BorchelReinertzHumbert2005

Peter Micheuz and Peter Antonitsch and Mittermeir, Roland Ed.

Innovative Concepts for Teaching Informatics. Informatics in Secondary Schools: Evolution and Perspectives

Ueberreuter Verlag Wien 2005, pp. 53--63

Design of an Informatics System to Bridge the Gap Between Using and Understanding in Informatics

Christiane Borchel[1]
Ludger Humbert[2]
Martin Reinertz[1]

Didaktik der Informatik – Universität Dortmund [1]
Fachgruppe Informatik – Willy-Brandt-Gesamtschule, Bergkamen [2]

Abstract. Informatics nowadays is a secondary school subject. Some contributions consider the object-oriented thinking pattern particularly useful in teaching informatics to 6th graders. Based on this approach, students learn to analyze documents and, as a consequence, reflect upon their structure—a process that is guided by instructors who teach the »deconstruction« of documents/products (i. e. how to break them down into well-known elements, such as sections and characters, with their respective attributes and methods). To actually carry out the changing of attributes and observe this procedure's effects, students commonly use standard text processing systems. The link between the more »theoretical« construct and the concrete document then, is established by making use of the actual text processing software's menus. This way, students are provided with a model that functions as a guideline for using concrete software.
However, this approach is limited by design. Students never have an opportunity to build documents from formal descriptions. Consequently, neither do they get a chance to predict / hypothesize about attributes and methods given, for instance, to a character or section. Modern object-oriented systems allow the programmer to look at such elements—by »asking« an instance for the messages you may send to an object, by taking a closer look at the values of attributes, etc. For all these reasons, we decided to create a prototype meant to fill the gap between the object-oriented concept of deconstructing given documents on the one hand, and constructing documents in an object-oriented manner on the other—and hence offer students a new level of insight into the topic.

1 Basics

Schools are supposed to fulfill their mission of education by taking up informatics contents into the range of compulsory subjects (cf. [1], [2]). However, mere computer literacy (cf. [3]) should not be the only goal of informatics as a secondary school subject. A »computer driving license« as the aim of informatics education which only documents a learner's knowledge of and ability to use standard software would never yield any long-lasting, product independent and comprehensive problem-oriented abilities or skills—not to mention informatics literacy (cf. [4]).

On the other hand, in any kind of basic informatics instruction—theoretically well-based as it may/should be—there always has to be an obvious link to »real« systems and

their range of application, or more generally: the area of applied informatics (cf. [5]). It is this very field of tension (theory vs. practical experience) that informatics as a school subject is situated in. Consequently, this issue is also the starting point for hot discussions surrounding the contents of basic informatics instruction, i. e. the fundamentum for teaching informatics in secondary schools. A major question that arises against this background, then, is this: What might theoretically well-based, yet at the same time practically oriented informatics instruction look like? (cf. [6])

1.1 Informatics Education—a Structure

At a conference in Dagstuhl (2004), the Focus Group »Educational Standards of Informatics« of the GI-Dagstuhl-Seminar »Concepts of Empirical Research and Standardisation of Measurement in the Area of Didactics of Informatics« defined certain key elements to create a didactic framework for educational standards concering informatics in secondary schools. Eventually the group formulated content lines which are supposed to provide a structure to informatics education.

- information and data
- algorithms
- informatics systems
- technology
- theory
- society

Formal languages, models and structured decomposition are not represented as lines (i. e. categories) of their own—yet each of these aspects is included in several of the other content lines listed above. In addition, the group worked out the following process lines.

- problem solving and modeling
- interpreting and reasoning
- communication
- connections
- representation

If one moves down from these rather theorectical considerations to concrete elements taken from hands-on experience, there is (among others) one major component which students work on in the course of their informatics eduaction: documents. This element offers a promising connection between modeling (as an abstract process) and implementing the modeled structure. Certainly, this view does not originate from informatics as science, but rather represents a user- or learner-oriented perspective: after all, wordprocessors, spreadsheet applications, hypertext editors and imaging software are all fairly common standard applications (i. e. well-known informatics systems) and they all create various kinds of documents. Students usually benefit from any prior knowledge they may have gained early on in their lives, about the use of such systems; especially as far as other subjects are concerned. Documents and their components can be conceived of as instances of pre-defined classes which, by their very nature, also provide methods and attributes—a perspective that is much more theoretical and scientific than a mere focus on documents as products of standard software.

As Voß indicated in [7] an object-oriented, theoretical approach can also be extremely rewarding with respect to the acquisition of skills and abilities surrounding the creation and manipulation of text documents: in her study, those students who had been

able to adopt an object-oriented view on text documents proved to be much more confident with regard to handling text documents on a general level. More confident, that is, than students who had only been provided with instructions on how to use the word processor or, in other words, which buttons to click on in order to trigger the desired functionality.

In addition to the obvious success of approaches to teaching »word processing« that are based on a combination of theory and practical experiences, there is yet another genuinely informatic link between these two areas: a careful, intuitive transition from object-orientied modeling (OOM) to object-oriented programming (OOP). After all, why should it not be a feasible undertaking to take methods that have been introduced on the blackboard (and in the students' notes) and put them into tangible reality with the help of an informatics system specifically designed for this purpose?

1.2 *Ponto*—an object oriented approach

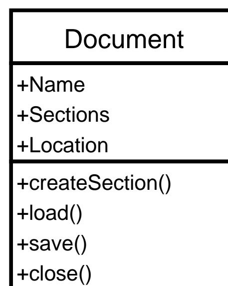

Fig. 1. class diagram *Document*

Informatics instruction frequently makes use of class diagrams like the one depicted in Fig. 1. It is certainly a valid assumption that, if students are able to understand and make use of these diagrams 1, they are also able to invoke *Ponto's* constructor method for the *Document* class in a *Python* shell.

```
document1=Document()
```

This results in the visible creation (it is being displayed, which provides instant visual feedback) of a new document in *OpenOfficeWriter* (wordprocessor part of the open source office package *OpenOffice.org*). The following invocation then closes the document.

```
document1.close()
```

All in all, students only have to apply the object-oriented terminology they are already familiar with in order to gain successful results in their first contact with actual programming languages. In addition to teaching the properties of a (hypothetical) *Document* class via a wordprocessor's menu structure or keyboard shortcuts, *Ponto* can serve as another way to »try out« the class's methods introduced in the classroom beforehand.

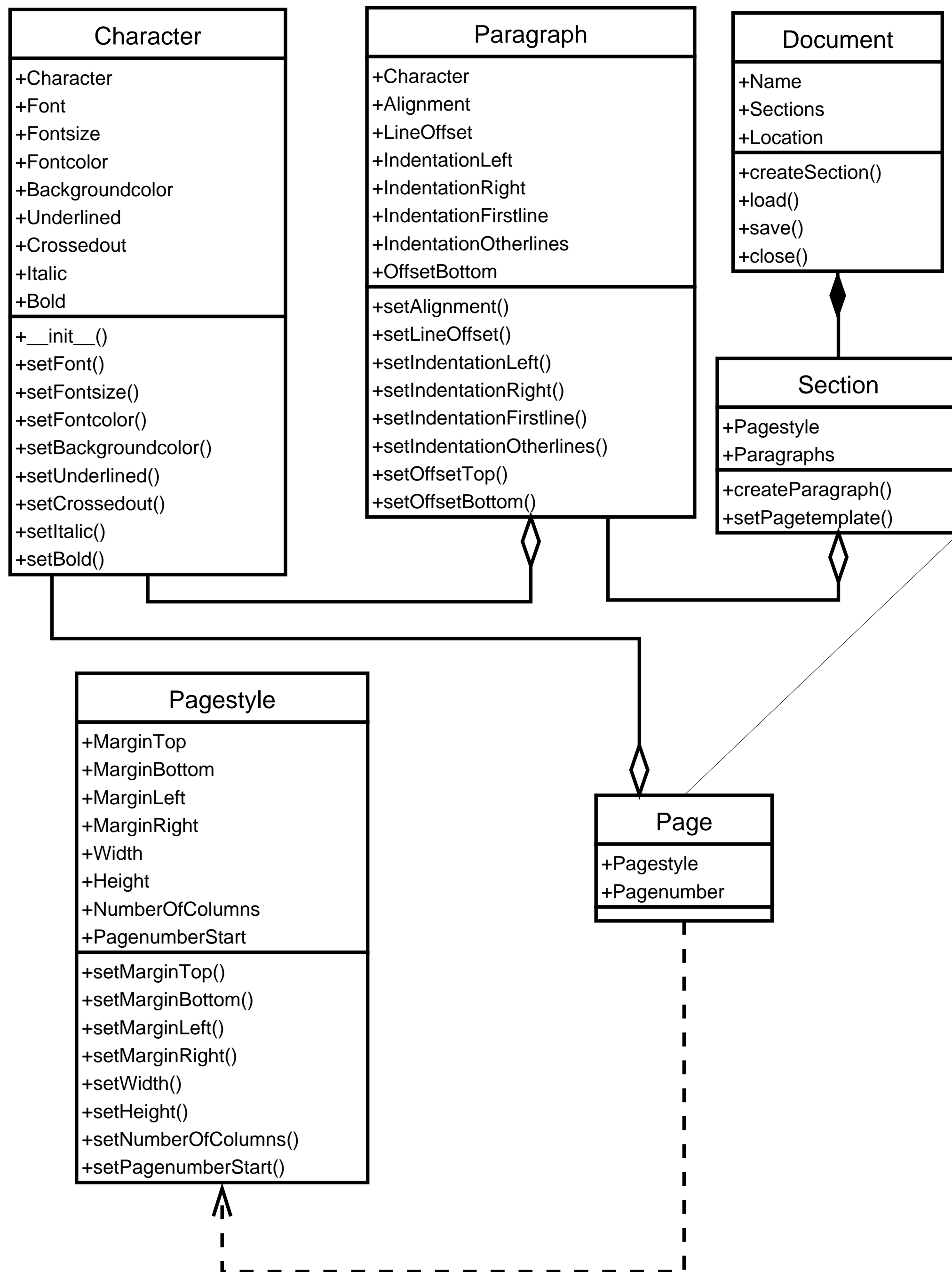

Fig. 2. class diagram *Ponto*

The *Python* module *Ponto* makes it possible to control the wordprocessing software *OpenOfficeWriter* (part of *OpenOffice.org*, see above) from the *Python* shell in a convenient manner by using object-oriented terminology. In more detail, *Ponto* provides an interface between *Python* and the *PyUNO* bridge software which is part of any current *OpenOffice.org* standard installation. In order to use *Ponto* in school, instructors therefore, in addition to a working *OpenOffice* installation, need a *Python* module (*Ponto*) and the documentation. *Ponto* provides exactly those classes which are specified in the concept for basic informatics instruction (with regard to word processing as a topic) that has already been introduced: *Document, Section, Paragraph, Character, Page* and *Page Style*. These classes are grouped with their corresponding attributes and methods, the latter of which can be invoked and this way create visible results. Fig. 2 depicts a simplified version of a class diagram that contains *Ponto's* main elements.

Furthermore, *Ponto* also provides a *Cursor* class which allows for navigation of a text document's cursor within the text and also meets the requirements of the content line algorithms. [6] refers to this content line as »Verarbeitung von Dokumenten und Automatisierung dieser Verarbeitung« and describes it as

- specification of methods in natural language for objects found in standard software
- building blocks of algorithms: sequence, repetition and choosing.

It should also be mentioned in this context that the Dagstuhl »standards group« lists propaedeutic algorithmics (sequence, case differentiation, repetition, event control).

1.3 The building blocks of algorithms with *Ponto*

By now it should be fairly obvious that in *Ponto*, invocations of methods can be placed in almost natural language. At least, the module supports the object-oriented terminology known from the theoretical concept (and thus known to students and instructors alike). Moreover, the *Cursor* class makes it possible to execute algorithmic building blocks on documents and their components (and visually observe the resulting effects). For example, one can retrieve a sequence of characters from a paragraph through iterative code, and then color these characters red. Also, students can use *Ponto* to write small methods on their own, e. g. with case differentiation or loops. As opposed to tools like *Karel the robot* or robots like the *Lego Mindstorm* series *Ponto* can easily be integrated into the same »document« context that is usually introduced on a theoretical basis. As a consequence female and male students might very well have about the same degree of motivation to develop skills in algorithmic thinking. In addition, word processing as a larger context seems to be a sensible choice, because it allows for the use of real systems (such as *OpenOffice.org* in this case), i.e. not only didactic tools. Even on a global level, office scripting is anything but a mere toy; in fact, the *PyUNO* bridge is used on a professional level to develop software for businesses to process standard business-specific documents easier and faster. Although this aspect is only of minor importance from a didactical standpoint, it indicates that the chosen context is not just a didactic invention, but on the contrary has its real-life applications. In the course of a spirally structured curriculum *Ponto*, at first, might only be used to instantiate classes like *Document, Section, Paragraph*, etc. and invoke their methods first through the wordprocessor, then with the help of *Ponto*. Later, *Ponto* could then be used to introduce the fundamentals of algorithms.

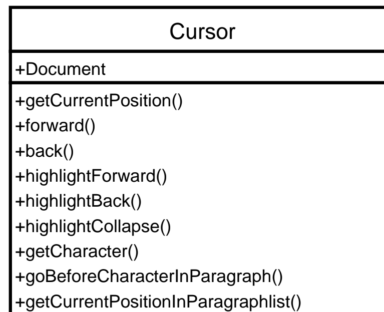

Fig. 3. class diagram *Cursor*

1.4 *Ponto* and informatics literacy

Back in 2003 a major debate concerning informatics literacy started, which remains yet to be finished. It was initiated against the background of the OECD's PISA study from 2000 in the course of which mathematical, science and language literacy were tested in 15-year olds. In order to test anything such as informatics literacy, a consesus on what makes up this kind of literacy is absolutely crucial—which is why didactic scholars need to agree on certain basic informatic competencies. A contribution that does exactly this is [8], in which informatic modeling is protrayed as key. Modeling processes allow for informatic knowledge to be transferred onto and applied to real-life phenomena. This transfer process can then be applied to areas deeply rooted in informatics systems (e. g. wordprocessing software as for *Ponto*), but also to areas only remotely (or even not at all) related to such systems (e. g. modeling a cash-register system for a grocery store or modeling components of a game). In addition to this focus on modeling competences, the authors of [8] also emphasize the importance of testing for formal knowledge and for formal competences as far as informatics literacy tests are concerned. Formal competences, according to the authors are (among other aspects) basic programming skills which are to be examined in a mostly programming language independent manner. *Ponto* offers teachers the chance to teach exactly these formal competences to their students. Compared to other means of instruction, *Ponto* offers instruction with real-life phenomena (problems related to wordprocessing) and real software products (*OpenOffice.org*). In addition, if an entire school year is structured around the handling of documents anyway, *Ponto* certainly fits into this context.

2 Full particulars and didactic decisions

2.1 Why *Python*?

The main reason for using *Python* to put *Ponto* into reality lies in the former's pseudo-code-like syntax which eliminates complicated declaration procedures. This certainly is a major aspects that make *Python* a programming language well-suited for use in

the classroom (cf. [9]). Yet, this is not to say there are no other tools to realize an interface for *OpenOffice.org*; one candidate might be the *Basic* variant that is integrated in *OpenOffice.org*. There would be an immediate benefit of this apparoach: students could put their desired actions into macros. However, there is also a major downside to the idea which is of a didactic nature: students would only learn how to use one standard wordprocessing system—the knowledge acquired in the course of learning *OpenOffice.org's Basic* variant could not be easily tranferred to other products. The *Python* based approach shows that students are able to operate a fully GUI-based piece of software by remotely controlling it from a text console.

A further advantage of using *Python* to put the project into reality was (as plain as it sounds) that *Python* supports object-oriented programming. Also, not to complete all tasks directly in the office application, but instead to work on a text-based console (somewhat like a »hacker«) can be an additional thrill for students—although this approach should, for obvious reasons, not be taken too far. One has to be aware though that this danger, in fact, seems fairly remote—the »natural language like« code that is to be written by students has to be rather clear and direct.

2.2 Why *OpenOffice.org*?

OpenOffice.org is equipped with a *Python* interface, which simply means it can be controlled using *Python*. In addition, *OpenOffice.org* is platform independent and as a consequence so is *Ponto*: it can be used on *Unix/Linux* systems as well as *Microsoft* operating systems (or any future operating system that supports both *Python* and *OpenOffice.org*).

2.3 The Cursor class—a (not too) deliberate decision

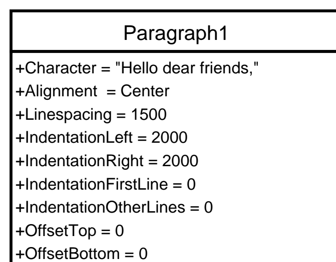

Fig. 4. Object reference card for a paragraph object

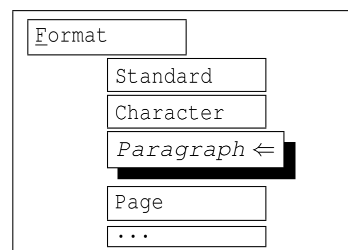

Fig. 5. Menu structure in OpenOffice.org

At the project's beginning it was our intention to implement the elements from the Bavarian concept into *Ponto* as exactly as possible. Ms. Voß (in a collborative effort between didactical institutions) was so kind as to provide the class diagrams she had developed. We figured that, if only the contents of these class diagrams were implemented as closely as possible, *Ponto's* functionality would be an exact match with the concepts

Algorithm 1 Opening and changing attribute-values

```
openingParagraph=sectX.createParagraph("Hello␣dear␣friends,")

openingParagraph.setAlignment(Center)
openingParagraph.setLinespacing(1500)
openingParagraph.setIndentationLeft(2000)
openingParagraph.setIndentationRight(2000)
```

introduced in school textbooks and other instruction materials. This way *Ponto* could have been integrated into current curricula almost without effort.

Eventually, we had to adopt a different view: a mere collection of classes which mirror the (mostly) static structure of text documents is simply not sufficient to develop a module whose aim is interaction with these structures. Yet, the basic requirement (namely that *Ponto* was supposed to be as close to the already existing concept as possible), naturally, was not to be given up. As a result of these considerations, it was more than clear that *Ponto* would nevertheless have to provide the same classes that could be found in the instrucion materials (including their methods and attributes). However, in the processes of designing *Ponto*, these classes had to be extended—primarly as far as their methods were concerned. Additionaly, *Ponto* required a new class to make navigation within a document possible: the *Cursor* class. Without a doubt the corresponding methods and attributes could also have been placed inside the *Document* class, but this would have dramtically altered the latter's character: about 70% would have been dedicated to cursor control—a conception that is certainly not a good template for instances of the *Document* class (text documents, after all). As a result, the *Cursor* class was called to life—so that any required cursor control methods would not have to be hidden inside other classes.

Any informatics model is merely a specific view on a discrete problem that, given a target-oriented perspective, takes certain static and dynamic structural criteria into account, in order to solve problems from the range of the model's target specifications. The *OpenOffice.org*-API contains the following structural elements: structures, services and factories. To elaborate on these concepts in detail would certainly go beyond the scope of this paper. However, one should be aware of the fact that providing students with a didactically oriented interface that is so dramtically limited by design that it has only one specific application does not seem reasonable (even if the solution is rather elegant/efficient). It makes much more sense to support object-oriented modeling on a general level.

Despite these points for criticism, teaching with the help of didactically designed class diagrams like the ones just described, is a reasonable approach. After all, there are certainly different applications for different models. With this in mind, the class diagrams serve two purposes at once: First, they provide an insight into the important field of object-oriented modeling and second, they contribute to a better acquisition of skills concerning the use of word processing tools (cf. [7]). Since *Ponto* also offers the chance to gain first insights into dynamic text processing via small bits of code, the

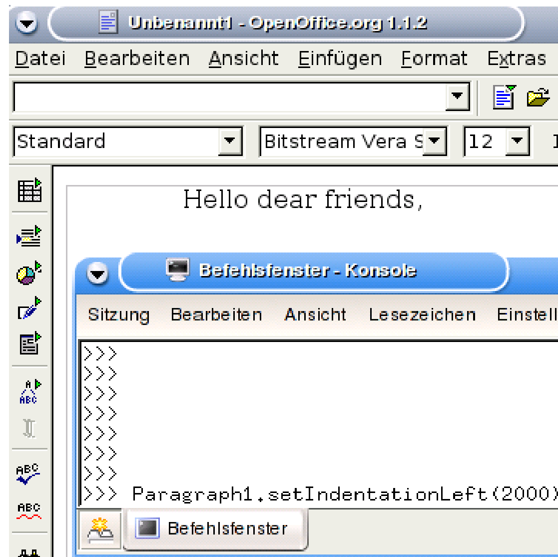

Fig. 6. Indenting a paragraph

original static model was not sufficent and thus had to be extended. If texts are not only to be created, but also manipulated with *Ponto*, means to navigate within the text are absolutely indispensable. This is the very functionality provided via the *Cursor* class (Fig. 3).

3 Examples

Sample tasks include for example object reference cards, like Fig. 4. Students are able

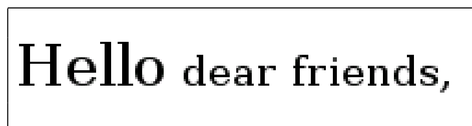

Fig. 7. Result of running algorithm 2

to instantiate an object that corresponds to the one provided and then assign the given values to the object's attributes. In *OpenOffice.org*, attribute values are usually assigned via the program's menus (Fig. 5). In order to create a Fig. 4 object with *Ponto*, a user must first instantiate a *Paragraph* object. In the next step, the attribute's values are assigned, as shown in algorithm 1. The values for identations and linespacing have to

Algorithm 2 Changing the attribute *fontsize*

```
from ponto import Document
# Script to change the size of a word in a paragraph
doc1=Document()
cursor=doc1.getCursor()
paragraph1=doc1.createParagraph('Hello␣dear␣friends,')
#back to the begin of the paragraph:
counter=0
while counter<=19:
 cursor.back()
 counter=counter+1

oneCharacter=cursor.getCharacter()
while oneCharacter.Character <> "␣"
  oneCharacter.setCharacterHeight(16)
  cursor.forward()
  oneCharacter=cursor.getCharacter()
```

be provided in 1/100 mm.[1] Fig. 6 is an actual screenshot that was taken right after the attribute *IndentationLeft* had been assigned a value other than 0. Algorithm 2 sets the attribute *Fontsize* of the next five characters to the value 16. Figure 7 shows the result.

4 Conclusions

Ponto is able to bridge the gap between wordprocessing software and the related theoretical object-oriented aspects currently considered important by scholars primarily concerned with didactics of informatics. It is certainly desirable for this kind of combined instruction in theory & practice to also be extended to other areas of informatic didactics. Programs like *Ponto* could serve as a bridge between many fields of applied informatics and their theoretical object-oriented foundations (e. g. vector graphics, hypertext documents). More information on *Ponto* (including the source code files) can be found at the website `http://ddi.cs.uni-dortmund.de/projekte/ponto`.

[1] Though we do realize that this might pose a problem to students in those countries which still primarily use inch-based measurements the use of metric units is clearly in line with e. g. science instruction in America as defined in the National Science Education Standards.

References

1. Humbert, L.: Let's teach informatics – empowering pupils, students and teachers. In van Weert, T.J., Munro, R.K., eds.: Informatics and the Digital Society – Social, Ethical and Cognitive Issues, Norwell, Massachusetts, IFIP TC 3, Kluwer Academic Publishers (2003) 141–147 – July 22-26, 2002, University of Dortmund, Germany.
2. Brinda, T., Claus, V., Diethelm, I., Humbert, L., Johlen, D., Magenheim, J., Micheuz, P., Modrow, E., Puhlmann, H., Scheel, O., Schneider, M., Schubert, S., Schulte, C., Schwill, A., Zündorf, A.: Zweite Dagstuhler Empfehlung zur Aufnahme des Fachs Informatik in den Pflichtbereich der Sekundarstufe I (2004) `http://www.informatikdidaktik.de/HyFISCH/Informieren/politik/DagstuhlerEmpfehlung2004.htm` – geprüft: 24. September 2004.
3. Wirth, J., Klieme, E.: Computernutzung. In Baumert, J., Artelt, C., Klieme, E., Neubrand, M., Prenzel, M., Schiefele, U., Schneider, W., Tillmann, K.J., Weiß, M., eds.: PISA 2000 – Ein differenzierter Blick auf die Länder der Bundesrepublik Deutschland, Opladen, Leske + Budrich (2003) 195–211
4. Puhlmann, H.: Informatische Literalität nach dem PISA-Muster. [10] 145–154
5. Humbert, L.: Zur wissenschaftlichen Fundierung der Schulinformatik. pad-Verlag, Witten (2003) zugl. Dissertation an der Universität Siegen `http://www.ham.nw.schule.de/pub/bscw.cgi/d38820/` – last visited: 28th August 2004.
6. Hubwieser, P.: Informatik als Pflichtfach an bayerischen Gymnasien. In Schwill, A., ed.: Informatik und Schule – Fachspezifische und fachübergreifende didaktische Konzepte. Informatik aktuell, Berlin, Springer (1999) 165–174
7. Voß, S.: Objektorientierte Modellierung von Software zur Textgestaltung. [10] 211–223
8. Humbert, L., Puhlmann, H.: Essential Ingredients of Literacy in Informatics. In Magenheim, J., Schubert, S., eds.: Informatics and Student Assessment. Concepts of Empirical Research and Standardisation of Measurement in the Area of Didactics of Informatics. Volume 1 of GI-Edition – Lecture Notes in Informatics (LNI) – Seminars., Bonn, Dagstuhl-Seminar of the German Informatics Society (GI) 19.–24. September 2004, Köllen Druck+Verlag GmbH (2004) 65–76
9. Humbert, L.: Which programming language supports my concepts for education in informatics at school? (2002) Abstract: `http://didaktik-der-informatik.de/dortmund2002/nrw/humbert.html` – presentation, manuscript: `http://www.ham.nw.schule.de/pub/bscw.cgi/0/20866`.
10. Hubwieser, P., ed.: Informatik und Schule – Informatische Fachkonzepte im Unterricht – INFOS 2003 – 10. GI-Fachtagung 17.–19. September 2003, München. Number P 32 in GI-Edition – Lecture Notes in Informatics – Proceedings, Bonn, Gesellschaft für Informatik, Köllen Druck + Verlag GmbH (2003)