfLocalParameters>
                <localParameter>
                    <id>kon</id>
                    <value>0.1</value>
                    <units>per_second</units>
```

```
                    <constant>true</constant>
                </localParameter>
            </listOfLocalParameters>
            <listOfReferencedParameters>
            </listOfReferencedParameters>
            <listOfConstraints>
            </listOfConstraints>
        </forwardKineticLaw>
        <reverseKineticLaw>
            <math>
              <apply>
                <times/>
                <ci> koff </ci>
                <ci> AHL_out </ci>
                <ci> E_coli </ci>
              </apply>
            </math>
            <listOfLocalParameters>
                <localParameter>
                    <id>koff</id>
                    <value>0.1</value>
                    <units>per_second</units>
                    <constant>true</constant>
                </localParameter>
            </listOfLocalParameters>
            <listOfReferencedParameters>
            </listOfReferencedParameters>
            <listOfConstraints>
            </listOfConstraints>
        </reverseKineticLaw>
    </kineticLaw>
</reaction>
</MoDeL>
```

Last is the binding reactions, they are all written with a prefix of bid. The reaction rate is normally handled. One of the reactions: bid_AHL_luxR is written below.

```
<MoDeL>
    <reaction  id="bid_AHL_luxR"  name="binding  AHL:luxR"  reversible="true"
fast="false">
        <interface>
            <url>http://2010.igem.org/Team:USTC_Software</url>
            <short_desc>binding of AHL compound and luxR</short_desc>
        </interface>
```

```
        <listOfCompartments>
            <compartment>
                <compartmentReference>E_coli</compartmentReference>
                <currentCompartmentLabel>E_coli</currentCompartmentLabel>
                <parentCompartmentLabel>ROOT</parentCompartmentLabel>
            </compartment>
        </listOfCompartments>
        <listOfReactants>
            <reactant>
                <speciesReference>AHL_compound</speciesReference>
                <speciesLabel>AHL_compound</speciesLabel>
                <compartmentLabel>E_coli</compartmentLabel>
            </reactant>
            <reactant>
                <speciesReference>luxR_prot</speciesReference>
                <speciesLabel>luxR_prot</speciesLabel>
                <compartmentLabel>E_coli</compartmentLabel>
            </reactant>
        </listOfReactants>
        <listOfModifiers>
        </listOfModifiers>
        <listOfProducts>
            <product>
                <speciesReference>AHL_luxR</speciesReference>
                <speciesLabel>AHL_luxR</speciesLabel>
                <compartmentLabel>E_coli</compartmentLabel>
            </product>
        </listOfProducts>
        <listOfSubstituentTransfers>
            <substituentTransfer>
                <from speciesLabel="luxR_prot">X0</from>
                <to speciesLabel="AHL_luxR">X0</to>
            </substituentTransfer>
            <substituentTransfer>
                <from speciesLabel="luxR_prot">X1</from>
                <to speciesLabel="AHL_luxR">X1</to>
            </substituentTransfer>
        </listOfSubstituentTransfers>
        <kineticLaw>
            <forwardKineticLaw>
                <math>
                  <apply>
```

```
                <times/>
                <ci> kon </ci>
                <ci> AHL_compound </ci>
                <ci> luxR_prot </ci>
                <ci> E_coli </ci>
              </apply>
            </math>
            <listOfLocalParameters>
                <localParameter>
                    <id>kon</id>
                    <name>rate of binding</name>
                    <value>1E+4</value>
                    <units>litre_per_mole_per_second</units>
                    <constant>true</constant>
                </localParameter>
            </listOfLocalParameters>
        </forwardKineticLaw>
        <reverseKineticLaw>
            <math>
              <apply>
                <times/>
                <ci> koff </ci>
                <ci> AHL_luxR </ci>
                <ci> E_coli </ci>
              </apply>
            </math>
            <listOfLocalParameters>
                <localParameter>
                    <id>koff</id>
                    <name>rate_of_unbinding</name>
                    <value>0.01</value>
                    <units>per_second</units>
                    <constant>true</constant>
                </localParameter>
            </listOfLocalParameters>
            <listOfReferencedParameters>
            </listOfReferencedParameters>
            <listOfConstraints>
            </listOfConstraints>
        </reverseKineticLaw>
    </kineticLaw>
</reaction>
```

```
</MoDeL>
```

And this is the end of the example. You can find the whole database on our Github page: http://github.com/jkdirac/igame. Hopefully this example will answer most of your questions in database construction.

12. Future Work

MoDeL is young, to be exact, just born. There **must** be a long way before he can stand up and run by himself. At this stage, we have to take extra care of him.
Knowing the drawbacks of MoDeL, we have already got a list of improvements waiting to be implemented in the next version of MoDeL. They are:

- **Enhance MoDeL to support description of intra-molecular reactions.**

If it is applied, our software will become a useful tool to study the competition mechanism between intra-molecular and inter-molecular reactions.

- **More powerful template.**

Only six kinds of substituents are far from meeting the demands of more accurate modeling and complex reaction mechanism. At present, substituent parts are limited to match only parts on one chain, and they are not allowed to represent a structure with more than one chains and trees. This needs the extension of definition of substituent parts by designing a unified format that can provide enough information.

- **Enable description of transcription and translation reactions.**

Currently, transcription and translation are handled differently with other reactions. It is necessary to modify MoDeL to incorporate description of the two reactions.

- **Eliminate the inconsistencies.**

In the current version of MoDeL, a bunch of inconsistencies exist. For example, the identifiers of compartments follow different restrictions from other global identifiers, and some components' attributes are required to be a certain value. These problems make MoDeL not so elegant when looking at details. Work **must** be done to improve its consistency.

- **Be more independent.**

Currently, some of MoDeL syntaxes are defined for specific algorithms of iGaME. Being a language for automatic modeling, cooperation with algorithm is reasonable. However, we believe there are too many compromises, thus causing a number of inconsistencies. In the next version, we are looking forward to a more independent MoDeL.

13. Author's Contact Information

Zhen Wang xqqixp@mail.ustc.edu.cn
Chen Liao liaochen@mail.ustc.edu.cn

Hao Jiang haojiang@mail.ustc.edu.cn
Xiaomo Yao soimort@mail.ustc.edu.cn
Kun Jiang jkdirac@mail.ustc.edu.cn

14. Known Bugs

1. The labels in a math element **must not** include another label in the same math element as its sub-string. Otherwise it will cause error in iGaME. This is because the XML elements in math are not analyzed separately.

15. References

1. Hucka, M., et al., *Systems Biology Markup Language (SBML) Level 2 : Structures and Facilities for Model Definitions ere California Institute of Technology , USA.* Systems Biology, 2008.
2. *W3C (2000b). W3C's math home page. Available via the World Wide Web at http://www.w3.org/Math/.*