

Why not Empower Knowledge Workers and Lifelong Learners to Develop their own Environments?

Felix Mödritscher

(Vienna University of Economics and Business, Vienna, Austria
felix.moedritscher@wu.ac.at)

Fridolin Wild

(The Open University, Milton Keynes, United Kingdom
f.wild@open.ac.uk)

Abstract: In industrial and educational practice, learning environments are designed and implemented by experts from many different fields, reaching from traditional software development and product management to pedagogy and didactics. Workplace and lifelong learning, however, implicate that learners are more self-motivated, capable, and self-confident in achieving their goals and, consequently, tempt to consider that certain development tasks can be shifted to end-users in order to facilitate a more flexible, open, and responsive learning environment. With respect to streams like end-user development and opportunistic design, this paper elaborates a methodology for user-driven environment design for action-based activities. Based on a former research approach named ‘Mash-Up Personal Learning Environments’ (MUPPLE) we demonstrate how workplace and lifelong learners can be empowered to develop their own environment for collaborating in learner networks and which prerequisites and support facilities are necessary for this methodology.

Keywords: User-Driven Environment Design, End-User Development, Semantic Interoperability, Workplace Learning, Lifelong Learning

Categories: D2.1, D2.2, L3.1, L3.6

1 Introduction

In our everyday life we experience a lot of environments and tools that were designed by others; houses are constructed for different usages by different building enterprises, cars have been developed for more than one century to what we know nowadays, computers are built and assembled by many vendors, and software is realised by developers and teams, usually according to their mental models and design expertise – just to name a few examples. However, when it comes to regular usage of these environments and tools people tend to customise them in their specific way: they furnish each room of their houses, they decorate their cars and (must) adjust the mirrors to their needs, and they adapt the operating system and install their preferred programs.

Nowadays, the development of learning technology is driven by companies and open-source communities, whereby software solutions for knowledge workers and lifelong learners have to fulfil many requirements and sophisticated mechanisms of socio-technical systems, as shown with the multi-activity distributed participators design process for work-integrated learning systems in [Jones & Lindstaedt, 08] or the

concept of learner networks in [Koper et al., 05; Wild, 09]. In this scope, knowledge management aims at supporting the designers (domain experts and software developers) through various business- and technology-driven models, like the SECI model [Nonaka & Takeuchi, 95]. Again, such approaches put the organisation central stage prescribing how knowledge transfer ought to work and, at the same time, ignoring idiosyncrasies of learners and communities.

Due to the complexity and dynamics of such socio-technical systems, expert-driven software engineering is not sufficient any more, and alternatives are considered to be useful or even urgently required for workplace and lifelong learning. In this paper, we sketch the paradigm shift from traditional software engineering to new streams like opportunistic design and end-user development and, furthermore, describe a methodological approach to user-driven environment design applicable for knowledge workers and lifelong learners. In the end, we summarise our experiences gained with that software development method so far and argue for the potential of this approach.

2 Moving from Traditional Software Engineering to User-Driven Development

Looking back on the history of **software engineering**, development methods have developed from rather static and product-centric process like the V model or the spiral model to more dynamical methodologies taking into account the ever shorter software life cycles with fast changing requirements as well as issues of socio-technical systems, i.e. networks of humans and machines and their interrelationship and interaction [Trist, 81; Geels, 04]. Recently, many application areas require iterative processes interweaving design and prototyping phases in combination with extreme programming [Beck & Andres, 00] and agile software development [Cockburn, 02].

On the other hand, user-centred design dating back at least to the 1970s has a long tradition in the field of **human-computer interaction**. [Preece et al., 98] state that user-centred design dedicates extensive attention to the user for all steps of the design process. Therefore, [Maguire, 01] proposes various user-centred design methods for each phase of the software development process, beginning with the planning and usage context (e.g. stakeholder identification, user observations, task analysis, etc.) up to designing and evaluating software (parallel design, prototyping, usability inspection and testing, questionnaires, and the forth).

However, expert-driven software engineering and human-computer interaction still lack the ideas of recent streams, like community platforms and related **Web 2.0** concepts. Placing the focus on a more active participation of users and interactions with others has led to dealing with requirements engineering of socio-technical systems and, furthermore, to methodologies like the RESCUE process by [Jones & Lindstaedt, 08].

Additionally, the dynamics and complexity of socio-technical systems require development processes going beyond the methods depicted so far. [Hartmann et al., 06] come up with the idea of **opportunistic design** which considers users to hack, mash and glue software (and hardware!) artefacts to achieve their goals. According to these authors, mashing up existing and own code pieces provides more functionality

up front, although the ‘last mile’ (finalising the full application) may be slow (but not always necessary). For their approach they are using AppleScript for enabling (experienced) users to realise their solutions on the basis of existing and own source code.

Similarly, **end-user development** aims at empowering users to design their software environments for their purposes, whereas this development process ranges from customising user interfaces over visually assembling applications up to real programming tasks [Fischer et al., 04]. [Lieberman et al., 06] state that end-user development changes systems from being “*easy to use*” to being “*easy to develop*” in order to increase their responsiveness towards the diversity of users, i.e. people with different skills, knowledge, cultural background, etc., as well as towards the dynamics of work and learning practices. Lieberman and his colleagues differentiate between (a) parameterisation/customisation and (b) program creation/modification. Moreover, they highlight typical activities, like parameterisation of software components, annotations, programming by examples, incremental programming, model-based development, and so forth.

According to [Fischer et al., 04], it is necessary to motivate users to practice the hand-on skills required for an end-user development approach, e.g. by giving examples or demonstrators. Beyond that, opportunistic design of a learning environment requires even more **competences of end-users**. According to the classification by [Stahl, 08], these competencies comprise technical ones, like basic computer and internet skills, methodological ones, for instance being capable to know and utilise a specific tool for a certain purpose, personal ones, i.e. self-motivation, self-confidence or self-efficacy in working and learning with online tools, as well as social ones, such as the capacity to collaborate with others, communicate ideas and information, or to connect to learner networks.

Overall, [Kraus, 05] introduces the idea of the ‘**long tail of software development**’ and sees the opportunity to satisfy the needs of many if (experienced and capable) end-users are willing to create opportunistic assets and share them with others. Specifically for higher education, [Wild, 09] states that end-user development is more powerful than instructional design or personalised adaptive e-learning as it does not take away important learning experiences from learners, e.g. by calculating the optimal path through a course, but supports them in constructing knowledge within their activities, e.g. through recommendations or good practice sharing [Mödritscher & Wild, 09].

3 A User-Driven Methodology for Environment Design and Learner Interactions

The end-user development method described in this section is built on a former conceptual approach called ‘Mash-Up Personal Learning Environment’ (MUPPLE) and developed within the scope of higher education [Wild, 09]. Without going into details of this approach, some preliminaries have to be stated at this point:

- MUPPLE is based on a **web application mashup infrastructure** which allows learners to integrate arbitrary web-based learning tools into a single

user experience (see also Fig. 3). This mashup paradigm is one possible front-end and might be realised in other ways.

- Requirements for integrating learning tools into MUPPLE comprise a certain degree of **widgetisation and semantic interoperability**. Widgetisation means that a web application can generate web-based output (widgets) for certain functionality through RESTful requests, i.e. a Uniform Resource Locator (URL) address. Semantic interoperability deals with the possibility to plug one system to another one so that an exchange of data and its processing in a meaningful way is achievable. Such a mechanism can be regarded as a powerful approach for enabling users to create new functionality. For instance, a PLE including 10 interoperable widgets does not only have 10 basic functions for learning (e.g. bookmarking, searching, browsing, creating artefacts, etc.) but also $(n-1) \times \frac{n}{2} = 45$ possibilities to combine two different widgets (e.g. bookmarking results from the search widget). Most of these combinations might not make sense but it should lie with the users to play around and identify a new, valuable functionality. Aiming at a generic solution for web tools, we build on RSS feeds (distributed feed networks) to achieve semantic interoperability [Wild & Sigurdarson, 08].
- Similarly to the Activity Theory introduced by [Engeström, 87], the usage context of MUPPLE is structured through a very **simple model of a learning activity** which consists of a set of user interactions, each interaction being formalised as a triple of (a) an action, (b) an outcome, and (c) at least one tool. The actor is the user in front of the screen (user-centred approach), while the action labels the user interaction and the outcome describes a real or an abstract achievement (e.g. an artefact or a user goal). The tool (including a URL) is necessary to formalise how a user can achieve the outcome. Examples for such generic user interactions are ‘find paper using Google Scholar’ (with the URL <http://scholar.google.com>), ‘publish self-description using VideoWiki’ (with the URL <http://distance.ktu.lt/videowiki/addvideo>) or, to describe an everyday situation, ‘sign contract using pen’.

Based on these preliminaries, we differentiate between functional and non-functional requirements. On the one hand, we consider **functional requirements** of the learning environment to be the features which are directly related to the learning process, typically involving a specific tool. Such requirements are materialised with action-outcome-tool triples which include RESTful requests to a widget or even a tool combination if semantic interoperability is fulfilled for the web applications involved. The set of functional requirements is open, so that users can define their own interactions and bring in their own tools. In the context of traditional software engineering, the model of a learning activity would be equal to the concept of a *class* having the user interaction statements as its *methods*. Normally, creating a new activity would instantiate an empty activity-*object*, whereas users can add new methods (*mix-ins*) and each user-given functional requirement would stand for one ‘interaction method’ (with the parameters action, outcome, tool, and URL). As part of

our practice sharing strategy, it is also possible to derive activities from a so-called activity pattern which can be seen as a class pre-initialised with abstract or real methods by another user (cf. [Mödritscher & Wild, 09]).

On the other hand, **non-functional requirements** address the end-user development framework, in our case the MUPPLE prototype, which includes all facilities enabling and supporting users in creating web application mashups for their activities and interacting with the tools. These non-functional requirements have to be (iteratively) gathered and realised by the provider of the end-user development framework; they are highly dependent on the presentation layer of a solution and the characteristics of the tools involved, like semantic interoperability or widgetisation issues. [Wild, 09] derives these requirements from different research perspectives, like learning design, semantic interoperability, social networking, practice sharing or personalised learning, and summarises them for our first MUPPLE prototype.

Fig. 1 visualises our **methodology for end-user development of personal learning environments**. The top left corner indicates the user interaction scheme which can be used to instantiate a learner interaction either by specifying the action-outcome-tool triple manually, by taking advantage of the recommendation service, or by reusing them from a pre-defined activity pattern. The start of the actions of one activity leads to the web application mashup, as shown in the centre of the figure.

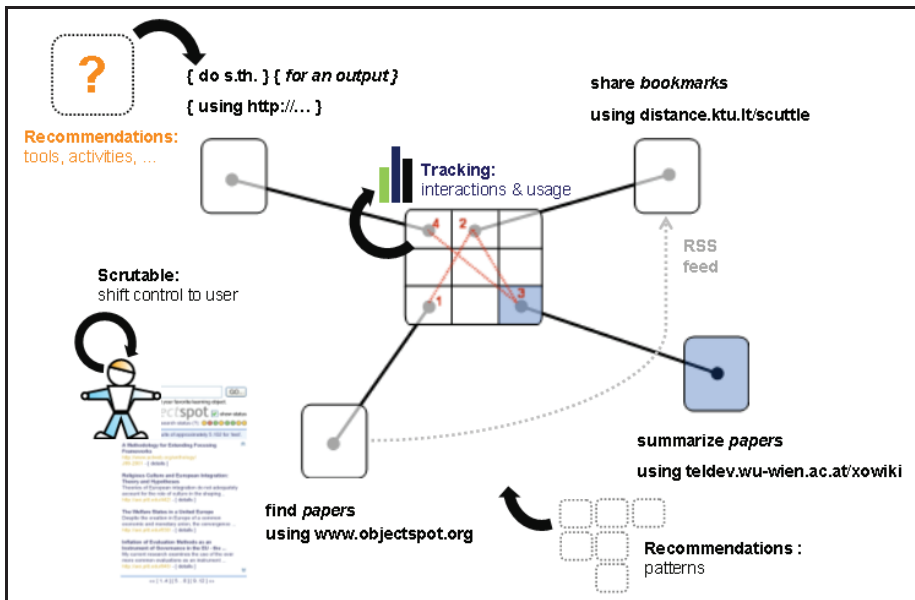


Figure 1: User-driven methodology for environment design and learner interactions

Now, the user has the control over the user interface, is able to work with the tools in any specific order, fully rewrite the activity, complete or resume actions, and export new patterns from this activity. Certainly, users can bring in new tools if they require them for successfully finishing an activity. These new user interactions with the environment are subject to automated generation of recommendations or to user-

driven good practice sharing through the activity patterns (top left and bottom right). Fig. 1 also gives an example for applying a tool combination, for example the interaction ‘find papers using <http://www.objectspot.org>’ enables users to send selected search results to their preferred bookmarking tool and share these bookmarks afterwards (cf. interaction ‘share bookmarks using <http://distance.ktu.lt/scuttle>’).

Specifically, the MUPPLE approach has been developed within the scope of higher education, assuming that learners are self-motivated enough to design their environments to collaborate in meaningful PLE-based activities. Such approaches to self-directed learning are also observable in the fields of workplace and lifelong learning [Ulbrich et al., 06; Koper & Specht, 06]. More likely, the user-driven methodology for environment design is not only applicable for these two areas but for any action-oriented activity, even beyond the context of learning. The non-functional part of MUPPLE, however, is dependent on the technological solution and usage context and, therefore, not transferable to other domains.

4 Implementation of and Experiences with User-Driven Environment Design

In this section we use our first MUPPLE prototype to explain how end-user development of and interactions with these environments can be supported. As a first step towards implementing environment design we have designed a domain-specific language, namely the ‘Learner Interaction Scripting Language’ (LISL), to be able to materialise how learners design and use their environments [Mödritscher et al., 08].

Fig. 2 gives an example of a LISL script which describes a MUPPLE activity consisting of three actions (lines 1-3: compose, browse and bookmark), each one bound to a specific outcome (lines 4-6: self-descriptions, peers and selected descriptions) and displaying a windowed widget with a specific RESTful request to the VideoWiki application (lines 10-12). The lines 7 to 9 define two tools (VideoWiki and Scuttle) and the channel between these tools, which is set up by the connect statement and requires a certain degree of semantic interoperability.

```

1> define action compose with url http://[...]?action=create
2> define action browse with url http://[...]/%%peers%%
3> define action bookmark
4> define object 'self-description'
5> define object 'peers' with value 'group_a'
6> define object 'selected descriptions'
7> define tool VideoWiki with url http://videowiki.icamp.eu
8> define tool Scuttle with url http://scuttle.icamp.eu
9> connect tool VideoWiki with tool Scuttle
10> compose object 'self-description' using tool VideoWiki
11> browse object 'peers' using tool VideoWiki
12> bookmark object 'selected descriptions' using tool VideoWiki
13> drag tool VideoWiki to column 1

```

Figure 2: Example LISL script for activity ‘getting to know each other’

As mentioned before, we differentiate between a functional and a non-functional view on learning environments. On the one hand, the action statements (lines 1-8 and 10-12) refer to the functional requirements of the learning environments and comprise the learning tools required and defined by the learners for their specific situation (activity). Connecting two or more tools, furthermore, leads to new functionality (e.g. bookmarking VideoWiki entries in the Scuttle tool) and increases the powerfulness of this approach so that learners can reuse and mix functionality for more sophisticated (real-life) situations.

On the other hand, non-functional requirements cover the learner interactions with the environment and are therefore dependent on the overall environment, the single tools, and the learner interactions with it. Such statements are hard-coded for specific integrative frontends, like Yahoo Pipes, iGoogle, or our web application mashup solution. For instance, line 13 indicates that learners can drag and drop windowed tools from one to another position along a grid-based mashup space on the screen. Further statements are about connecting, minimising, maximising or closing tools, and so forth.

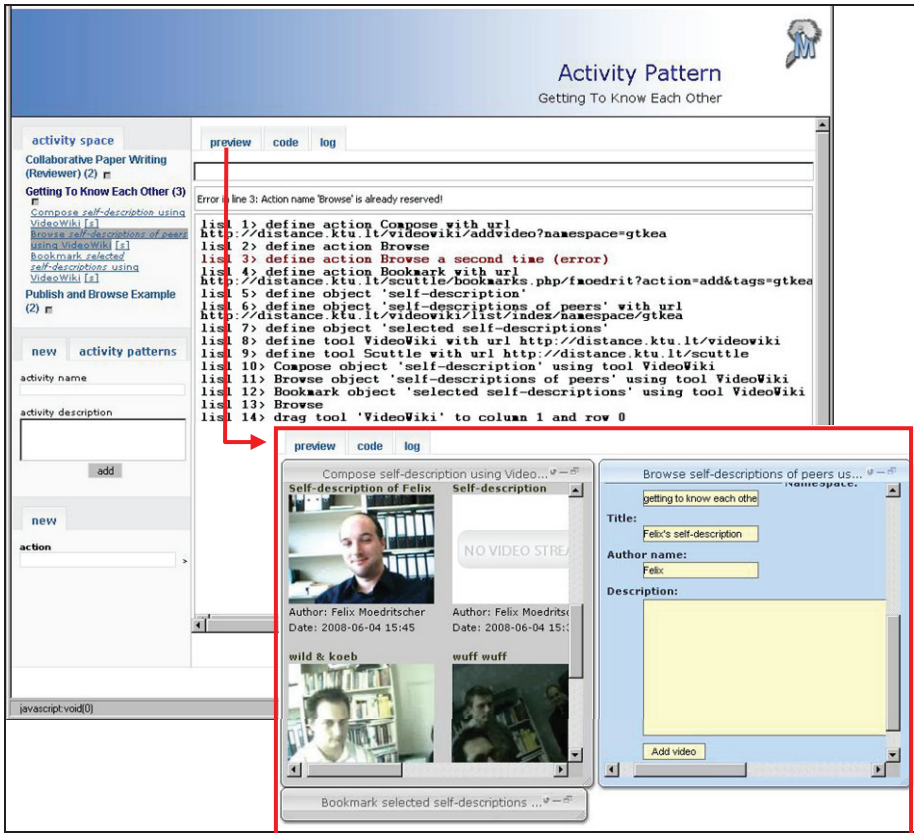


Figure 3: MUPPLE page for the LISL script in Fig. 2

Using this scripting language, it is possible to capture (‘externalise’) environment design capabilities and learner interactions with MUPPLE and, more importantly, to exploit these scripts in some way, e.g. for automated generation of recommendations or user-driven practice sharing. Within our first prototype, the result of such a LISL script is a so-called MUPPLE page, as shown in Fig. 3. Learners, in fact, do not need to create these pages by manually writing LISL code but can use web-based widgets, which materialises their interactions by appending commands to the LISL script.

The header of a MUPPLE page, as shown on top of Fig. 3, displays the type (activity or pattern) and the title of the learning situation (activity). On the right-hand side, users can navigate through their activity space, create new activities from blank or from the given patterns, and add new action-outcome-tool triples to the opened activity, either by specifying them manually or by making use of the recommendations. Opening one activity executes the LISL script of this page within a web-based interpreter which reconstructs the last state of the MUPPLE page. Thereafter, the user can choose between three possible viewing modes of the page: (a) the tab ‘preview’ displays the tool mashup (bottom right of Fig. 3); (b) the tab ‘code’ shows an in-line editor for the whole LISL code of the page; (c) the tab ‘log’ visualises the result of the web-based interpreter, i.e. also warnings and errors in connection with the current LISL script (in the centre of Fig. 3).

Within our prototype, starting a new action within one activity is equivalent to having a learner to ‘implement’ one functional requirement. In this context, the end-user can manually specify the details of the interaction (action-outcome-tool triple) or use recommendations provided by our prototype. Therefore, MUPPLE empowers learners to design their environment. The non-functional requirements are more or less realised by functions of the whole platform, comprise the learner interactions with their mash-up personal learning environments and are subject to being adapted or extended by us, the PLE developers. So far, the non-functional part of MUPPLE monitors how learners use the environment which is materialised by adding additional LISL code to the current page (like the statement in line 13 of Fig. 2).

In a preliminary evaluation study, we observed that new MUPPLE users who are not at all experienced in environment design prefer working only with the web-based widgets and extensively use the recommendations provided by the system. Those users being familiar with MUPPLE having programming skills or having to prepare a lot of exemplary scenarios (facilitators) slowly proceed to script the functional requirements of their environment. Adding the LISL code for learner interactions, however, is always left to the widget-based wrappers of the platform.

5 Conclusions and Outlook

In this paper, we have sketched a methodology for end-user development of personal learning or working environments and demonstrated how it can be set into practice. Furthermore, we argued for the need to support learners in designing their environment. Overall, user-driven environment design seems to be a promising approach for both knowledge workers and lifelong learners, particularly if they are self-motivated enough. In this context, [Alvesson, 04] argues for specific competencies necessary for knowledge workers (e.g. social competencies or capabilities to orchestrate the interaction process), which we tried to consider with

different features of our MUPPLE platform, like a recommendation service. In terms of important usability components, learnability and efficiency, the required hand-on skills need to be trained by inexperienced users which we tried to realise through emergent behaviour of our PLE solution and help facilities.

However, this first prototype needs to be extended and improved, e.g. with regulation and reflection facilities for collaboration in learning networks or good practice sharing functions. Moreover, the whole MUPPLE approach as well as our methodology for user-driven environment design is based on specific technical requirements, like widgetisation and semantic interoperability of tools, which are often not fulfilled. Although a lot of research and development still needs to be undertaken, user-driven environment design offers a great potential for knowledge workers and lifelong learners if this approach centres the learners and their specific needs and not organisational requirements. Finally, the organisation also might benefit from this learner-centred approach, as the materialisation of environment design and learner interactions is more or less the externalisation of implicit knowledge. From this perspective, the concept of a mash-up personal learning environment can be seen as an enabler for action-oriented, community-driven knowledge management.

Acknowledgements

This work has been produced in the context of ROLE, a research and development project financially supported by the European Union under the ICT programme of the 7th Framework Programme (Contract number: 231396).

References

- [Alvesson, 04] Alvesson, M.: Knowledge work: ambiguity, image and identity, In K. Starkey, S. Tempest, A. McKinlay (Eds.): *How Organizations Learn: Managing the Search for Knowledge*, 2nd edition, Thomson Learning, London, 2004, p. 385-405.
- [Beck & Andres, 00] Beck, K., Andres, C.: *Extreme Programming Explained*, Addison-Wesley, Reading, 2000.
- [Cockburn, 02] Cockburn, A.: *Agile Software Development*, Addison-Wesley, Boston, 2002.
- [Engeström, 87] Engeström, Y.: *Learning by expanding: an activity-theoretical approach to developmental research*, Orienta-Konsultit Oy, Helsinki, 1987.
- [Fischer et al., 04] Fischer, G., Giaccardi, E., Ye, Y., Sutcliffe, A.G., Mehandiev, N.: *Meta-design: a manifesto for end-user development*, *Comm. of the ACM* 47(9), 2004, pp. 33-37.
- [Geels, 04] Geels, F.W.: *From sectoral systems of innovation to socio-technical systems Insights about dynamics and change from sociology and institutional theory*, *Research Policy* 33, 2004, pp. 897-920.
- [Hartmann et al., 06] Hartmann, B., Doorley, S., Klemmer, S.R.: *Hacking, Mashing, Gluing: A Study of Opportunistic Design and Development*, *IEEE Pervasive Computing* 7(3), 2006, pp. 46-54.

- [Jones & Lindstaedt, 08] Jones, S, Lindstaedt, S.: A Multi-Activity Distributed Participatory Design Process for Stimulating Creativity in the Specification of Requirements for a Work-Integrated Learning System, Proc. on Distributed Participatory Design Workshop at the Computer Human Interaction Conference (CHI), Florence, 2008.
- [Koper et al., 05] Koper, R., Giesbers, B., Van Rosmalen, P., Sloep, P., Van Bruggen, J., Tattersall, C., Vogten, H., Brouns, F.: A design model for lifelong learning networks, *Interactive Learning Environments* 13(1-2), 2005, pp. 71-92.
- [Koper & Specht, 06] Koper, R., Specht, M.: Ten-Competence: Life-Long Competence Development and Learning, In M-A. Cicilia (Ed.): *Competencies in Organizational e-learning: concepts and tools*, IGI-Global, Hershey, pp. 234-252.
- [Kraus, 05] Kraus, J.: The long tail of software, 2005, http://bnoopy.typepad.com/bnoopy/2005/03/the_long_tail_o.html (2009-04-02).
- [Lieberman et al., 06] Lieberman, H., Paterno, F., Klann, M., Wulf, V.: End-User Development: An Emerging Paradigm. In H. Lieberman, F. Paterno, V. Wulf (Eds.): *End-User Development*. LNCS, vol. 4321, Springer, Dordrecht, 2006, pp. 1-8.
- [Maguire, 01] Maguire, M.: Methods to support human-centred design, *International Journal of Human-Computer Studies* 55, pp. 587-634.
- [Mödritscher & Wild, 09] Mödritscher, F., Wild, F.: Sharing Good Practice through Mash-Up Personal Learning Environments, *Proceedings of the International Conference on Web-based Learning (ICWL)*, Aachen, 2009. (to appear)
- [Mödritscher et al., 08] Mödritscher, F., Wild, F., Sigurdarson, S.E.: Language Design for a Personal Learning Environment Design Language. *Proceedings of the Workshop on Mash-Up Personal Learning Environments at the European Conference on Technology Enhanced Learning (EC-TEL)*, Maastricht, 2008, pp. 5-13.
- [Nonaka& Takeuchi, 95] Nonaka, I., Takeuchi, H.: *The Knowledge-creating Company: How Japanese Companies Create the Dynamics of Innovation*, Oxford University Press, New York, 1995.
- [Preece et al., 98] Preece, J., Finley, A., Abowd, G., Beale, R.: *Human-Computer Interaction*, Addison-Wesley, Essex, 1998.
- [Stahl, 08] Stahl, C.: *Developing a Framework for Competence Assessment*, Dissertation, Vienna University of Economics and Business Administration, Vienna, 2008.
- [Trist, 81] Trist, E.L.: *The evolution of socio-technical systems: A conceptual framework and an action research program*, Ontario Quality of Working Life Centre, Toronto, 1981.
- [Ulbrich et al., 06] Ulbrich, A., Scheir, P., Lindstaedt, S.N., Görtz, M.: A Context-Model for Supporting Work-Integrated Learning, In W. Nejdil, K. Tochtermann (Eds.): *Innovative Approaches for Learning and Knowledge Sharing*. LNCS, vol. 4227, Springer, Berlin, 2006, pp. 525-530.
- [Wild, 09] Wild, F. (Ed.): *Mash-Up Personal Learning Environments*, iCamp Deliverable D3.4, http://www.icamp.eu/wp-content/uploads/2009/01/d34_icamp_final.pdf (2009-03-30).
- [Wild & Sigurdarson, 08] Wild, F., Sigurdarson, S.E.: Distributed Feed Networks for Learning, *European Journal for the Informatics Professional (UPGRADE)* 9(3), 2008, pp. 51-56.