**MANUFACTURING SCHEDULE**

**Introduction**

This document will describe the kind of chart that would be useful for our company. Our customers, for whom the charts are intended, are manufacturing plants. The charts are used to schedule their daily work. They illustrate to the plant operators what needs to be done and for the person that plans the production they are a useful tool to plan the machines as optimal as possible.

We have investigated the current set of Infragistics WPF controls and we noticed that there are 2 controls that are possible candidates for a solution: the xamGantt and the xamSchedule. However, we are unsure if they fully meet our requirements.

We would like your assistance in determining (1) whether any of the aforementioned controls are suitable for our scenario, and (2) which one. If it is currently not possible to achieve the desired result with the current IG controls then we would like to know if you would consider either extending the current controls, or creating a completely new one. You could then see this document as a product idea.

The software product that we are making is being developed in multiple stages. Currently, we are still in the first stage, and in this stage, we only want to *display* the schedule information to the user. Almost no user interaction is needed. Later, we will need more user interaction, but these requirements are not yet clear so they are mentioned only briefly.

**Demo case**

The requirements will be explained using a demo scenario, an imaginary factory where various kinds of pies are made. These scenarios are of course purely fictional and are only meant to illustrate certain principals.

Suppose we need to create 500 apple pies and 600 fruit pies. For the sake of simplicity, we want to create all apple pies at once, and (separately) all fruit pies at once. This means we have 2 batches that need to be made; 1 batch of 500 apple pies, and 1 batch of 600 fruit pies. First, the batch of apple pies will be produced, and later the batch of fruit pies.

For creating the apple pies, the following steps (tasks) need to be done:

* Cut the apples
* Cut the dough
* Combine the dough and the apples (“create apple pies”)
* Bake the apple pies

Creating fruit pies works in a similar fashion, the only difference is that more kinds of fruit need to be cut (apples, pears and bananas).

Creating fruit pies is a dirty task. When a batch of fruit pies has been made the pie maker machine needs to be cleaned. Cleaning is not necessary after the creation of apple pies.

Later this week maintenance is planned for the pie maker (e.g. it needs to be re- oiled), and the cutting blades of the dough cutter need to be inspected.

The following illustration demonstrates such a schedule:



**Analysis of the parts**

This paragraph contains an explanation of all parts that make up the entire chart. The parts are marked blue.



|  |  |
| --- | --- |
| **Part** | **Description** |
| Time axis | The horizontal axis is a time axis. |
| Date indicator | The time axis consists of 2 regions; this region displays date information. |
| Time indicator | The time axis consists of 2 regions; this region displays time information. |
| Current time indicator | Marks the current time. |
| Task executor | The vertical axis contains a list of task executors, which are entities (usually machines) that can execute certain tasks.  In the example these are the machines that can execute various cooking-related operations.  Note that 1 such machine may be able to execute multiple kinds of tasks (e.g. the fruit cutter can cut all kinds of fruit, it is not restricted to cutting apples only) |
| Task | A single bar on the chart represents 1 task that has been scheduled.  1 bar displays the start, end, duration and nature of that task.  The nature is indicated by the color and label. |
| Lane | All tasks are projected on a background lane, so that it is easier to group tasks for each task executor. Like rows in a datagrid, the background colors alternate. |
| Downtime | A period of time a certain task executor is incapable of executing any tasks.  Examples are maintenance, inspections, cleaning, reparations, …  Remarks:   1. Some downtimes, such as cleaning, are a direct result of a related task. 2. Other downtimes, such as maintenance, have no relation with any task. 3. Downtime periods cannot be overlapped like ordinary tasks. |

**Requirements**

This paragraph describes our requirements for the chart. The priorities are indicated using the MoSCoW method (M = Must have, S = Should have, C = Could have, W = Want to have).

Requirements regarding the visual aspects:

|  |  |
| --- | --- |
| **Requirement** | **Prio** |
| Display multiple task executors | M |
| Display multiple tasks for each task executor (i.e. multiple tasks in 1 lane) | M |
| Configure visual aspects of a task by means of databindings:   * The background color * The foreground (text) color * The text itself * The border style   + Border color   + Border thickness | M |
| Display a current time indicator | M |
| Configure the time axis:   * Set the exact start and end time (allowing the chart to “zoom” in and out) * Date / time formatting * Visibilities: display only date information, only time information or both (separately, on 2 lines) | M  S  S |

Requirements regarding user interactions:

|  |  |
| --- | --- |
| **Requirement** | **Prio** |
| When people hover over a task, a tooltip should be shown displaying detailed information about that task. | M |
| When the user right-clicks a task, a contextmenu should pop up, and when the user clicks one of the menu items, it should execute a command on a viewmodel. | S |

Future requirements regarding user interactions:

The following requirements are intended for the next stage of development. For now, they can be considered ‘W’-requirements. They are mentioned here only to give an idea of the direction our developments are going to.

* The user should be able to drag ‘n drop tasks in the same lane
* The user should be able to drag ‘n drop tasks to another lane (e.g. from fruit cutter #1 to fruit cutter #2)
* During the drag operations the system should be able to evaluate the current drop target (possibly by setting commands which provide CanExecute-logic).