

Complex Project Manager

Manual

Version: 0.1
Author: W.Wagner
Web: cpmanager.sourceforge.net

Content

1. Introduction.....	3
1.1. Overview.....	3
1.2. Simulation?.....	3
2. Basics.....	5
2.1. Project modelling.....	5
2.1.1. Elements of a project model.....	5
2.1.2. Effort Action constraints.....	6
2.1.3. Linking Elements.....	6
2.2. Project flow simulation.....	8
2.2.1. Introduction.....	8
2.2.2. Discrete simulation / Steady intervalls.....	8
2.2.3. Rollbacks.....	9
2.2.4. The simulation.....	9
2.2.5. Starting in the middle.....	10
3. The Engine - technical.....	11
3.1. Project modelling network.....	11
3.1.1. Project Structure Classes.....	11
3.1.2. Resources.....	11
3.2. Usefull helpers.....	11
3.2.1. Date and Time handling.....	11
3.2.2. Calendars, Calendar entries and Generics.....	12
3.2.3. Debug.....	12
3.3. The engine.....	12
3.4. Tests.....	13
3.5. Sample Project.....	14
4. Designing a Front End.....	20
4.1. General requirements.....	20
4.2. Components.....	20
4.2.1. Views.....	21
4.2.2. Environment.....	21
4.2.3. Editors / Modules.....	21
4.2.4. User Roles.....	21
4.3. Project Generation / Editing.....	22
4.4. Simulator integration.....	22
4.5. Data structure.....	23

1.Introduction

1.1.Overview

This project creates an engine that allows you to simulate the project flow of one or many projects.

This engine is meant to be used as a back end of an open source project management tool. It surely will not implement everything you can think of, because the complexity can quickly grow into unimplementability. It focuses more to the real live situations and tries to keep it as simple as possible.

The engine implements the project elements:

- Project
- Project Phase
- Action (the real work that is to do)
- Milestone

With them you are able to model most project flows.

An Action can be defined to use DURATION or EFFORT(s) to calculate its end. If it is defined by effort, the engine takes in respect the concurrencies for human and technical resources between different parallel actions.

As in real life there are many restrictions to ones work time or the availability of technical resources. To reflect this issue the engine supports calendars. With their help, the engine can respect weekends, holidays, vacations, regular work breaks and special days ('Friday, I work until 1 P.M.')...

You see there are many things making this project complex. If it weren't there would exist many project management tools already.

1.2.Simulation?

Most project planning tools provide nice Gantt diagrams and allow you to store much information about your project. You define begin and end of projects periods (Phases, Actions etc) and get a nice visualization of your definitions.

These tools can be helpfull without doubt. Most helpful they are if you use them to document your project flow, means, if you store how things went in your project.

The accuracy of your predictions for the future depends on the ability of the user to guess the future. Especially if you have many, complex and dependent projects it is hard to guess how things will happen and we should not speak about 'accuracy' anymore.

This project was born out of an urgent need for realistic statements about the future project flow. Instead of defining the start and end dates of project elements it should be possible to define the flow (predecessors, successors) and structure of a project.

A simulation engine will work on these project flow definitions and deliver the information about begin and end date of the project elements as well as a work plan for each resource. This work plan is necessary to make sure that all resources are available when a cooperative work needs to be done.

After the simulation the results may be displayed in nice Gantt charts.

But what's about the accuracy of this simulation engine? To use it an organisation must accept some issues:

(1) Work times

As we told above, this simulation engine supports the definition of all kind of individual work times. Individual work day begins and ends, breaks, vacations and holidays can be maintained and will be taken into respect.

If you are working in a field that requires a high amount of cooperative work it should be clear that a loose working day with less hours in which all resources are available makes it hard to gain progress in a project.

(2) Effort predictions

The real work is done in project elements called Actions. An Action can be defined by Durance or Effort.

A *durance* definition is good for special tasks where the needed time is constant, for instance a shipping time, or an external work with guaranteed delivery in time. Duration is defined in real time. Scheduler will not take respect to calendars.

An *effort* definition reflects the effort needed to gain the progress to finish an action. It is a guess like you guess the effort in other project management tools. The accuracy of the simulation depends on the accuracy of this effort. In general it is better to define a higher effort than necessary in best case. The best case should be about 60% of what you define.

(3) Acceptance of the WorkPlans

The simulation creates work plans for each resource. This work must be accepted by everybody in the company. The plans ensure that all necessary resources are available when the action is to be done.

So management has to care about, that these plans are mandatory.

(4) Accept the truth

Using a project simulation engine correctly, will reduce the cases in which managers declare phantastic due dates. The engine will show them when it may be ready if **all assumptions and effort guesses are true**. An unavailable resource or extraordinary delay in one of the projects may make the results invalid.

So please don't blame us if your project misses the result of a simulation run. It stays a simulation without electrical power or machine failures, sick people and huricanes.

A good idea is to take the simulation result as best case and publish dates basing on your experience and on your own responsibility. It is a tool not magic.

2. Basics

Let us talk about the basic assumptions behind the simulation engine. The logical way starts with the modelling of the projects and goes to the simulation engine...

2.1. Project modelling

2.1.1. Elements of a project model

A project model can be designed out of four elements which are basically 'periods in time' (have begin and end) and a lot in common:

=> Project, Project phase, Action, Milestone

2.1.1.1. Project

The *Project* element is a container for all other elements. As a container its end is defined by its content.

A project has a priority. This priority is valid for all contained elements.

2.1.1.2. Project phase

The *Project phase* element is a container for Actions and Milestones and is contained in a project element. It is usually used to structure the project.

As a container its end is defined by its content.

2.1.1.3. Actions

An *Action element* reflects the real work. It can be defined either by duration or by Effort.

Actions defined by Duration are meant to reflect real world situations that need a defined time, without respect to resources, workdays etc. For instance: Shipping time of a product, external work with guaranteed delivery in time. In this case End date can be calculated directly out of start date.

Actions defined by Effort are the object of our simulation. For flexibility each action can have more than one effort assigned. Each Effort needs at least one resource which does the work.

The work referenced by an Effort needs to be done by all assigned resources together, so all of them must be available at the same time.

Examples:

- (1) Four guys have to carry a piano: one effort and 4 resources
- (2) Two guys have to move two chairs: two efforts and 2 resources

2.1.1.4. Milestones

Milestones are special periods in time without a duration. So begin and end are equal. They are used to track the project progress.

2.1.2. Effort Action constraints

Constraints for can be defined to reflect real live scenarios:

MinWorkPerEntry: Defines the minimum length of one calendar entry. In most cases it makes no sense to switch between tasks frequently. To make progress you need a minimum work time. Default is 2 hours.

Note: This value should be less than the minimum work time between defined breaks. Otherwise this Action will never be done.

ExclusiveOnDay: This action is done solely on a day. It will be started when it has the highest importance for all cooperating resources. Nothing else will be planned on that day.

This feature can be used for work like being on the road, unavailable for all other tasks.

ExclusiveOnWork: This action is done solely until it is done. It will be started when it has the highest importance for all cooperating resources. Nothing else will be planned on these days.

This feature can be used for work like being on the road for many days, unavailable for all other tasks.

2.1.3.Linking Elements

Each of the objects above can have multiple predecessors and successors. So you can easily model even complex real live scenarios.

Start date

The start date defines itself out of the latest end date of its predecessors.

Special case Project: If a project is not linked to another project it may have a start date defined. If there is no start date defined it is assumed to start immediately.

Container

A contained element can be added at begin, at the end or between existing contained elements.

If it is added at the begin its start date is defined by its containers start date. (In this case it has no predecessors.)

If it is added at the end its end date is one of the dates that define the containers end date. (In this case it has no successors.)

All these things are automatically calculated by the project modelling network.

Creating and configuring the elements and linking them together (set predecessor or successor, add to container) is all that needs to be done.

2.2. Project flow simulation

2.2.1. Introduction

The main problem with the simulation was the reduction of the complexity. Before the simulation could start many small problems had to be solved, for instance:

- Date/Time calculations (class `ProjectDate`)
- Definition of Workday and special days (classes `GenericSpecialDay`, `GenericCalendarEntry`, `Calendar` classes)
- the project modelling network (package `cpm.core`)

All these classes encapsulate functionality and reduce it to a understandable minimum.

Another basic concept was to use roles. Class `ProjectRepository` cares for instance about the accesses and maintenance of Project Modelling Network. Class `WorkSimulator` behaves like a moderator coordinating the project flow simulation. It gets all active Actions from project repository and lets the `ResourceAttourneys` (objects that 'speak' for the resources) negotiate when the actions are to be done.

2.2.2. Discrete simulation / Steady intervalls

A project flow contains many discrete points: Everytime a new period starts or ends there is a change in which periods are active. (Periods can be done, active and start in future)

Target of the simulation is to distribute the available work time between the efforts in respect to the importance of these efforts. (Note: We distinguish between importance and priority!)

As the simulation needs to respect the discrete points we define *steady intervalls*, in which there is no discrete point. So a steady intervall has a constant amount of active efforts and the simulator may freely plan the calendar items within it.

The project repository delivers the list of active efforts ordered by *importance*.

Importance: Messure to compare efforts by (1) priority, (2) count of cooperating resources and (3) amount of progress left.

So an effort with higher priority comes first, within the same priority, the effort with higher amount of cooperating resources and within the same priority and same amount of coop resources, the one that has left to do to be finished.

Explanation: (2) is important because many resources make it hard to find common work time, and (3) is a pragmatic way to reduce the amount of open efforts. (Finish what is almost ready...)

Begin of a steady intervall is always the last calculated end of a steady intervall. The intervall ends when the first active effort is brought or another project with defined start date starts. But this is the result of the simulation and so an egg or hen problem.

The problem is solved by using end of the best case scenario, which is in fact the shortest length of the interval: We simply add the lowest amount of work to do to the begin of the interval.

Result is a short steady interval which is not necessarily the time between two points of change in the project structure.

2.2.3. Rollbacks

Short intervals

In complex projects it may be that steady intervals become too short to be planned. For instance you may get the ends of two parallel but independent actions within 5 minutes.

To get a realistic workday and avoid fragmentation we define a minimum length of the interval of one day. So we ignore the calculated end of the steady interval at first.

After that we continue with the calculated end of the interval. If the active effort list has changed, it may contain entries with higher importance as before. When the ResourceAttorneys find an existing calendar entry that has lower importance than the new effort, a *RollbackException* is fired.

A *Rollback* is done by going back in history to the begin of the steady interval in which the conflict occurred. Then it is being simulated again. This time with knowledge that an action with higher importance will start at the conflict date.

In next cycle there will be no conflict anymore and the simulator continues.

2.2.4. The simulation

`WorkSimulator` represents the moderator in the negotiations between the `ResourceAttorneys`.

In each steady interval:

The simulator contacts `ProjectRepository` and gets an ordered list of active efforts. Then it works on these efforts starting with the one with highest importance: It retrieves the cooperating resources for this effort and asks the attorney of the first resource for a proposal, when this effort can be brought. The proposal is a set of `CalendarEntry` objects.

Then it turns to the next attorney and asks it to check it. The attorney checks the calendar of its resource and reduces the proposal (shrinks single calendar entries/ removes entries if they are too short) accordingly.

When all attorneys have checked it, the result will be booked in the calendars of the resources as well as in the effort.

The next effort is done until no one is left, then the next steady interval is processed.

This process ensures that

- Efforts with higher importance are done first,
- Efforts with lower importance fill up the gaps or wait until there is nothing else to do.

Result are work plans for each resource and the begin and end dates of the project elements.

2.2.5.Starting in the middle

Until now we assumed a project repository simulated from begin using a ReferenceDate or the current time as begin.

If we think on, we will have an application that has a big repository with solved projects, active projects and projects that start later. This application will have done at least one simulation run, the database contains the begin and end dates of former runs.

Our engine supports the reporting of project progress by the project members. They can set the progress percentage of an effort. If the engine starts in the middle it respects this percentage setting to calculate the progress left to do.

The engine is not touching the dates of project elements in the past.

3.The Engine - technical

3.1.Project modelling network

The Project Modelling Network is located in Java package `cpm.core`. Its classes can be divided into two groups: Project structure classes and Resource classes:

3.1.1.Project Structure Classes

The project elements `PeriodProject`, `PeriodPhase`, `PeriodAction` and `PeriodMilestone` are all inheritans of `PeriodInTime`, an abstract class which delivers the common functionality.

The inheritors implement their specific features. For instance: `PeriodPhase` implements the functionality to be a container, `PeriodProject` inherits these extentions and adds the feature to save its priority, `PeriodMilestone` implements the fix duration of zero, so end date is always equals to start date. Finally `PeriodAction` extends the basic funtionality by the features to store and provide Effort support.

The listeners `PeriodDateListener` and `PeriodStructure listener` are interfaces which allow to notify other classes when a `PeriodInTime` inheritant changes. In the network they ensure the different change notifications between the project elements. (For instance: The project is being notified when an element at the end gets an end date. This situation may mean that the end date of the project is known.)

To be able to define the efforts of an `PeriodAction` there is a class `Effort`. Each `Effort` has at least one `HumanResource` that does the work.

The project repository is the storage for the project elements and acts as interface to the simulation engine.

3.1.2.Resources

Even if we don't feel comfortable by doing it =) we have inherited `HumanResource` and `TechResource` from one class. Class `Resource` contains the common functionality.

3.2.Usefull helpers

The package `cpm.util` contains some very usefull helpers.

3.2.1. Date and Time handling

The simulation engine has got its own date handling. It still uses internally some features of `java.util.Date` but works basing on minutes. The class `ProjectDate` provides all basic funtionality like, setting , adding , subtracting and comparing points in time.

It implements the interface `Compareable`. So it can be used in Java Collection API.

3.2.2. Calendars, Calendar entries and Generics

We have implemented two calendar classes which have nothing to do with Java's calendar classes.

`ResourceCalendar` is a class that stores the work days, including all special days like Weekend, Holidays, Fridays as well as the personal breaks.

These definitions are done using the classes `GenericSpecialDay` and `GenericCalendarEntry`. `GenericSpecialDays` are defined by using all kinds of pattern (for instance: every day, every first Monday, every first day in month,...) Please see the code of this class for explanations and examples.

For each `GenericSpecialDay` object you may define `GenericCalendarEntry` objects. These represent busy times for spare time or unavailability times and can be converted into `CalendarEntry` objects (introduced soon).

`ProjectCalendar` works as a store for the personal work plan. It provides functionality to add, remove, retrieve `CalendarEntry` objects and find gaps between them. `CalendarEntry` objects have basically a begin and end and store to which object they belong in `CalendarReference`. `CalendarEntry` class is implementing interface `Comparable` too. (for Java Collection API)

When `ProjectCalendar` searches for gaps in calendar for work to do, it resolves the `GenericSpecialDays` in range to `CalendarEntry` objects, adds the already booked `CalendarEntry` objects for `PeriodActions` and examines the gaps between them.

3.2.3. Debug

The class `debug` implements some methods to output messages depending on debug levels. In tests `Debug.TRACE` may be very usefull. Later `Debug.NORM` is sufficient.

3.3. The engine

The classes of the simulation engine are stored in package `cpm.engine`.

Main point is the class `WorkSimulator` which act as the moderator between the `ResourceAttourney` objects. These objects encapsulate all resource specific features in the simulation whereas `WorkSimulator` controls the central flow. How the simulation works has been explained already in chapter *Project Flow Simulation*.

`PeriodList` is used as a container for the active `cpm.core.PeriodAction` objects. It delivers a `OrderedEffortList` to the engine which contains the efforts of the active actions ordered by their importance. So sorting them is encapsulated in those classes and simulator gets automatically the most important effort first.

Whenever the simulator detects a conflict between an existing `CalendarEntry` with lower importance and a new, more important `Effort` to be planned, the `RollbackException` is being fired.

The classes `OrderedActionList` and `ProjectNavigator` are not used in the moment.

3.4. Tests

The package `cpm.test` contains the `Tester` class and the `SampleProject`. Class `Tester` contains several small projects to test the features of `Complex Project Management engine`.

The `SampleProject` class is explained in next chapter.

3.5. Sample Project

The sample project consists is a heavily commented program which is reproduced here to give you an idea how the simulation engine has to be used by the Front End.

File: cpm.test.SampleProject.java:

```
/*  
 *  
 * CPM - Complex Project Manager  
 *  
 *  
 *  
 * WWW:          cpmanager.sf.net  
 * Project:      www.sf.net/projects/cpmanager  
 *  
 *  
 *  
 * Copyright (C) 2003 CPManager team  
 *  
 * This program is free software; you can redistribute it and/or modify  
 * it under the terms of the GNU General Public License as published by  
 * the Free Software Foundation; either version 2 of the License, or  
 * (at your option) any later version.  
 *  
 * This program is distributed in the hope that it will be useful,  
 * but WITHOUT ANY WARRANTY; without even the implied warranty of  
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the  
 * GNU General Public License for more details.  
 *  
 * You should have received a copy of the GNU General Public License  
 * along with this program; if not, write to the Free Software  
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA  
 *  
 */  
package cpm.test;  
  
import cpm.core.*;  
import cpm.engine.*;  
import cpm.util.*;  
  
import java.util.*;  
  
/**  
 * This class is part of the project documentation. It contains a sample project  
 * to show, how a front end has to use the simulation engine.  
 *  
 * @author Wolfram Wagner (wwagner@users.sf.net)  
 */  
public class SampleProject {
```

```

public static void main(String[] args) {

    sampleProject();
}
/*
 * Project, Phase -----
 * Action.durance ~~~~~
 * Action.effort #####
 *
 *
 * Project 1 *-----*
 * Phase 1 *-----*
 * A1, A2 *~~~~~#####*
 * A3 *#####*
 * Phase 2 * *-----*
 * A4, A5 * *#####~::~*
 */
public static void sampleProject() {
    /*
     At the begin we set the debug level to Debug.TRACE wich will create
     much more output about the simulation. In productivity this line
     should be removed to gain speed.
     */
    Debug.setDebugLevel(Debug.TRACE);

    /*
     PROJECT MODELLING
     =====
     Before we simulate the project we need to model the project flow. Container
     for the projects is ProjectRepository..
     */
    ProjectRepository pRepository = new ProjectRepository();

    // CREATING RESOURCES

    // Now let's create a first HumanResource object for Samual Smith:
    HumanResource sam = new HumanResource("1000", "Samual Smith");
    // Sam is using the build-in default work day (7:00-9:00,9:15-12:00,12:45-16:00)
    // and will have free weekends. This is defined by the arguments of
    // ResourceCalendar.
    ResourceCalendar resCal = new ResourceCalendar(null,true);
    sam.setResourceCalendar(resCal);

    // the second resource Marie will have another workday, at first we create an
    // HumanResource
    // object again...

```

```

HumanResource marie = new HumanResource("1001","Marie");
// now we create a GenericSpecialDay which is her default work day
// note: The string "1/0/*" means that it is valid for every day. This is
// mandatory for DefaultWorkDays.
GenericSpecialDay dwdMarie = new GenericSpecialDay("Workday",
            "Marie's Workday","1/0/*", 0,null, null);
// now we define the times in which marie is not available by creating
// GenericCalendarEntries for these times.
dwdMarie.addCalendarEntry(new GenericCalendarEntry(0,0,8,00,"Morning"));
dwdMarie.addCalendarEntry(new GenericCalendarEntry(12,0,13,00,"Lunch break"));
dwdMarie.addCalendarEntry(new GenericCalendarEntry(17,0,24,00,"Evening"));
// and create a ResourceCalendar using this new default work day. The second
// argument (false) means that marie works on weekends too.
ResourceCalendar resCal2 = new ResourceCalendar(dwdMarie,false);
marie.setResourceCalendar(resCal2);

// we want to use tech resources too. Let us create an object for a car.
TechResource techRes = new TechResource("t1","Car 1");
// Tech resources have another default workday: they are available 24h 7days a
// week (see constructor of TechResource)

// PROJECT CREATION
/*
 * Project 1 *-----*
 * Phase 1   *-----*
 * A1, A2    *~~~~~#####*
 * A3        *#####*
 * ...
 */

// we create a first project with Priority 60
PeriodProject proj1 = new PeriodProject("P01", "Project 1",
            "This is project 1",60);
// defining a start date is not necessary if it starts immediately
ProjectDate pdProlStart = ProjectDate.now();
proj1.setStartDate(pdProlStart);
// each project element needs to be added to the Repository
pRepository.add(proj1);

// Phase 1
// Now let us create a ProjectPhase and add it at the begin of Project 1
// Note: project phases can be omitted by adding actions and milestones directly
// to projects
PeriodPhase ph1_1 = new PeriodPhase("P01-01","Prol-Phase 1","Prol-Phase 1");
pRepository.add(ph1_1);
// This phase will be added to the begin of Project1, so the start dates will be
// linked...

```

```
proj1.addPeriodAtBegin(ph1_1);
```

```

// Action 1
// the first action is defined by Duration.
PeriodAction a1 = new PeriodAction("P01-01-A1", "Pro1-Phase 1-A1",
                                   "Pro1-Phase 1-A1", PeriodInTime.RULE_DURATION);
// it will take 40 hours and 0 minutes:
a1.setDuration(40,0);
pRepository.add(a1);
ph1_1.addPeriodAtBegin(a1);

//Action 2
// the second action is defined by an effort
PeriodAction a2 = new PeriodAction("P01-01-A2", "Pro1-Phase 1-A2",
                                   "Pro1-Phase 1-A2", PeriodInTime.RULE_EFFORT);
// an effort object is being created
Effort e2 = new Effort(a2,"e2","");
// we add sam and the car to it
e2.addHumanResource(sam);
e2.addTechResource(techRes);
// the effort to work is 40 work hours and 0 minutes
e2.setEffort(40,0);
// now we add the effort to the action
a2.addEffort(e2);
pRepository.add(a2);
// now we have to LINK the actions this can be done either by
a2.addPredecessor(a1);
// or a1.addSucessor(a2); which result in the same link
ph1_1.addPeriodAtEnd(a2);

//Action 3
// parallel to action1 and action2 we have another one defined by effort and
// also to be done by Sam
PeriodAction a3 = new PeriodAction("P01-01-A3", "Pro1-Phase 1-A3",
                                   "Pro1-Phase 1-A3", PeriodInTime.RULE_EFFORT);
Effort e3 = new Effort(a3,"e3","");
e3.addHumanResource(sam);
e3.setEffort(80,0); // effort is 80 working hours
a3.addEffort(e3);
pRepository.add(a3);
// because it goes from begin to end of phase1 we have to add it on both
ph1_1.addPeriodAtBegin(a3);
ph1_1.addPeriodAtEnd(a3);

```

```

/*
 * ...
 * Phase 2      *
 * A4, A5      *
 */
//
// phase 2 will follow phase 1, that means when action2 and action3 are done
PeriodPhase ph1_2 = new PeriodPhase("P01-02", "Prol-Phase 2", "Prol-Phase 2");
pRepository.add(ph1_2);
// again we can link elements either by successor or by predecessor
ph1_1.addSucessor(ph1_2); //or ph1_2.addPredecessor(ph1_1);
// the project will end when phase2 is ready, add it at the end:
proj1.addPeriodAtEnd(ph1_2);

//Action 4
PeriodAction a4 = new PeriodAction("P01-02-A4", "Prol-Phase 2-A4",
                                   "Prol-Phase 2-A4", PeriodInTime.RULE_EFFORT);
Effort e4 = new Effort(a4, "e4", "");
e4.addHumanResource(sam);
e4.setEffort(40,0);
a4.addEffort(e4);
pRepository.add(a4);
ph1_2.addPeriodAtBegin(a4);

//Action 5
PeriodAction a5 = new PeriodAction("P01-02-A5", "Prol-Phase 2-A5",
                                   "Prol-Phase 2-A5", PeriodInTime.RULE_DURATION);
a5.setDuration(40,0);
pRepository.add(a5);
a4.addSucessor(a5); // or a5.addPredecessor(a4);
ph1_2.addPeriodAtEnd(a5);

/*
 * THE SIMULATION
 *
 * After modelling the project we can go on and simulate it...
 */
WorkSimulator sim = new WorkSimulator();
sim.setRepository(pRepository);
sim.update(ProjectDate.now());

// thats it!

```

```

/*
 * Display the results
 */
System.out.println("\n\r Sams Calendar:");
sam.getProjectCalendar().printEntriesInRange(sam,new ProjectDate(2002,1,1,0,0),
                                             new ProjectDate(2008,1,1,0,0), true,false);
System.out.println("\n\r Calendar of tech res:");
techRes.getProjectCalendar().printEntriesInRange(techRes,
                                                new ProjectDate(2002,1,1,0,0), new ProjectDate(2008,1,1,0,0), true,false);

// To proof the results, let us check the calendars against the efforts
// Note: if you someone sets the progress of an effort manually,
//       the sum of calendar entries and effort may differ from each other
System.out.println();
System.out.println("Calendar checks started:");

sam.getProjectCalendar().checkConsistency(sam.getProjectCalendar().getBegin(),
                                           sam.getProjectCalendar().getEnd(),sam);
techRes.getProjectCalendar().checkConsistency(techRes.getProjectCalendar().
                                              getBegin(),techRes.getProjectCalendar().getEnd(),techRes);
System.out.println("Calendar checks ready.");

// now print the results:

System.out.println();
System.out.println(proj1.toString());
System.out.println(ph1_1.toString());
System.out.println(a1.toString());
System.out.println(a2.toString());
System.out.println(a3.toString());
System.out.println(ph1_2.toString());
System.out.println(a4.toString());
System.out.println(a5.toString());
System.out.println(proj2.toString());
System.out.println(a6.toString());
System.out.println();
}
}

```

4. Designing a Front End

This chapter is a collection of thoughts, how a front end may be designed.

4.1. General requirements

There are some targets that should be clear and published before one starts to implement. Most of them do not require additional discussion:

- Easy, intuitive handling (the object "Project management" is complex enough)
- Concentrate on the necessary features (no overkill)
- Multiusersupport
- RightManagement
- Support of scenarios (Productive + test scenarios)
- Server - Client
- Pure JAVA (do gain OS independance)
- Database support: Postgress for development+ universal interface via JDBC
- Help on the spot (Clearness, Transparency, Tooltip help)
- ProgressBar support for longer processes (simulation phase) where applicable
- audit log

4.2. Components

In our world most people tend to think that GANTT charts are THE tool for project planning. GANTT charts are a good way to display the flow of projects. But if you want to model a project flow it has some disadvantages. Most important is that the GANTT chart always indicates a time needed for an project element. We are using a project flow simulator, so we don't have the information when an element begins and ends.

A much better way (because it does not indicate the time needed) to model a project flow is to use flow charts. Using them you create chains of project elements, split the flow for parallel actions and join them again where needed.

The editing person thinks in actions. When the model is ready it may be tested in a separate scenario. After the simulation you are able to tell something about when it will be done, because in a complex case a human being will have problems to have the overview about all pending things to do and about which resource does what.

So we suggest to use flow charts to model the projects and but display the results in GANTT chart.

4.2.1.Views

There are a couple of views onto the projects, oriented on the users role:

- **Supervisor view** (existing Projects with Priorities and Status)
- **Department leader view:** Department leaders are often asked what their department members are doing. This view helps them to answer:
 - Calendar like "Who in my department is doing what to which time?"
H-V: People - Time
- **Project leader view**
 - Calendar like "When is something done, on which of my projects?"
H-V: Projects - Time
- **Project views**
 - (Table of project data)
 - Project flow view (flow chart)
 - GANTT
- **Personal view / work calendar**

4.2.2.Environment

Depending in which role the user has to act, the application should adapt itself to the needs. We call these adaptations environments:

- Supervisor environment (to proof and release projects)
- Project manager environment
- Project member environment

4.2.3.Editors / Modules

An application will need graphical components/modules to provide its functions:

- see views
- Project flow editor
- Editor for Project, Phase, Action and Milestone data
- SysConfiguration editor

4.2.4.User Roles

The users of the project planning tool act in different roles. This can be:

- Admin (configures system)
- Supervisor per Project domain (can publish projects into scenarios)
- Project manager (models project flow, (can create personal scenario))
- Project member (can set progress of their own efforts, see project plan, work plan)
- Viewer (see only)

4.3. Project Generation / Editing

The engine is working with published projects only. A publish operation is forcing a new run

of simulation engine.

If a NEW PROJECT should be ADDED it is created outside the scope of the simulator. The project manager is creating it using something like flow diagrams. When it is ready, he might be able to fork out a temporary scenario for himself (internally copy the published projects + his new project) and see it influencing the way things go.

EXISTING PROJECTS to be EDITED are copied out of the scope of simulator and edited there.

When the project is modelled it is marked to be released by the "Supervisor". Supervisor has a view onto the projects, which shows him the priority of the other projects. He is able to create a test scenario if needed and can use all other project views to determine the impacts and side effects of the new project.

Superuser can influence everything by setting START DATE and PRIORITY. If necessary the project will be rejected and corrected by the project manager.

If the project is released it is "moved" into the scope of the simulation engine. And is not editable anymore. (If the project is a copy of an existing project, the original will be overwritten.)

Simulation engine is running again, all temporary scenarios are removed or deactivated at least. (POINT FOR DISCUSSION)

4.4. Simulator integration

Simulation on released data runs only when new projects are released. So the released projects are semistatic.

Personal temporary scenarios can be simulated independently by a separate instance of the simulation engine. It runs on a complete copy of released projects plus the additional modifications done by the project manager. This separation should be planned and implemented especially careful and clean.

A mix between scenario and productive data would mess up everything.

4.5. Data structure

This is a loose collection of data to be stored.

Definitions

- Project definition (Project, Phase, Action, Milestone)
- Resource definition (Resource, Special days, Generic calendar)
- GenericCalendar entries for DefaultWorkday and SpecialDays
- Sysconf

Simulation results

- ProjectCalendar data per Resource

States of project data

- Planning
- To be released
- Released
- Scenario copy (not editable)