

# GRIP developer documentation v 0.01

Mathijs Vogelzang

February 12, 2007

# Contents

<b>1</b>	<b>Introduction</b>	<b>3</b>
1.1	About GRIP . . . . .	3
1.2	Technology used . . . . .	3
1.3	Development process . . . . .	4
<b>2</b>	<b>Implementation notes</b>	<b>5</b>
2.1	Precautions . . . . .	5
2.2	Implementation tips . . . . .	5
<b>3</b>	<b>Overview of the code</b>	<b>7</b>
3.1	Package overview . . . . .	7
3.2	Multithreading . . . . .	7
<b>4</b>	<b>Communication with the hospital information system (HIS)</b>	<b>9</b>
4.1	Polling the HIS . . . . .	9
4.2	Processing of patient location changes . . . . .	9
<b>5</b>	<b>Storage of data by GRIP</b>	<b>11</b>
5.1	Database layout . . . . .	11
<b>6</b>	<b>Graphical user interface (GUI)</b>	<b>12</b>
<b>7</b>	<b>Recommendation algorithms</b>	<b>13</b>
7.1	Linear interpolator . . . . .	13
<b>8</b>	<b>Additional utilities</b>	<b>14</b>
8.1	Backup script . . . . .	14
8.2	Report generator . . . . .	14

# List of Tables

3.1	Overview of packages in the GRIP source tree . . . . .	8
4.1	Handling of different cases according to admission status and HIS status	10
5.1	Overview of tables in the GRIP database . . . . .	11

# Chapter 1

## Introduction

This chapter provides an introduction to the GRIP system.

### 1.1 About GRIP

GRIP (Glucose Regulation for ICU Patients) is a Clinical Decision Support System (CDSS) designed to give recommendations on insulin pump rates for intensive care unit patients. It was developed starting from 2002 in the University Medical Center, University of Groningen, the Netherlands.

After a period of development, GRIP was implemented in the surgical ICU of the UMCG in December 2004. Initial results were very promising and a paper describing the implementation was published in the open access journal BMC Medical Informatics and Decision Making. The article can be accessed at <http://www.biomedcentral.com/1472-6947/5/38>.

At the time of writing this introduction (January 2007), GRIP is being used at three intensive care units. Source code will shortly be accessible at <http://grip-glucose.sf.net/> under a free-software license.

### 1.2 Technology used

GRIP was built with the possibility to release it as free software in mind. During development, Linux was used as development platform, and therefore GRIP is capable of running on a computer using only free software. For conformance with the standard hospital IT infrastructure, our production computers are running Microsoft Windows XP, however.

GRIP is built using the Java programming language developed by Sun. Code to run and develop java programs is freely available from <http://java.sun.com/>. The version used to develop and run GRIP was JDK 5.0, but it should also run fine on later versions.

The GRIP graphical user interface is built using the Java Swing user interface. For many GUI components, the tableLayout library (<https://tablelayout.dev.java.net/>) was used as layout manager. In our production environment, we have used the

JGoodies Looks pluggable Look-And-Feel to let the user interface look more like standard Windows XP applications. The JGoodies Looks library is available from <http://www.jgoodies.com/downloads/libraries.html>.

Other libraries used by GRIP are:

- The Apache Xerces XML parser (<http://xerces.apache.org/xerces2-j/>) for serialization to and from XML.
- The jakarta commons-lang package (<http://jakarta.apache.org/commons/lang/>) for some date and time functions.
- The MySQL connector/J package (<http://www.mysql.com/products/connector-j/>) to connect to a MySQL database.
- The apache log4j package (<http://logging.apache.org/log4j/docs/>) for flexible logging of warnings and errors.

### 1.3 Development process

During its development, a source tree of GRIP was kept in a version control system (first **CVS** and later **SVN**). During production, new releases were always saved under a new filename, so that in case of new errors, we could always revert back to an older version.

Development was done on a Linux (**Ubuntu 6.06**) machine, using the **Eclipse** java IDE. We used the Eclipse **visual editor** for development of some of the GUI components.

## Chapter 2

# Implementation notes

### 2.1 Precautions

*Implementing GRIP is no easy process. It should be well thought-out, and appropriate measures should be taken to minimize the chance of accidents or failure.*

Even though implementation went smooth in our center, we cannot make any guarantees about the workings of GRIP, and the UMCG is not in any way responsible for GRIP. The sole persons responsible for anything that happens before, during, or after implementation of GRIP are the ones implementing and using GRIP.

Legally, by downloading and using GRIP, the users are bound to the Gnu Public License, under which GRIP is distributed. This license also explicitly states that the makers of the software provide it "as is", without any warranty or guarantee on fitness for a specific purpose. The user of the software takes complete responsibility for any risks and damages related to the program.

### 2.2 Implementation tips

In the UMCG, we took a careful approach in implementing GRIP. It is very important that technical problems can be solved on a 24/7 basis. In the initial implementation period, there should be frequent moments during which users can ask questions about the program.

At the first ICU where we implemented GRIP, we had a three month period in which GRIP's recommendations were not followed, only data entry was performed. Although the algorithm was designed to be conservative and was based on a review of protocols from the literature, we regularly compared the recommendations of GRIP to the actual amount of administered insulin. We also assessed whether the recommended time to wait until the next glucose measurement was safe.

The decision that GRIP's recommendations were safe, and likely to give an improvement in glucose control was made by the full medical staff of the ICU. From that moment, we began implementing the actual use of GRIP recommendations bed-by-bed. The nurses were instructed to ask a physician for confirmation when in doubt

whether a recommendation was logical. They were also instructed to feel free to measure the blood glucose at an earlier time point when they considered the recommended interval to be too long.

In the initial implementation phase, the programmer of GRIP visited the ICU daily to see whether there were questions about the program and if things were running smoothly. Additionally, the cellphone number of the programmer was on the monitor. (During the three year period GRIP has been running, I have been called ~20 times outside of office hours, of which I only had to go to the hospital 3 times to solve the problem).

During subsequent implementations at other ICUs in our hospital, we took the same approach, with a much shorter run-in time, because we knew the combination of our hospital information system and GRIP was stable and reliable. Two to four nurses of the new ICU were explained how the program works and appointed as 'superuser' of GRIP for the new ICU. During the course of 2-3 weeks, they taught their colleagues how to use GRIP, and after that period, implementation was quickly started, again increasing the number of beds for which GRIP's recommendation was used each day.

## Chapter 3

# Overview of the code

This chapter gives an overview over the high-level layout of the code

### 3.1 Package overview

The overview of source packages is shown in Table 3.1.

### 3.2 Multithreading

GRIP is multithreaded. In a multithreading application like GRIP, correct locking is of paramount importance. Wrong locking may either result in data loss or deadlocks. When a thread wants to query what patients are admitted, or make changes, it should acquire the lock of the ICU. When a thread wants to read or alter any information within a Patient object it should acquire the lock for that Patient. The Events, Lab or ID data don't have separate locks, the Patient lock counts as the general lock for all these data. When different threads acquire locks in different order, deadlock may result. Therefore, when reading or making changes to both the ICU and Patient objects, the ICU lock should ALWAYS be acquired first, and then the Patient objects. If multiple patient locks should be acquired, they should be acquired in order of their respective patient ID. Because the change listener linking is done in the opposite direction of the locking (i.e., when a Patient's fireChange() method is called, it is propagated up to the ICU's fireChange method()), the fireChange() method may NEVER be called within a locked section, because a changelister of the ICU may then likely acquire the ICU lock after a patient lock has already been acquired.

In the GRIP code, patient methods actively check whether the corresponding lock is acquired when the variable ICU.LOCKCHECKING is set to true (default).

<b>Package</b>	<b>Description</b>
nl.umcg.grip.main	The main classes needed for running GRIP.
nl.umcg.grip.models	Some GRIP-specific data models, such as a model for recommendations generated by GRIP. Most other data models are contained in nl.umcg.models.
nl.umcg.grip.gui	The classes making up the graphical user interface of GRIP.
nl.umcg.grip.process	Classes implementing the 'business-logic'.
nl.umcg.grip.storage	Classes for the persistency of GRIP, both abstract interfaces and concrete implementations.
nl.umcg.his	Classes defining the interface between GRIP and the hospital information system (HIS). (HIS is named "ZIS" in a number of places, as this is the Dutch name).
nl.umcg.grip.recommendation	The classes implementing the generation of recommendations GRIP makes.
nl.umcg.util	A number of utility classes.
nl.umcg.tests	A number of tests of the code, mostly JUnit testcases.

Table 3.1: Overview of packages in the GRIP source tree

## Chapter 4

# Communication with the hospital information system (HIS)

This chapter gives details on how GRIP communicates with the HIS.

### 4.1 Polling the HIS

Periodically, or on the user's request, a synchronization with the HIS (Hospital Information System) is performed. Apart from providing glucose and other laboratory measurements, the HIS determines which patients are admitted to the unit. GRIP knows what values to treat as glucose values from the "labvalues.txt" configuration file. The labvalues.txt also allows to define a number of laboratory parameters to retain in GRIP's database, for instance for inclusion in future control algorithms. Only the glucose and potassium values are asked to be validated by the user.

### 4.2 Processing of patient location changes

When a patient is leaving the unit, and according to the HIS the patient is gone, GRIP still retains the patient as being admitted to allow the user to indicate where the patient has gone. These patients are included in the Department object as "former" patients, and can only be removed by user action. This gives 6 different possibilities for the status of a patient when the query to the HIS is complete, as tabulated in Table 4.1.

<b>Current status</b>	<b>Presence in query result</b>	<b>Explanation</b>
Admitted	TRUE	Patient was admitted, and remains so.
Admitted	FALSE	Patient was admitted, is gone now, and so will become "former" patient.
Former	TRUE	Patient is still a former patient, but also gets a new admission. It will thus have two open admissions.
Former	FALSE	Patient remains a former patient.
Not admitted	TRUE	Patient was not admitted, and is now. The patient should become an admitted patient.

Table 4.1: Handling of different cases according to admission status and HIS status

## Chapter 5

# Storage of data by GRIP

This chapter discusses briefly how GRIP stores the data it gathers.

### 5.1 Database layout

GRIP stores its information in a MySQL database. 6 tables are primarily used, with names which currently are still Dutch. Table 5.1 lists the names and function of each database table.

Table name (Dutch)	Translation	Contents
patienten	patients	General patient identification information (i.e., name, birthdate, etc.)
opnames	admissions	Information on admissions
events	events	Events (either program generated or input by the user)
labdata	laboratory data	Raw laboratory data, and whether measurements have been accepted or not
foutmeldingen	error messages	Error/warning messages logged by log4j
verpleegkundigen	nurses	The names of nurses

Table 5.1: Overview of tables in the GRIP database

## **Chapter 6**

# **Graphical user interface (GUI)**

This chapter gives an overview how the graphical user interface is constructed. Configuration of the user interface is mainly controlled by a number of configuration files in the configuration directory. Things the nurse should fill out when a patient is either admitted or re-admitted, or when a new measurement result comes in are specified in `varpages.properties`. The definition of how variables are represented when being asked from the user is defined in `variables.properties`.

## Chapter 7

# Recommendation algorithms

This chapter will describe the recommendation algorithm of GRIP in more detail.  
[TODO]

### 7.1 Linear interpolator

In the `KaliumRecommender` and the measurement-interval calculation part of the recommendation generator code, linear interpolation is extensively used. The `nl.umcg.util.LinearInterpolator` class allows an easy mapping to be defined by two strings, representing x and y values. Queries can then be made, and an y value is returned by piecewise-linearly interpolating y values.

## Chapter 8

# Additional utilities

This chapter describes additional GRIP code.

### 8.1 Backup script

In our production environment, we have configured a simple cron job on a separate computer that does a mysqldump once daily of all computers with GRIP installed on it. As all data is stored in the MySQL database, this assures a minimization of data loss in case of a system failure.

### 8.2 Report generator

We will later release a simple report generator. This program produces a series of HTML pages that show glucose control of all currently admitted patients for the past 3 days.