

Fachgebiet für Offene
Kommunikationssysteme (OKS)

Time Switches for VHE User Profiles

Project in SS 2002

Bjoern Schuenemann (schueni@cs.tu-berlin.de)

Table of Contents

1	Introduction.....	3
2	Structure of Time Switches.....	3
2.1	XML structure	3
2.2	Structure of class TimeSwitch	6
2.3	Structure of Java code of the CPL time-switch (Columbia University).....	7
3	VHE User and Service Profiles.....	8
3.1	Profile Management Entities	8
3.2	Class model.....	9
4	Start the Program	11
5	GUI.....	11
5.1	Graphical representation of User Records, User Profiles and Time Switches	11
5.2	Create a new or edit an existing Time Switch	12
6	Bugs.....	14
7	To Do.....	14
8	Author's Address	14
9	Bibliography	14

1 Introduction

I have dealt in this project with capabilities to make decisions based on the time and date an event is being started or finished. Time switches allows such decisions. The structure of time switches is based closely on the time switch tag of Call Processing Language (CPL) [1].

There is a sample implementation with a graphical user interface allows creating and editing time switches for VHE user profiles and VHE user records. This implementation also can used for other sceneries. There are only short adaptation necessary, because the code has a modular structure .

2 Structure of Time Switches

The class **TimeSwitch** has all the accessories necessary for setting the parameters of a time switch. The field **_xmlFile** in this class contains the file with the XML code of the time switch. Because of complexity of the XML code chapter 2.1 goes deeply into the structure of this XML code. Chapter 2.2 explains the class **TimeSwitch**. A short description of Java code of the CPL time-switch (Columbia University) follows in chapter 2.3. This code is used checking the activity of time switches.

2.1 XML structure

The XML structure of time switches is based closely on the time switch tag of Call Processing Language (CPL) [1]. But there are some restrictions:

- The “secondly”, “minutely”, and “hourly” recurrences, and the “bysecond”, “byminute”, and “byhour” rules, are eliminated.
- The “count” and “bysetpos” rules are eliminated.
- The duration of the event (**duration** or **dtend - dtstart**) must be less than 24 hours.

The syntax of important parameters of the time-switch node is summarized in Figure 1.

dtstart	Start of interval (RFC 2445 DATE-TIME) [3]
dtend	End of interval (RFC 2445 DATE-TIME) [3]
duration	Length of interval (RFC 2445 DURATION) [3]
freq	Frequency of recurrence (one of “daily”, “weekly”, “monthly”, or “yearly”)
interval	How often the recurrence repeats
until	Bound of recurrence (RFC 2445 DATE-TIME) [3]
byday	List of days of the week
bymonthday	List of days of the month
byyearday	List of days of the year
byweekno	List of weeks of the year
wkst	First day of the work week

Figure 1: Syntax of the **time-switch** node

The **time-switch** tag takes two optional parameters, **tzid** (Time Zone Identifier) and **tzurl** (Time Zone URL), both of which are defined in RFC 2445 [3]. These parameters are insignificant at this stage of the project, but in future implementations they can be used.

Time nodes specify a list of periods during which their output should be taken. They have two required parameters: **dtstart**, which specifies the beginning of the first period of the list, and exactly one of **dtend** or **duration**, which specify the ending time or the duration of the period, respectively.

For a recurring interval, the **duration** parameter MUST be smaller than 24 hours. Zero-length and negative-length durations are not allowed.

If no other parameters are specified, a time node indicates only a single period of time. More complicated sets of periods and intervals are constructed as recurrences. A recurrence is specified by including the **freq** parameter, which indicates the type of recurrence rule. No parameters other than **dtstart**, **dtend**, and **duration** SHOULD be specified unless **freq** is present.

The **freq** parameter takes one of the following values: “daily”, to specify repeating periods based on an interval of a day or more; “weekly”, to specify repeating periods based on an interval of a week or more; “monthly”, to specify repeating periods based on an interval of a month or more; and “yearly”, to specify repeating periods based on an interval of a year or more. These values are not case-sensitive.

The **interval** parameter contains a positive integer representing how often the recurrence rule repeats. The default value is “1”, meaning every day for a “daily” rule, every week for a “weekly” rule, every month for a “monthly” rule and every year for a “yearly” rule.

If the value specified by **until** is synchronized with the specified recurrence, this date or date-time becomes the last instance of the recurrence. If not present, the recurrence is considered to repeat forever.

The **byday** parameter specifies a list of days of the week. “MO” indicates Monday; “TU” indicates Tuesday; “WE” indicates Wednesday; “TH” indicates Thursday; “FR” indicates Friday; “SA” indicates Saturday; “SU” indicates Sunday.

Each **byday** value can also be preceded by a positive (+n) or negative (-n) integer. If present, this indicates the nth occurrence of the specific day within the “monthly” or “yearly” recurrence. For example, within a “monthly” rule, +1MO (or simply 1MO) represents the first Monday within the month, whereas -1MO represents the last Monday of the month. If an integer modifier is not present, it means all days of this type within the specified frequency. For example, within a “monthly” rule, MO represents all Mondays within the month.

The **bymonthday** parameter specifies a comma-separated list of days of the month. Valid values are 1 to 31 or -31 to -1. For example, -10 represents the tenth to the last day of the month.

The **byyearday** parameter specifies a comma-separated list of days of the year. Valid values are 1 to 366 or -366 to -1. For example, -1 represents the last day of the year (December 31st) and -306 represents the 306th to the last day of the year (March 1st).

The **byweekno** parameter specifies a comma-separated list of ordinals specifying weeks of the year. Valid values are 1 to 53 or -53 to -1. A week is defined as a seven day period, starting on the day of the week defined to be the week start (see **wkst**). Week number one of the calendar year is

the first week which contains at least four (4) days in that calendar year. This parameter is only valid for “yearly” rules. For example, 3 represents the third week of the year.

The **bymonth** parameter specifies a comma-separated list of months of the year. Valid values are 1 to 12.

The **wkst** parameter specifies the day on which the work week starts. Valid values are “MO”, “TU”, “WE”, “TH”, “FR”, “SA” and “SU”. This is significant when a “weekly” recurrence has an interval greater than 1, and a **byday** parameter is specified. This is also significant in a “yearly” recurrence when a **byweekno** parameter is specified. The default value is “MO”.

Byxxx parameters modify the recurrence in some manner. **Byxxx** rule parts for a period of time which is the same or greater than the frequency generally reduce or limit the number of occurrences of the recurrence generated. For example, **freq=“daily” bymonth=“1”** reduces the number of recurrence instances from all days (if the **bymonth** parameter is not present) to all days in January.

Byxxx parameters for a period of time less than the frequency generally increase or expand the number of occurrences of the recurrence. For example, **freq=“yearly” bymonth=“1 2”** increases the number of days within the yearly recurrence set from 1 (if **bymonth** parameter is not present) to 2.

If multiple **Byxxx** parameters are specified, then after evaluating the specified **freq** and **interval** parameters, the **Byxxx** parameters are applied to the current set of evaluated occurrences in the following order: “bymonth”, “byweekno”, “byyearday”, “bymonthday” and “byday”; then “until” is evaluated.

Here is an example of evaluating multiple **Byxxx** parameters.

```
<time dtstart="2000-01-24T10:15:00-05:00" duration="PT0H10M"
      freq="yearly" interval="2" bymonth="1" byday="SU">
```

First, the **interval=“2”** would be applied to **freq=“YEARLY”** to arrive at “every other year”. Then, **bymonth=“1”** would be applied to arrive at “every January, every other year”. Then, **byday=“SU”** would be applied to arrive at “every Sunday in January, every other year. Then the hour, minute and second is derived from **dtstart** and the **duration** is 10 minutes to end up in “every Sunday in January from 10:15:00 AM to 10:25:00 AM, every other year”. Similarly, if the **byday**, **bymonthday** or **bymonth** parameter were missing, the appropriate day or month would have been retrieved from the **dtstart** parameter.

2.1.1 Reasons for restrictions of parameter of time switch nodes

- The **count** parameter cannot be checked in constant running time, since it requires that the server enumerate all recurrences from **dtstart** to the present time, in order to determine whether the current recurrence satisfies the parameter.
- Similarly, the **bysetpos** parameter requires that the server enumerate all instances of the occurrence from the start of the current recurrence set until the present time. This requires somewhat more complex pre-processing, but generally, a single recurrence with a **bysetpos** parameter can be split up into several recurrences without them.
- Finally, constant running time of time switches also requires that a candidate starting time for a recurrence can be established quickly and uniquely, to check whether it satisfies the

other restrictions. This requires that a recurrence's duration not be longer than its repetition interval, so that a given instant cannot fall within several consecutive potential repetitions of the recurrence. That's why duration of the event (**duration** or **dtend - dtstart**) must be less than 24 hours.

2.2 Structure of class **TimeSwitch**

The class **TimeSwitch** is the core of this project. It has all the accessories necessary for setting the parameters of a time switch. It encloses methods to make decisions based on the time and date the user profile is active or not. The class contains methods to parse, validate and check time switches. Attributes of time switches can be get and set.

The class contains four fields.

_priority: Every time switch has a priority. If two active time switches exists in the same time period, the time switch with the higher priority will be executed. The lowest priority is zero.

_negative: Negative time switches are in activity if and only if conditions of the time switch are not fulfilled.

_xmlFile: This file contains the XML code of the time switch (see chapter 2.1 XML structure). This code cannot be contained in a String, because parsing of XML code with XERCES is supported for files only.

_exists: The file name of the XML file is an abstract name. **_exists** is true, if this file exists, that is the xml file contains data.

Three methods checking correct structure of time switch exist in class.

parse(): Tests with JDOM if the XML structure of the file is valid.

validate(): Tests with JDOM/XERCES if the XML file is conform to the schema **TimeSwitch.xsd**.

check(): There are some illegal values are valid against the schema. This method tests if the time switch has no illegal values.

There are many methods to get the attributes of the time switch (**getPriority()**, **isNegative()**, **exists()**, ...). For more information see javadoc documentation.

An important method is **isActive()**. This method tests if the time switch is in activity in the specified time periode. **isActive()** use "Cal-Code: Java code for CPL time-switches" [2] (see chapter 2.3 Structure of Java code of the CPL time-switch (Columbia University))

For more information about methods setting attributes of the time switch or change them (**set()**, **toString()**, **delete()**, ...) see javadoc documentation.

The Structure of Class **TimeSwitch** is summarized in Figure 2.

TimeSwitch
boolean _exists boolean _negative short _priority File _xmlFile
boolean parse() boolean validate() int check()
boolean isActive() boolean exists() short getPriority() File getXmlFile() boolean isNegative()
boolean set() String toString() boolean delete() void setNegativity() void setPriority()

Figure 2: Structure of Class **TimeSwitch**

2.3 Structure of Java code of the CPL time-switch (Columbia University)

This implementation [2] contains sample Java code implementing the logic underneath the CPL <time-switch> tag [1] and is written by Jonathan Lennox <lennox@cs.columbia.edu>. The software bears the standard 3-clause BSD license.

The class `edu.columbia.cpl.Recurrence` implements a recurrence. It has all the accessories necessary for setting the parameters of a recurrence. There are two support classes, `edu.columbia.cpl.Duration` (which represents a duration interval) and `edu.columbia.cpl.DayAndPosition` (which represents byday rules like "fourth Thursday").

The method `isActive()` in class **TimeSwitch** uses the method `isInRecurrence()` in class **edu.columbia.cpl.Recurrence** to test of activity of the time switch. That's why it is necessary to convert XML code of the time switch to an object of class **Recurrence**.

The Structure of Class **Recurrence** is summarized in Figure 3.

Recurrence
int[] byXXX rules Calendar dtStart Duration duration int frequency int interval Calendar until ...
boolean isInRecurrence() ...

Figure 3: Structure of Class **Recurrence**

3 VHE User and Service Profiles

Virtual Home Environment (VHE) is defined as a concept for personal service environment(PSE) portability across network boundaries and between terminals. The concept of the VHE is such that users are consistently presented with the same personalized features, User Interface customization and services in whatever network and whatever terminal (within the capabilities of the terminal and network), where ever the user may be located.

The key requirements of the VHE are to provide a user with a personal service environment which consist of:

- personalized services;
- personalized User Interface (within the capabilities of terminals);
- consistent set of services from the user's perspective irrespective of access e.g. (fixed, mobile, wireless etc. Global service availability when roaming).

3.1 Profile Management Entities

The **Profile Manager** is responsible for the management of the users /subscribers.

As a “root” to the profile management the **User Record (UR)** denotes the end-user within VHE system. It contains all required information that can identify the end-user uniquely when she/he enters the system.

Once established a relation, the end-user has the option of selecting a **User Profile (UP)**. It is well imaginable that one user can have more than one profile, e.g. for private and business usage, or even more sophisticated. The enumeration of **UPs** can be provided through a prior arrangement with the VHE provider or it can be declared by the end-user on demand.

The **UP** defines a unique tuple of information elements that suggest preferences for the end-users environment and of information elements that state the preferences for the subscribed services set.

Time Switches make decisions based on the time and date an **UP** is being selected (activated) or not.

See figure 4.

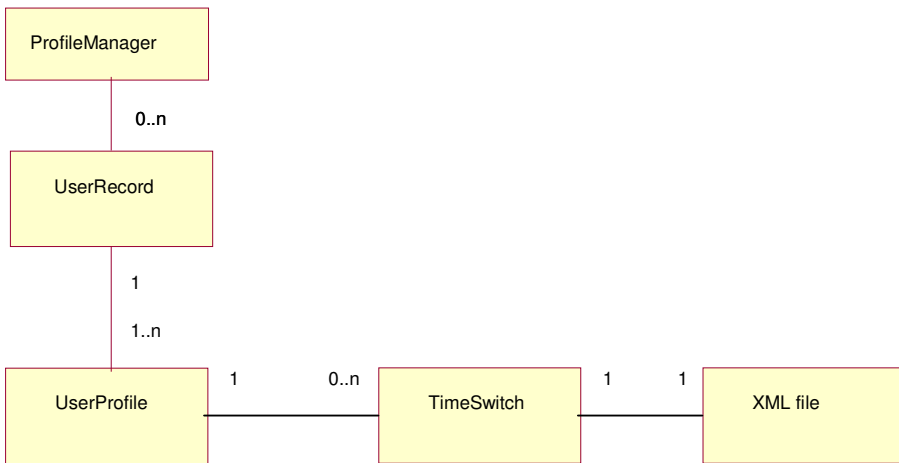


Figure 4: Profile Management Entities

3.2 Class model

Implementation (Example)

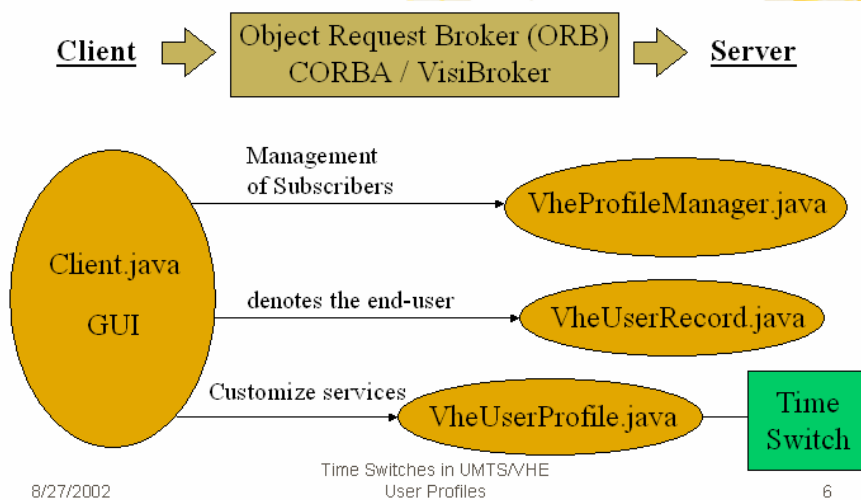


Figure 5: Class model

There are four classes on client site: *GUI.java*, *TimeSwitchFrame.java*, *TimeSwitch.java* and *Client.java*.

GUI.java contains the GUI. The User can create, manipulate and delete Time Switches. It's possible to show information about User Records, User Profiles and Time Switches. User Records and User Profiles cannot be manipulated, even though these methods exist. These methods wasn't tested very much because they don't belong to the abandonment of this project.

TimeSwitchFrame.java is called from *GUI.java* to edit and create Time Switches.

TimeSwitch.java: see chapter 2.2 Structure of class TimeSwitch

Client.java initializes the ORB, binds to the *VheProfileManager* and *VheUserRecord* objects and invokes the methods to communicate with the server site.

There are five classes on server site: *Server.java*, *TimeSwitch.java*, *VheProfileManagerImpl.java*, *VheUserRecordImpl.java* and *VheUserProfileImpl*.

Server.java does the following:

- Initializes the ORB.
- Creates a Portable Object Adapter with the required policies.
- Creates the profile manager servant object.
- Activates the servant object.
- Activates the POA manager (and the POA).
- Waits for Incoming requests.

TimeSwitch.java: see chapter 2.2 Structure of class TimeSwitch

VheProfileManagerImpl.java contains methods to:

- create and delete User Records
- show information about User Records

VheUserRecordImpl.java contains methods to:

- create, manipulate and delete User Profiles
- show information about User Profiles and User records
- change User Credentials

The Structure of Class **VheUserRecordImpl** is summarized in Figure 6. There are mentioned methods with direct influence on time switches only. For more information see javadoc documentation.

VheUserProfileImpl
existsActiveUserProfile() getActiveUserProfile() listActiveUserProfiles() ...

Figure 6: Structure of Class **VheUserRecordImpl**

VheUserProfileImpl.java contains methods to:

- manipulate User Profiles
- show information about User Profiles
- create, manipulate and delete Time Switches

The Structure of Class **VheUserProfileImpl** is summarized in Figure 7. There are mentioned methods with direct influence on time switches only. For more information see javadoc documentation.

VheUserProfileImpl
countTimeSwitches() deleteAllTimeSwitches() deleteTimeSwitch() existsActiveTimeSwitch() existsTimeSwitch() getAllActiveTimeSwitches() getAllTimeSwitches() getHighestPriorityActiveTimeSwitch() getHighestPriorityTimeSwitch() getTimeSwitch() setNewTimeSwitch() setTimeSwitch() ...

Figure 7: Structure of Class **VheUserProfileImpl**

4 Start the Program

There is a sample implementation with a graphical user interface allows creating and editing time switches for VHE user profiles and VHE user records.

To start this program do following:

- Install VisiBroker
- Start VisiBroker Smart Agent
- Add xerces.jar and jdom.jar to the classpath
- Start Server (*start vbj Server*)
- Start graphical client (*vbj GUI*)

5 GUI

Note: You can get information about User Records and User Profiles with this GUI, but there is no way to manage (create, delete, change, ...) this entities. These methods are implemented but not tested so much. You can use the methods with the textually client *Client_txt.java*.

Methods of time switches can be used completely.

5.1 Graphical representation of User Records, User Profiles and Time Switches

After starting the GUI you can see a window with two parts. The left part shows (after creating data) a tree with User Records, User Profiles and Time Switches. After selecting a node of the tree with the left mouse button information about this node are shown in the right part.

In the bottom of the window three buttons exist:

- Demo: creates data for demonstration
- Refresh: refreshes the tree
- Exit: Finishes the program

The Children of the root node are User Records. These nodes has User Profiles as their children. The nodes in the next level are time switches.

After selecting a node you can open an context menu with the right mouse button. The structure of this context menu depends on the kind of the node.

For more information see figure 8 and test the GUI.

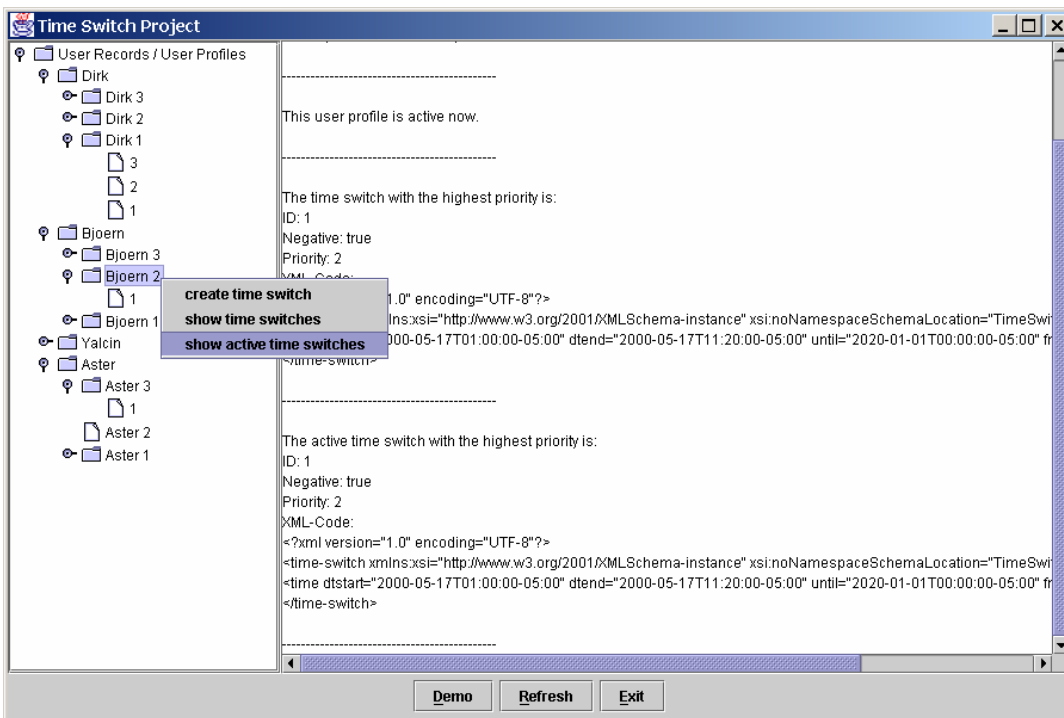


Figure 8: Structure of the GUI

5.2 Create a new or edit an existing Time Switch

The context menu of the User Profile node has an item to create a new time switch (“create time switch”). To edit an selected Time Switch node you can open the context menu and choose “show/edit”. In both cases a new window is opened. Because of the complexity of this window I want to explain important details.

For more information see chapter “2.1 XML structure” and chapter “2.2 Structure of class TimeSwitch”.

On top of the window **priority** and **negativity** are set. You can choose “End Time” or “Duration”, which specify the **ending time** or the **duration** of the period. The **start time** specifies the beginning of the first period of the list. **Until time** stands for the lifespan of the time switch.

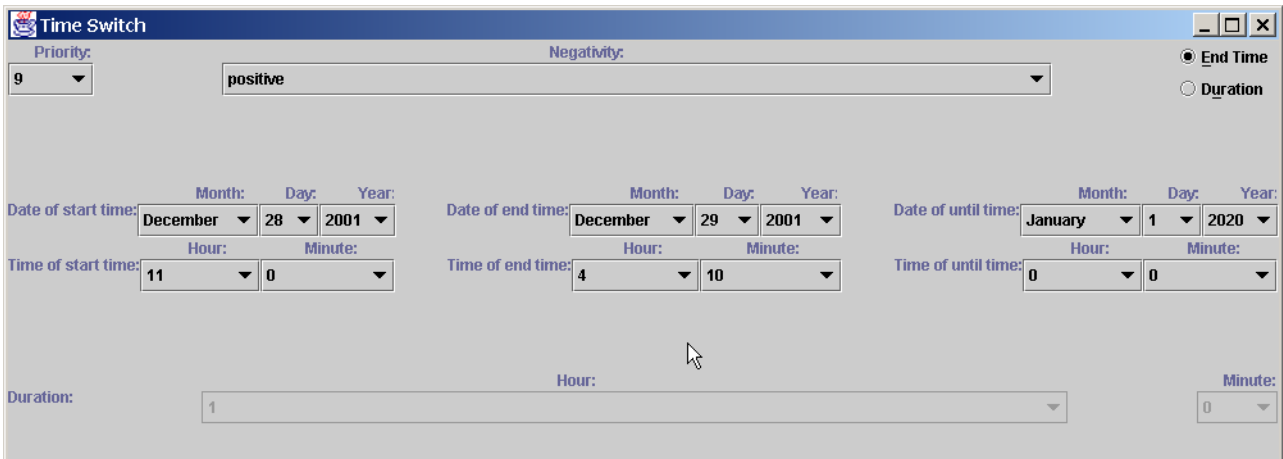


Figure 9: Upper part of window “creating/editing time switch”

Now you can choose the **frequency** the time switch:

- “Daily” specifies repeating periods based on an interval of a day or more
- “Weekly” specifies repeating periods based on an interval of a week or more
- “Monthly” specifies repeating periods based on an interval of a month or more
- “yearly” specifies repeating periods based on an interval of a year or more

Depending on the frequency you can set different **byXXX** parameter.

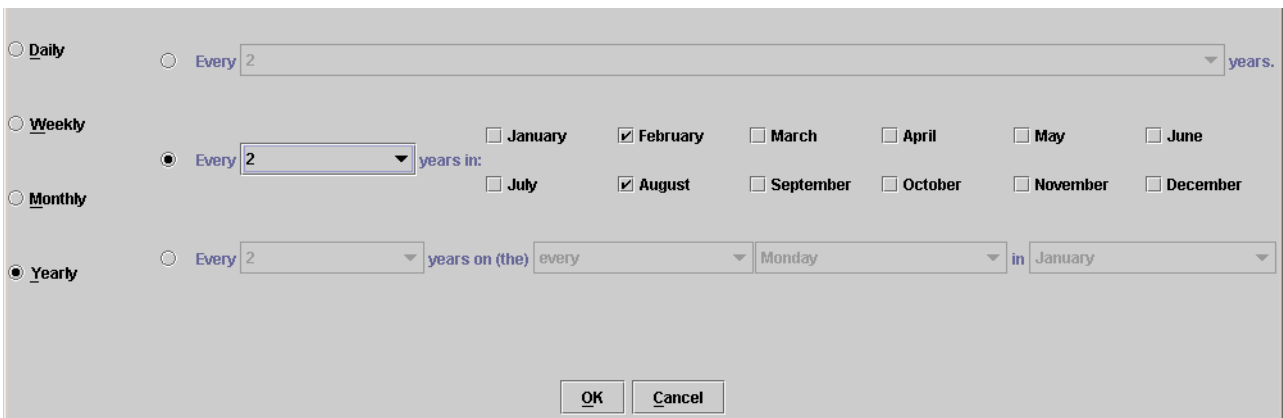


Figure 10: Lower part of window “creating/editing time switch”

5.5.1 About this example

When our time switch will be active? Do you remind of chapter 2.1? The XML structure of this time switch is:

```
<time dtstart="2001-12-28T11:00:00-05:00"
dtend="2001-12-29T04:10:00-05:00"
until="2020-01-01T00:00:00-05:00"
freq="yearly" interval="2"
bymonth="2 8 "
wkst="MO" />
```

The time switch is active “on 28th in February and August from 11:00:00 AM to 04:10:00 AM, every other year”.

6 Bugs

The activity of a Time Switch isn't correct, if a yearly frequency is used in combination with byday rules with an integer unequal zero (e.g. +1MO). This error occurs because there is a bug in Java code of the CPL time-switch (Columbia University) [2]. The author of the code is working on a revision.

7 To Do

There is no way to use time zones in the current state of the project. The XML file of a Time Switch contains two optional parameters, **tzid** (Time Zone Identifier) and **tzurl** (Time Zone URL), both of which are defined in RFC 2445 [3]. This parameters are insignificantly at this stage of the project. A future implementation should enhanced the method **isActive()** in class **TimeSwitch** in that way, that it convert actual time in local time. Then Time Switches with different time zones can be evaluated.

8 Author's Address

Björn Schünemann

TU-Berlin

e-mail: schueni@cs.tu-berlin.de

9 Bibliography

- [1] Lennox / Schulzrinne, “CPL: A Language for User Control of Internet Telephony Services”, <http://www.cs.columbia.edu/~lennox/draft-ietf-iptel-cpl-06.pdf>
- [2] Jonathan Lennox, “Cal-Code: Java code for CPL time-switches”, <http://www.cs.columbia.edu/~lennox/Cal-Code/>
- [3] F. Dawson and D. Stenerson, “Internet calendaring and scheduling core object specification (icalendar)”, Request for Comments 2445, Internet Engineering Task Force, Nov. 1998, <http://www.ietf.org/rfc/rfc2445.txt>