Enabling Metaphor Evolution for Improving Systems’ Usability

Roland Kaschek, Alexei Tretiakov
Department of Information Systems
Massey University, Private Bag 11 222
Palmerston North, New Zealand
(R.H.Kaschek_A.Tretiakov}@massey.ac.nz

ABSTRACT: In this paper we consider both the interface language and domain language of a software system. That system co-evolves with its user group, with its usage context, application domain, and usage conditions. We presuppose that the evolution of that system is triggered by and reflected in the use of those languages. Since human language use is essentially metaphorical we expect the metaphors employed by system users to evolve when the software system evolves. To assure maximum usability of that system we suggest using a signal interception and platform for a metaphor controller for implementing a controlled change of the metaphors that are available in the mentioned languages. We show how our prototype of such a component can be used for implementing the evolution of the used metaphors. Our approach focuses on the maintainability of the metaphors rather than on a once-and-for-all method for optimally choosing them.

Categories and Subject Descriptors
C.0 [Computer Systems Organization]:Hardware/software interfaces; H.3.4 [System and Software]

General Terms
Software systems, Information systems, Software development

Keywords: Software evolution, Metaphor, Information system

Received 8 April 2006; Reviewed and accepted 24 April 2006

Introduction
It is common to distinguish (see [28]) software systems as systems of type S (specifiable or static), type E (i.e., evolving), or type R (i.e., all other software systems). Specifiable systems are for example those that implement functions such as sin, cos, or similar in a mathematical library. Software systems that implement environments within which humans perform their daily routine are, however, different. Competition forces to change these routines. Popular operating systems, text-processing, image-processing, process-control typically evolve with the use that is made of them. The size/complexity relation to price is another obstacle for completing specification. Competition enforces to be satisfied with good-enough software since software business based on good software might be too expensive and the time to market might be too long. In short good systems might not be sustainable. Software of the latter kind mentioned typically is E-type software. It is maintained in versions and releases and has completed specifications in the best case only for major releases. It is not likely that the specified software has been proven correct. The composition of these systems into a platform for everyday work (such as a typical MS Windows based PC) is typically not fully controlled. Consequently one has the typical problem of COTS (commercial off-the-shelf) systems that various base systems (i.e., components) interact with each other that are instantiated in releases that not necessarily fully comply with each others interface. Consequently for the everyday use work platform no full specification exists and it is of E-type. See facts 15 – 20 of [14] for evidence that the cost of resolving these problems might actually outweigh the benefits of doing so.

The evolution of E-type software affects its functionality as well as its further qualities. The latter accompanies an evolution of non-functional requirements, as these typically correspond to system quality. See for example [13] for a respective recent discussion of software quality. One of these aspects is usability. According to that source one could define the usability of a system as the extent to which the system users find that system easy to use. Usability can be decomposed into finer granular system quality aspects. Among these we consider understandability as of specific importance. If the software system is an information system (IS), and thus involves a user interface enabling human users to verbally communicate with the system, the respective computer-generated outputs can be interpreted as responses of a human expert to inquiries of human users. These responses (or the rules to generate them) are stored in the computer in a compressed, standardized format. Modern cognitive linguistics considers human communication as essentially metaphorical. It is therefore consistent with cognitive linguistics to assume that messages exchanged between humans and computers contain metaphors. Software evolution therefore includes metaphor evolution. Both of these are related to knowledge evolution as software can be considered as knowledge encoded in executable form and metaphors can be understood as a relational knowledge that enables the interpretation of a source domain in terms of a target domain.

We assume that metaphors will be used anyway. To boost understandability and thus usability of IS one therefore should be able to flexibly and explicitly manage the metaphors available to the user via the interface. The minimum requirement that a system must meet so we can try to improve its understandability with our approach is that in fact has a standardized interface such as a Web interface. Speedy adaptation of usable metaphors requires the metaphors being maintained by a dedicated component. Rather than the standard four-tier software architecture we suggest using a five-tier architecture that enables isolating the actual processing of the used metaphors from the core of the standard tiers. Effects are not supposed to be implied neither on the application logic tier nor on the data management tier. Therefore we consider our approach as a metaphor change enabling.

Paper outline The paper is structured as follows. In the next section we discuss related work. In the conceptual foundations section we provide a discussion of software evolution, knowledge evolution, and of metaphor. After that, in section 4, we discuss the architectural aspects of the use of metaphors in information systems and, finally, in section 5 we outline a case study of architecting and implementing metaphor support that we have conducted (including user evaluation of the resulting system). Finally we provide conclusions and our references.

2. Related work
The discussion of metaphor with respect to its use in information systems at least goes back to early eighties of the twentieth century (see [43] and [9]). While in [43] the discussion of the topic is rather brief and covers only about two pages it is the matter of the whole paper [9]. In that paper the discussion of metaphor for information systems is justified with the suggestions of two cited studies according to which learning is not simply an intake of data but rather a creation of a new cognitive structure and connecting it to already existing cognitive structures. The respective findings are consistent with “... the simple observation (dating at least back to William James, 1890) that people tend to learn about new things by making use of their past learning ...” ([p. 107, 9]). That paper’s authors, Carroll and Thomas, state then what they call metaphor principle, i.e., that people “... develop new cognitive structures by using metaphors to...” These authors use the term “computing system” that here is specialized to IS
cognitive structures they have already learned.”

Carroll and Thomas’ recommendations regarding the use of metaphors for information systems are here paraphrased as:

1. Find and use appropriate metaphors.
2. Suitability of metaphors with respect to an IS is proportional to the degree to which they are congruent to that IS.
3. The tone of a metaphor should be conducive to the desired emotional attitude of the user.
4. If several metaphors are going to be used with respect to an information system then these metaphors should be chosen from a given particular domain such that they are not mutually exclusive.
5. For each metaphor used the probable consequences to users and system designer should be considered.
6. When a metaphor is used then the users should be notified of that and made aware of the limitations of that metaphor.
7. At least for continual users metaphors might cease to be useful, since these users just get used to a particular way of using the system.
8. Exciting metaphors should be used for routine tasks and if alternative but structurally equivalent scenarios are available then the user should have a choice.

The first five of these recommendations address novice users while the last three address expert users. Thus metaphors in information systems right from the beginning were used as a concept for adapting systems to their human users. In the above mentioned papers Carroll and Thomas do not clearly specify which of the languages that are relevant with respect to an information system they focus on. There are, however, at least five different roles that languages play and which deserve attention. These are the

1. Architecture language, i.e., the language used for referring to IS architecture and usage issues;
2. Interface language, i.e., the language for interaction with the IS;
3. Domain language, i.e., the language for conceptualizing the application domain that the IS is aiding;
4. Organization language, i.e., the language used for talking about the organization that uses the IS;
5. Market language, i.e., the language that is used for talking about competing systems, their vendors and their marketing strategy etc.;
6. Development language, i.e., the language used for talking about developing the IS.

A number of metaphors have been identified that in particular could be used in the architecture language: game, machine, journey, jungle, family, zoo, society, war, and organism, [20]. A lot of further metaphors that are frequently used in computer discourse are pointed out in [20]. Much of the work known to us refers to the interface language [15, 49, 31, 41, 16, 17], and [7]. The papers [22, 37, 10] refer to the development language. The domain language is targeted with [4, 30, 39, 40, 44]. Obviously our language roles do not constitute a classification of languages, as one language (such as natural language, Logic, or Mathematics) could be used in several of the above mentioned roles.

The paper [30] recommends a procedure for finding good domain language metaphors. It thus enhances the Carroll and Thomas’ recommendations as restricted to metaphors of the domain language. That recommendation essentially consists in interviewing information system users in their work environment and asking them to explain their work. The respective interview logs are analyzed and metaphors are extracted from them. The author of [30] further more points out two common failures regarding the use of metaphors. Firstly, metaphors are chosen that suit the designers but not the users. Secondly, metaphors are modified after it was agreed to use them.

The papers [4] and [16] challenge the view that using metaphors in the interface language is beneficial. It is thus of particular interest to note that approaches for evaluating metaphors have been published. In that regard we firstly mention quality aspects of metaphor systems that were supposed to make metaphor use more comprehensible. In [29] the quality aspects covered are base specificity, richness, abstractness, systematicity, base exhaustiveness, and transparency. [31] proposes the quality aspects conceptualizability, learning aid, usage aid, extensibility, number, appeal added, memorizing aid, and deception. Finally we mention the most recent of these approaches which include its paraphrased quality aspect definitions. In [49] the CARSE approach is suggested. It proposes contextual suitability (the extent to which the target domain structurally resembles the source domain), applicability of structure (the extent to which the metaphor is relevant to the concept that shall be explained), representability (the ease with which the metaphor can be represented graphically on a computer screen), salience (the extent to which the source domain contains highly explicit elements that are very subtle in the target domain), emotional tone (extent to which the emotion triggered by the metaphor is suitable for the task at hand). Personae are recommended in [48], [45] for validating Web information systems (WIS). Obviously they may also be used for validating metaphors.

Employing metaphors is obviously not the only way of adapting software systems to users. In particular, alternative approaches covered in [3, 34, 6, 46, 35, 5] deserve mentioning. These approaches involve a run-time adaptation, i.e., a user model included in the software and parameters of that model being updated according to the evolving human-computer interaction. Also [38] promotes user modeling. However, it currently, is restricted to analysis-time adaptation, i.e., to user models with no parameters that could be updated. Obviously it is possible to combine the approaches by incorporating a user model, a user-interface model, as well as a domain model into the IS and update the parameters of these models based on the evolving human-computer interaction.

We finally point out that occasionally (in particular in the architecture language) metaphors are used without that necessarily being clearly understood. Take for example the process metaphor. Users conceptualize using a workflow application as a process in which documents flows from one clerk to another. What really happens is that the database management system used behind the scenes gives users access to data in a coordinated way, which is based on process shaped rules. Similarly, of course, in data mining no real digging takes place. The term software evolution that we have used for addressing the ongoing change of software systems is a metaphorical one, since software (at least currently) is not alive.

3. Conceptual foundations

3.1 Software systems

While the system concept is an important one we are not going to discuss in detail as it is not in the focus of this paper. For sake of brevity we rather refer the reader to chapter 4 of [1]. We consider software systems as systems that by automata of a given kind, such as general purpose computers, can be used for self-controlled activity. In that sense software systems are fully formal. They, however, not necessarily need to be encoded in a programming language. Rather, models’ encoded in diagrammatic languages such as the UML or subsets of it can be regarded as software provided they are sufficiently consistent and complete. From a pragmatic point of view it is important that software development is a process of knowledge acquisition and that a part of the required knowledge is represented in the developed software. The deployed artifacts are used in everyday life for solving a number of quite different problems. It is known since long that the majority of these problems are outside the range of mathematical problems and their solutions, see [47]. Terry Winograd notes: “Computers are not primarily a process for solving well-structured mathematical problems ...”. From a functional point of view and following Langefors (see [p. 11, 18]) we define an information system as “… a technologically implemented medium for the purpose of recording, storing, and disseminating

---

1 For details about models see, e.g.[21].
linguistic expressions as well as for the supporting of inference making.” For this paper we choose to ignore non-IS software systems such as embedded systems or digital games. The majority of IS thus would have to be considered as E-type systems, i.e., as evolving, as the experience of using a deployed IS is likely to change the work processes in which an IS is used as well as the users’ expectations and capabilities. IS typically are used as a medium of interaction of a user with an expert. That expert might have shallower knowledge (e.g. regarding phone number of people living in the city) or deeper knowledge (e.g. regarding geometric objects such as the ratio of length and radius of a circle) and less advanced (e.g. print an article from a digital library) or more advanced (e.g. suggest a strategy that is most likely helpful in solving a given problem) capabilities. One of the respective discriminating factors is the expert’s domain of expertise.

Developing information systems aims at overcoming the obvious limitations of a human expert and so exploiting his or her knowledge and skills more efficiently, effectively, and economically. Information systems also can integrate knowledge and skills of different experts. The human-computer interaction thus necessarily and at least in so far as the domain language is concerned involves the use of metaphors. Thus, it is not up to the system designers or developers whether metaphors will be used at all. It is only up to them which metaphors can be used easily by whom and what for. In [41] it has even shown that not reflecting user generated metaphors in systems design may decrease the system’s usability.

3.2 Knowledge evolution

The environment of a system S is the set of systems S interacts with and that do not belong to S. This interaction often can be modeled as exchange of discrete signals. These signals can be classified as stimulus (request for interaction) and response (provided interaction) respectively, depending on the role in such an exchange. Many systems such as animals, humans or organizations can learn, i.e., they can adapt the inner parameter (such as memory or reasoning capability) they use for controlling the responses they generate to stimuli. That way they maintain their capability to respond in a variety of ways to an event that occurs repeatedly. That inner system parameter is obviously heavily related to what one would call that system’s knowledge.

The natural world overlaps the man-made. Both of them are changing. We use the metaphor of evolution for describing this process of change. That suggests a number of points: Firstly, we do not presuppose a goal or controlling agent of that change, i.e., changes may become necessary without being desired by the involved human being. Secondly, survival depends on general aspects of the world only. For certain software systems niches may exist in which they can stay alive, i.e., in use while at the world level they are extinct. Thirdly, death, i.e., retirement of a software system can be prevented from happening if that system stays fit in its environment. While for certain systems (such as assistants or agents) staying fit is a matter of self adaptation for the majority of systems it is a matter of being adapted by purposefully operating humans. At this point of course the metaphor brakes down, as at least the ratio of the two in biological evolution differs currently from the one in software evolution.

According to a frequently used view (that for example is present in the ER-model or comparable semantic models) knowledge can be understood as being self-contained or relational. Factual knowledge such as “George W. Bush is the President of the United States”, or “Napoleon Buonaparte crowned himself emperor of France” would here considered to be self-contained. It represents judgments of individuals about entities in the world. Relational knowledge such as “time is money” or “John is a lion” relates such entities to each other and thus may enable understanding one entity in terms of another one. Of course for that actually to work it must be the case that the one who is supposed to understand knows the referred-to-concept reasonably well. Metaphors are a kind of relational knowledge, as they allow one to understand concepts in one domain (“source” domain) in terms of structurally, behaviorally, or otherwise similar (and thus, related) concepts in another, “target” domain. The metaphors employed in a software system obviously are subject to co-evolution with that system’s environment to increase the system’s chance of survival.

3.3 Metaphor

Given that humans perceive phenomena in their environment as something, i.e., have a categorizing perception (see, for example [12]) it is not surprising that humans tend to understand and conceptualize phenomena as something, i.e., as entities and reason about one entity in terms of another one. Metaphor is a mechanism to achieve that. Since the genesis of Western civilization in ancient Greece and until recently, metaphor was mainly considered as a powerful tool of deception that would do more harm than good and that should and could be left to the fine arts and rhetoric, see for example [17]. In particular the work of George Lakoff and his associates has contributed to change that view. Nowadays many cognition linguists (see for example [19]) and philosophers (see for example [2]) believe that metaphor is a key tool of human interaction with the world. Also in Artificial Intelligence, see for example [32], metaphor is nowadays considered as important.

The Oxford English Dictionary Online defines metaphor as a “... figure of speech in which a name or descriptive word or phrase is transferred to an object or action different from, but analogous to, that to which it is literally applicable; an instance of this, a metaphorical expression.” Similarly, more general and a bit more fuzzy [33] defines metaphor as “… the understanding of one concept in terms of another.” A more recent text, i.e., [24] that wholly dedicated to metaphors agrees with that and proceeds with defining “… (a) conceptual metaphor consists of two concepts that in some way share such a fundamental aspect that one domain is understood in terms of the other. A conceptual domain is any coherent organization of experience. …” The conceptual domain from which the metaphorical expressions (i.e., words or other phrases) are drawn is called target domain. The conceptual domain that is understood that way is called source domain.

We do not aim at covering the latest results of the cognitive linguistic theory of metaphor, i.e. [26]. For easily accessible further detail see, for example [chapter 2, 27]. Ignoring the results of new inheritance among metaphors and following [27] we define a metaphor as a partial and structure preserving mapping m: S → T from a source domain S into a target domain T. By mapping parts of S into T one is enabled to understand parts of S in terms of their image in T. If, for example, for a question q one can be raised regarding a concept t = m(s) of T one has found an answer q then there is a chance that it can be turned into an answer as regarding an analogous question q that is raised with respect to S. One of the examples frequently discussed is the metaphor LOVE IS A JOURNEY that enables interpreting aspects of the domain of “love” in terms of the domain of “journey,” as is the case in saying “Our relationship is at a crossing”. It was one of the riddles of the traditional analysis of metaphor how humans with relative ease could understand expressions like “Our relationship is at a crossing”, even if they have never heard it before, see [42]. Lakoff’s answer is that if humans are aware of an appropriate partial structure mapping then they have a clue of how to interpret such expressions even if these are new to them. An expression, as the one above, is called metaphorical expression if in it with respect to a term (i.e. our relationship) characteristics are ascribed that usually do not apply to that term (i.e. being at a crossing). According to Lakoff and Johnson metaphor permeates every aspect of language use and is a basic mechanism of cognition. For a critical review of several theories of metaphor see chapter 2 of [11]. In chapter 12 of [24] it is argued that certain metaphors are very similar in English, Hungarian, and Japanese and thus may be independent of human culture and that some metaphors even might be universal in the sense of being common to all humans.

Understanding of metaphor as a structure-preserving partial mapping does not imply the structure of the domains put into relation to be equal. Rather, as it was mentioned above, the limitations of a metaphor as rooted in two different domain structures or derivation rules should be pointed out to the IS user.

Lakoff and Johnson say, [25], “... metaphor is pervasive in everyday life ... Our ordinary conceptual system in terms of which we both think and act is fundamentally metaphorical in nature.” Thus, for IS it is not a question whether metaphors are used or not. Rather, the question is which metaphors are used and how easy it is to use them.

4. Metaphors in information systems

It seems worth mentioning that the term “computer” as applied to
modern standard computers is a metaphorical expression, as a computer does not compute. It rather only transforms patterns of magnetization, electrical charge, or similar. Humans interpret these transformations in a way that allows imposing on computers a formal system suitable for designing, implementing, and using self-controlled machines. That interpretation identifies then a computer-internal correlate to computing. Also, rather than the machines used for carrying out computations, originally, the humans who were using advanced technology were called “computer” while at that time the machines they used were called “calculators”.

Content delivery systems such as the ones discussed in e-learning are generic and need customization so that the metaphors used in human-computer interaction as the metaphors in the content cannot be controlled by the systems developers. This is because their various domain languages may be quite incoherent (due to the nature of the documents provided) and due to ongoing change (as content adaptation may take place frequently). At the same time these metaphors are potentially very important for successful system use as the users aim at understanding that content and might have no alternative source of information. The situation is similar with respect to legacy IS as it might be impossible, impractical, or undesirable etc. to actually modify them. However, understandability issues might be apparent, as large numbers of complaints are made or usage extent is decreasing. An approach would be needed that addresses that situation.

By definition “E-type software” tends to evolve. Obviously, this evolution may affect the system functionality, as well as the system quality and in particular, the system’s interface. The source [28] reports (p. 283) that 60 to 80 percent of the resources allocated to software evolution need to be spent after deployment of the first release. Obviously software evolution is a co-evolution of software systems and their user community. We suggest simplifying this co-evolution by providing tools that intercept the signals exchanged between user and IS and that these tools aid users in using appropriate metaphors. Updating the metaphors registered to such tools should be fairly easy so the burden of finding appropriate metaphors would be lowered in favor of changing them more frequently, according to user demand. Respective tool architectures might even make it possible to fully leave maintaining the employed metaphors to the users. In this paper we focus on the domain language. It is, however, obvious that this also is an option for the interface language, at least in so far as the icons that are used in graphical user interfaces are concerned. For fully resolving the reported difficulties regarding such icons (see for example, the discussion of the respective trash-can example in the Mac user-interface [16, 17, 41]), however, it should be possible to flexibly, on-demand, alter the mapping of icons to the behavior they invoke.

5. Case study: Metaphor-enabling an E-learning system

We draw here from [44] and keep therefore the discussion short. The paraphrased purpose of the study reported in the source was the demonstration of the feasibility of domain language metaphor evolution. We aimed at a non-invasive approach that would not require access to the implementation of the information system providing the content. We refer to the prototype implemented as the metaphor enabler (ME).

5.1 The metaphor enabler

The ME’s architecture is based on the PAW (Pro-active Web Filter) open-source filtering HTTP proxy based on the Brazil Framework. The PAW proxy offers the capability to plug in custom filters (which have to comply with filter APIs defined as part of the Web application development environment, as defined by the Brazil Project). At present, the PAW proxy comes with a RegExp filter, allowing us to match Web page data included in HTTP responses as a string against a set of regular expressions, and to replace matches by arbitrary strings. To fully implement the ME one would have to develop a custom filter, combining the capability to match and replace strings with access to session and request data (such as user identity and user type assignment, the URL of the page being retrieved etc.). For the purposes of initial evaluation, we opted to limit ourselves to relying on the functionality of the already available RegExp filter, so that the concept mapping was not context-sensitive: rules defined for string replacement were defined for all web pages of the target WIS (the “base service”) in a uniform way. It would be easy to introduce a form of adaptivity to the user’s type by running several instances of the proxy simultaneously. One proxy would be available for each user type. The users then would be instructed to use the appropriate proxy settings. However, the number and the variety of users available for the initial evaluation were not sufficient to validate the value of adaptivity to user types. Furthermore, our primary purpose was to demonstrate the technical feasibility of our approach. Hence, we defined the transformations necessary to add metaphor support for a single proxy instance, supporting a single user type, and assuming a uniform context for target domain concepts.

To guarantee that adding a metaphor support layer does not affect the ease of use experienced even for users who do not consider the metaphor support to be useful), rather than replacing the occurrences of target domain concepts by source domain concepts, we transformed the web page HTML in such a way, that each target domain concept would become an anchor that, when activated by the user, would display a pop-up showing the corresponding source domain concept. The RegExp filter was configured to add JavaScript allowing us to achieve this effect on all HTML pages served via the PAW proxy. Thus, from a user perspective, the metaphor support layer was introduced transparently. Users could see through it without taking note of it, if they wished.

Words corresponding to target domain concepts were highlighted in blue to make them appear similar to HTML links in default HTML styling, but they were not underlined (as HTML links are), so that they could be distinguished from HTML links. By adopting a styling of references to HTML links, we intended the user would understand that clicking on that word would trigger some relevant information being displayed. We assumed that highlighting words would not decrease the usability of the interface for a user who opts not to use metaphor enabling software-tier. With this technology one could actually register source- and target domains, as well as partial mappings from the former into the latter. The source domain would be the application domain of the IS the usability of which is under scrutiny. The target domain would be another domain with an expected evolution may affect the system functionality, as well as the system quality and in particular, the system’s interface. The source [28] reports (p. 283) that 60 to 80 percent of the resources allocated to software evolution need to be spent after deployment of the first release. Obviously software evolution is a co-evolution of software systems and their user community. We suggest simplifying this co-evolution by providing tools that intercept the signals exchanged between user and IS and that these tools aid users in using appropriate metaphors. Updating the metaphors registered to such tools should be fairly easy so the burden of finding appropriate metaphors would be lowered in favor of changing them more frequently, according to user demand. Respective tool architectures might even make it possible to fully leave maintaining the employed metaphors to the users. In this paper we focus on the domain language. It is, however, obvious that this also is an option for the interface language, at least in so far as the icons that are used in graphical user interfaces are concerned. For fully resolving the reported difficulties regarding such icons (see for example, the discussion of the respective trash-can example in the Mac user-interface [16, 17, 41]), however, it should be possible to flexibly, on-demand, alter the mapping of icons to the behavior they invoke.

5.1 The target domain

As a source domain for the ME evaluation, we chose graph theory. The content required for that was drawn from Chris K. Caldwell’s respective online tutorials, see [8]. They introduce Graph Theory in general, and, in more detail, Euler Circuits and Euler Paths. Below we often refer to them collectively as “the tutorial”. The reasons for this choice were as follows: the tutorials are highly interactive, and involve static pages, dynamic pages, and HTML forms, thus representing a full-featured WIS. The tutorials have a linear structure, and involve static pages, dynamic pages, and HTML forms, thus representing a full-featured WIS. The tutorials have a linear structure, which ensured that all test users followed the same path, and had comparable exposure to the WIS information. The tutorials include a large number of quizzes, which in principle would allow us to consider the impact of the metaphor enabling on the success of the users in learning the subject. The topic covered by the tutorials is relevant for the background of the test users we had available (postgraduate students in Information Systems with no in-depth background in mathematics). Since the students are supposed to be teaching graph theory, the source domain contained various graph theory terms. As the target domain, we chose plumbing with pipes standing for graph edges, and pipe junctions for vertices, on the assumption that all of the prospective users have a degree of everyday experience relevant to this domain. We have built a system of metaphors based on that assumption. For example, movement along edges and vertices interpreted as “swimming” along “pipes” (for example in a fun-bath). In addition to textual representation, we added the relevant drawings. We assured that each target concept representation would stand on its own, and
would not require any further clarification via cross-referencing etc. All of the metaphorical expressions employed make up an overbearing metaphor of a pipe work and swimming through it.

5.2 Evaluation setup and results

The evaluation was conducted with 11 users of similar background (most of them postgraduate students employed as graduate assistants, neither majoring nor particularly educated in mathematics). The users were instructed to reconfigure their browsers to use the PAW proxy imposing the metaphor support layer, and to complete the tutorial with as few quiz resubmissions as possible. The tutorial was set up (by its original creator) in such a way that the user had to proceed the user needed to answer all questions in each of the intervening quizzes correctly; if a quiz was answered incorrectly, the user was prompted to re-do it until all answers were correct.

There was no time limit, and the users were not asked to maximize the speed with which questions are answered, so the number of quiz resubmissions was the only quality criterion (the fewer resubmissions, the better, with zero resubmissions corresponding to all questions answered correctly at the first attempt). The evaluation instructions we used were available in the departmental intranet. All HTTP requests were logged at the proxy. We attributed web page and metaphor invocations to user sessions via IP addresses, which were stored by the proxy as part of each log entry. As IP addresses in our environment are allocated dynamically, it was not possible to determine the identity of the user, which ensured user anonymity. In addition to completing the tutorial, users were asked to anonymously provide free-form comments on how useful they found the metaphor layer.

A Python script that was created for that purpose was used to filter out log entries for each session, and to represent the session as an animated replay of user interactions with the tutorial and with the metaphor layer. For each user, the animated replay was analyzed visually, to make sure that any spurious or irrelevant interactions are not taken into account. While most users navigated along the tutorial linearly, only once, from beginning to end, some completed only half of the tutorial on the first attempt; later, they resumed from the beginning, and this time, completed it, or almost completed it. For such users, we disregarded the part of the second attempt session duplicating the first attempt, so that all data corresponds to the first encounter of the user with the particular part of the system’s hyperspace. We distinguish metaphor invocations as the ones in the context of exploration (such as in context of a concept being introduced or elaborated upon), and the ones in the context of fulfilling short-term goals (such as in context of a concept used in a quiz question, or in answer feedback).

Most of the users employed the metaphor layer feature, with the mean number of metaphor invocations per user of about 8, which, considering that there were 34 pages in the tutorial, constitutes about 1 invocation per 4 pages visited. The number of metaphor invocations differed significantly from user to user, the standard deviation estimation was 4.6 (more than half of the mean value of 8). An unexpected and highly interesting outcome was that the users were significantly different from user to user, the standard deviation estimation was 4.6 (more than half of the mean value of 8).

6. Conclusions

We have argued that metaphors will be used by IS users independently of whether or not that was anticipated in systems development. The available literature suggests that metaphors in the interface language or the domain language of an IS impacts the systems’ usability. Metaphors above were considered as a relational knowledge. The group of users of an IS is likely to change when the system remains accessible for a long time. This evolution of users is, as we have argued, likely to require a co-evolution of the metaphors that are supported by the IS. Referring back to a study that included the development of a prototype for metaphor maintenance we have shown that this form of knowledge evolution actually can be supported by technical means.

References


