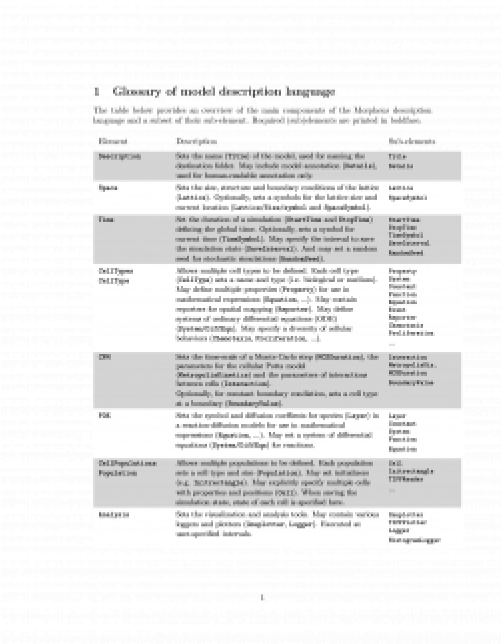
Model description language

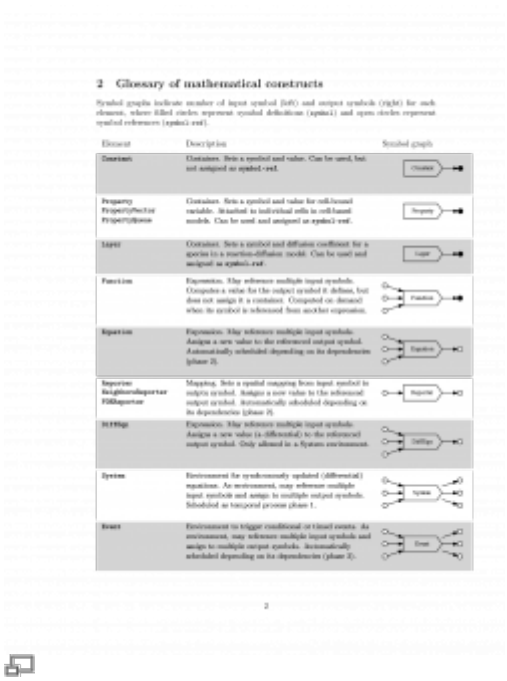
The Morpheus model description language (MDL) is a domain-specific extensible mark-up language (XML) for the specification of multiscale multicellular simulation models. It has been developed to separate the process of modeling from numerical implementation to allow users to specify simulation models in terms of familiar biological and mathematical concepts rather than in a general-purpose programming language (see sections [sub:Domain-specific-language] and [sec:Glossary-of-main]). This makes modeling of multiscale multicellular systems accessible for researchers and students without computational expertise. And, importantly, it structures the scientific workflow in the development of simulation models is which becoming increasingly important as the systems and models under investigation are growing more complex.

In Morpheus, models are fully specified by single XML files written in a human-readable format. This circumvents complex models to be distributed among multiple source and parameter files, and avoids the need for (re)compilation of source code after model changes. Moreover, encapsulation of model description in single files eases editing, archiving, checkpointing and the exchange of models among users.

The key enabling technique that allows concise descriptions of (potentially complex) multiscale models is the use of symbols, symbol references and expressions (section [sub:Symbols-and-expressions] and [sec:Glossary-of-mathematical]). This provides an intuitive mathematical interface and provide a high level of flexibility in the model description language. Importantly, it enables integration of models to be performed automatically, as explained in section [sec:Model-integration].


1 Glossary of model description language

Glossary of main elements of Morpheus model description language



2 Glossary of mathematical constructs

Glossary of mathematical constructs in Morpheus model description language

glossaries.pdf

Mark-up language (XML) and XML schema description (XSD)

The Morpheus model description language (MDL) is based on the extensible mark-up language (XML) that enables the use of human-readable domain-specific terminology and provides an extensible interface to adapt to the ongoing development of the simulation environment. From a computation perspective, it provides a clear structure that can be easily parsed and is independent of operating systems.

The terminology, grammatical rules and constraints for the structure and content of valid MDL documents are laid down in a XML schema description (XSD). As such, the XSD defines the MDL. The XSD describes e.g. whether elements are required or optional, how many occurrences are allows, what attributes are allowed and what value types each attribute requires, and contains documentation describing elements and attributes.

The standalone graphical user interface application (morpheus-gui) uses these rules to validate model files (checking whether a certain XML file conforms to these rules) as well as to restore outdated model files (adding or removing elements and attributes to make model compliant to newer version of the MDL). Most importantly, however, morpheus-gui uses the XSD in the editor to constrain the editing of model descriptions as to guarantee its validity. For instance, it ensures that required elements (such as Time) cannot be removed, or that only certain data types (double or string) can be entered for specific attributes. Morpheus-gui also uses the XSD to provide context-sensitive documentation, upon selection of a particular element or attribute in the editor.

Unfortunately, the XSD schema language has several shortcoming that may lead to unintuitive behavior of the GUI. For instance, the XSD schema language does not support mutually exclusive attributes nor co-occurrence constraints which would allow the occurrence of elements to depend on values of other elements.

Nevertheless, the XML and XSD files allow for a separation of simulator and the graphical user interface by forming interfaces between the two standalone applications (morpheus and morpheus-gui). (see figure [fig:XML-XSD-1]). During compilation of the simulator, the XSD document describing the MDL is assembled from many small XSD files that specify the rules for each individual part of the simulator. This XSD document is subsequently built into morpheus-gui to use it for editing, as explained above. Upon execution of a model from morpheus-gui, the XML document is written to file, and passed to morpheus as a command line argument.

This separation between simulator and GUI is useful to allow headless simulations, without graphical interface, and enables simulation on remote computing on high performance computing (HPC) resources. If morpheus simulator is installed on a remote HPC, it can be controlled remotely from a desktop computer by sending XML files over a network connection.

Domain specific language

An important design goal for the Morpheus MDL has been to be able to express multiscale multicellular models in terms of familiar biological and mathematical terms instead of a general-purpose programming code or scripting language. In other words, the MDL should be a domain-specific language, a descriptive formal language that is suitable for computer processing while allowing models to be expressed in the terminology and at the level of abstraction of the intended application domain. This makes models easier to develop and renders them self-documenting.

Having multicellular systems biology as its application domain, the choice of biological and mathematical terminology follows naturally. The MDL uses biological concepts such as CellType and Populations to define simulated entities, and concepts such as Proliferation and Chemotaxis to define cellular processes. In addition, the MDL employs mathematical terms such as Equation and Systems of DiffEqn (differential equations) to specify relations and changes of variables. Yet, the inclusion of some computational terms related to the numerical implementation cannot be avoided. For instance, a time-step must be specified for numerical solvers. Moreover, it is of course necessary to specify the duration of simulation and the size or resolution of the simulated domain. The MDL therefore uses a mixture of biological, mathematical and computational terminology. A glossary of the terminology of the Morpheus model description language is available in section [sec:Glossary-of-main].

The domain-specificity of the MDL also influences its structure. For instance, whereas reaction-diffusion models describing morphogens must be specified outside of the definition of a CellType, the specification of ordinary differential equations (ODE) describing intracellular processes is only allowed inside a CellType. As another example, the specification of cellular Potts-related parameters (CPM) is separated from the definition of CellTypes to allow cell-based models to be specified without CPM-like cell motility.

Symbols and expressions

The use of symbolic references and expressions provides an important level of model flexibility in the model description language. Mathematical expressions can be used to relate model variables to each other, describe how variables change or to analyze model behavior. They are written as plain-text formulae in terms of user-defined symbols. This makes it straightforward to translate handwritten or published mathematical models into simulation models (see table [tab:Translation-of-DiffEqn]).

%\marginnote{\protect\includegraphics[width=5mm]{marginnote}}

A glossary of the available mathematical constructs included in the Morpheus model description language is included in section [sec:Glossary-of-main].

Symbol definitions

Declaration of each model variable requires the specification of a symbol that is used as a reference to the quantity it represents. Symbols must be provided for user-specified constants (Constant) and variables such as cell-bound properties in cell-based models (CellType\fshyp{}Property) and species in reaction-diffusion models (PDE\fshyp{}Layer). Optionally, symbols can also be specified for built-in model variables such as lattice size (Lattice\fshyp{}Size), location (Lattice\fshyp{}SpaceSymbol) and time (Time\fshyp{}TimeSymbol). These allow users to define spatial gradients, time-dependent functions, etc.

Symbol references and expressions

Symbol definitions and references

Symbols can be referenced in two ways (see figure [fig:symbol-def-ref]): First, symbols can be used in expressions (Expression) written as plain text using the typical operators like +,-,*,^ to form algebraic expressions. Such expressions are used as the right-hand-side (rhs) of functions, (differential) equations and event conditions (Function, Equation, DiffEqn, Condition). Symbols in expressions represent the independent variables or input symbols whose values can be read but not assigned.

Second, symbols can also be referenced using the attribute symbol-ref, which provides an interface for reading values as well as assigning new values to symbols. These can act both as independent variable, input symbols, and dependent variables, i.e. output symbols. Note that some symbols, such as those referring to constants, lattice size or time, are read-only and cannot be assigned via symbol references.

Metadata

Symbols are more than a simple reference to a value. They contain metadata about the (sub)model in which they are defined. This metadata is derived from the XML path in which their definition occurs. At initialization, this is used to associate each symbol with a particular type describing its identity. This way, the simulator knows whether a symbol refers to either a property of a cell, or a species in a reaction-diffusion system. Likewise, built-in symbols like time, space, or cell ID are also associated with their type. The metadata allows Morpheus to differentiate between the various symbol types, and adjust the reading or assigning of values according to each type.

Resolving symbol references

How a symbol reference should be resolved depends on (1) the context in which it is used, and (2) the context in which the symbol has been defined. In multi-scale models these contexts may be different. Therefore, like symbol definitions, symbol references are also associated with their context, derived from their occurrence in the model (XML path). Yet, whereas symbol definitions and their metadata

are registered during initialization, the resolving symbol references is performed at runtime, as explained in more detail in section [sub:Mapping-spatial-data].