19 Datenstrukturen in MEDICI-PK

Im folgenden werden die konstruierten Datenstrukturen erläutert, um die Flexibilität und Erweiterbarkeit des Programmtools MEDICI-PK zu zeigen. In den Teilen, die die Code-Zeilen zur Implementierung zeigen, werden nur die **wesentlichen** Member und Memberfunktionen gezeigt.

19.1 Einheitentypen und Einheitensystem

19.1.1 Definition

Wird in einem Programmpaket ein Einheitensystem mit den gängigsten Einheitentypen fest implementiert, so wird sich schon nach kurzer Zeit die Notwendigkeit ergeben, dieses zu erweitern. Je nach Sprachraum können z.Bsp. die benutzten Einheitenbezeichnungen unterschiedlich sein (m, cm, mm, foot, ect), und es werden, auch abhängig vom Fortschritt in der Modellierung, zusätzliche Einheiten nötig werden. Das Problem der unterschiedlichen Bezeichner kann man durch Vorschreiben bzw. ausschliessliche Benutzung des SI-Systems umgehen, nicht aber die Einführung von bisher nicht vorgesehenen Einheitentypen. Dieses Problem kann umgangen werden, indem man die Definition des Einheitensystems dem Benutzer überläßt und ihn damit in die Lage versetzt, sein System allen Anforderungen anzupassen. Wir haben dazu folgenden Ansatz gewählt:

Das Einheitensystem besteht aus einer offenen Liste von Einheitentypen.

Definition 32 *Ein Einheitentyp ist ein (selbstgewählter) Bezeichner für eine physikalische Größe. Er ist durch seinen Namen eindeutig definiert. Ein Einheitentyp hat eine Liste von Einheitenbeschreibungen, von denen eine seine Darstellung nach SI-Norm ist.*

Beispiel 33 *Ein Einheitentyp ist die „Länge". Er bezeichnet damit einen Abstand oder eine Ausdehnung. Er hat als SI-Einheit das Meter m.*

Einheitentypen sind eindeutig gekennzeichnet durch ihren Namen. Es dürfen nur verschiedene Einheitentypen in einem Einheitensystem definiert werden.

Definition 34 *Eine Einheitenbeschreibung ist ein Bezeichner, der eine bestimmte Ausformung eines Einheitentyps beschreibt. Dazu muss sein Name gegeben werden sowie der Umrechnungsfaktor (bzw. Offset) zwischen ihm und der SI-Einheit des Einheitentyps, dem die Einheitenbeschreibung zugeordnet ist. Die Einheitenbeschreibung, die die SI-Einheit darstellt, hat den Umrechnungsfaktor 1 und den Offset 0.*

Beispiel 35 *Dem Einheitentyp Länge wird die Einheitenbeschreibung „mm" zugewiesen, Umrechnungsfaktor zur SI-Einheit m ist 1/1000.*

19.1.2 Einheiten innerhalb der Projekt-Datenstrukturen

Offensichtlich besitzen alle benutzten Parameter (siehe auch Parameterverwaltung) eine Einheit. Weniger offensichtlich, aber ebenso natürlich, besitzen auch die Modelle, die in der Modellbasis definiert werden, eine Einheit. Sie geben einen Massenfluss zurück, dessen Bedeutung vom Modelltyp abhängt. Parameter können damit innerhalb eines Modells in einer gewünschten Einheit(enbeschreibung) benutzt werden; damit kann man zum Beispiel Literaturwerte ohne Umrechnung eingeben und verwenden.

19.1.3 Implementierung

class VUnitDescr : public MObject {

protected:

AnsiString unitName, comment;

double factorToSI; // faktor von einer Einheit von SI, z.B. 1 kg = 1000 g, also factorToSI = 1000

double offsetToSI; // offset einer Einheit von SI, z.B. 1 C = 1 F + 273.15, also offsetToSI = 273.15

}

class VUnitType : public MObject {

protected:

int unitType;

AnsiString typeName;

AnsiString siUnit;

AnsiString comment;

AnsiString outUnit;

VUnitDescrList* descriptions;

public:

bool ConvertUnit(AnsiString actUnit, AnsiString aimUnit, double mm, double& val);

}

class VUnitSystem : public MObject {

VUnitTypeList* unitTypes;

public:

```
bool InitializeOurUnits();

VUnitType* GetTypeFromUnitString(const AnsiString& str) const;

}
```

19.1.4 Parameterverwaltung

Definition 36 *Ein Parameter in MEDICI-PK ist eine beliebige, aber festgehaltene Variable, der man eine bestimmte Bedeutung zuweisen kann. Jeder Parameter ist daher von einem bestimmten, nicht änderbaren Einheitentyp. Jedem Parameter kann man eindeutig seine Abhängigkeit von Wirkstoff, Individuum, von beidem oder von keinem zuweisen. Je nachdem bezeichnen wir ihn als compound dependent, individual dependent, mixed dependent parameter oder independent parameter. Der Wert eines Parameters kann vom Kompartiment (Organ) abhängen, innerhalb dessen er in einer funktionalen Abhängigkeit benutzt und ausgewertet wird. Auch diese Eigenschaft ist fester Teil eines Parameters.*

Die mögliche Abhängigkeit des Wertes eines Parameters vom Organ, innerhalb dessen er ausgewertet wird, führt dazu, dass nicht ein Wert, sondern eine Liste von Werten für diesen Parameter gespeichert werden muss. Pro Organ, welches man betrachten will, muss ein eigener Wert mit einer eigenen Einheit vorzugeben sein. Die Parameter, die zur Modellierung eingesetzt werden, sind als ebenso eindeutig anzusehen wie die Einheitentypen des Einheitensystems. Daher werden sie in einer offenen Liste gesammelt und für alle Modellierprojekte zur Verfügung gestellt.

Definition 37 *Die Liste von Parametern, die definiert werden, bildet die Parameterbasis.*

Einem Parameter aus der Parameterbasis muss kein Wert zugewiesen werden; man könnte eine Standardbelegung vorsehen.

Innerhalb eines Projektes zur Modellierung werden nicht alle definierten Parameter benutzt werden. Daher ist eine weitere Aufteilung der Parameter nach ihren Abhängigkeiten an dieser Stelle nicht sinnvoll. Die Abhängigkeiten, die neben der Organabhängigkeit festgelegt sind, werden innerhalb eines Projektes ausgenutzt, um gleichartige Parameter in je einem Datenobjekt zu sammeln. Damit erhält man in einem Projekt eine Liste von Parametern, die sämtlich Eigenschaften eines Wirkstoffs beschreiben, sowie eine Liste von Parametern, die sämtlich Eigenschaften eines Individuums oder Eigenschaften, die sowohl von Wirkstoff als auch von Indivduum abhängen.

19.1.5 Parameter innerhalb der Projekt-Datenstrukturen

Der Modellierer beschreibt seine gewünschte Kinetik in Form von Differentialgleichungen in Abhängigkeit von Parametern, aber unabhängig vom Wert des Parameters. Bei

der Definition eines Modells ist daher anzugeben, welche Parameter genutzt werden sollen und in welcher Einheit sie genutzt werden sollen.

19.1.6 Implementierung

class VKenngr : public VKennProp // VObject

{

int id;

int depOrgan; //einzelgröße oder verteilte größe (pro Organ)

int depMix; //Abhängigkeit von wirkstoff, individuum, beidem oder keinem

double computeValue; //für das Organ, welches aktuell ist, eingetragener Wert

bool computeValueSet; //ist gesetzt oder nicht

AnsiString unitTypeStr; //Einheitentyp

AnsiString modelUnit; //eine Kenngr/Parameter im Modell hat eine Einheit, in der dieser Wert genutzt wird,

// die Einheiten aus den

// organabh. subkgs müssen entsprechend berücksichtigt werden

SubKGList* subList; //jedes element hat denselben !! Einheitentyp; pro Organ ein eigener Wert

int favorite;

}

class VKenngrList : public EListCit<VKenngr>;

19.2 Modellbasis

Definition 38 *Als Modell bezeichnen wir die differentielle Beschreibung eines der von uns zugelassenen (wirkstoff)konzentrationsändernden Prozesse Flow, Transfer, Bindung oder Metabolismus innerhalb der von uns gewählten kleinsten Betrachtungseinheit Subkompartiment. Ein Modell wird in Abhängigkeit von Konzentrationen und von Parametern aus der Parameterbasis beschrieben, und gibt einen Wert in einer definierten Einheit zurück. Ein Modell hat einen Typ (Flow, Transfer, Bindung,Metabolismus), der aber im wesentlichen nur zur Benutzerführung genutzt wird. Die Konzentration eines Wirkstoffs wird über einen festgelegten Bezeichner angesprochen.*

Ein Modell gibt über einen integer-Wert an, wieviele Wirkstoffe es behandelt.

Definition 39 *Ein 1-compound-Modell ist ein Modell, welches die Konzentrationsänderung genau eines Wirkstoffes beschreibt; in dieses Modell kann die aktuelle Konzentration dieses Wirkstoffes sowie (im Falle eines Transfermodells) die aktuelle Konzentration dieses Wirkstoffes in einem benachbarten Subkompartiments übergeben werden. Ein 1-compound-Modell wird für jeden Wirkstoff ausgewertet, der zur Simulation ausgewählt wird.*

Definition 40 *Ein n-compound-Modell ist ein Modell, welches die Konzentrationsänderungen von n Wirkstoffen beschreibt. In dieses Modell werden die Konzentrationen von n der ausgewählten Wirkstoffe übergeben. Das Modell gibt über n Rückgabewerte die entsprechenden Konzentrationsänderungen zurück. Ein n-compound-Modell wird innerhalb einer Simulation unabhängig von der Anzahl ausgewählter Wirkstoffe nur einmal (pro Integrationsschritt) ausgewertet.*

Wird ein n-compound-Modell benutzt, so müssen mindestens n Wirkstoffe zur Simulation ausgewählt sein.

Ein Modell gibt bekannt, welche Parameter es auswerten will und in welcher Einheit diese Parameter auszuwerten sind. Dazu muss es eine Liste von Parametern geben, die ausschliesslich aus Elementen der Parameterbasis bestehen darf und bei der jedes Element eine ausgewählte Einheit besitzt.

Die Modelle, die zur Modellierung eingesetzt werden, sind unabhängig von bestimmten Wirkstoffen und unabhängig von bestimmten Parameterwerten zu definieren. Daher sind sie als ebenso eindeutig anzusehen wie die Einheitentypen des Einheitensystems und werden in einer offenen Liste gesammelt und für alle Modellierprojekte zur Verfügung gestellt.

Definition 41 *Die Liste von Modellen, die definiert werden, bildet die Modellbasis.*

19.2.1 Mappings

Wird ein n-compound-Modell benutzt, so muss es eine Zuordnung zwischen im Modell genutzten Bezeichnern für die Konzentration eines bestimmten Wirkstoffs (z.Bsp. „Comp1") und dem Wirkstoffnamen des tatsächlich ausgewählten Wirkstoffs geben. Diese Zuordnung kann erst nach Fertigstellung eines Simulationsobjektes erfolgen.

19.2.2 Implementierung

```
class Map : public MObject

{

AnsiString name; //Comp1, Comp2 ect.

AnsiString refName; //verwaltet den Namen des compound
```

VComponent* reference;

}

class Mappings : public EListCit<Map>

class VModel : public VObject

{

AnsiString descr; //Interpretertext, wird an rule übergeben

VComputationalRule* rule; //verkapseltes Objekt zur Auswertung eines skriptes

AnsiString name, type, options, comment;

AnsiString xAxis, yAxis; //für beschriftung in libvariablen

VUnitType* requUnitType; //Einheitentyp

VKenngrList* reqs; //Liste von Parameter, die benutzt werden

int noTreatedCompounds; //diese sollten korrespondieren

Mappings* compNames; //diese sollten korrespondieren

VComponents* addCompounds; //im Metabolism-Modell (oder auch andere) können zusätzliche Stoffe gerechnet werden

AnsiString compSel; //für libvariablen

AnsiString organSel; //für libvariablen

AnsiString phaseSel; //für libvariablen

Histories* histories; //Historykurve für eine Libvariable

public:

void DistributeRequirements(VKenngrList* list, char* organName, VComponent* compPt); //

void ActualizeKGsWithName(char* organName);

void AdaptModel(VModel* modIn);

void AdaptMappings(Mappings* map);

void ActualizeMapping(char* compName);

Mappings* GetMappings() {return compNames;}

bool ConnectData(VComponents* compList, DString& errMsg);

bool CreateAddSolComps(VSimulation* sim, VOrgan* organ, VPhaseDescription* p, EListCit<VSolComp>* solComps, CompMap* compMap, int& counter);

void CreateAddSolPerComp(VSimulation* sim, VComponent* mainCompound, VComponent* ompound,VOrgan* organ, VPhaseDescription* p, EListCit<VSolComp>* solComps, CompMap* compMap, int& counter);

bool CheckMapping(int noComps);

bool MapArgs(list<VComponent*>* comps, IntVec& index);

bool Mapped() {return true;}

void PrepareRule(VComputationalRule* rule, char* text, list<VComponent*>* simComps, bool topPhase, bool typ);

}

typedef EListCit<VModel> VModels;

class VPackage : public MObject //Modellbasis

{

AnsiString name, nameSpace;

AnsiString ownFileName;

VModels* models;

}

19.2.3 Additional compounds

Innerhalb eines Modells können (lokal) ODEs für weitere Substanzen mitgerechnet werden. Für die Modellierung sinnvoll ist es, wenn diese ODEs mit den Differentialgleichungen für die Wirkstoffe verkoppelt sind. Diese Verkopplung wird durch das Skript abgedeckt (siehe AnsiString descr). Damit diese Substanzen als Systemgrößen erkannt und behandelt werden können, werden sie wie ein Wirkstoff über das Datenobjekt VComponent definiert und in der entsprechenden offenen Liste VComponents gesammelt.

19.3 Organtypen

Kompartimentabhängige Parameter müssen für jedes Kompartiment, innerhalb dessen sie ausgewertet werden, einen eigenen Wert liefern. Jeder Eintrag in der Liste von Werten (siehe SubKGList unter Parameter) ist durch den Namen des Kompartiments und den zugehörigen Wert festgelegt. Jeder Modellierer muss vorab festlegen, welche Kompartimente er betrachten will. Da dies im allgemeinen Organe sein werden und

diese Bezeichnung sehr anschaulich ist, sprechen wir im weiteren von Organen und Organtypen, obwohl es sich auch um feinere Kompartimente oder zusammengeschlossene Verbände (Cluster) von Organen handeln kann.

Definition 42 *Die Organtypenbasis ist die Liste von Kompartimentbezeichnern, die der Modellierer im folgenden untersuchen möchte.*

Jedes Element dieser Liste besitzt als wesentliche Eigenschaft einen eindeutigen Namen, unter dem es wiedererkennbar ist. Wir haben die Organtypenliste nicht als einfache String-liste implementiert, sondern als komplette VOrgans, so dass jedes Element bereits als ein VOrgan mit entsprechenden Eigenschaften vorliegt. Zur Implementierung der VOrgans siehe 19.4.6 . Ein neuer organabhängiger Parameter wird auf Basis der Organtypenliste angelegt, d.h. seine SubKG-Liste erhält entsprechend viele Einträge. Wird die Organtypenliste erweitert, so muss jeder zuvor angelegte Parameter entsprechend erweitert werden.

19.4 Projekte

19.4.1 Datenbank Individuum

Ein Individuum ist softwaretechnisch wie ein Wirkstoff als eine Datenbank zur Sammlung von Parametern anzusehen. Daher wird die genauere Beschreibung unter 19.4.2 zu finden sein. Das Datenobjekt ist abgeleitet von VComponent und hat zusätzliche Eigenschaften, die aber nur beschreibend wirken.

Implementierung class VIndividual : public VComponent

{

int age;

double weight;

double height;

int gender; //geschlecht

int race;

}

19.4.2 Datenbank Wirkstoff

Ein Wirkstoff kann im Wesentlichen als Liste von Parametern betrachtet werden. Da diese Datenstruktur auch für systembiologische Größen benutzt wird, enthält sie

Eigenschaften wie participatesInFlow oder createPerCompound, um festzuhalten, ob diese Größen auch am Flow teilnehmen oder ob sie vervielfältigt werden müssen.

Ein Wirkstoff muss seine Molmasse bekanntgeben, damit Umrechnungen der Konzentration von einer Massendarstellung in eine Molendarstellung möglich ist.

Ein Wirkstoff wird dosiert. Diese Dosierung wird unabhängig von einer tatsächlichen Auswahl eines Wirkstoffs formuliert, da sie nur den Zugabemodus einer beliebigen Substanz beschreibt. Die Zuordnung zu einem Wirkstoff muss daher noch erfolgen und wird über die Kopie einer im Projekt definierten Dosierung in ein Wirkstoffobjekt erreicht. Es können beliebig viele Dosierungen an einen Wirkstoff übergeben werden. VDosings ist eine offene Liste.

Implementierung class VComponent : public VGroupMember, public VCompInterface

{

private:

int id;

MString name;

AnsiString shortName;

bool participatesInFlow; //nur für addCompounds wichtig

bool createPerCompound; //nur für addCompounds wichtig

AnsiString initialName; //damit kann der ursprüngliche Name im Simobjekt erhalten bleiben, während er überschrieben wird

AnsiString aliasName;

AnsiString trivialName;

AnsiString iupacName;

MString formula;

MString casNo;

AnsiString comment;

VKenngrList* kenngrs; //enthält die Kenngrößen/Parameter

double mw; //Molmasse

AnsiString mwUnit; //Einheit molmasse

VMixParameter* mixPar; //dies wird erst in einem simulationsobject bestückt

VDosings* dosings; //dies wird erst in einem simulationsobject bestückt

AnsiString mappedAs;

AnsiString imageFileName;

//

VMassBal* mb; //Massenbilanz

}

Ein General parameter set ist ebenso ein VComponent-Objekt wie ein compound.

19.4.3 Datenbank Mischparameter

Ebenso wie ein Wirkstoff ist auch ein Mischparameter (set) eine Liste von Parametern, deren Eigenschaft es ist, in ihrem Wert sowohl von einem speziellen Wirkstoff als auch von einem speziellen Individuum abzuhängen. Diese Abhängigkeit wird durch die Namen realisiert, die als AnsiStrings in einem Mixparameter objekt gespeichert sind.

class VMixParameter : public VObject

{

AnsiString name, comment;

int id;

AnsiString s1, s2; //Stoff - Individuum

VKenngrList* constProps; //Mixparameter als Konstante implementiert

}

19.4.4 Organe/Kompartimente

class VOrgan : public VObject

{

private:

//grundsätzliche Eigenschaften zum wiedererkennen:

int typeProperty; // (Arterie, Vene, normales Organ, Cluster...)

MString propName; //dies ist der eigene Name, der zu dem OrganType gehört

MString abbreviation; //dies ist der Name, mit dessen Hilfe die Parameter erkannt werden! ==Abbreviaition

MString name; //dies ist der eigene Name, mit dessen Hilfe Organe gleichen Typs unterschieden werden können

//Muskelaussenschicht / Muskelinnenschicht als Beispiel

//

VOrganTopology* topology;

MString importedFrom;

int inOutFlow; //vene rein/raus, arterie rein/raus...

PhaseList* modelsInPhases;

AnsiString comment;

double QPuffer, VPuffer; //

}

19.4.5 Organ-Topologien

Implementierung class VOrganTopology : public MObject

{

MString ownName; //für welches Organ bin ich?

int typeFlow; // 1:1, 1:n, n:1, n:m beschreibt Blutfluss durch dieses Organ

EListCit<MString>* flowIn; //dies sind die Namen (!) der Organe, von denen dieses organ bekommt

EListCit<MString>* flowOut; //dies sind die Namen (!) der Organe, denen dieses organ weitergibt

public:...

}

19.4.6 FullBodyTemplates

Implementierung class VOrgans : public VObject

{

private:

VOrgansList* organsList;

VVolumeTreatment* volTreat;

MString name;

int typeBloodModel;

bool isAdded;

bool useStandard;

AnsiString comment;

public:..

}

19.4.7 Dosierungen

Implementierung class VDosing : public VObject

{

MString name;

int type; //function oder impulse

MString organ;

MString phase;

bool isAdded;

double startTime;

double period; //Abstand zwischen zwei Dosierungen

double periodNo; //Anzahl Dosierungen

VIntervals* intervals;

AnsiString iniConcName;

VModel *model;

public:..

}

19.4.8 Simulationsobjekte

1. das FullBodyTemplate wird als ganzes Objekt in das Simulationsobjekt kopiert; damit ist gewährleistet, dass auch Modelle, die als Kopie im FBT vorliegen, in einer bestimmten Definition im Simulationsobjekt erhalten bleiben. Rechnungen bleiben dadurch reproduzierbar.

2. die Individuen und Compounds werden als ganze Objekte in das Simulationsobjekt kopiert. Rechnungen bleiben dadurch reproduzierbar.

3. Es gibt Funktionen, die den schnellen Austausch der Daten ermöglichen: wird ein Individuum(sparameter)/Compound(parameter) im Simulationsobjekt geändert, so muss diese Änderung auch zurück in das Individuum des Projektes (von dem es ja eine Kopie ist) möglich gemacht werden.

Ein neues Simulationsobjekt wird gefüllt.

19.4.9 Projekte

Da es sich hier um ein ausschliesslich verwaltungstechnisches Objekt handelt, verweisen wir auf den Code des Programms.

19.4.10 Bibliothek

Da es sich hier um ein verwaltungstechnisches Objekt, verknüpft mit bereits beschriebenen Strukturen (Modelle) handelt, verweisen wir auf den Code des Programms.