Heta language: Modularity and Reusability in QSP modeling platforms

Evgeny Metelkin

InSysBio, Moscow, Russia

MOTIVATION & OBJECTIVES

The size and complexity of mechanistic **Systems Biology** (SB) and **Quantitative Systems Pharmacology** (QSP) models continue to increase. The typical modern QSP model includes hundreds components and is based on many experimental findings and literature data.

The size is a challenging problem for efficient handling such models and modeling platforms because of several reasons:

- Large-scale models is not easy to modify because it requires reviewing of many items and mathematical expressions to find and update the required one.
- Debugging and testing models require substantial resources.
- Large models are usually created by teams. Working with a model as a whole it is hardly to develop the effective modeling workflow.
- Simulation and analysis requires substantial time.

The **modularity** which is a typical approach in software engineering might be a solution for the SB/QSP modeling problems. The decomposition of the whole modeling code into smaller parts allows working more effectively, to share job within team and to **reuse code** in many projects.

This study presents the approach for modularity implemented in the Heta modeling language. The approach can be effectively applied for Heta-based modeling platforms and implemented in many simulation tools like Matlab, Simbiology, Mrgsolve, HetaSimulator.jl, and others.

HETA LANGUAGE

Heta is a human-readable and writable modeling language for Quantitative Systems Pharmacology (QSP) and Systems Biology (SB).

Heta language represents the dynamic model in a process-description format i.e. as interacting components that describe volumes, concentrations, amounts, rates, and others. The Heta code can be represented as ordinary differential equations (ODEs) in-place.

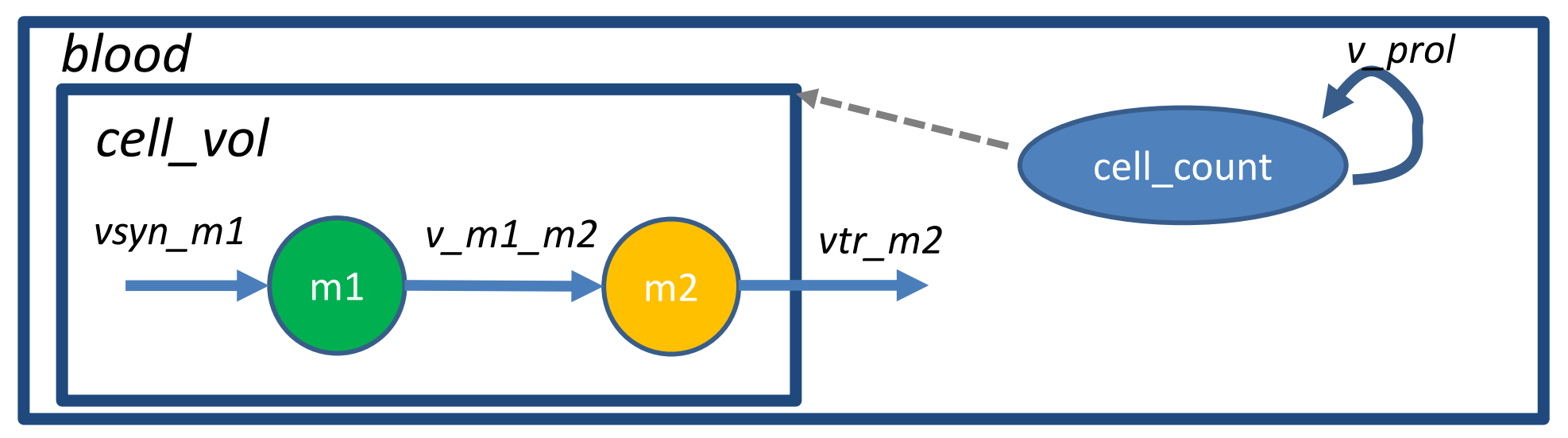
Features:

- **Human-readable/writable** code can be used for model creation, modification, or integration.
- **Easy code parsing** and transformation for potential implementation into different tools and frameworks.
- **Modularity:** QSP platform can be subdivided into several files and spaces for better project management.
- **Reusability:** modeling platforms should be easily extended for other projects.
- Reach **annotation** capabilities for better code revision and reporting.
- **Simple transformation** to popular modeling formats: Matlab, R, Simbiology, DBSolve, SBML, etc.

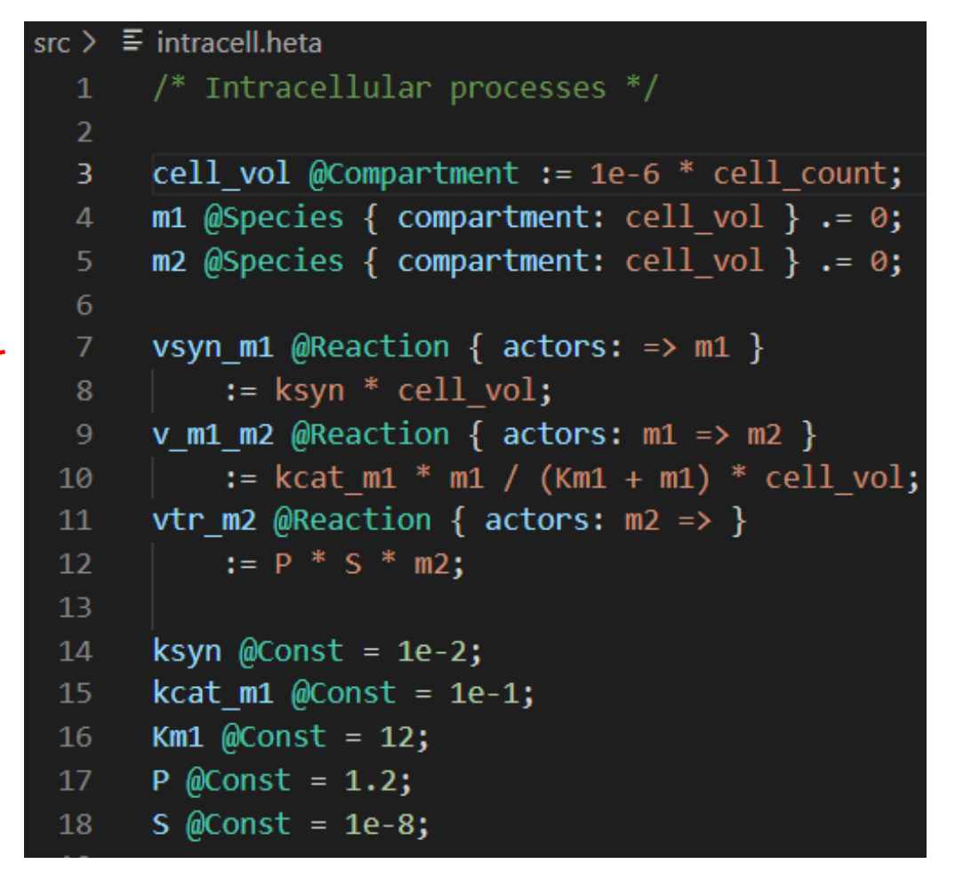
#INCLUDE

The **modules** in Heta platforms are files of one of supported types. One of the files (usually having name *index.heta*) is used as an entry point for platform compilation. The content of other files can be attached to platform using the *#include* action.

- #include acts like we put the code from one module into another one.
- Any module can include another modules. It is possible to organize a hierarchical module structure with many files and directories.
- There are several module's types: Heta code, Excel table, JSON/YAML or SBML. We can use a hybrid Heta platform. Part of the platform will be written in Heta code, another part in SBML,excel files etc.
- Modules may be developed by team members or taken from external sources.

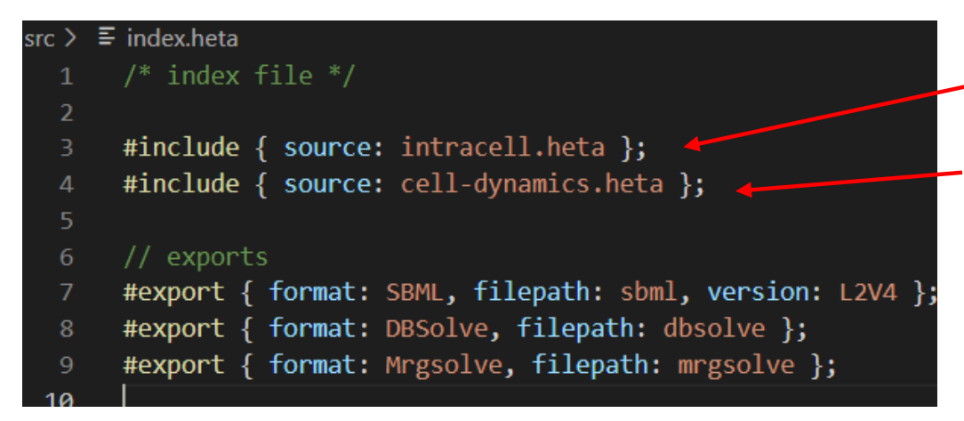


intracell.heta



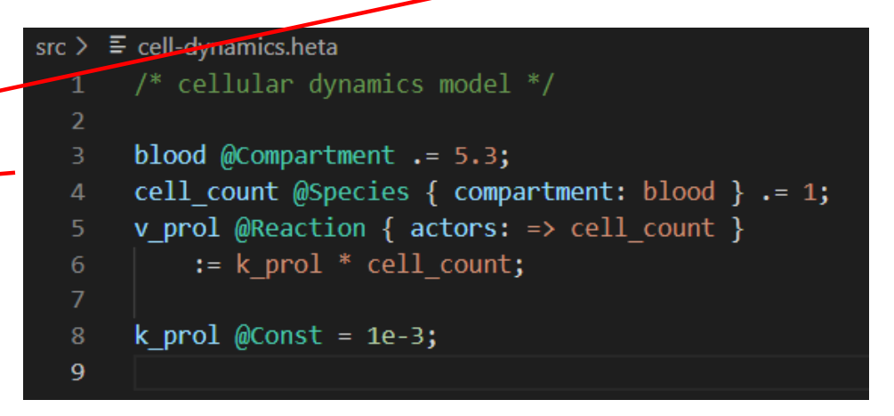
```
src > intracell.heta
/* Intracellular processes */

cell_vol @Compartment := 1e-6 * cell_count;
m1 @Species { compartment: cell_vol } .= 0;
m2 @Species { compartment: cell_vol } .= 0;

vsyn_m1 @Reaction { actors: => m1 }
    := ksyn * cell_vol;
v_m1_m2 @Reaction { actors: m1 => m2 }
    := kcat_m1 * m1 / (Km1 + m1) * cell_vol;
vtr_m2 @Reaction { actors: m2 => }
    := P * S * m2;

ksyn @Const = 1e-2;
kcat_m1 @Const = 1e-1;
Km1 @Const = 12;
P @Const = 1.2;
S @Const = 1e-8;
```

index.heta



```
src > index.heta
/* index file */

#include { source: intracell.heta };
#include { source: cell-dynamics.heta };

// exports
#export { format: SBML, filepath: sbml, version: L2V4 };
#export { format: DBSolve, filepath: dbsolve };
#export { format: Mrgsolve, filepath: mrgsolve };
```

cell-dynamics.heta



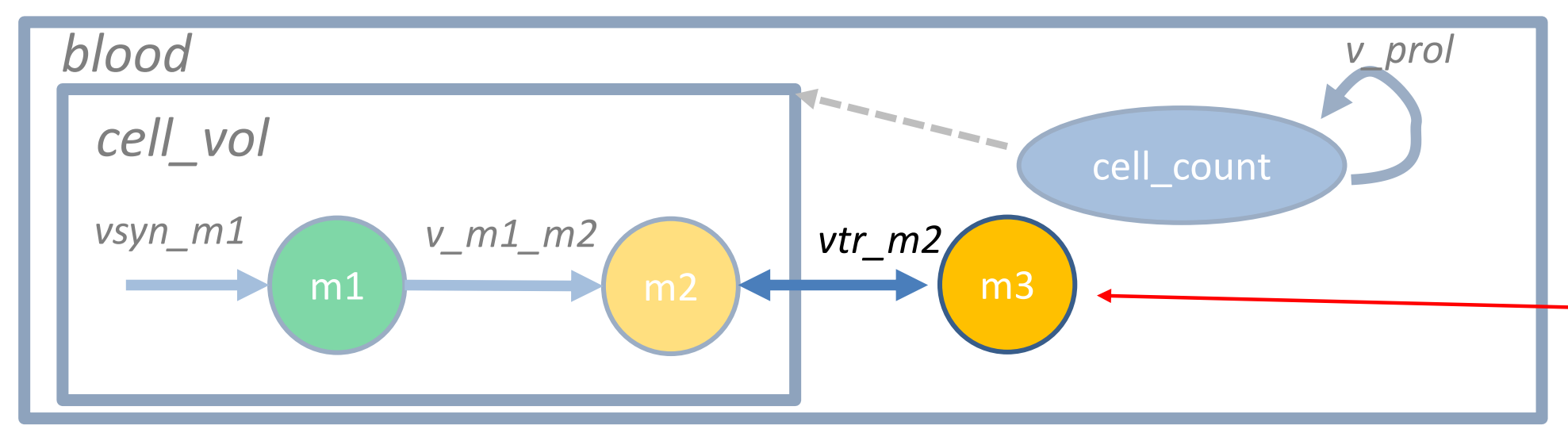
```
src > cell-dynamics.heta
/* cellular dynamics model */

blood @Compartment .= 5.3;
cell_count @Species { compartment: blood } .= 1;
v_prol @Reaction { actors: => cell_count }
    := k_prol * cell_count;

k_prol @Const = 1e-3;
```

Conclusion: The "include" mechanism allows to split the whole code into any number of separate files. Combining modules one can create different models.

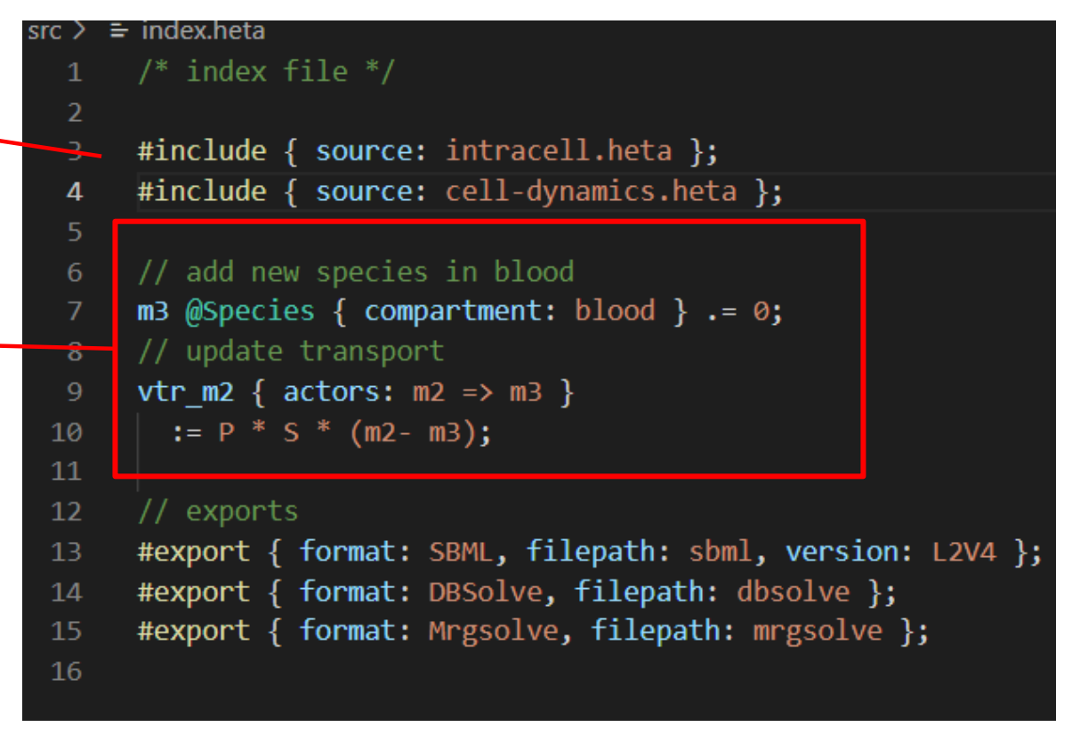
#INSERT, #UPDATE, #DELETE

#insert, #update and #delete are three base actions for modification of model components. They are used correspondently to include/replace, change properties and delete component by selecting id.

- The actions modify components declared above code part including those loaded from modules.
- This mechanism allow to modify a modeling platform without rewriting the original source code.



index.heta (updated)



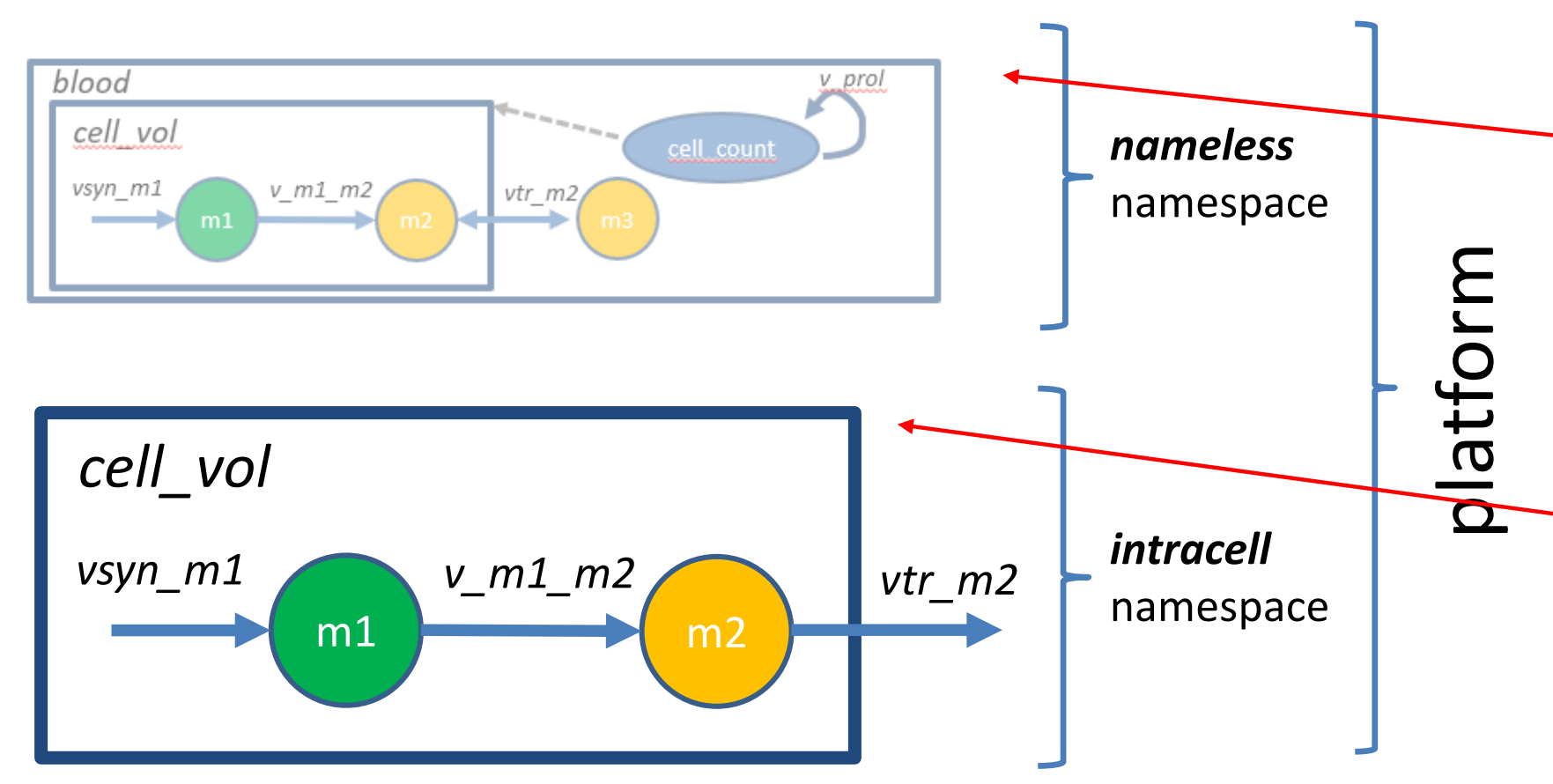
```
src > index.heta
/* index file */

#include { source: intracell.heta };
#include { source: cell-dynamics.heta };

// add new species in blood
m3 @Species { compartment: blood } .= 0;
// update transport
vtr_m2 { actors: m2 => m3 }
  := P * S * (m2- m3);

// exports
#export { format: SBML, filepath: sbml, version: L2V4 };
#export { format: DBSolve, filepath: dbsolve };
#export { format: Mrgsolve, filepath: mrgsolve };
```

Conclusion: Heta format allows to update components without changing the source code. This is powerful mechanism to tune model created from shared building blocks.

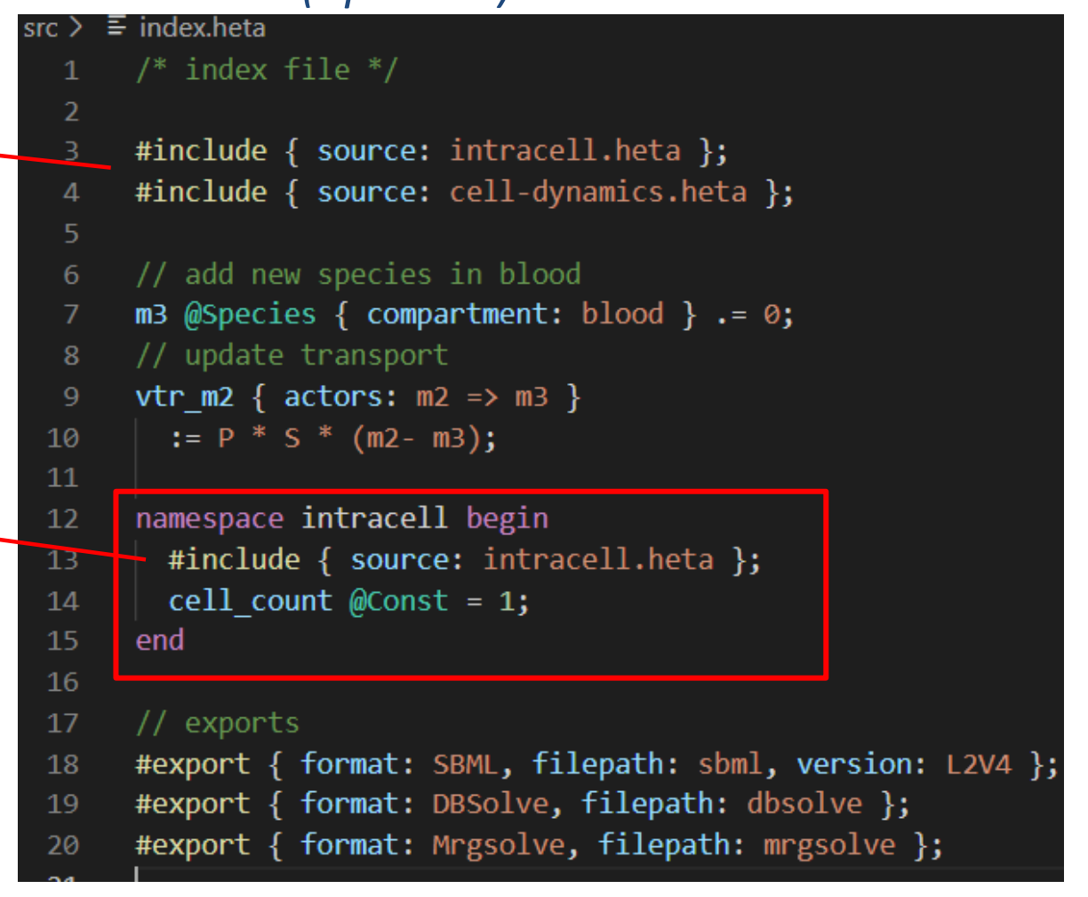
NAMESPACE

The **namespace** in Heta standard is a way to create several models in one modeling platform. The models can share some parts or components or be independent.

- Namespace is a storage of components availably by an identifier.
- There is a default namespace with the id "nameless" created at platform initialization.
- A user can create any number of namespaces using the **namespace statement** or the **#setNS** action.



index.heta (updated)



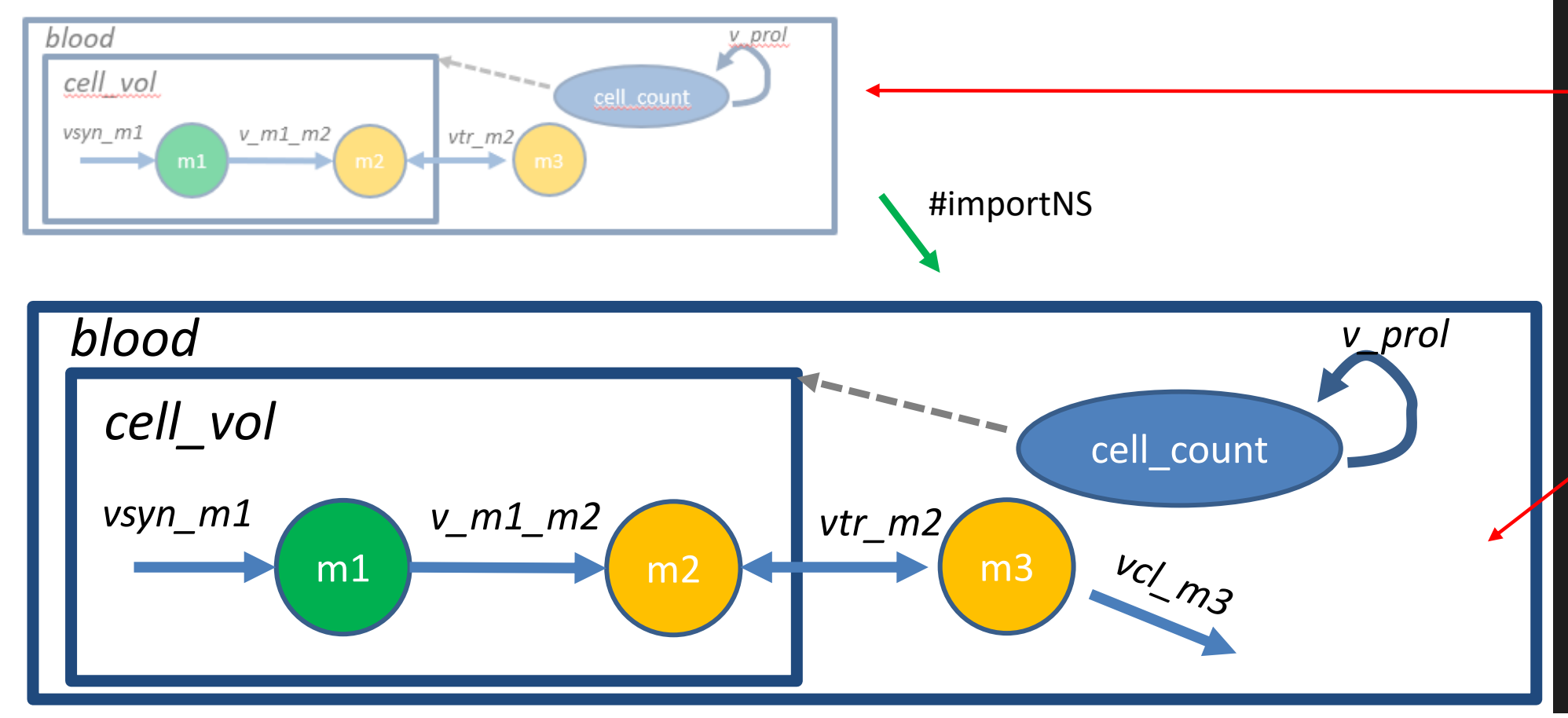
```
src > index.heta
/* index file */

#include { source: intracell.heta };
#include { source: cell-dynamics.heta };

// add new species in blood
m3 @Species { compartment: blood } .= 0;
// update transport
vtr_m2 { actors: m2 => m3 }
  := P * S * (m2- m3);

namespace intracell begin
  #include { source: intracell.heta };
  cell_count @Const = 1;
end

// exports
#export { format: SBML, filepath: sbml, version: L2V4 };
#export { format: DBSolve, filepath: dbsolve };
#export { format: Mrgsolve, filepath: mrgsolve };
```

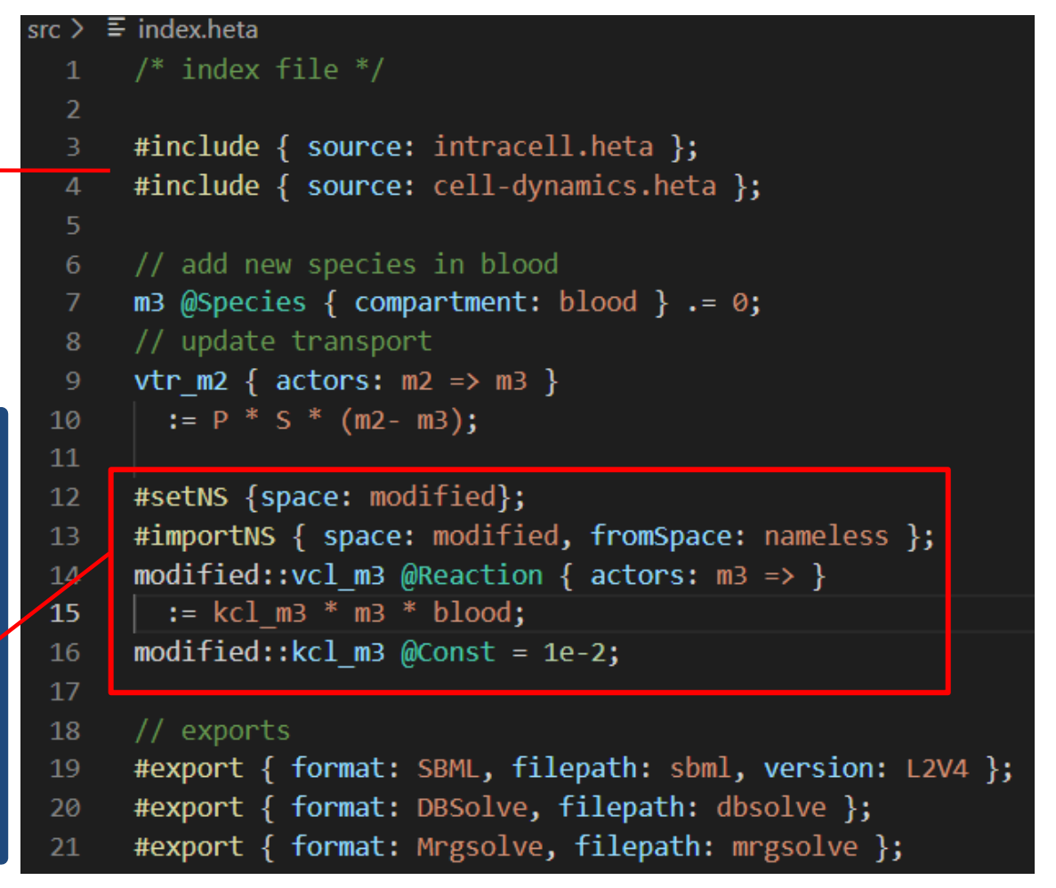
Conclusion: Namespaces allow to reuse modules for different models of the same platform. This can be helpful when it is necessary to use model variants applicable for *in vitro*, animal species and human body conditions.

#IMPORT, #IMPORTNS

A namespace content can be copied into another namespace using **#importNS** action. It is also possible to borrow one or several components with the **#import** action.

- The #importNS action copies all namespace's content into another one. The components with the same id will be merged.
- The #import action copies a single component.





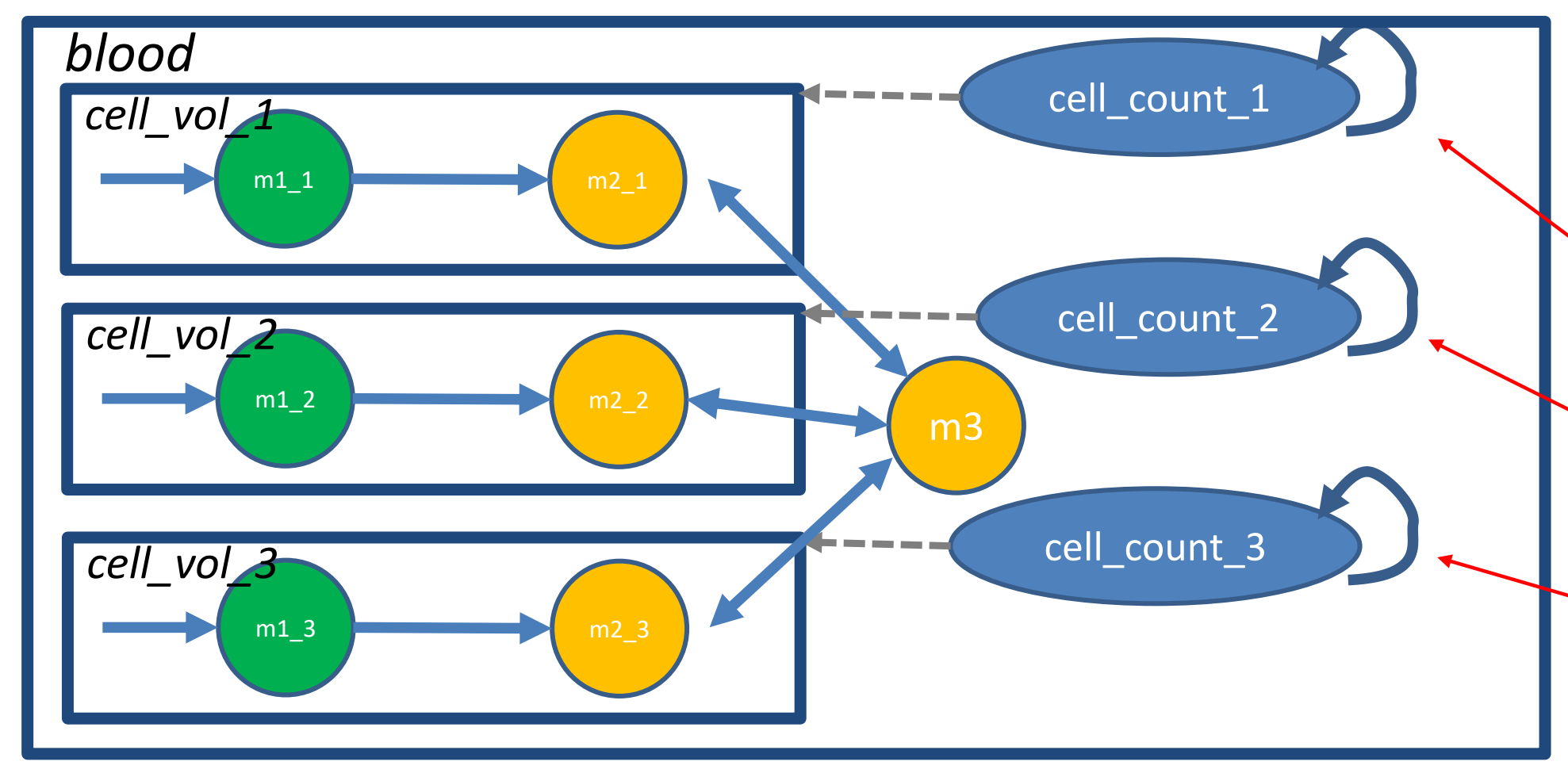
```
src > index.heta
/* index file */

#include { source: intracell.heta };
#include { source: cell-dynamics.heta };

// add new species in blood
m3 @Species { compartment: blood } .= 0;
// update transport
vtr_m2 { actors: m2 => m3 }
  := P * S * (m2- m3);

#setNS {space: modified};
#importNS { space: modified, fromSpace: nameless };
modified::vcl_m3 @Reaction { actors: m3 => }
  := kcl_m3 * m3 * blood;
modified::kcl_m3 @Const = 1e-2;

// exports
#export { format: SBML, filepath: sbml, version: L2V4 };
#export { format: DBSolve, filepath: dbsolve };
#export { format: Mrgsolve, filepath: mrgsolve };
```

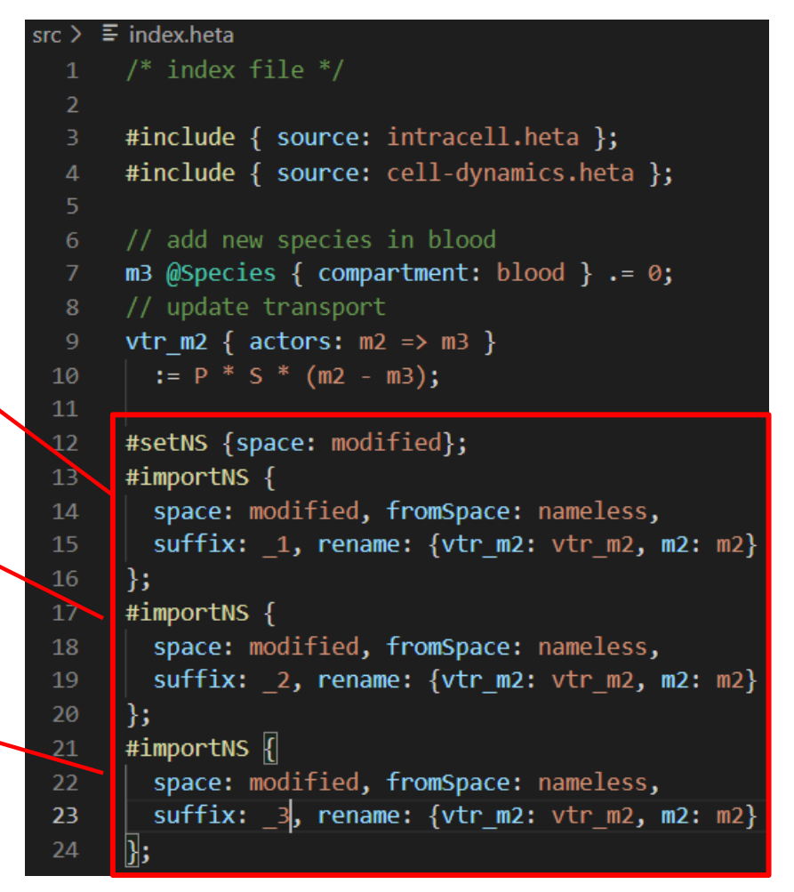
Conclusion: It is possible to reuse not only modules but namespaces too. It is the easiest way to create model's variants in the same platform or to borrow some necessary components from another model.

#IMPORT + RENAME

#importNS and **#import** actions have a series of additional properties which can be used to modify copied elements "on-the-fly". This approach can be used for creating repeated modeling structures.

- The "prefix" and "suffix" properties updates identifiers of all copied components with a simple rule.
- The "rename" property is the way how the identifiers can be updated manually one by one.





```
src > index.heta
/* index file */

#include { source: intracell.heta };
#include { source: cell-dynamics.heta };

// add new species in blood
m3 @Species { compartment: blood } .= 0;
// update transport
vtr_m2 { actors: m2 => m3 }
  := P * S * (m2 - m3);

#setNS {space: modified};
#importNS {
  space: modified, fromSpace: nameless,
  suffix: _1, rename: {vtr_m2: vtr_m2, m2: m2}
};
#importNS {
  space: modified, fromSpace: nameless,
  suffix: _2, rename: {vtr_m2: vtr_m2, m2: m2}
};
#importNS {
  space: modified, fromSpace: nameless,
  suffix: _3, rename: {vtr_m2: vtr_m2, m2: m2}
};
```

Conclusion: It is possible to create really complex models with #importNS action writing very simple code. It is not necessary to write similar model parts many times.

MORE EXAMPLES

Open source QSP platforms (shared on GitHub)

- Demo example of QSP platform developed with Heta and qs3p-js.
  https://github.com/insysbio/heta-case-mini/
- Platform describing Fatty Acid Amide Hydrolase inhibition in human.
  https://github.com/insysbio/faah-inhibitor
- Model describing SARS-CoV-2 virus and host cell life cycles
  https://github.com/insysbio/covid19-qsp-model
- PBPK Modeling For Therapeutic Nanoparticles Loaded With Drug
  https://github.com/insysbio/drug-loaded-nanoparticles

REFERENCES & CONTACTS

Heta project homepage:
https://hetalang.github.io

Video tutorial
https://rb.gy/xgpkft

GitHub repository
hetalang/heta-compiler

dev@insysbio.com
http://insysbio.com

PIA-010 9-13 November 2021, ACOP 12, Virtual