X Semantics in the field of widgets: a case study in public transportation departure notifications

Alena Kovárová and Lucia Szalayová

Faculty of informatics and information technologies, Slovak University of Technology, Bratislava, Slovakia

Faculty of Information Technology, Brno University of Technology, Brno, Czech Republic

Abstract. Widgets are becoming increasingly present in our everyday routines, which makes their portability and reusability desirable properties. As a particular example, we consider public transportation passengers who are extensively using the internet to make their lives simpler. In order to minimize the time spent at the bus stop, they use to check their bus line departures on the web before they leave their homes or offices. For this purpose, there exist several internet portals providing information on local transportation time schedules. This chapter starts by presenting a better way (quicker and easier) of obtaining the same information – using an adaptive desktop widget with comfortable user interface. The second step is the utilization of the semantics of considered data in order to make the widget portable through different data sources of the same domain.

X.1 Introduction

Due to the continually growing volume of information that is made freely available online, people often find themselves in the inconvenient situation where they have to invest disproportional effort and time in order to interact with the information sources they use. Everyone subconsciously or consciously estimates how long it will take to obtain the desired information and more importantly whether this information is worth this time and effort.

This process includes for example decisions such as which electronic newspaper to read, which sports section to monitor, which broadcast to watch, which web pages contain relevant information and so on. This is of course a daily struggle; most of us would appreciate the time-saving and effort-saving option of having this "personalized" information wait for us somewhere nicely aligned. To come as close as possible to this vision, we come to the point of choosing a favorite newspaper, favorite channels and programs, favorite web pages; simply said: favorite

information sources. But this is still not enough; even within these favorite sources it is still needed to search and to filter. This simply reflects the fact that the majority of the available information sources are built for the masses and therefore do not have any implemented personalization / personal adaptation features to serve the needs of each individual person.

The abovementioned quest for information can be divided into three distinct types, which refer to a case in which someone is searching for:

1. general knowledge (whether in an unknown or a known field)
2. specific information in an unknown field
3. specific information in a known field

In this work we focus on information quests of the third type. This means that a user is interested in specific information from some known area and that he knows where to search for it and how to filter the information that is available at that location; the user already has a favorite source for this information. In other words, in our case the user is able to formulate his requirements in greater details and to be explicit. Examples of such requirements could be: “I want to monitor this specific list of stocks on the stock-market and I have no interest in the fluctuation of other stocks or of the general index.” or “I need to have the current weather forecast for the city where I live and I prefer to have it in textual and image form.” While this is a known area and the experienced user knows where and how to find (manually) the information he is interested in, the problem that remains is how to transform such a requirement into a computer language so that a computer can look for the information (automatically) instead of the user.

To understand this problem practically, let’s have a look at a specific case, the average morning of a “John”. John looks for information on bus departures from home to work. It takes John a little bit of time till he opens the relative web page in his web browser – it always takes John a few seconds to perform this task. The time depends on the degree to which John is capable of customizing the system he is working with and also on how much different settings will allow him (if they exist as an option) to speed up obtaining his desired information. Our goal is twofold: to minimize this time and to relieve the user from the manual customization of the information source.

Clearly John's (and also our) requirement can be formulated like this: "I want to know, when my bus is going from where I am now and in the usual direction." It is important to notice words "my", "where" and "usual" because these assume an application is able to estimate the number of his bus, where he is and which direction he wants to travel. Once an application fulfilling this requirement is developed, a second question rises: "If John would move, could he still use the application?" If the widget worked at a semantic, rather than flat information, level, then that kind of portability would be possible too, allowing John to continue using the tool he is accustomed to, even when his own circumstances and context change. The design and development of such a tool is the objective of this chapter.

The remainder of this chapter is structured as follows: Section 2 contains a brief survey of different solutions for the retrieval of desired information via browsers and widgets. We point on their pros and cons in view of our purpose. Section 3 describes our widget starting with possible data sources, through system overview and widget basic functionality. We also explain widget architecture and give a closer view at its data model. Section 3 is closed by widget evaluation. Section 4 deals with semantics and the corresponding ontology model, which could make the widget independent on data source. We compare our model with other ontology models, which belong to the same area, but they are based on different requirements. Section 6 lists our concluding remarks.

X.2 Related background

Let's have a closer view of the user's possibilities of

- searching,
- filtering,
- retrieving the web data within specific site,
- customizing web-application for his own benefit
- how to obtain some web page the quickest way

It is the same for any kind of problem of the third type – when the user knows where to search and how to filter. So our first question is: What is the usual way to obtain information from Internet? Omitting the highly specialized web-applications, it is the well-known browsing.

X.2.1 Traditional access to resources on the web using web browsers

The most used are *internet browsers* e.g. Microsoft Internet Explorer, Mozilla Firefox, Safari or Opera. And how can the average internet browser save time? The user can set up some settings e.g. to save his favorite web page via “Add to Favorites”, to make some pages as his “home-page”, to “Show the windows and tabs from the last time” when internet browser starts. Such settings allow the user to set up different things about web pages but there is no possibility to specify or to ask for specific information within the web page (if the user wants just a part of the page). A little improvement brought Microsoft Internet Explorer 8 with Web

Slices, which use simple HTML markup to represent a clipping of a web page, enabling users to subscribe to content directly within a web page[1].

To move closer to user needs, next to generic internet browsers we find *site specific browsers*. This type of browser is designed to create a more comfortable environment for the user, especially when browsing "the favorite" sites e.g. for e-mails or on different types of social networks. Examples of the site specific browsers are: Fluid (for Mac OS X), Mozilla Prism, Google Chrome or Bubbles. They are web-applications, which have the same core as web browsers but from the outside they look like desktop applications. They offer drag & drop function and have many other nice features, maybe they have some settings, which can be manually set up and then the user can obtain his information even quicker as in web browser, but they still do not guess user's focus, do not give a chance to filter (specify which part of which web page) and do not offer the way of presentation.

Apart from bookmarking systems built-in web browsers, users can take advantage of *bookmarking web services*, such as social bookmarking system Delicious[2] (formerly del.icio.us). Such services provide them with the possibility to organize their bookmarks by using tags and to have their bookmarks available independently of user's location and browser.

Another option, which can significantly speed-up user access to relevant information are *personalized and adaptive web-based systems* [2], especially when combined with site-specific browsers. Appropriately trained personalized web based system can often display the information the user is looking for directly on the first page.

X.2.2 Widgets

Without implementing own engine or robust system, a chance for solving our problem could be found between *widgets* (sometimes also called *gadgets*). In our context, they are not some elements, which help the user to navigate or to orientate or to pick a choice, but they are single-purpose mini-(web-)applications, which typically have a minimal size and are dedicated to bring simple solution based effect while a user is working with a computer. Their functionality is oriented to one, specific goal – to display very specific information. They can be of two types, either for the web (web-widgets) or for the desktop (widgets) [3]. The latter one can be for computer as well as mobile devices [1]. In this work we focus on the desktop widget for computers, which can be freely located and easily combined within the desktop. Most often used engines for widgets or gadgets are:

---

[1] Internet Explorer 8: Features – Web Slices http://www.microsoft.com/windows/internet-explorer/features/easier.aspx

[2] Delicious – social bookmarking http://delicious.com/

- *Konfabulator*[3] from Yahoo! for Windows XP+ and MacOS
  - known as Yahoo! widgets[4]
- *Windows Sidebar* from Microsoft for Windows Vista
  - sidebar with gadgets on Windows Vista desktop
- *Google Desktop Gadgets*[5] from Google for Windows XP+
  - in a form of Google Desktop
- *Opera Widgets*[6] from Opera for Beta MacOS 10.5 and Windows XP+
- *Dashboard*[7] from Apple for MacOS 10.5
  - as the 2nd desktop with widgets
- *Joost Widgets* Joost 1.0 Beta Mac OS 10.5, Windows XP, Windows Vista

Most of them use a kind of API which processes mainly HTML, JavaScript, XML and CSS files.

There are some differences between different enterprises of widgets and gadgets for desktops. From the user perspective some widgets are represented by views or icons which are located in a standard sidebar of the desktop and the widget become active only after click initiation where the icon spreads itself to the desktop. After this the widget can be relocated as wished. On the other hand some gadgets have almost double sized sidebar wideness as the widgets where gadgets are providing the service during all the time of activeness. After clicking on it gadget spreads itself and increases the service quality or quantity whereas relocation is limited within the sidebar.

From an implementation point of view there are three possibilities for the user on how to have their own personal widget. The user should first decide which API he wants to use and if it is not already a part of his system or application, he needs to install it. Then those three choices are:

1. To find it on the web page with plenty of complete widgets, download it, manually set up it and use it.
2. To read a tutorial for extending a generic widget and follow simple instructions to created a specific one.
3. To read a tutorial for developers and program their own widget.

Which of these three will be chosen is highly dependent on the type of information, which should be displayed (the way of displaying is not now taken in to account). Just like the site specific browser, the complete widgets cover the demand of the majority. Therefore non standard requirements are not covered by the first choice. If there is already a service as an RSS or a web-service, which can be requested for information, the second choice is sometimes enough. But in case of

---

[3] Konfabulator, Reference manual, Version 4.5 http://manual.widgets.yahoo.com/

[4] Yahoo! widgets, http://widgets.yahoo.com/win

[5] Google Desktop, http://desktop.google.com/index.html

[6] Dev.Opera,, http://dev.opera.com/articles/view/creating-your-first-opera-widget/

[7] Dashboard widgets for Mac OS X Dashboard, http://www.apple.com/downloads/dashboard

non-existent complete widget or service, the only choice is the third. The last one also gives a space for developer to implement some features, which would offer to the user some kind of personalization. But generally there is no effort to implement widgets for one purpose with a broad usage (i.e. independent of information source); those, which obtain information from Internet, all are exactly one site or exactly one web-service oriented. It is because there is no standardization for these sites or services, which would be applied in such widget.

X.3 Public transportation departure widget

Based on the survey presented in the previous section, we can implement a widget, in order to John's request: "I want to know, when my bus is going from where I am now and in the usual direction." Widget technology is suitable for this purpose, while the request needs a very little space of user's desktop to display relevant information - the closest departures of chosen (guessed) stop, direction and line from public city transport. This tiny desktop application is mostly suitable for laptop owners (where the mobility can increase the need of extensive transportation) as well as for any computer user who is interested in his/her favorite line schedules.

Here and in the following section we explain what the needs of our user are, which the features of the widget fulfilling these needs are and how they work together. Finally, we tackle the two main related theoretical questions: "Would it be possible to use metadata describing data semantics in order to make the widget independent of an information source?" and "What should the ontology model look like?"

Our first key point was to look for suitable information sources (to show, they are not good enough and to choose one of them as our data source) and the second is to gather user's requirements for the application.

X.3.1 Sources of public city transport departures in Bratislava

There are three well-known web sources of public transport information for Bratislava. In following lines are shortly described all of them with emphasis on user possibilities.

The very first source is the web site http://www.dpb.sk [4]. This web site is administrated by the public transportation provider for the area of the capital city Bratislava in Slovakia. Process of reaching information (there are only timetables with departures) is relatively complicated and there is required a manual action – there are six steps needed within the browser. Thus, this source is not very favored between users. More over, it is not possible to personalize these pages.

The second solution can be found within the web site http://www.imhd.sk [7] (imhd). This site is probably the most used. There is for example a useful feature where the user can search also the stop to stop combination. An attractive service of the imhd is an email notification possibility - where actual changes, exclusions, news and useful information can be provided. Personalization possibility is very limited – thus, searching for relevant information is not brisk.

The last and the most recent source is web site from http://www.cp.sk [5], what is the National information system of timetables for Slovakia. This web site offer all kind of timetables for trains, buses, flights and different public city transports within Slovakia. Taking in to account only public city transport, the user can find his route by setting the starting stop, the last stop and time of departure or desired arrival. The connection is found within interchanges, but user can ask also for direct connections only. The other choice is to get the entire timetable for one line at some stop for set date or to get the schedule for one bus and its route. The only possible personalization is to save the displayed page as the favorite one.

From previous lines it is clear, that there is no service, which would give us the required information on demand; there are only different web sites. Since our requirement is so specific, we had to choose the third choice: to program our own widget. Evaluating the pros and cons of different widgets APIs we have decided to implement the widget using the Konfabulator and we chose imhd as a data source for our widget, while it had structured html code good enough to parse it to our database.

X.3.2 *System overview*

The idea of widget with line departures shall not substitute any of above mentioned information sources. To explain the difference closer, imagine a following scenario: John is at work. He knows which buses stop next the building and knows which one is suitable for him. But he does not remember its departures and just wants to know what the closest time his bus comes is, because he does not want to stand on that bus stop for ages. Of course, he does not want to browse internet, where he either has to click many times or has to fill some input boxes always with the same strings. He used to print out the entire timetable for his bus, but he always needed to check time and search for relevant value in paper. John is not interested in transfer between lines, he does not search for the quickest or the cheapest route. He does not need to know, when he will arrive to his destination.

As we already mentioned, our two goals are to minimize time/effort and manual customization, in other words, we want to fulfill John's requirement the way, which would minimize the number of his actions and accelerate the access to the information. The widget, which can follow this, has to have at first some input and output. Example of input is when the user chooses a number of a line. This input is continuously monitored, what enables our widget to adjust to the user. The output

is displayed to the user - view. Our output is desired departures, which are loaded either from a local database, or downloaded from a web. When downloading is induced, new data are stored in local database. The last case for user is the possibility to set up predefined locations (Fig. X.1, upper part), that enables the user to adjust the widget from the first touch.

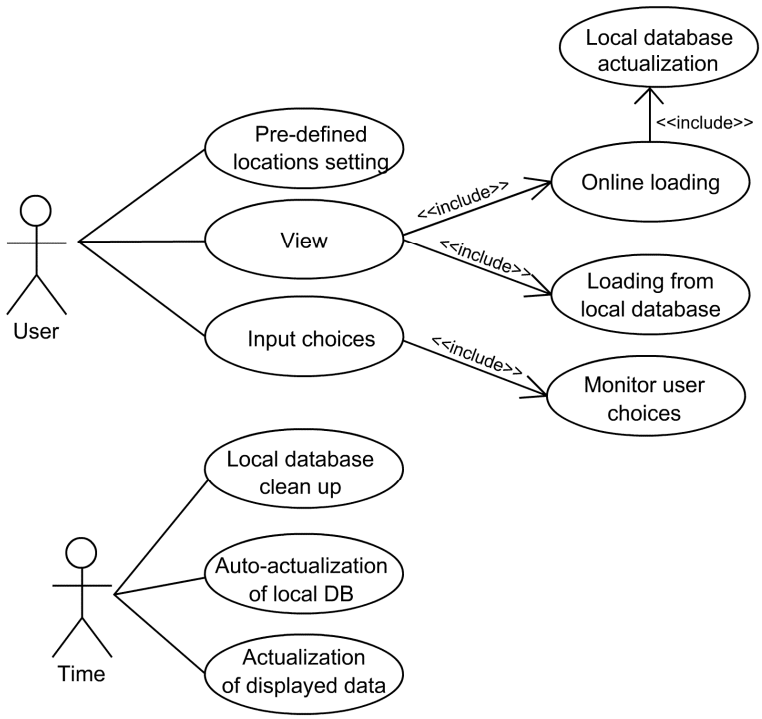

Figure X.1. Use case diagram of widget system

As departure schedules are from time to time changed, these changes need to be translated into the local database update to provide the user with the most up to date information. This updating process can run automatically every week, but the user can at any time, switch off this updating. There is also case for automatic clean up to erase data which are not used and are old. And the most important is to keep fresh data in displayed area – current departures, what is the last case of time actor (Fig. X.1, lower part).

X.3.3 *Widget Basic Functionality*

Basic widget functionality is to display the upcoming five departures of selected line from the chosen stop in a set direction (Fig. X.1, case Input choices). To get this, the user has to go through three steps, which should be done in proper and intuitive order:

1. Select a line number – from a list within the dropdown-menu (Fig X.2, point 1), selection is needed only if the user does not want the automatically chosen.
2. Change a direction – simple click (Fig. X.2, point 3), needed only if the widget wrongly proposed the inverse one
3. Choose a stop – from a list within the dropdown-menu (Fig. X.2, point 2), shown are only those stops which belong to the previously selected line. This selection has to be done only if the automatically chosen stop is not the wanted one. In the case of the first-time line selection, the first stop of selected line is pre-selected.

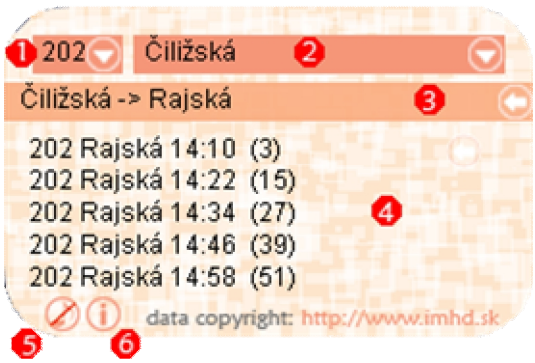

Figure X.2. Widget description

After these three steps, whether they were done automatically or by the user, the upcoming five departures are displayed (from current time). The widget displays exactly: line number + direction + departure time + time-left in minutes (Fig. X.2, point 4). To have current data at any time, actualization is performed every minute.

To alleviate the user from permanent time checking – how many minutes remains to a departure - we implemented also one extra feature – sound. The widget can announce the time of the next departure e.g., "Next bus arrived at 12:00. That is in 3 minutes." Of course, this function can be turned off (Fig. X.2, point 5).

Finally, every application should have a Help (Fig. X.2, point 6). Our Help contains a user manual.

To make it more user friendly, we gave the user the possibility to set up his favorite locations manually (Fig. X.1, case Pre-defined location settings): The user can for every location choose several lines (with respective stops and directions), which he usually travels with, for example from school or office. The user can name it e.g., route "school->home". The output is the same as within the basic functionality, only the upcoming five departures differ in line number and name of stop. Departures are ordered in the usual way – according to time of departure (Fig. X.3)

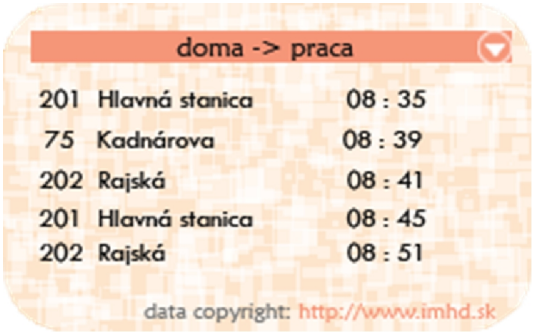

Figure X.3. Widget setup for multiple lines within one route (in Slovak language, translation of route: Home -> Work)

The last of the basic widget functionalities is widget ability to adjust to the user's needs. As we do not use any other information sources (e.g. browsing history) to find out what are user's usual bus stops and bus lines, the widget has empty database (except default data) at the beginning. While the user uses the widget, it monitors his choices and stores number of selection of each choice in the local database (together with downloaded data). Finally, the most often chosen option can be pre-selected automatically and thus accelerate the service access.

X.3.4 *System architecture*

We chose the Konfabulator as an engine for our widget. It means, we used mainly XML and JavaScript for programming and supported SQLite for our local database. Our system can be divided in following parts (Fig. X.4):

GUI – Graphics User Interface, which use to send data (user choices) to the Task manager and according to them can ask the Task manager for new data from local database. The GUI can also send information about user's choices to User profiler

The User profiler updates in database the number of user's selections. And remember the user's settings including his favorite locations / routes. Anytime the user chooses a line number, stop or direction, its relevancy raises.

The Task manager

- updates GUI (departures) either because of time or user's different choice,
- updates the local database (data downloaded from Public transport information provider, if there was an Internet connection) and
- cleans up the local database - due to performance optimization the Task Manager will erase the least selected lines out of the database in certain period

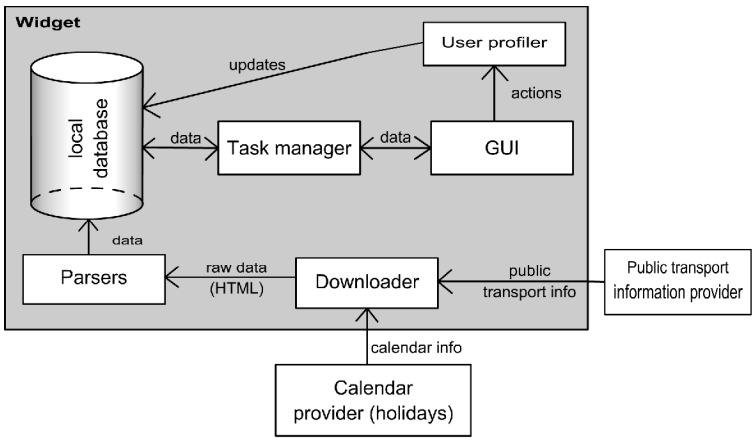

Fig. X.4 Conceptual architecture of the public transportation departures widget

The Downloader downloads entered web page, therefore it is needed an Internet connection, when user wants to download new time tables or a new calendar.

An input of the Parser is raw data (HTML code of a web page), which is parsed and stored in respective columns of the local database – wherefrom it will be loaded for the user as requested.

X.3.5 *Data model*

To parse one web page takes several seconds, what was contrary to our goal. Therefore we needed to store the data in our local database. The most important is to store lines, their stops and departures for terminal stops. While there is a difference in timetables depending on day type, we enlarged our database with two small separated tables – public and school holidays (Fig. X.5).

The line table contains data about the line previously loaded by system. By lines there is a learning ability applied - so one of the attributes is used to specify the incremental value of line selection count.

The line stops table is loaded by data parsing of the left part of the schedule list. It contains information about stops of a respective line and time lag between each two upcoming stops in a route. Here is the learning capacity of the system done by

incrementing the station selection count - selection of the station for specific line and direction.

The departure table, in database, represents departure times out of the base station - so the time of arrivals for specific station is calculated using the initial departure time and summary of time lags until the desired station. As departures are differentiated based on the actual day (working day, weekend, public holiday or school holiday) this feature is taken into consideration.

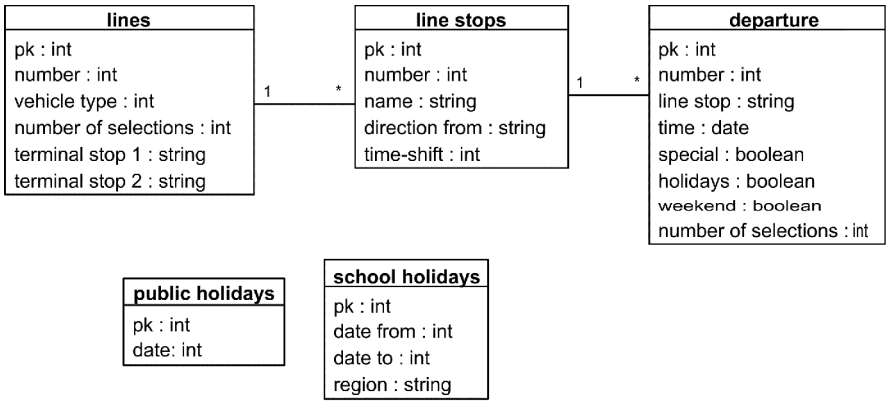

Fig. X.5 Logical data model of the widget database

Previously mentioned day differentiation is being done by recognizing a week day (working or not) whereas a special feature for recognition of public or school holidays is represented within separate tables with these special days. A list containing the school holidays is updated yearly - this list can be gathered in the site of The Ministry of Education in Slovakia[8].Attribute region is necessary, while school holidays in our country differ on the basis of it.

X.3.6 *Evaluation*

Evaluation was done among the students of the Faculty of informatics and information technologies of Slovak University of Technology in Bratislava. Tests were performed by 10 volunteers who use the computer on a daily basis.

Their task was to download a new line (of public transport) in the application to display departures for one of its stops. By starting the widget, instruction guidelines were displayed, but were usually skipped by the testers. As testers realized during their first attempts that the widget displays only one default line, guidelines were used to get the information on how to extend the widget's functionality.

[8] The web site of The Ministry of Education in Slovakia, http://www.minedu.sk

Overall, it took generally less than three minutes for users to find the desired link information. Testers observed the specific feature of the application - due to data parsing after the URL was set – that it was not possible to influence the widget for a moment. This feature has been previously well documented also within the guidelines.

One special feature of the widget is sound – the widget can announce the time of the next departure. This feature was also tested (the speech was realized by using the Windows functionality of automatic reading of given text). This voice functionality was evaluated as being very popular by the users, whereas the widget was rated in a very positive way as a whole. No negative features were found. Testers came out with one recommendation: to display departures in centralized printout within the frame.

The system has been implemented according to its design. During the implementation several traps occurred. One of the most complicated was not well-structured HTML code of imhd pages. Pair tag rules were many times broken, what forces us to deep study of the source code. It was necessary to identify key points within the HTML code which were used to identify the load sections. This way is complicated for the implementation and execution as well. Due to this fact implementation of automatic data updating has not been implemented – to update database (departures of a few lines) would take several minutes and during this time widget would be out of order. Due to this fact updating can be done if initiated by user in the same way as adding a new line.

X.4 Extending the widget with semantics

Coming back to question posed earlier in this chapter: "If John would move, could he still use the application?" It would be suitable, if our widget would work although it will have different information source with the same type of information – line departures. This idea assumes that the provider provides data also with their semantics. Such providers are very rare as well as widgets working with such data; more often are web widgets e.g., in project of Eetu Mäkelä with colleagues [6]. But the principle is the same, so we created our own ontology model (Fig. X.6) to represent the semantics and relations within data we are working with – parsing, storing and displaying in our widget. This ontology model includes all three main tables and their attributes from our data model.

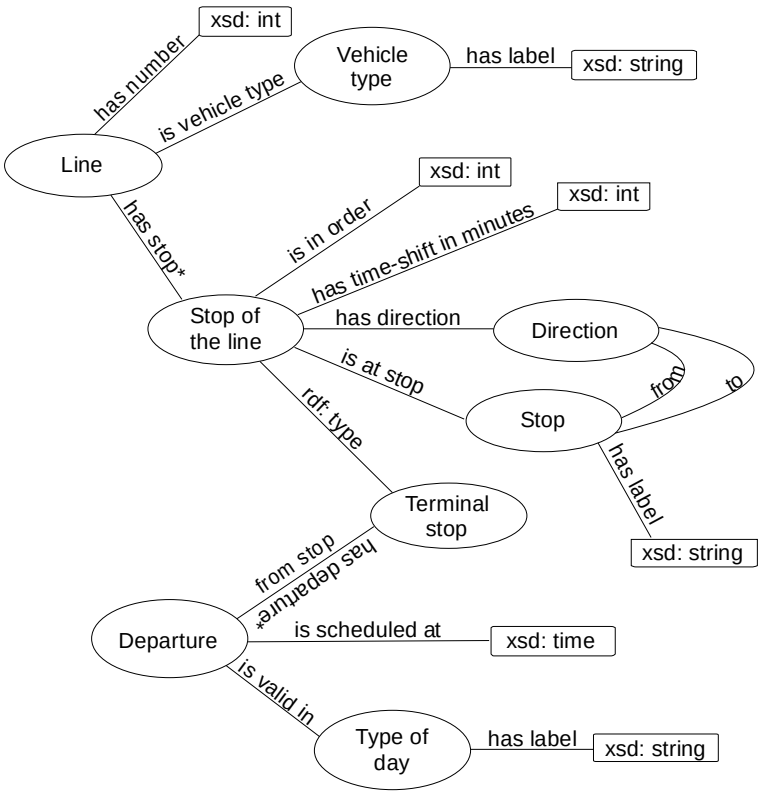

Fig. X.6 Ontology model of data from public transportation departures widget

To check the compatibility with provider, let's assume that the provider provides the same model as presented Junli Wang and his colleagues in their work [8]. Their model is not meant for widgets, but it also deals with public transportation. Their purpose is oriented on public transport query as transfer trip scheme, route query and station query. That is a wider range of public transport domain than ours, thus also their ontology model is wider (Fig. X.7). Omitting the concepts, we do not use in our model, and leave the same ones out, it is noticeable only one serious difference: the concept of route with its timetable. We do not have anything like this in our model, while we can calculate it from departure from terminal stop plus time-shift to selected stop. Our model expect the timetable of departures (from terminal) without knowing the last stop, but their always need to have set the first and final one. This leads us to two conclusions. The first one is, our widget would not work on their ontology model unless we would reimplement our widget, and the second is that our ontology model is better, since we used departures, what is semantically lower concept than route – route can be easily calculated from departures.

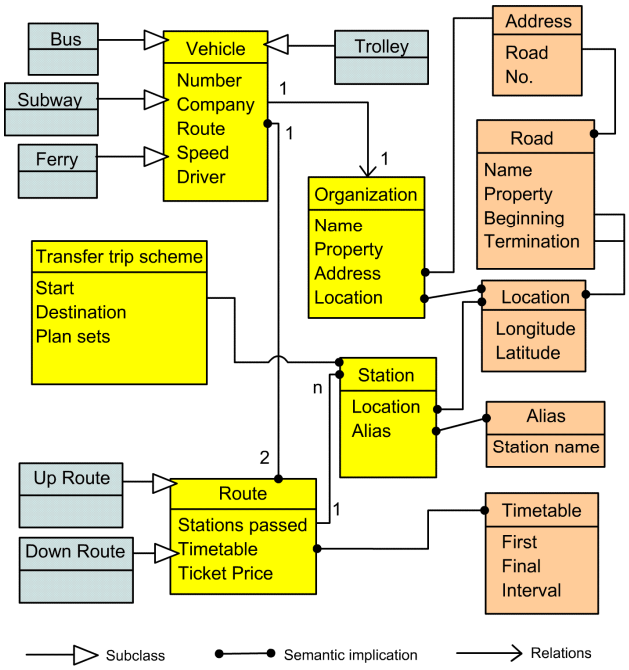

Fig. X.7. Urban public transport ontology [8]

X.5 Conclusions

It was already well known that it is possible to implement a widget (as a client), which downloads and parses data from some web source (server side). Moreover, such a widget can be personalized, because it can adjust itself to best serve the user, thus making the retrieving of information more comfortable and quick. This accommodation is achieved by monitoring the user's choices and storing the number of selection for each choice in the local database. The only one disadvantage is that such widget is totally dependent on the data source. In this chapter, in order to make such widgets portable through different web sources in the same domain, we proposed the creation of an ontology model which can reflect data semantics.

We created such a model and compared it with an other one from the same domain but with a different purpose. The comparison showed that the two ontological models differed in the main concept. This conclusion implies that although it is useful to use semantics in the widget (as in any other client application), it will work only if the server provides data with the same semantics.

Regarding further applications of the work presented herein, the widget could take a benefit of such semantic model which could be applied also in other kinds

of systems with regular departures e.g., logistics or catering. In the same time, our ontology model can be extended so it would serve also for other purposes e.g., route planning.

Acknowledgement: This work was partially supported by the Scientific Grant Agency of Slovak Republic under the contract No. VG 1/0848/08.

References

[1] Boström, F., Nurmi, P., Floréen, P., Liu, T., Oikarinen, T., Vetek, A., and Boda, P. Capricorn - an intelligent user interface for mobile widgets. In: Proceedings of the 10th international Conference on Human Computer interaction with Mobile Devices and Services. MobileHCI '08. ACM, New York, NY, pp 327-330. (2008)

[2] Brusilovsky, P., Millán, E.: User Models for Adaptive Hypermedia and Adaptive Educational Systems. In: P. Brusilovsky P, Kobsa A, Nejdl W (Eds) The Adaptive Web, LNSC 4321 Springer (2007)

[3] Caceres, M. Widgets 1.0: The Widget Landscape. W3C. http://www.w3.org/TR/widgets-land/ (2008) Accessed 17 September 2009

[4] Dopravný podnik Bratislava, a.s (company, provider), Public transportation for the area of the capital city Bratislava (web site), http://www.dpb.sk Accessed 17 September 2009

[5] INPROP, s. r. o (company, provider), National information system of timetables for Slovakia (web site), http://www.cp.sk/ Accessed 17 September 2009

[6] Mäkelä, E. Enabling the Semantic Web with Ready-to-Use Web Widgets Export. In: Lyndon J. B. Nixon, Roberta Cuel, Claudio Bergamini (Eds) Proc. of the First Industrial Results of Semantic Technologies Workshop (FIRST'07). pp. 56-69. (2007)

[7] mhd.sk (citizen union, provider), imhd.sk (web site of public transportation for the area of the capital city Bratislava), http://www.imhd.sk Accessed 17 September 2009

[8] Wang, J, Ding, Z., Jiang, Ch.: An Ontology-based Public Transport Query System. In: Proceedings of the First International Conference on Semantics, Knowledge and Grid table of contents. IEEE Computer Society. pp 62-64 (2005)