oTN-2008-01 openlab Technical Note

Improving display and customization of timetable in Indico

Pierre-Luc, Hémery
Jose Benito, Gonzalez Lopez
20 August 2008
Version 1
Distribution:: Public

Plan of the report

1 A quick description of the topic of my work 1
1.1 Abstract 1
1.2 Résumé 2
2 General presentation of the project 2
2.1 Description 2
2.2 Structure and modifications 2
3 Technical point of view for the project 3
3.1 Creation of the JSON Object 3
3.2 Standard Algorithm 5
3.2.1 Principle of the algorithm 5
3.2.2 Application to the timetable in Indico 5
3.2.3 Schema for the algorithm 9
3.3 Filtering 10
3.4 Miscellaneous 11
3.4.1 The new html structure of the timetable 11
3.4.2 CSS 12
4 The future … 12
5 Conclusion 12
6 Appendices 13
6.1 CSS Files for timetable 13
6.2 Python Code for generating the JSON dictionary 14
6.3 JavaScript code used for the display of timetable 16
6.3.1 Basic functions for the display of a standard grid 17
6.3.2 Functions for generating block for the entry 17
6.3.3 Functions for entry placing 18
6.3.4 Filtering Function 18

1 A quick description of the topic of my work

For this first part, as you will see in a few lines, I've written an abstract of the work I've realized during my two months as an Openlab student. The main content of my work is sum up in that lines, that's why you will find it in two languages: English and French.

1.1 Abstract

This project takes place in the Indico Team that is located in CERN. Indico is a Rich Internet Application developed and uses at CERN (but it also uses in other places). Indico is software made for conference management (content, time, place, people and more…).

For this project, I have worked with the development version of Indico, and more precisely with the timetable display. For the moment, Indico used some "old fashioned' HTML way of coding based on "tab" tag and in addition the code is generated by the server in Python. So the idea is to generate this code on the

side on the client and this code will now be based on “div” tag. It will allow some filtering functions in JavaScript, and the final goal will be to the user to create his own schedule on Indico.

1.2 Résumé

Ce projet s’est déroulé au sein de l’équipe Indico du CERN. Indico est un *Rich Internet Application* développe et utilise au CERN (elle est également utilisée en dehors) permettant la gestion de conférence (plus généralement d’événements) tant au niveau temporel, matériel et du contenu (slides, posters…).

Mon projet s’est axe sur la version en développement d’Indico et plus particulièrement sur l’affichage des emplois du temps (appelles timetable). Actuellement, Indico affiche les emplois du temps avec du code HTML reposant sur des balises tab et génère cote serveur (en Python). L’idée est donc d’utiliser une structure HTML plus « moderne » reposant sur des div permettant ainsi l’utilisation de fonction de filtrage par l’utilisateur (afficher les événements par salle par exemple). De plus, cette fois, le code HTML serait génère en JavaScript, cote client, ce qui permet ainsi une meilleure flexibilité. La fin de cette logique est que chaque utilisateur puisse générer, gérer, composer son propre emploi du temps.

2 General presentation of the project

2.1 Description

Indico can manage several types of events and among those kind of events there is conference. In our application, conference can be associated with timetables. Here is an example of timetable in the current production version of Indico :

This project starts with a simple fact: actually, Indico displays timetable with HTML tags tables. It is not convenient for several reasons:

1. Difficult to place dynamically the different events on the timetable
2. Not scalable
3. A lot of space is lost when we used that

In addition, when the user wants to custom the timetable, he has to click several times and a request is sent to the server. It’s slow and not very interactive. So the idea is to use a better way to display the timetable and this way should allow a dynamic filtering for the user.

That’s why we have decided to use this new method:

1. We replace the table tag by div tag : more easy to place and to display
2. No more useless request to the server: JSON is generated by the server and sent once to the client’s browser. Then all the process is done in JavaScript.
3. A standard display is made and the user can play with the filter menu to display is own time table.

Now we will explain the development in details.

2.2 Structure and modifications

The following schema will show where we have made the modification in the Indico’s structure to realize the new display and the filtering functions. You’ll see that the most important modifications are in fact made in new files dedicated specially for the new display and filtering functions.

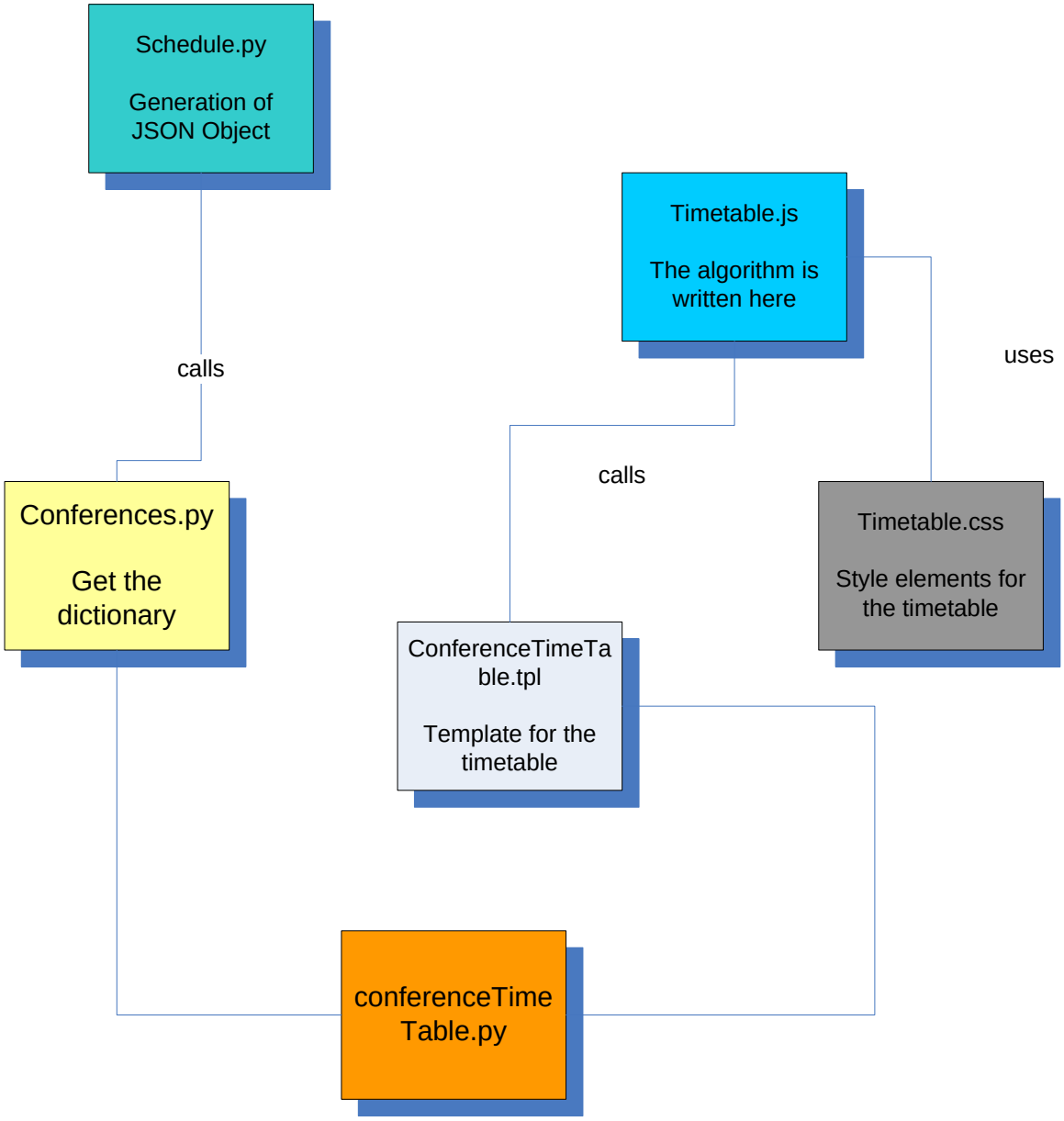


Figure 1Structure

3 Technical point of view for the project

3.1 Creation of the JSON Object

We have already explained that the main point on the server side was the creation of the JSON Object. We will now give some details about its creation and its design.

Originally for the timetable, the server generated the HTML code for it. Now, we just use a static method to create the JSON dictionary on the server side. We place this method in the schedule.py file. This method received a list of entry sort by days and then call some methods from the type of the entry by the way : entry.getOwner().getXX().

We use getOwner() to determine the specific type of the entry we are looking for instance is it a break, a session slot or contribution.

So we fill a dictionary first organised by day, then each day contains dictionaries for each entry that the day has. For each entry, we give the following information:

- Title
- Description
- Start and end hour
- Duration
- Content of the session slot (if it's the session) and the master session Id.
- Type of the entry

At the same level of the day dictionary, we have also another dictionary that manages session (and not session slot). We simply give the title, the id and the color of the session.

Here we sum up this text with this image that describes the JSON Object:

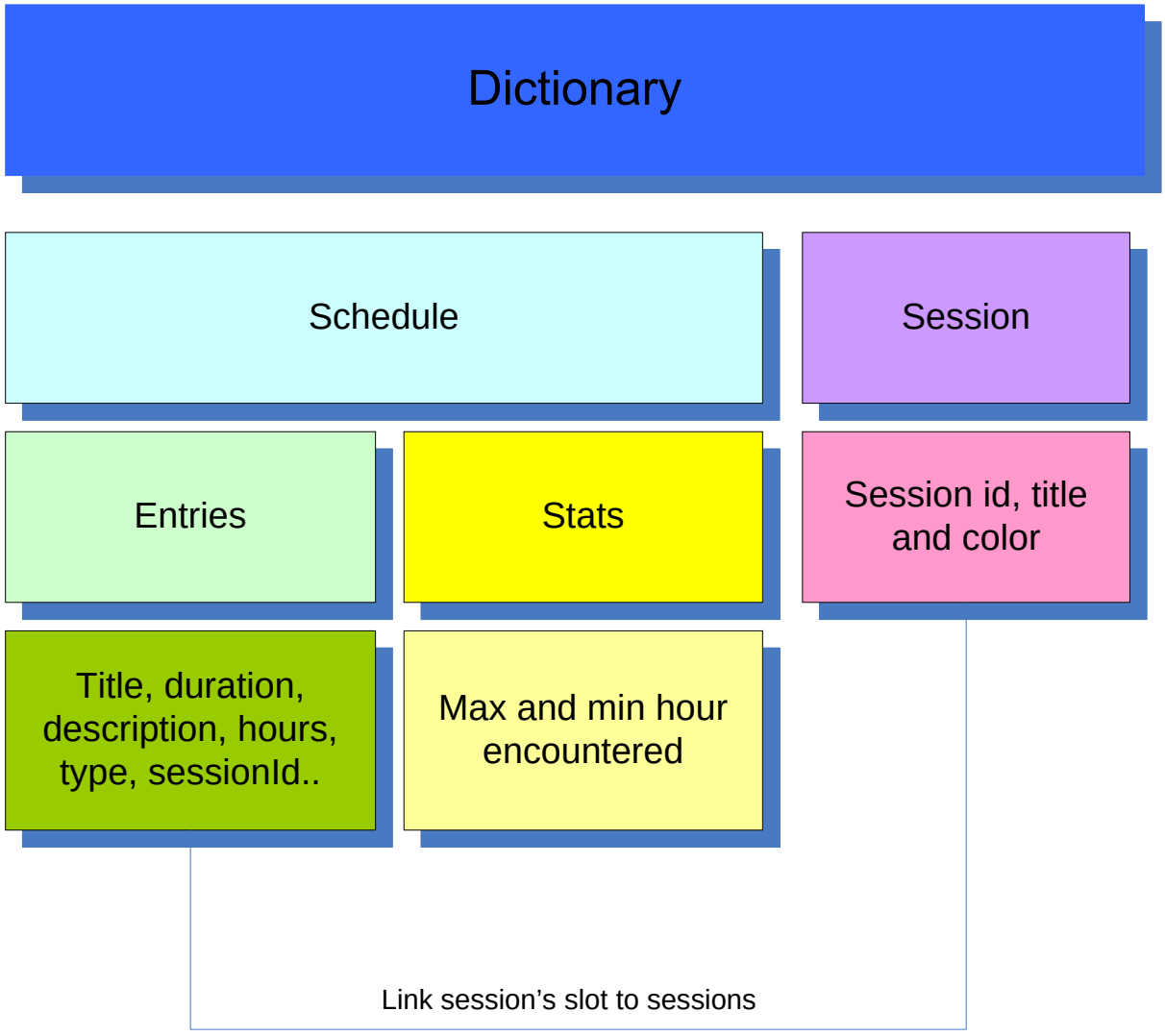


Figure 2 The Dictionary used to transmit informations

Now we will explain what we do when the server sends the timetable on the client side.

3.2 Standard Algorithm

3.2.1 Principle of the algorithm

This algorithm starts with a general idea, so it can be applied, I suppose for displaying other thing than timetable in Indico.

Here are the important steps:

1. Build a standard repair with known coordinates
2. Place elements in that repair to mark the vertical position of the element that we want to display
3. Then for each vertical zone find, analyze the horizontal position
    a. Determine how many columns you need
    b. For each element in that zone, you have to give a number of column where it has to stay
4. With those numbers we can calculate the exact position each element.

3.2.2 Application to the timetable in Indico

When the user clicks for displaying a timetable, the server sends to his browser the object that we have described in the last part.

Then, the script analyzes the day dictionary to know how many days to display. Then, it creates a standard grid for the hour as seen on the next screenshot:

Figure 3 The empty grid first generated

After that, the script iterates on the entry to check in which hour, we can find some entries and then we place correctly the hour on the grid as seen on the next screenshot:

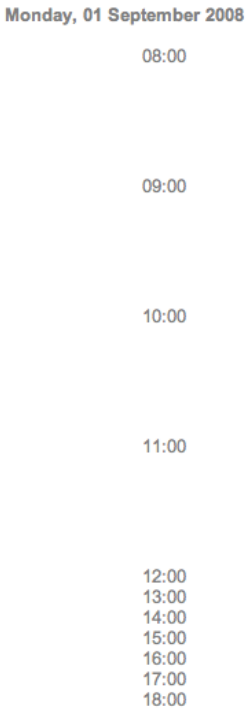
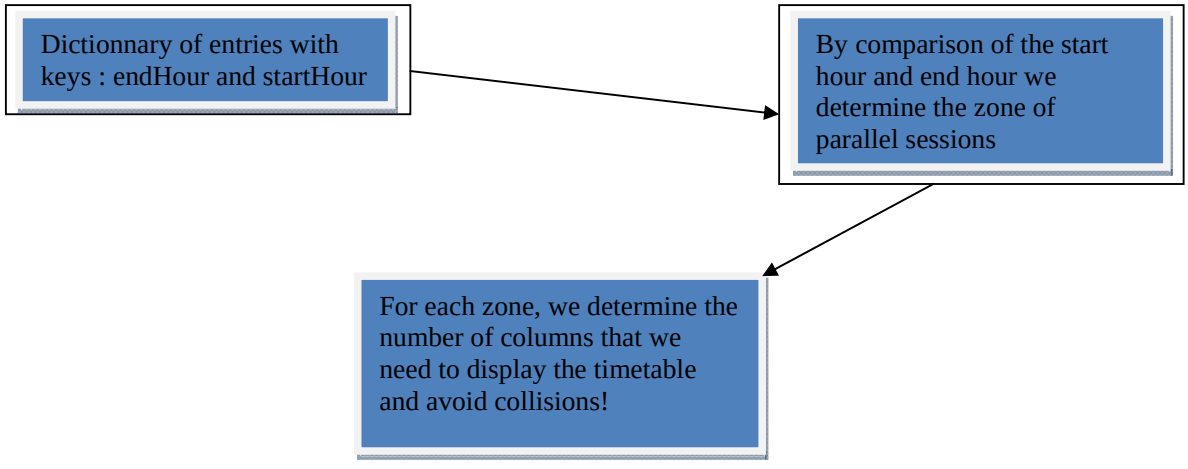


Figure 4 The "analyzed" grid

Then we have a well dimensioned grid for placing our entries.

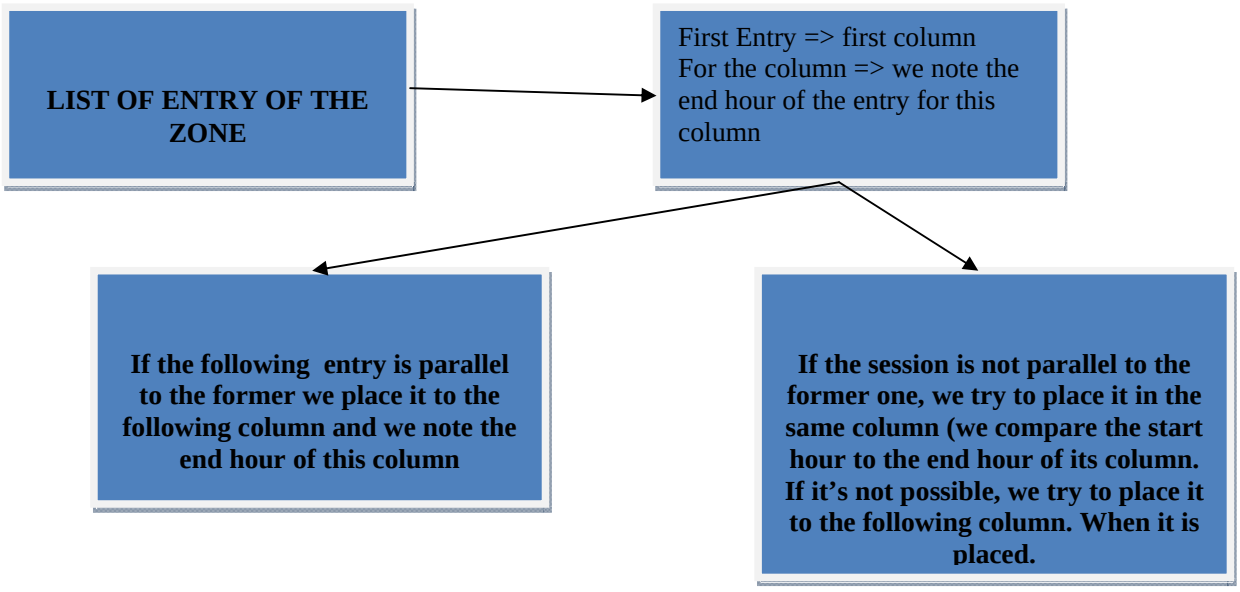
So now, we know where to place vertically the different entries (sessions' slots, breaks and contributions). But we must also determine the horizontal placement of each block, because sometimes we have some parallels entries. So we had to find an elegant way to address this issue.

The following schemas explain the algorithm that detects, calculates and places the entries.



We want to underline the fact that now, each zone has its own number of columns and in the previous version it was the zone with the maximum number of parallels sessions that imposed to the others its number so we lose a lot of space with the former version! Now, we have a better usage of space…

Now the algorithm will lead an analysis by zone of parallel sessions, and for each entry of this zone, we will find a column for the entry. The following schema explains the basic of the algorithm:



Now, for each zone, we have the number of column and for each entry the number of column where it has to be put! So we have enough information to place the entry with the style property left and top (x and y coordinates if you prefer…). We have also the duration to determine the width of the entry) and then we obtain a result like the following figure:

I want to underline the fact that this algorithm is independent of the time zone chosen by the user to display the conference, so when a change of time zone is asked by the user to the server, a new dictionary is generated by the server with the correct information for the current time zone. There is a problem that can happen with a time zone change but it is identified and in the conclusion I give a possible solution to fix it. It is important to notice that the problem is not in the algorithm but in the way we generate the information that it needs.

Figure 5 A completed timetable

You can see on the screenshot that there is a filtering menu. We have now implemented some “live” filtering functions (i.e. without requests to the server). We will explain that in the next part(3.3).

3.2.3 Schema for the algorithm

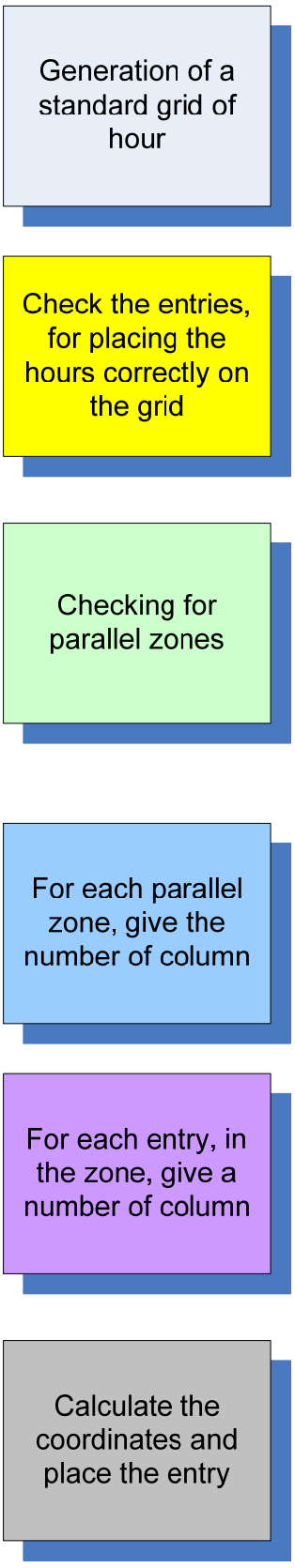


Figure 6 Schema of the algorithm

3.3 Filtering

Filtering functions were already present in the production version of Indico. But it is slow to use because we have to make several clicks and it takes time to the user to see the result.

That's why we have continued the work with JavaScript. With the library Presentation, we defined some HTML input (checkbox and radio button) to observe some parameters:

1. Days to display (implemented)
2. Sessions to display (implemented)
3. Display Session or content of the session (contributions) (in development)

When the user clicks on one input, a new dictionary is created with the selected content by the user. And we launch with this new JSON object that contains the selected content by the user. Now for enabling or disabling, content only one click is needed. The result appears in live for the user. So he can easily select what he wants to display.

So in fact, we can say that the filtering functions are in fact in the continuation of the algorithm explained in the previous part. The essential tip results in the creation of the new dictionary with the good content inside. With the library Presentation it's easy to realize, because when we create HTML inputs such as radio buttons or checkbox, we just have to place an observer.

The following schema sums up the principle:

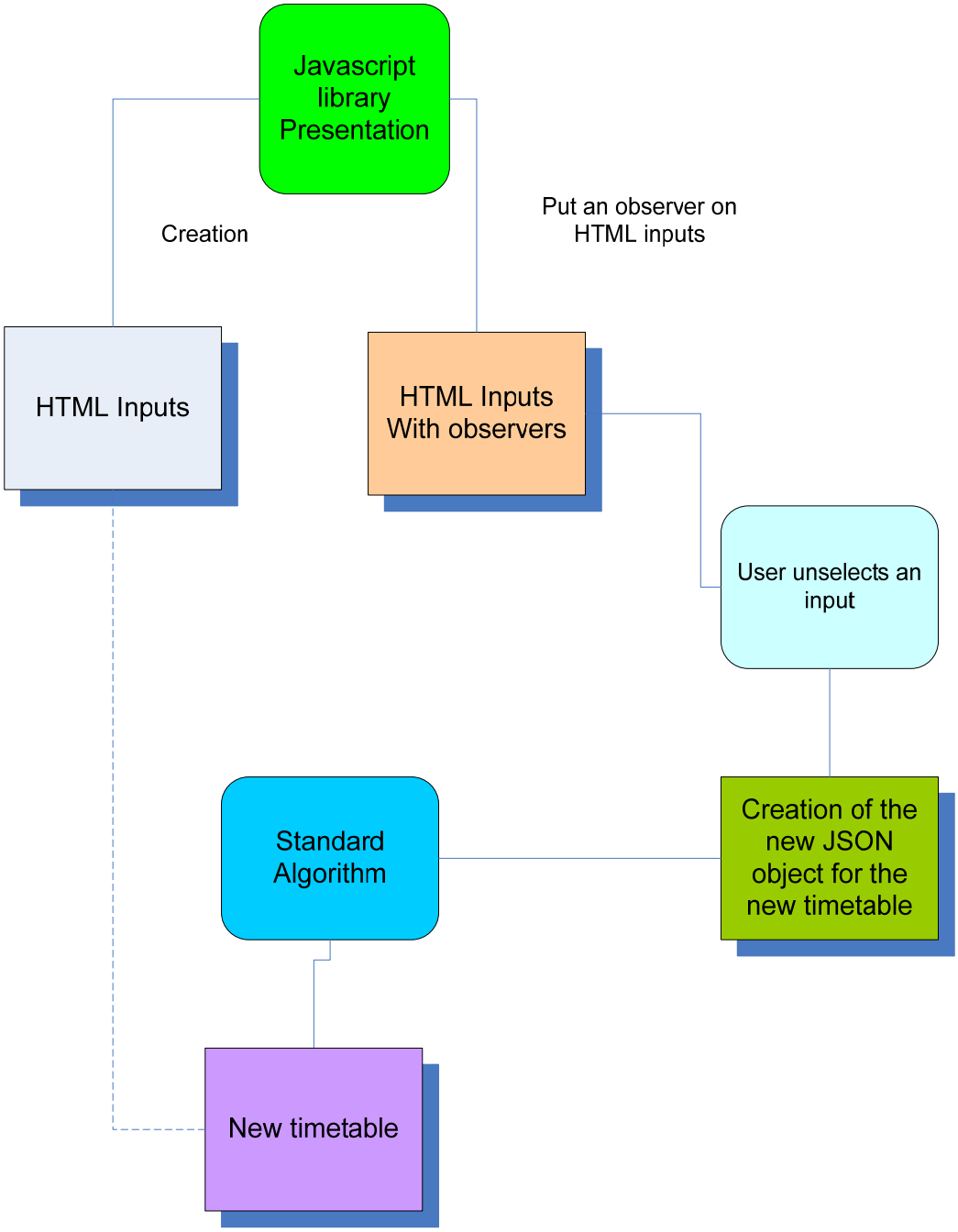


Figure 7 Principle for filtering

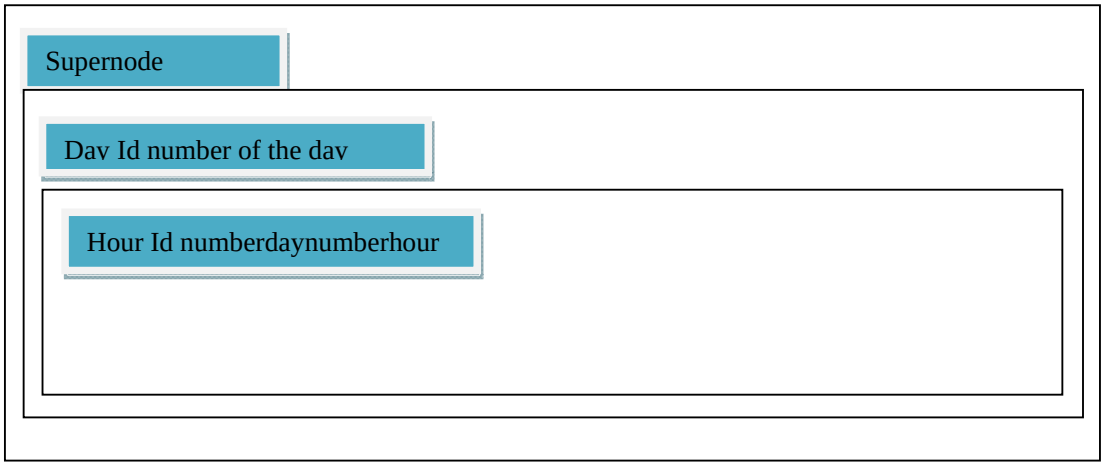
We will now explain some other details about the project in the next part.

3.4 Miscellaneous

3.4.1 The new html structure of the timetable

We have already said that the production version of Indico used HTML tag table for displaying timetable. For commodity, better use of space, simplification of the code, we have chosen to adopt a div structure.

We explain the new structure with the following schema:



The entries are also displayed as div. They are place inside their day div and are at the same level in the HTML tree as the hour.

Every div is placed with absolute positioning.

We also used a separated div for the filtering menu.

3.4.2 CSS

We have made a CSS style dedicated to timetable. It is a very simple CSS that only precise some details like the absolute positioning. In fact, the main part of the style attributes is given directly in the HTML code because it is a more simple way to set them dynamically.

However, the CSS used element defined by Pedro Ferreira in the administration panel of Indico for the preview of timetable.

4 The future …

There is a next step for this project. In fact, it is pretty simple to understand: it will be excellent if the user can save his filter preference for his timetable. For instance, it will be very interesting if the user can save the days where it will assist to the conference and the sessions that he wants to see.

In a more technical point of view, the code of the algorithm is very modular i.e. there are a lot of sub functions that are very specialized. So if someone has an idea to optimize one part of the algorithm he will just have to replace the sub function that is concerned.

5 Conclusion

The main aim of this project has been achieved: the experience for the user is now better display of timetables and filtering are faster than before. Several things are also important to notice before ending this report.

The script has been well tested on several points:

1. Compatibility with browser: Internet Explorer 5.5 to 8 (beta), Mozilla FireFox 2 and 3, Safari and Opera.

2. Different situations of conference: the algorithm is always fast and avoids collisions between entries! One point to underline, sometimes the algorithm overestimates the number of column needed (especially when we have more than three unparallel sessions in one column). If a better algorithm is found for that function, we just have to replace the actual one…
3. The current engine supports time zone changes. However, sometimes, when a session is split between two days during the change of time zone, the script crashes. It is possible to correct it hopefully. A solution could be to check if there is a split before adding the whole entry in a particular day, and if there is split to separate the two parts, and for the first one set the end hour to midnight and for the second one the hour start to midnight. The script only needs this information for running without problem.
4. The filtering menu is not defined in term of design definitively. We should find a way to have this menu easy of access for the user and it should be fast to use! Pedro Ferreira has the idea to set up a vertical dynamic line of the size of the web page and where the user clicks on it, the menu could be displayed. It is a possible elegant way to address this issue. But users' feedback are I think necessary to have a better view of this problem.

As we have already said the logic evolution for that project is to set up a way to the user to simply save his filtering information.

6 Appendices

In that part I will present the commented code, I've realized.

6.1 CSS Files for timetable

In the first time of this project, the CSS plays a bigger role than now. But because we have adopted a general algorithm for every entry, this file is now only for policy font size and for absolute positionning.

Here is the file:

```
// for hour like 08:00
time{
   color:gray;
   background-color:white;
   font-size:-1;
   padding-right:5px;
   font-weight:normal;
   left:65px;
   height:12px;
   margin-left:35px;
   position:absolute;
}

//for entry
.blockSession{
   margin-left:0px;
   margin-top:0px;
   margin-bottom: 0px;
   margin-right:0px;
   position: absolute;
   border-top: 2px solid  white;
   border-left: 1px solid  white;
   border-right: 1px solid  white;
}
```

```
// for the content of the entry
.contentTitleBlock{
    font-size:12;
    position:absolute;
    color:gray;
    text-align: left;

}

//for the block menu
.filterBlock{
    position:absolute;
    font-weight:900;
    background: #DDDDDD;
}

// for subcategory in filter menu
.filterType{
    margin-top: 15px;
    font-weight:bold;
    max-height: 200px;
    overflow: auto;
}

//for input in filter menu
.filterElement{
    font-weight:normal;
}
```

6.2 Python Code for generating the JSON dictionary

We are generating the JSON object in the file Schedule.py. Basicaly, we call method from the ScheduleEntry class or from the parent 'object' class (that explains sometimes the use of getOwner()).

It can look pretty long and horrible but it is not too terrible, we just fill information in the dictionary, the main to know is the structure of the dictionary after it is just a game with getXX() function to obtain the information XX().

Here is the code:

```
class ScheduleToJson:
## Method Static
    @staticmethod
## We work with the timetable and the timezone to generate correct hours
    def process(timetable, tz):
## We initiliaze all the levels of dictionaries
        D={}
        D['Sessions']={}
        D['Schedule']={}
        dejaVuSession = []
## We use those paremeters to use the correct scale for the timetable in function of the timezone
        maxHour=0
        minHour=24
//counter of days
        c=0
```

```
//counter of sessions
      count=0
//initialize others dictionaries
      D['Schedule']['entries']={}
      D['Schedule']['Stats']={}
      D['Schedule']['Stats']['min']=0
      D['Schedule']['Stats']['max']= 0
//We get the day list of the timetable
      for day in timetable.getDayList():
         D['Schedule']['entries'][c]={}
//for each day we get the entry list
         for entry in day.getEntryList():
            interDate = entry.getAdjustedStartDate(tz=tz)
            interDate = interDate.replace(hour=0,minute=0,second=0)
            dEntry = D['Schedule']['entries'][c][int(entry.getId())] = {}
//We are just picking up information from timetable
            if(interDate == day.getDate()):
               initTime = entry.getStartDate().ctime()
               dEntry['title']=entry.getTitle()
               dEntry['date']=entry.getAdjustedStartDate(tz=tz).ctime()
               dEntry['hour']=entry.getAdjustedStartDate(tz=tz).time().isoformat() + "-" +
entry.getAdjustedEndDate(tz=tz).time().isoformat()
               dEntry['description']=entry.getOwner().getDescription()
//We made some distinctions between sessions, contributions and break about colors and titles
               if isinstance(entry.getOwner(),MaKaC.conference.Contribution):
                  dEntry['color']="#fff4e4"
                  dEntry['type']='contribution'
               elif isinstance(entry.getOwner(),MaKaC.conference.Conference):
                  dEntry['color']="#90c0f0"
                  dEntry['type']='break'
               else:
                  countEntries = 0
                  dEntry['content']={}
                  dEntry['color']=entry.getOwner().getColor()
                  dEntry['sessionId']=entry.getOwner().getSession().getUniqueId()
                  dEntry['type']='session'
                  dEntry['sessionTitle']=entry.getOwner().getSession().getTitle()
//We check if the session has some sub contributions, we set up a sub dictionary build on the same model
                  for contrib in entry.getOwner().getSchedule().getEntries():
                     dEntry['content'][countEntries]={}
                     dEntry['content'][countEntries]['title']= contrib.getTitle()
                     dEntry['content'][countEntries]['color']= entry.getOwner().getColor()
                     dEntry['content'][countEntries]['date']=contrib.getStartDate().ctime()
                     dEntry['content'][countEntries]['hour']=contrib.getStartDate().time().isoformat() + "-" +
contrib.getEndDate().time().isoformat()
                     dEntry['content'][countEntries]['description']=contrib.getOwner().getDescription()
                     dEntry['content'][countEntries]['type']='contribution'
                     dEntry['content'][countEntries]['dayId']=c+1
                     dEntry['content'][countEntries]['duration']=str(contrib.getDuration())
//we build some ids that are used to link our entries to the  div in html
                     if(c<9):

dEntry['content'][countEntries]['hourStartId']=str(0)+str(c+1)+""+contrib.getAdjustedStartDate(tz=tz).time().
isoformat()[0:2]
```

```
dEntry['content'][countEntries]['hourEndId']=str(0)+str(c+1)+""+contrib.getAdjustedEndDate(tz=tz).time().isoformat()[0:2]
                    if(c>9):

dEntry['content'][countEntries]['hourStartId']=str(c+1)+""+contrib.getAdjustedStartDate(tz=tz).time().isoformat()[0:2]

dEntry['content'][countEntries]['hourEndId']=str(c+1)+""+contrib.getAdjustedEndDate(tz=tz).time().isoformat()[0:2]

dEntry['content'][countEntries]['hourStart']=contrib.getAdjustedStartDate(tz=tz).time().isoformat()[0:2]+""+contrib.getAdjustedStartDate(tz=tz).time().isoformat()[3:5]

dEntry['content'][countEntries]['hourEnd']=contrib.getAdjustedEndDate(tz=tz).time().isoformat()[0:2]+""+contrib.getAdjustedEndDate(tz=tz).time().isoformat()[3:5]
                    countEntries = countEntries + 1
// Here we simply build a dictionary that make the link between
                if entry.getOwner().getSession().getUniqueId() not in dejaVuSession:
                    D['Sessions'][count]={}
                    D['Sessions'][count]['title']=entry.getOwner().getSession().getTitle()
                    D['Sessions'][count]['sessionId']=entry.getOwner().getSession().getUniqueId()
                    D['Sessions'][count]['color']=entry.getOwner().getSession().getColor()
                    dejaVuSession.append(entry.getOwner().getSession().getUniqueId())
                    count=count + 1
            dEntry['dayId']=c+1
            dEntry['duration']=str(entry.getDuration())
            if(c<9):
dEntry['hourStartId']=str(0)+str(c+1)+""+entry.getAdjustedStartDate(tz=tz).time().isoformat()[0:2]

dEntry['hourEndId']=str(0)+str(c+1)+""+entry.getAdjustedEndDate(tz=tz).time().isoformat()[0:2]
            if(c>9):
                dEntry['hourStartId']=str(c+1)+""+entry.getAdjustedStartDate(tz=tz).time().isoformat()[0:2]
                dEntry['hourEndId']=str(c+1)+""+entry.getAdjustedEndDate(tz=tz).time().isoformat()[0:2]

dEntry['hourStart']=entry.getAdjustedStartDate(tz=tz).time().isoformat()[0:2]+""+entry.getAdjustedStartDate(tz=tz).time().isoformat()[3:5]

dEntry['hourEnd']=entry.getAdjustedEndDate(tz=tz).time().isoformat()[0:2]+""+entry.getAdjustedEndDate(tz=tz).time().isoformat()[3:5]
            if int(entry.getAdjustedStartDate(tz=tz).time().isoformat()[0:2])<minHour:
                D['Schedule']['Stats']['min']=int(entry.getAdjustedStartDate(tz=tz).time().isoformat()[0:2])
                minHour = int(entry.getAdjustedStartDate(tz=tz).time().isoformat()[0:2])
            if int(entry.getAdjustedEndDate(tz=tz).time().isoformat()[0:2])>maxHour:
                D['Schedule']['Stats']['max']=int(entry.getAdjustedEndDate(tz=tz).time().isoformat()[0:2])
                maxHour = int(entry.getAdjustedEndDate(tz=tz).time().isoformat()[0:2])
        c=c+1
    return D
```

6.3 JavaScript code used for the display of timetable

6.3.1 Basic functions for the display of a standard grid

setMinMaxHour :
parameter: entry in the dictionary that gives the maximum and the minimum hour encounter for the timetable
description: set up the max hour and the min hour for the timetable

layout:
parameter: dictionary JSON. This function generates the div for the days
description: calls the function to fill the day divs.

fillWithHours:
parameter: day's div, name and number.
description: for each day, using the max and the min hour, the day is fill with a grid of hour.

6.3.2 Functions for generating block for the entry

creationBlock :
parameter: an entry, the day, and the dictionary.
description: this method is used to call the different methods explained in the next lines and creates the div for the entry.

fillBlock :
parameter: the div for the entry.
description: the div is filled with this function with the information of the div.

detectParaSession:
parameter: the dictionary with all the entries.
description: this function will basically detect every zone with parallel sessions.
return: a list of list with a list for each zone with parallel sessions.

getNumberOfColumns :
parameter; the dictionary with all the entries and a list of parallel session for a zone.
return: for the examined zone an integer that gives the number of columns needed to avoid collisions when we display the timetable.

testParaSession:
parameter: two entries and the dictionary of entry.
description: Check if the entry are parallel (comparison of start and end hours).
return: a boolean.

searchTheParaListOfAnEntry : parameter an entry. Return the zone of parallel sessions where the entry is.

linkBlockToColumn:
parameter: number of column, dictionary of entry and a list of parallel session.
return: a number of column for each session
description: it avoids collision during the session by noticed for each column the end hour of each element inserted in it and then check that the start hour of the element we want to insert is after the end hour noticed.

6.3.3 Functions for entry placing

positionBlock:
parameter: the div for the entry, the entry, the number of column and the column for the entry.
description: It calculates the position for the block with the parameter given (rule of three).

displayGrid :
parameter: the dictionary.
description: It calls the method describes in the next few lines.

linkEntriesToGrid:
parameter: the dictionary.
*description:*It will iterate through the dictionary to localize where the entries are.
return: It returns a list of list containing hour and entry linked.

wellDimensionnedGrid:
parameter: the list returned by the previous function.
description: With the information given by the list it moves the hour in order to display a well dimensioned grid. When there are parallel sessions the hour is only moved once.

6.3.4 Filtering Function

Those functions use the Presentation JavaScript library.

createFilterDiv:
parameter: the dictionary.
*description:*It creates a menu with all the input needed for the filtering. It also the functions describe in the next few lines to create the input.

createDayFilter:
parameter: a div created by the previous function.
description: This function observes the value of the input selected by the user and when an input is unselected a list of the selected input is created d and the function redraw is called.

createSessionFilter: idem

createTypeFilter: idem

redraw:
description: we use the list of selected inputs to create a new dictionary and to recall the algorithm to generate a new timetable.