Fischer, G. & Scharff, E. *Learning Technologies in Support of Self-Directed Learning*. Journal of Interactive Media in Education, 98 (4) [www-jime.open.ac.uk/98/4]

Published 22 Oct. 1998
ISSN: 1365-893X

# Learning Technologies in Support of Self-Directed Learning

**Gerhard Fischer, Eric Scharff**

## Abstract:

Self-directed learning is a continuous engagement in acquiring, applying and creating knowledge and skills in the context of an individual learner's unique problems. Effectively supporting self-directed learning is one of the critical challenges in supporting lifelong learning. Self-directed learning creates new challenging requirements for learning technologies. *Domain-oriented design environments* address these challenges by allowing learners to engage in their own problems, by providing contextualized support, and by exploiting breakdowns as opportunities for learning.

*Economies of educational knowledge* constitute an emerging concept in which communities contribute toward the creation of information repositories, which can be reused and evolved by all members of the community for the creation of new environments. We argue and demonstrate that domain-oriented design environments can serve as models for these economies, that a software reuse perspective provides us with insights into the challenges these developments face, and that the creation and evolution of these economies are best understood as problems in self-directed learning.

## Keywords:

Self-directed learning; lifelong learning; domain-oriented design environments; economy of educational knowledge; reuse; seeding, evolutionary growth, reseeding

## Demonstrations:

A demonstration of the *WebNet* system described in this article can be found at <http://www.cs.colorado.edu/~gerry/WebNet/site/webnet.htm>

## Commentaries:

All JIME articles are published with links to a commentaries area, which includes part of the article's original review debate. Readers are invited to make use of this resource, and to add their own commentaries. The authors, reviewers, and anyone else who has 'subscribed' to this article via the website will receive email copies of your postings.

Gerhard Fischer & Eric Scharff. *Center for LifeLong Learning and Design (L3D), Department of Computer Science and Institute of Cognitive Science, University of Colorado, Boulder, CO 80309-0430, U.S.A.* http://www.cs.colorado.edu/~gerhard, *http://rtt.colorado.edu/~scharffe, {gerhard, Eric.Scharff}@cs.colorado.edu*

## 1. Introduction

The previous notion of a divided lifetime—education followed by work—is no longer tenable. Learning can no longer be dichotomized, spatially and temporally, into a place and time to acquire knowledge (school) and a place and time to apply knowledge (the workplace). In the emerging knowledge society (Drucker, 1994), an educated person will be someone who is willing to consider learning as a lifelong process. More and more knowledge, especially specialized knowledge, is acquired well past the age of formal schooling, and in many situations through educational processes that do not center on traditional schooling (Illich, 1971). Seen in this context, working, learning, and collaboration become intimately intertwined rather than being three different and distinct activities.

Lifelong learning has emerged as one of the major challenges for the worldwide knowledge society of the future. Lifelong learning has been given considerable international attention by the European Community, which proclaimed 1996 to be the "European Year of Lifelong Learning," and by UNESCO, who has included "Lifetime Education" as one of the key issues in planning for the future. The G7 group of industrialized nations has named "Lifelong Learning" as a main strategy in the fight against unemployment. Despite this great interest, there are few encompassing efforts to tackle the problem in a coherent way. Lifelong learning cannot be investigated in isolation by looking at only one small part of it, such as K-12 education, university education, or worker re-education. Lifelong learning needs to promote effective educational opportunities in the many learning settings through which people pass, including home, school, work, and communities.

Supporting lifelong learning requires a suite of complementary approaches including intelligent tutoring systems, design environments, performance support systems, on-demand learning, coached simulation systems, intelligent help and advisory systems, and collaborative systems. Exploring these systems has been the shared objective of the East/West Consortium (see Spohrer, et al, 1998, this issue). This article focuses on one of the most important approaches in support of lifelong learning: *supported self-directed learning*. Self-directed learning is critical when learning becomes an integral part of life—driven by our desire and need to understand something or to get something done, instead of merely solving a problem given in a classroom setting. A lifelong learning perspective implies that schools and universities need to prepare learners to engage in self-directed learning processes because this is what they will have to do in their professional and private lives outsides of the classroom.

The challenge for environments supporting self-directed learning is to allow learners to work on authentic problems and tasks of their own choosing, and yet still provide them with learning support contextualized to their chosen problem. Although repositories of objects are an

important resource for designers creating artifacts within a certain domain, economies of educational *knowledge*, not just of *objects*, must address the self-directed learning needs of the community that the economy is meant to support. These challenges, some of our approaches to them, and their illustrations in the context of educational economies of knowledge form the core of this article.

This article will first discuss in section 2 the need for and the theoretical grounding of self-directed learning. Section 3 describes domain-oriented design environments, their structure, and process models for their creation and evolution. Section 4 uses different systems, which represent pieces of economies of educational knowledge (such as Gamelan, the Educational Object Economy, and the AgentSheets Behavior Exchange), to illustrate the need to support self-directed learning to make these environments sustainable efforts that enhance the practice of knowledge workers. Section 5 discusses the strengths and weaknesses of different economies of educational knowledge and articulates some specific challenges for future research.

## 2. Self-Directed Learning

### 2.1 Current Theories of Learning

Current trends in educational theory make the following fundamental assumptions about *learning* (Resnick, 1989):

Learning is a process of **knowledge construction** not of knowledge recording or absorption (Harel & Papert, 1991)—and it requires environments in which learners can be active designers and contributors rather than passive consumers (Fischer, 1998). Research in end-user programming and end-user modifiability (Nardi, 1993) contributes toward this goal. In systems such as AgentSheets and Hypergami (see Repenning *et al.* and Eisenberg and Eisenberg, 1998, this issue), learners construct new knowledge through interaction with the system and the creation of new artifacts with these software tools. Actively constructing knowledge engages learners and emphasizes the need for learners to construct knowledge in a manner appropriate for them.

Learning is highly **tuned to the situation** in which it takes place (Lave & Wenger, 1991; Suchman, 1987)—requiring environments that are domain-oriented and support human problem-domain interaction (the connection between people and the domain specific problems that they face) and not just human-computer interaction. In a typical activity (such as working or playing), individuals are acting until they encounter a *breakdown* and they reflect about the breakdown (Fischer *et al.*, 1993). These breakdowns (originating from missing knowledge, misunderstandings about the consequences of actions, and so on) are key to situated learning.

Schön (Schön, 1983) calls this reflection-in-action, Norman (Norman, 1993) calls it experiential and reflective mode. Because self-reflection is difficult, a human coach, a design critic, or a teacher can help the learner to identify the breakdown situation and to provide task-relevant information for reflection. In our own work, we have explored the possibility of using computational critics (Fischer *et al.*, 1993; Sumner *et al.*, 1997) to provide some of this support when humans are not present. Critics support learners in their own activities by contextualizing existing knowledge within a certain design situation. The information spaces presented and the information provided should be made relevant to the task at hand—something that computational media can achieve, but is impossible for paper and pencil technologies.

Learning is **knowledge-dependent**, meaning people use their existing knowledge to construct new knowledge, requiring environments that support user-tailored information presentations such as *differential descriptions* of new information. For example, if someone who knows MS Word wants to learn HTML, the explanations and examples provided should be different than those given to a learner who knows Framemaker. The cognitive models of users constructed by Intelligent Tutoring Systems (see Ritter et al., 1998, this issue) provide some support for presenting knowledge in a form targeted to a specific user. Design critics may be used to tailor information so that it is relevant to the current task.

Learning needs to account for **distributed cognition** (Norman, 1993), by which the knowledge and effort required to solve a problem is distributed among various participants. The distribution of knowledge among humans is based on the "symmetry of ignorance" (Rittel, 1984) or asymmetry of knowledge between different stakeholders in problem solving. Teaching in classrooms is often conceptualized very differently: it is often fitted "into a mold in which a single, presumably omniscient teacher explicitly tells or shows presumably unknowing learners something they presumably know nothing about" (Bruner, 1996). A critical challenge is a reformulation and reconceptualization of this impoverished and misleading conception. Although this model may be more realistic for early school years (Hirsch, 1996), it is obviously inadequate for self-directed learning processes as they occur in lifelong learning, where knowledge is distributed among many stakeholders and "the answer" does not exist or is not known. Group discussions, conversations around dinner tables, and classrooms have the potential to be places where knowledge is created and constructed by communities of mutual learners. Distribution of knowledge is a central concept for collaboratively constructed knowledge repositories such as Gamelan and the Educational Object Economy (see Sections 4 and 5).

Learning is affected as much by **motivational issues** (Csikszentmihalyi, 1990) as by cognitive issues—requiring environments that let people experience and understand why they should learn and contribute something. For example, *learning-on-demand* (Fischer, 1991) lets users

access new knowledge in the context of actual problem situations and delivers information about which they are unaware in the context of *their* problem situations. Environments must allow users to take pride in their contributions and be awarded for them.

Learning is not limited to any discrete group of individuals. Even though educational systems deal with "teachers" and "learners" as separate groups, in reality these labels are not universally applicable. As tasks and responsibilities change, all individuals must be continuously learning. For example, educators who are traditionally "teachers" may desire to investigate new technologies to use in their classrooms. However, when exposed to new technology and methodologies for creating educational curricula, teachers become learners in order to understand how to use new opportunities effectively.

## 2.2 Self-Directed Learning: Beyond the "Gift Wrapping" Approach of New Media

One of the major misunderstandings in our current debate about enhancing learning with new media is the assumption that technological advances will, by virtue of their very existence, improve the quality of learning. New technologies and media must be more than add-ons to existing practices. New technologies and learning theories must together serve as catalysts for fundamentally rethinking what learning, working, and collaborating can be and should be in the next century.

A major finding in current business reengineering efforts is that the use of information technology had disappointing results compared to the investments made in it (Landauer, 1995). A detailed causal analysis for this shortcoming is difficult to obtain, but it is generally agreed that a major reason is that information technologies have been used to mechanize old ways of doing business, rather than fundamentally rethinking the underlying work processes and promoting new ways to create artifacts and knowledge.

We claim that a similar argument can be made for current uses of technology in education: it is often used as an add-on to existing practices rather than a catalyst for fundamentally rethinking what education should be about in the next century. For example, the "innovation" of making transparencies available on the Web rather than distributing copies of them in a class takes advantage of the Web as an electronic information medium. This may change the economics of teaching and learning, but it contributes little to introducing new epistemologies. Old frameworks, such as instructionism, fixed and "balkanized" curricula, memorization, decontextualized rote learning, and so forth, are not changed by technology itself. This is true whether we use computer-based training, intelligent tutoring systems, multimedia presentations, or the Web.

In the *"gift-wrapping"* approach, technology is merely wrapped around old frameworks for education. What is needed instead are richer conceptual frameworks, leading not just to the addition of technology to existing practices, but to the exploration of fundamentally new possibilities and limitations of computational media on how we think, create, work, learn, and collaborate. To move beyond "gift-wrapping" in the context of self-directed learning leads to the following requirements for computational environments.

Such systems must:

- be simultaneously *user-directed* and *supportive*, i.e., the choice of tasks and goals (including the learning opportunities offered) must be under the control of the user/learner, and the support provided by the system must be contextualized to the user's task;
- be sufficiently *open-ended* and *complex* that users will encounter breakdowns. The system must provide means for allowing users to understand, extricate themselves from, and learn from these breakdowns;
- provide means for significant modification, extension, and evolution by *users*;
- support a *range of expertise*, because such systems will be employed over long periods of time by their users and must be able to accommodate users at progressively different levels of expertise;
- must *promote collaboration* by supporting people to overcome the symmetry of ignorance and allow stakeholders to learn from each other and create mutual understanding.

## 3. Computational Support for Self-Directed Learning

Creating computational environments in support of self-directed learning prohibits designers from completely anticipating and determining the use context (as is done in intelligent tutoring systems), because this context is only partially known at design time. Figure 1 differentiates between two stages in the design and use of an artifact. Often, system developers create environments and tools, including help systems, guided tours, forms, etc., where they have tried to anticipate at design time the situational contexts and tasks users will be engaged in. For print media, a fixed context has to be decided at *design time*, whereas for computational media, the behavior of a system at use time can take advantage of contextual factors (such as background knowledge of a user, the specific goals and objectives of a user or the work context) *only known at use time* (Fischer *et al.*, 1993). The fundamental difference is that computational media have interpretive power: they can analyze and critique the artifacts created by users—and users acting

as designers will create artifacts of all kinds. The challenge is to create new innovative system components that allow users to articulate these contextual factors.

## 3.1 Contrasting Different Computational Approaches to Self-Directed Learning

*Intelligent tutoring systems* (Wenger, 1987) represent a teacher- or system-driven approach toward learning in which *the problem or the task is determined at design time*. At use time, these systems use their intelligence to individualize instruction with user modeling techniques and support the learning process by providing feedback to users' solutions, visualizations, and simulations. In general, users of a tutoring system are learners who interact with information and solve problems previously provided by teachers at design time. (Of course, the creators of the tutoring system were themselves learners when creating the tutoring system, but their activities of self-directed learning are not captured within the system).

*Interactive learning environments* (Papert, 1980) are learner-driven by providing powerful programming environments, often enriched with domain-specific abstractions and microworld support (Repenning & Ambach, 1997; Resnick, 1996) that enable learners to tackle complex problems. But during use time, they provide little support when the learner gets stuck, offer only limited feedback on the artifacts created, and have restricted access to information spaces behind the artifact (such as design rationale or a catalogue of related solutions). Thus, users of these systems must act as teachers and learners at the same time. Without capturing, selectively presenting, or sharing information, users are individually completely responsible for constructing and reflecting upon information.

*Domain-oriented design environment*s model domains but not individual tasks within the domain. These environments are intermediate between the other two approaches by providing a more distributed approach to interacting with domain problems. They face the challenge to (at least) partially "understand" the activity in which the learner is engaged. A framework for solving problems within a domain is provided at design time, and learners create artifacts of their choosing in a self-directed fashion (and extend the domain framework) at use time. Sources that are used to provide domain-specific assistance include: (1) the focus on the domain, (2) the partial construction of an artifact, (3) the partial specification provided by a learner, and (4) the information spaces visited (Fischer & Nakakoji, 1994).

## 3.2 DODEs: Environments for Self-Directed Learning

Domain-oriented design environments (DODEs) (Fischer, 1994) address the problem of self-directed learning by providing concrete learning support within a particular domain. Users

learn as they design artifacts following their unique interests and needs. Instead of attempting to incorporate knowledge about specific problems at design time, DODEs provide a framework that allows users to construct many different problems *within* a domain in which they are interested at use time. Example domains that we have explored in the past include kitchen design (Fischer et al., 1993), graphical user interface layout (Fischer *et al.*, 1990a), telephone voice messaging systems (Sumner, 1995), and Local Area Network design (Fischer *et al.*, 1992). We believe that some of the techniques used in the development of DODEs are important ingredients, comprising economies of educational knowledge as well as being effective tools for supporting such economies.

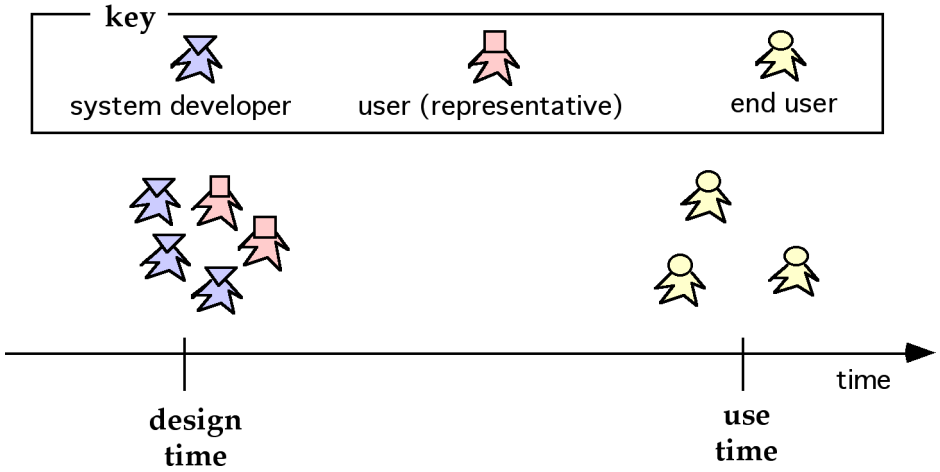


Figure 1: *Design and Use Time*

Figure 2 illustrates a prototype for a DODE for Local Area Network (LAN) design [1] . In contrast to general-purpose environments, DODE components are instantiated in the framework of the given domain. Components are expressed in terms of domain concepts, and information is presented in the context appropriate for that domain. DODEs provide specific functionality for manipulating, exploring, and communicating about domain entities. The components of a DODE (illustrated by the numbered elements in Figure 2) include the following:

- A specification component (4) allows the specification of design constraints and goals by users during use time. This provides the system with a more specific understanding about particular tasks at hand and enables the system to offer guidance and suggestions relevant to those situations.

[1] *WebNet is a prototype DODE for Local Area Network (LAN) design. <http://www.cs.colorado.edu/~gerry/WebNet/site/webnet.htm>*

- Domain-specific components (2) contain the individual elements that designers use to create design scenarios. These components, which can be shared and modified, provide a language of concrete items that augment communication through a domain.

- Critiquing mechanisms (not shown) represent the accumulated "wisdom" of a design community, such as criteria and rules about what constitutes "good" design. Critiquing mechanisms monitor design actions, provide explicit and task-relevant feedback, and identify breakdowns in a design developed by users at use time. This feedback leads to opportunities for self-directed learning.

- Organizational and artifact memories (1) support the capture of design rationale and argumentation embedded within design artifacts. Embedding rationale within designs allows users to explore the rationale behind specific parts of an artifact and to use the artifact as a tool to ground domain communication.

- Case libraries (5) allow reuse at a higher level of granularity than individual components. They facilitate a different kind of reuse and design-by-modification methodologies by modifying previously constructed artifacts.

- Simulation mechanisms (3) support users in understanding the behavior of a component or a complete artifact.

The combination of approaches used in the DODE framework go beyond the "gift-wrapping" approach and address the challenges faced in supporting self-directed learning. Specifically:

- Orienting use of the system through construction of objects within a domain gives the user choice over what artifact to construct (and grounds learning within authentic activity). *Construction is user-directed*, and critics support situated learning by analyzing the user's current construction.

- Employing *design critics* that can analyze different situations creates an environment in which users will be informed of problems as they arise. These breakdown situations provide opportunities for users to understand and learn from the breakdown within the context of their personal design activities.

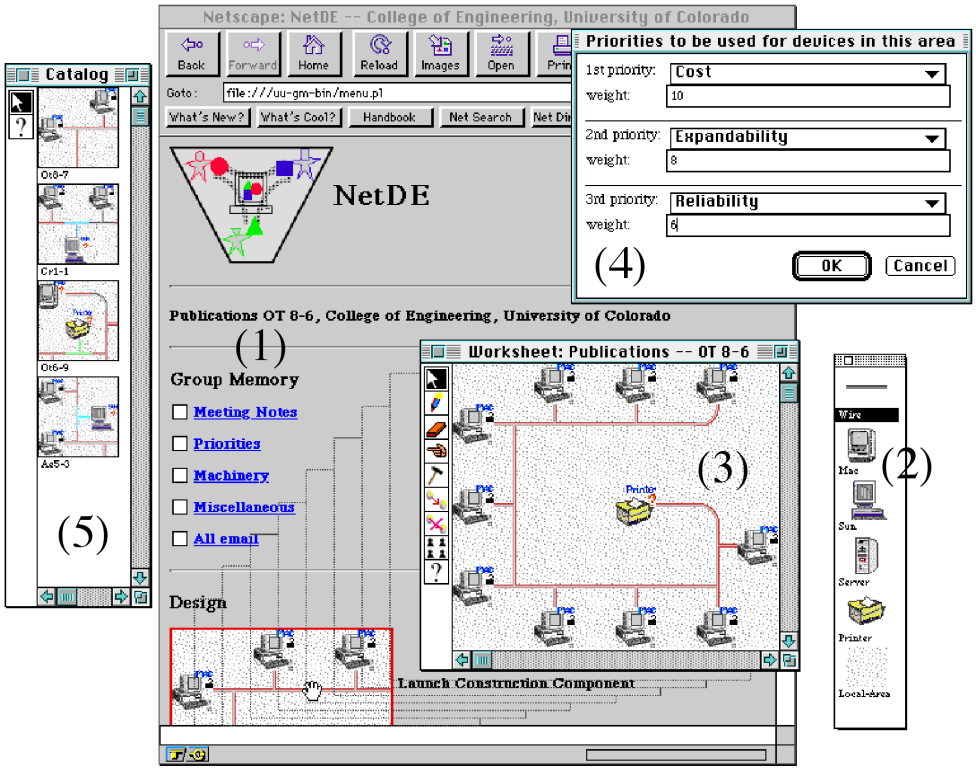


Figure 2: *Example of a Domain-Oriented Design Environment for Computer Network Design*

- Supporting different *levels* of modification (including changing organizational memories, adding new components, creating new cases for the case library, and altering the domain simulation) provide support for a wide range of expertise, providing the ability to augment the system when necessary but not requiring users to do so. Applying different sets of critics and using specification components also supports different levels of expertise by providing users with different schemas for analyzing their domain-centric activities.

- Providing *mechanisms* for evolution such as case libraries and organizational memories allows users to easily add knowledge to the existing framework. Simulations and critiquing systems serve as the driving support for evolution. End-user modification mechanisms are built to facilitate changes using domain concepts, minimizing the need for a shift from domain activity to changing low-level system implementation.

- Encouraging collaboration is inherent as users working within the same domain can easily share their new ideas and changes. Individual components, catalog examples, and rationale about designs are not static entities in DODEs. As users interact with a domain oriented environment, they create and compose new artifacts that themselves become part of the system. Therefore, components in a DODE are specifically designed to handle an ever-changing domain space to cope with the constant flux of problems in the real world.

## 3.3 The Seeding, Evolutionary Growth, and Reseeding Process Model

Most intelligent systems (including systems in support of learning such as Intelligent Tutoring Systems (Wenger, 1987), Expert Systems (Stefik, 1995), and Simulation Environments such as SimCity [2] ) have traditionally been developed as "closed" systems. The basic assumption was that during design time, a domain could be modeled completely by bringing domain experts (designers) and environment developers (knowledge engineers) together and the knowledge engineers would acquire the relevant knowledge from the domain experts and encode it into the system. This approach fails for the following reasons: (1) much knowledge is tacit and only surfaces in specific problem situations; and (2) the world changes, and intelligent systems that model this world must change accordingly. Thus, closed systems are inadequate to cope with the tacit nature of knowledge and the situatedness of real-world problems.

DODEs are designed as "open" systems, where opportunities for change are built in as a central part of the system. Tools and techniques developed for DODEs present important examples and highlight specific challenges for supported self-directed learning environments. By providing components that incrementally evolve, these environments allow a constant flow of input from designers and users, bridging the gap between "design time" and "use time." Although complex systems must evolve in order to be effective (Simon, 1996), it may not be clear how to conceptualize the changes that will take place over time in these systems. In our research, we have developed the Seeding, Evolutionary Growth, and Reseeding process model (Fischer *et al.*, 1994) to address these problems. This model has been explored in our work with DODEs and postulates three major phases:

A **seed** will be created through a participatory design process (Henderson & Kyng, 1991) between environment developers and domain designers. A seed is not a fully realized system but instead is a sufficiently expressive entity that can be used to address some specific real-world problems. Mechanisms for evolution must be built into these initial conceptualizations. Postulating a seed as an objective (rather then a complete domain model or a complete

[2] *SimCity is a registered trademark of Maxis, Inc.*
*See <http://www.maxis.com/games/simcity2000>*

knowledge base) sets this approach apart from other approaches in intelligent systems development and emphasizes evolution (Dawkins, 1987; Popper, 1965) as the central design concept.

**Evolutionary growth** takes place as individuals use the seeded environment to undertake specific projects. During these design efforts, new requirements may surface, new components may come into existence, and additional design knowledge not contained in the seed may be articulated. During the evolutionary growth phase, the environment developers are not present, making *end-user modification* (Girgensohn, 1992; Nardi, 1993) a necessity rather than a luxury. End-user programming both supports learning (by leading to the creation of a new computational artifact) and requires learning (in order to be able to create the new artifact). Figure 3 illustrates this dual relationship between learning on demand from a system and the simultaneous need to add knowledge to the system.

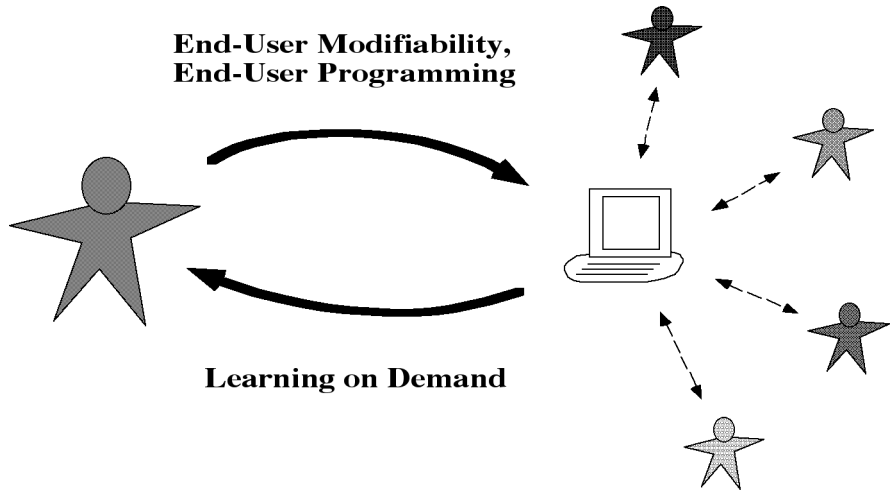


Figure 3: *The Duality between Learning on Demand and End-User Modifiability in Self Directed Learning*

**Reseeding**, a deliberate effort of revision and coordination of information and functionality, brings the environment developers back to collaborate with domain designers to organize, formalize, and generalize knowledge added during the evolutionary growth phases. Information about how the system has evolved is essential in determining how the system must be reconceptualized. By looking at the system evolution, it is possible to postulate which extensions created for specific design projects should be incorporated into future versions of the generic design environment. Drastic and large-scale changes occur during the reseeding phase.

The SER model is a useful framework for understanding the processes inherent in the development of *open systems*. For example, the development of open-source software systems such as the Linux operating system (Raymond, 1998) provides an interesting existence proof that reliable, useful, and usable complex systems can be built in a decentralized "Bazaar style" by many, rather than in a centralized, "Cathedral style" by a few. The Linux development model treats users simultaneously as co-developers and self-directed learners (see Figure 3) and is currently being tested in a number of new areas (e.g., in the *Netscape Communicator* [3] ). Gamelan, the Educational Object Economy, and the AgentSheets Behavior Exchange are three examples of open systems that are forming incipient economies of educational knowledge, we will now analyze using the SER framework .

## 4. Economies of Educational Knowledge

Economies of educational knowledge (EoEK) address the problem of self-directed learning by leveraging the knowledge of the (world-wide) community to promote knowledge dissemination. One might consider the entire Web to be an EoEK in which a distributed community presents publicly available information on any subject of interest to the contributors. However, the mere existence or availability of information does not imply that this information will be useful. Web-based EoEKs tend to focus on supporting certain kinds of knowledge. The idea of using a specific domain (and supporting a community of practice) that has emerged in successful EoEKs has been a central concept in the DODE framework for some time.

The idea for economies based on the interchange of educational knowledge is not new. More than 25 years ago, Illich (Illich, 1971) introduced the concept of "learning webs," a scheme for transforming the creation and dissemination of knowledge into a problem in which all people play an important role. Illich envisioned a world in which the mass distribution capabilities of the currently extant technology could be used to facilitate access to and sharing of information. Believing that people are capable of being both teachers and learners depending on the circumstances, Illich envisioned an economy that encouraged people to become active teachers and producers of educational knowledge as a result of self-directed learning activities.

The rapid growth and increasing ubiquity of the Web have made it a popular vehicle for establishing educational knowledge repositories. Three informative examples of current repositories are Gamelan, the Educational Object Economy, and the AgentSheets Behavior Exchange. All three are Web-based systems that share a common goal: to support the learning needs of a community of software developers. Gamelan is a resource for information about Java technology, the Educational Object Economy provides computational resources for designing educational software, and the Behavior Exchange allows AgentSheets simulation developers to

[3] *Mozilla.org is a group within Netscape that is chartered to act as a clearing-house for the Netscape source. <http://www.mozilla.org>*

share simulations and simulation components. However, as our analysis will show, these systems do not adequately support the interchange of educational *knowledge* per se. Repositories alone leave it up to learners to frame information in a way appropriate to their problems and self-directed learning activities. In their current state, they provide little support for locating, comprehending, and modifying information relevant to the task at hand (see Figure 8). Thus, while community repositories of information are certainly key components, the following analysis will highlight additional necessary components for a sustainable EoEK.

## 4.1 Gamelan

One prime example of a first step toward an EoEK based on community participation is taking form in the software design community. Java developers use the Web to facilitate the learning of Java and to share components created in Java. Due to the contributions of developers around the world, the Java programming community has used community repositories of knowledge to produce technical advances in a very short period of time. **Gamelan** [4] is one of the first community repositories for Java-related information. The primary users of Gamelan are Java developers looking for information about what other people are doing with Java. Gamelan is therefore a forum to facilitate the self-directed learning of members of the emerging Java community. The software developers who use the content are also the primary contributors, continuously adding new resources to the Gamelan repository. The thousands of developers who contribute to the Gamelan repository and the estimated thousands who search for information in Gamelan every day provide evidence that the Java community has taken a great deal of interest in using community repositories to share information. Gamelan was originally designed to be the official clearinghouse for all third-party uses of Java, and the site attempts to support any work that uses Java. Although this encyclopedic coverage attracts developers with many interests, such coverage does not make Gamelan the ideal educational tool. If one considers Gamelan an educational tool used to educate the community of Java developers, the system does not provide a great deal of support for the learning activities that developers must go through to use resources.

[4] *Gamelan was the original repository for Java components. It has grown substantially and now encompasses resources not only about Java but also other Web development technologies. The expanded system (of which Gamelan is now a subset) is called Developer.com <http://www.developer.com>.*

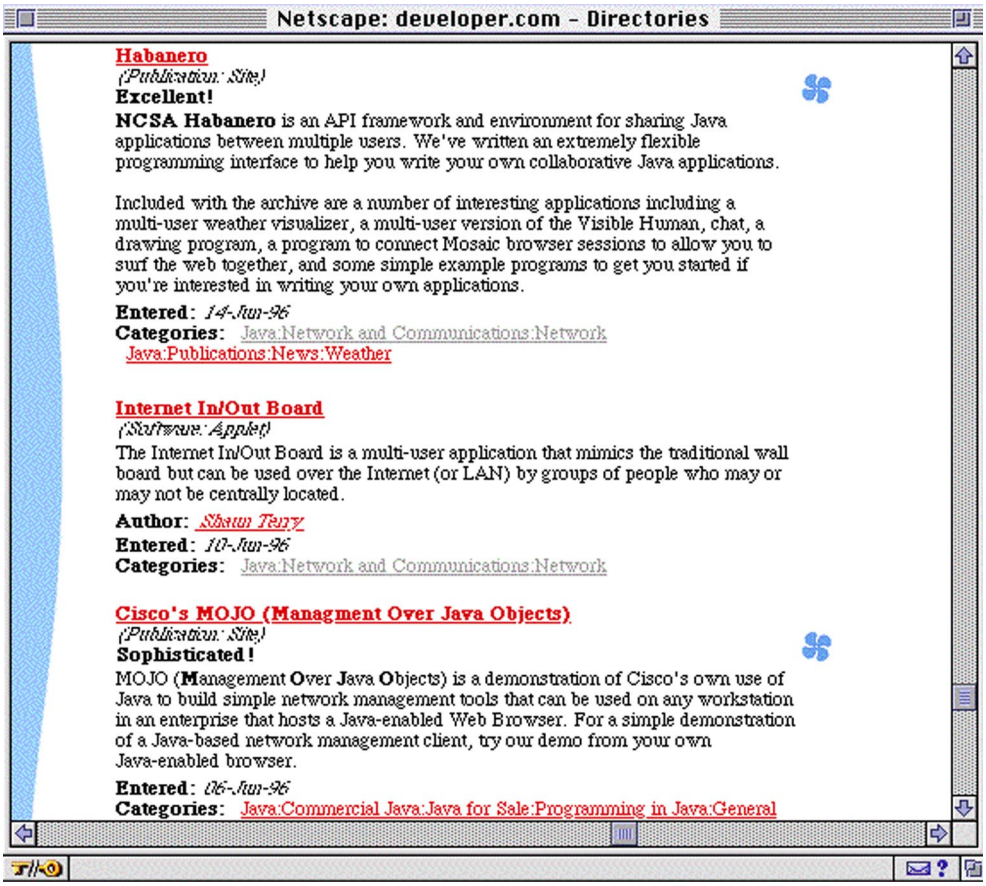


Figure 4: *Browsing Resources Returned by Gamelan*

Gamelan resources belong to one or more categories, which are organized hierarchically. Resources are retrieved by browsing through categories or by searching the titles and brief abstracts of resources. Figure 4 shows a keyword search for network visualization and management tools that resulted in a pointer to two Java categories, in this case the "Java -> Network and Communications -> Network" and "General" categories. Retrieved components appear in unsorted order. All the descriptions are provided by the contributors of the information. The only exceptions are the one-word annotations (marked with fans), which was added by the repository administrators. Gamelan does not provide any mechanisms for refining queries or organizing returned results.

## 4.2 Educational Object Economy

The **Educational Object Economy** (EOE) [5] (see Spohrer, et al., 1998, this issue) provides a more focused system than Gamelan. Currently realized as a collection of Java objects (mostly completed applets) designed specifically for education, the target users of the EOE are teachers (presumably acting as consumers of completed applets) wishing to use new interactive technology and instructional designers interested in producing educational software. The EOE's primary goal is to provide educators with a collection of useful resources ready to be used to help students learn. There is an interesting dichotomy apparent in the EOE. Educators wish to create tools that will facilitate the learning of their students, but the teachers are actually the learners as they search for useful components in the EOE repository.

The EOE has two major access mechanisms: a single level (but somewhat hierarchical) classification scheme based on the Dewey Decimal System, and a keyword or advanced search based on object descriptions.

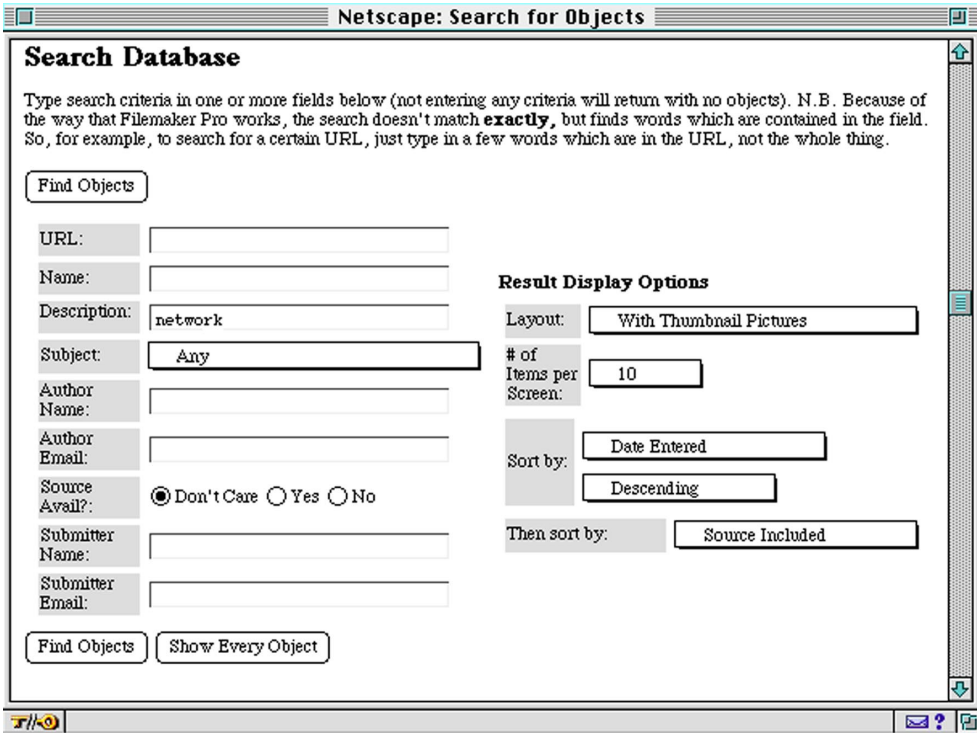


Figure 5: *Search Mechanism for the Educational Object Economy*

[5] *The Educational Object Economy, EOE Foundation <http://www.eoe.org>*

Figure 5 shows the search mechanism for the EOE. The EOE may be queried according to URLs and also by using keywords in the components descriptions. The *meta-data* [6] currently associated with EOE objects include the names, subjects, and descriptions in the left column and the submission dates and source code availability information in the right column. However, the "content-specific" query information must be specified with the predefined subject and keyword matches in the description.

Figure 6 shows the result of the query for (computer) network tools (Querying "Computer Network" yielded no matches, and the computer example shown seems to be the only computer-network related tool in the repository). Like the Dewey Decimal System, there is a confusion about whether computer networks belong in the engineering (621) or Computer Science (004) section. The one relevant EOE component falls only in the latter category. Because of the large number of Computer Science applets, manual browsing is prohibitive. Finding the right category or knowing how to appropriately categorize items is a large problem in both Gamelan and the EOE. This classic *vocabulary problem* (Furnas *et al.*, 1987) is not unique to computer repositories. Objects matching the search specification are returned in tabular form. There is no incremental refinement of queries. As with Gamelan, the repository acts more as an index, than as a storage location. The repository stores abstracts and meta-data, but the artifacts themselves are maintained on the personal pages of contributors. Users must go to individual web sites for further information about the artifacts.

A collection of educational resources is a valuable tool for creating systems that support self-directed learning. However, the act of locating and using educational objects is itself a self-directed learning activity that can benefit from specific support mechanisms. Thus, teachers and creators of educational resources can benefit from supported self-directed learning techniques as much as learners or "end-users" of educational resources. The EOE supports the exchange of domain-specific components for education, but falls short of supporting the self-directed learning needs of educational designers who need to locate useful components, understand how components work, and modify the retrieved components to fit a specific context.

6 *The IMS Meta-data Information Site <http://sdct-sunsrv1.ncsl.nist.gov/~boland/ims.html>*

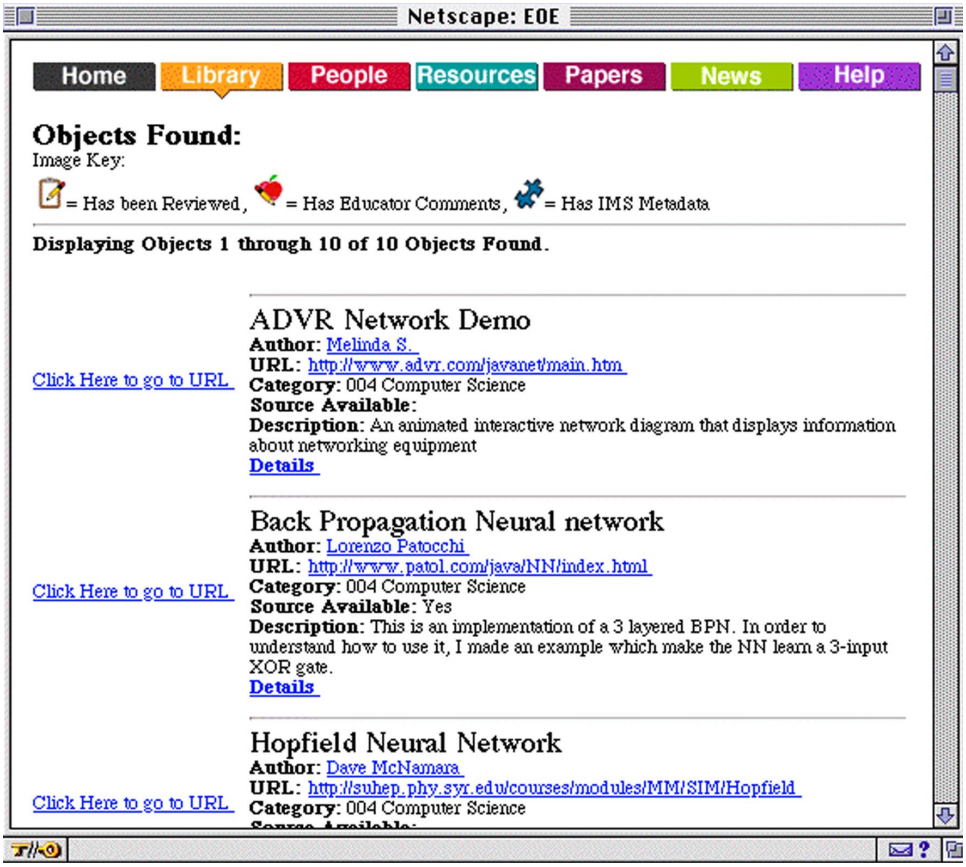


Figure 6: *Results of a Query for Networks Returned by the EOE*

## 4.3 Behavior Exchange

The AgentSheets **Behavior Exchange** [7] (Repenning *et al.*, this issue) is an initial prototype of a domain-specific system for sharing computational artifacts. It is a repository that stores *agents* (entities that can perform computation) created using the AgentSheets system. Like Gamelan and the EOE, the Behavior Exchange consists of a collection of computational artifacts and some "meta-data" specific to each component. The system's focus on agents (and sharing computation by exchanging agents) allows the system to present information in a manner appropriate for AgentSheets programmers. The Behavior Exchange was designed to support the needs of a specific audience who use the AgentSheets system to build simulations. These end-user programmers include college students, primary school students, and professionals in a

7 *The Agentsheets Behavior Exchange <http://agentsheets.cs.colorado.edu>*

variety of domains such as environmental design and biology.

Figure 7 shows a sample screen from the Behavior Exchange in the domain of "animal agents." Components are automatically added to the Behavior Exchange using an uploading mechanism built into the AgentSheets environment. This uploading process combines formal information from AgentSheets with informal information such as categories and descriptions provided by users. The system then uses these forms of information to provide various ways of presenting information in the repository. Multiple categorization schemes (such as projects and categories) are automatically maintained by the system. Agents may be subsequently sorted by name or modification date. Searches may be refined quickly using keyword mechanisms and formulating sets of target categories and projects. The information that can be automatically synthesized from the formal agent definition is combined with information contributed by users to provide a perspective relevant for simulation builders. Once a designer has found an agent that could be useful in a new simulation, this agent can be dragged directly from the Behavior Exchange into AgentSheets running on the user's local computer for immediate use (see Repenning, *et al*, 1998, this issue).


Figure 7: *Retrieving Components with the Behavior Exchange*

## 4.4 Summary: Economies of Educational Knowledge as Open Systems

One important feature common to all three of these systems is their support for evolution. As new educational knowledge becomes available, members of the community may share new developments with each other. In all three systems, the repository administrators set up an initial *seed* that structures how information is added, presented, and searched by users. The goal is to create useful information repositories in a decentralized fashion (Resnick, 1994). All three systems allow *evolution* by the community who uses the information, although the centralized authority plays different roles in the different systems. In Gamelan (and to a lesser extent, the EOE), all new resources are verified and filtered by the repository administrator. In the Behavior Exchange, contributions are unrestricted. In fact, contributors have some control over the categorization itself because of the ability to add new projects or categories. Evolution in all the repositories is limited to the *addition* of new content. The ability to *refine* content is limited in all the systems. Because users can add only new resources and all resources stand on their own, it is impossible to track the changes of individual components. Finally, over the past few years, all three systems have gone through dramatic redesign or *reseeding* phases in which content is checked and reformulated and revised, entries are related to each other (which might possibly have been captured during the evolution phase), categorization schemes change, information presentation goes through major changes, and different searching methods are employed. In all three cases, reseeding has been performed by the environment developers based on feedback from the community.

Because all three systems are envisioned as tools that evolve at the hands of a community of users, all three are prime candidates to study the challenges, strengths, and weaknesses of open systems. The SER process model that accounts for the evolution of DODEs is a useful framework for understanding the changes that have taken place in these systems. Table 1 summarizes the three systems and how they can be understood using the SER framework. Although most of the repositories were designed for evolution, the seeding and reseeding phases have also played important roles in their development. Having an understanding of the process model beforehand may be beneficial; for example, Gamelan went through a radical reconceptualization when the designers switched from using a single category to using multiple categories for a resource. This change was in large part due to the inflexibility of the categorization scheme - because users could not change categories and because existing categories were ambiguous or insufficient, the existing structure quickly became difficult to navigate. Because all changes are eventually handled by Gamelan staff, the evolution that the community could provide instead became the responsibility of the site managers. As illustrated by the other two systems, the more that users become 'co-developers', the more the repository begins to resemble an idealized EoEK.

| | Seeding | Evolutionary Growth | Reseeding |
|---|---|---|---|
| **Gamelan** | System builders create complete initial category structure, tags for special entries, etc. Initial content came from Sun third-party links (very few). | Users can add only new resources to the existing structure. Submissions are reviewed by site managers before inclusion in the repository. | Recatigorization and new special tags are added frequently by site managers. Broken links removed occasionally. Layout and search change often. Multiple categories for each entry added. |
| **Educational Object Economy** | System builders provide single-tier categorization, predefined slots for information, provision for meta-data. Initial content located by system builders. | Users can add resources to the existing structure. Users browsing the site can add comments and vote on resources but cannot change them. Meta-data are provided once on submission but are not updated by contributors. | Additional meta-data slots added by system developers, search facility has changed. |
| **Behavior Exchange** | System builders provide initial categories and browsing mechanisms. Initial content provided by system builders and small group of users. | Users can add resources to existing structures or create new categorization mechanisms by adding new categories or projects. Resources do not change after upload, but they may be down-loaded, changed, and resubmitted by anyone. | Reseeded by system builders to add more powerful search facility. |

Table 1: *A SER Perspective on Gamelan, the EOE, and the Behavior Exchange*

## 5. Challenges for Economies of Educational Knowledge

"Hit counts," the number of times a certain resource has been accessed, seem to be the most common way to evaluate the efficacy of Web-based information repositories. The success of systems such as Gamelan are usually given with database usage statistics. Frequently encountered database statistics include the number of people who visit a Web site or the number of resources indexed in a repository. Although these popularity statistics are important, neither of these metrics provides an insight into how well these systems truly support users engaged in self-directed learning activities. This is not to say that such database resources are worthless—

a centralized repository where software developers can find resources has proved to be invaluable for Java developers. Warehouse repositories are simply insufficient tools to support a learning community.

Instead, we must analyze the specific needs of users of an EoEK and use technology to help address these needs. The primary goal of the EoEKs described above is to support software developers acting in the context of their self directed learning activities. Designers (whether a Java programmer or an instructional designer) who want to take advantage of an existing EoEK will be driven by their own goals and objectives, which requires, by necessity, support for self-directed learning processes. To address the self directed learning efficacy of these systems, we present a model for understanding the self-directed learning needs of software developers and how the current repositories address (or might address) these needs.

## 5.1 Supporting the Location / Comprehension / Modification Cycle

The long-term goal of an EoEK is thwarted by an inherent design conflict: to be useful, an economy must provide many building blocks, but when many building blocks are available, finding and choosing an appropriate one becomes a difficult problem. Even if an appropriate block is found, it is rarely usable as is and usually must be modified to suit the new use context (see Roschelle, *et al*, 1998 for further discussion of this issue). Based on our investigations, as well as others, we are convinced that to make an EoEK a success, substantially more is required than creating objects and depositing them in a globally accessible information repository. Figure 8 illustrates three essential processes as they occur in using an EoEK: location, comprehension, and modification (Fischer *et al.*, 1991).

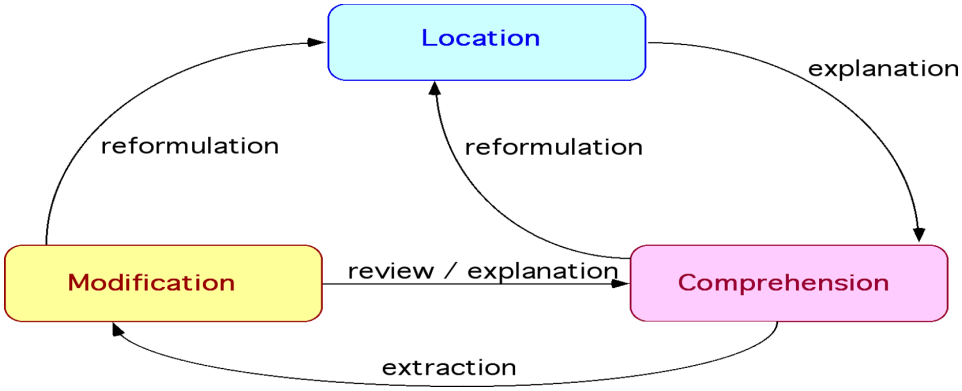


Figure 8: *A Conceptual Framework for Software Reuse*

The location, comprehension, and modification cycle serves as a model for understanding the processes involved in reusing existing components in new situations. Since systems like Gamelan and the EOE provide repositories of components that are intended to be retrieved and reformulated, it is instructive to determine how these systems would address a user's unique self directed learning needs.

We'll illustrate the self-directed learning challenge in software reuse using a specific software design task. In this example, an educator (perhaps a Computer Science professor) wishes to find tools to help students learn about the challenges in real-world computer networks. In particular, the teacher tries to find tools that simulate network behavior or visualize the communication that takes place between computational components. [8] First, the teacher attempts to locate components that could be used to construct a system to explain networks. Then, the teacher needs to *comprehend* the components, determining what each piece does and deciding how the pieces might be used together. The software components would then need to be modified for use in configurations not anticipated during their design time and to address the specific needs of the teacher. Finally, the teacher may wish to share the newly created educational tool with the community by adding it back into the repository. The cycle completes when another teacher wishes to locate a similar resource in another situation.

The first step for a designer would be to **locate** existing tools for simulating or visualizing networks. A typical "global" repository of software components such as Gamelan provides two mechanisms for locating relevant information: browsing a set of predefined categories or searching by keywords. The top-level Gamelan categories, such as "Games," "Commercial Java," and "Arts and Entertainment," may be a reasonable way of indexing the whole space of possible uses of Java, but they provide little guidance about how to proceed to find a specific component such as a network simulation. A viable alternative would be to provide multiple categorization schemes that present different views for browsing. For example, a designer may browse through genres of courses (looking perhaps for other networking courses), or different kinds of applications (such as LAN management). Although more selective queries are possible, a simple query search of an encyclopedic database may not provide useful information; a keyword search for "network" and "visualization" in Gamelan returned no matches (particularly surprising because there are quite a few tools for visualizing networks). Searching for "network" and "simulation" yields a few useful resources and categories (both Java -> Networking and Communications -> Network and "General"). This example demonstrates that query mechanisms that rely exclusively on keyword matching are extremely susceptible to the vocabulary problem (Furnas et al., 1987) common in the mismatch between a "system" model

8 *Although we will focus in this section on the teacher's self-directed learning, the artifact created by the teacher could be designed to facilitate the self-directed learning of the networking student. Using components to create self-directed learning environments is an interesting idea but a full discussion is beyond the scope of this example.*

(used by a system to represent information) and "situation" model (the perspective maintained by the user in a unique situation; see section 5.2 and Kintsch (Kintsch, 1998)). In Gamelan, the user is forced to make this association without support by manually chasing down relevant categories.

Fortunately, better solutions exist. One of our prototype systems for locating code resources (Fischer *et al.*, 1991) combined formal information (extracted from program source texts) and informal information (in the form of on-line documentation) in order to retrieve as much useful information as possible. The system also used spreading activation to help identify relevant objects and provided a mechanism for incrementally refining queries. If the system is capable of analyzing the artifact being constructed (as it is in the case in DODEs), a partial representation and a partial specification (see Figure 2) may be used as the basis for a query. The system could then return previous objects based on the state of the current design. We have developed systems that use text analysis mechanisms such as latent semantic analysis (LSA) (Landauer & Dumais, 1997) to find useful information. Ideally, a developer could type a description of the desired graph visualization system and then incrementally refine the query based on the returned results.

Once the designer finds a set of possibly relevant components, the designer must **comprehend** the retrieved resources. Annotations provided by the Gamelan staff such as "well commented," "sophisticated," "transglobal," and "Zowie!" offer little to aid comprehension. In fact, the repository stores little information about a resource other than a link to a Web page maintained by the resource contributor. It is up to the contributors to present information about the artifacts in question. Although many authors provide a page that demonstrates the functionality of a resource, usually in the form of a Java applet, these examples provided by the authors do not necessarily address the issues that the designer finds useful. It is unrealistic to assume that, with thousands of Gamelan developers, any documentation or demonstration will be written in a manner appropriate to the entire community. Instead, a developer would probably prefer a ranking of relevant resources based on features important for software reuse. The designer may use a specification component (see Figure 2) to say that modular code is more important than fast code, or perhaps that cross platform support cannot be sacrificed. The designer may browse comments created by other users that are associated with the individual components. These discussion forums would help link developers to the appropriate human contact points.

Hopefully, the designer will find a resource that is immediately relevant to the current task, such as a network simulator or visualization tool. Most likely, the developer will need to **modify** existing components such as a graph subsystem and a network statistic- gathering tool (both of which exist in Gamelan) and compose these to create a new tool. If the resource is a Java

component that uses JavaBeans[9], the standard Java component model, it would be possible to use a tool specifically designed to compose and modify these components. Most tools for composition require a significant amount of Java programming knowledge and low-level Java programming. Components that can be integrated or modified with domain-specific programming systems (see Repenning *et al.*, this issue) may make the modification task much easier. Although no single tool can be ideal for every programming task, visual and domain-specific programming languages for composing larger applications from existing modules would facilitate rapid system development and prototyping without requiring large low-level code modifications. A developer may use tools to embed design decisions into the components themselves. Providing the opportunity to track the evolution of individual components would allow future developers to see the specifics of how a system evolved and why those decisions were made.

Unfortunately, traditional repositories such as Gamelan or the EOE do not really support the notion of modifying or refining an artifact. If the designer has augmented an existing network simulator, there would be no way to submit the new "deluxe simulator" and associate it with the components that were extended. There are a large number of components in Gamelan built upon each other, but the relations among these components is virtually impossible to determine. Systems such as the EOE and the Behavior Exchange have recognized the need to associate "meta-data" with every component. Right now these "meta-data" do not contain information about the relations among components, but this kind of information would be an important kind of data about components. The infrastructure for the economy can then track the usage of the new component. Associating artifacts with "meta-data" about the evolution (such as peer reviews, comments, or revision histories) would use the intricate relations among components and associated comments to provide richer information to locate and comprehend components.

## 5.2 DODEs as Models for Economies of Educational Knowledge

In this article, we have identified some basic challenges for self-directed learners interacting with EoEKs (acting primarily as designers of some artifacts). They have to cope with the following challenges in such situations; (1) they do not know about the existence of components; (2) they do not know how to access components; (3) they do not know when to use components; (4) they do not understand the results that components produce for them; and (5) they often cannot combine, adapt, and modify components according to their specific needs without further help or support.

When designers attempt to locate components for use in a new application, they approach the

[9] *JavaBeans is the component architecture for Java. Both JavaBeans and Java are registered trademarks of Sun Microsystems, Inc. <http://java.sun.com/beans>*

search task with their own individual understanding of the world and their own vocabulary. The fundamental challenge of any EoEK is to bridge the gap between a user's application goals and vocabulary (situation models) and the repository's implementation or categorization schemes (system models). DODEs address this gap by providing information spaces and computational tools targeted at specific domains, of interest to certain communities of practices. Thus, the system model of a DODE should already reflect the situation model of the user. Our previous analysis of three different repositories illustrated how more-focused repositories such as the Behavior Exchange acted more like EoEK's than general purpose repositories such as Gamelan.

As previously discussed, the lifecycle of DODE's and other open systems follows the SER model. The SER model is motivated by how large software systems, such as Emacs, Unix, and Linux, have evolved over time. In such systems, users develop new techniques and extend the functionality of the system to solve problems that were not anticipated by the system's authors (following the observation that any artifact should be useful in the expected way, but a truly great artifact lends itself to uses the original designers never expected). New releases of the system incorporate ideas and code produced by users.

Unlike these large software systems, DODEs must address an additional challenge to make the SER model feasible: whereas the people in the above-mentioned environments are computationally sophisticated, DODEs need to be extended by domain designers who are neither interested in nor trained in the (low-level) details of computational environments (Nardi, 1993). Domain designers are more interested in their current task than in maintaining a knowledge base. At the same time, important knowledge is produced during daily design activities that should be captured. Rather than expect designers to spend extra time and effort to maintain the knowledge base as they design, DODEs provide tools to help designers record information quickly and without regard for how the information should be integrated into the environment. Knowledge-base integration is periodically performed during the reseeding phases by environment developers and domain designers as a collaborative activity.

One of our basic claims and assumptions is that most future EoEKs should and will have users who are more like users of DODEs than users of Linux and Gamelan. Because they are neither knowledgeable nor interested in specific programming issues, their self-directed learning activities and their own contributions will be more at the domain level than at the programming level (e.g., the target users of the EOE include teachers wishing to use and contribute new interactive technology and educational software).

Table 2 is an attempt to summarize and compare Gamelan and the EOE along these dimensions, and to derive recommendations for self-directed learning based on our work with DODEs.

| | Gamelan | Educational Object Economy | Recommendations for Self-Directed Learning |
|---|---|---|---|
| **Scope of Coverage** | Encyclopedic–indexes any resource related to Java technology | Wide–stores Java resources with educational uses | Domain-specific pieces classified in appropriate framework for the given domain. |
| **Target Users** | Java developers | Educators / software developers | Individuals interested in a given domain |
| **Content Providers ("Teachers")** | Java developers | Software developers and educators | Members of the design community |
| **Content Recipients ("Learners")** | Java developers | Students (who receive content from teachers) | Members of the design community |
| **When is Content Generated?** | Design Time–contributors submit fixed resources. Self-contained (completely planned) pieces are regarded favorably as "complete." | Design Time–current objects are "finished products" by the time they reach repository users. Many objects cannot be modified. | Design Time / Use Time–Systems are built to be extended. Components evolve as system evolves. |
| **Types of Artifacts** | Incorporates any relevant information–commercial tools, complete Java applets, reusable software components, tutorials, etc. | Most resources are complete educational applets. Reusable components exist but are less common. | Many levels of granularity, fined-grained pieces, component libraries, and complete examples that provide multiple levels for modification |
| **Mechanisms for Locating Information** | Browsing–all tools are placed somewhere in a large, strictly hierarchical space. Some items are cross referenced. Categories are maintained by system administrators. Searching–component descriptions (abstracts created by authors) may be searched by keyword. Search criteria may not be refined. | Browsing–one level of categories (sorted by educational discipline). Large number of resources returned at once. Searching–numerous search criteria based on "meta-data," author, subject area, description keywords, and submission date. Reviews of resources may not be searched. Meta-data are currently not extensible. | Browsing–multiple domain-specific hierarchies for different users (i.e., educators' subject view or developers' component view) Searching–combine formal (machine-derived) information such as usage information, code relationships, language keywords with informal information such as description and discussion. |
| **Mechanisms for Comprehending Information** | Gamelan staff may annotate resources with informative labels. Small amount of resource-related data stored in the system. Usage demonstrations must be provided by contributors. | Meta-data presented for all objects. Some meta-data may be used for further sort results, but only some filters are supported. Small amount of resource-related data stored in the system. Usage demonstrations must be provided by contributors. | Informal and formal data presented in a perspective appropriate for the user. Large amounts of design information available in the system. Interactive environments may provide demonstration of usage. Community may annotate artifacts with discussion. |
| **Mechanisms for Modifying Information** | None. Modification history of resources not captured by the system. | None. Modification history of objects not captured by the system. | Domain-specific environments may aid modification. Modification history tracked to aid developers and provide historical perspective of design decisions. |

Table 2: *Summary and Comparison of Systems from a Self-Directed Learning Context*

## 6. Conclusion

Industrial-age models of education and work are inadequate for preparing students to compete in a knowledge-based workplace and to be fully empowered citizens in an 'information society'. A major objective of our lifelong learning approach is to reduce the gap between school and workplace learning. When learning becomes a part of life, support for self-directed learning is a necessity.

New media and new technologies alone will not provide answers to the challenges that self-directed learning presents. When old processes are realized with new technologies, the possible benefits afforded by the new capabilities will be largely unrealized. Instead, we must rethink our basic assumptions and see how technology can be applied to best solve the fundamental problems that people encounter in actual learning situations.

Facilitating economies of educational knowledge is a promising direction that supports the needs of self-directed learners. Based on requirements for self-directed learning, we have argued that DODEs, support for the seeding, evolutionary growth and reseeding model, and support for the location, comprehension and modification cycle are necessary to create effective economies of educational knowledge. These economies must allow their designers to use them in self-directed ways, and their content must provide the components required for the creation of self-directed learning environments.

In our research we have explored fundamentally new possibilities and limitations of computational media as they complement existing media. The ongoing exploration of these issues will continue to raise important questions such as: How can sustainable environments be created for communities of practice? How can large complex information spaces be evolved over long periods of time? How can self-directed learning be facilitated and supported in its important role for making learning a part of life?

## Acknowledgments

The authors would like to thank the members of the Center for LifeLong Learning and Design (L3D) at the University of Colorado, who have made major contributions to the conceptual frameworks and systems described in this paper. The authors received important feedback from the JIME editors and reviewers to improve earlier versions of this paper. The research was supported by (1) the National Science Foundation, Grants REC-9631396 and IRI-9711951; (2) the McDonnell Foundation; (3) NYNEX Science and Technology Center, White Plains; (4) Software Research Associates, Tokyo, Japan; (5) PFU, Tokyo, Japan; and (6) Daimler-Benz Research, Ulm, Germany.

## References

Bruner, J. (1996). *The Culture of Education*. Cambridge, MA: Harvard University Press.

Csikszentmihalyi, M. (1990). *Flow: The Psychology of Optimal Experience*. New York: HarperCollins Publishers.

Dawkins, R. (1987). *The Blind Watchmaker.* New York - London: W.W. Norton and Company.

Drucker, P. F. (1994). *The Age of Social Transformation*. The Atlantic Monthly (November), 53-80.

Eisenberg, M. and Eisenberg, A.N. (1998) *Shop Class for the Next Millenium: Education through Computer-Enriched Handicrafts*. Journal of Interactive Media in Education, 98 (8). <http://www-jime.open.ac.uk/98/8>

Fischer, G. (1991). *Supporting Learning on Demand with Design Environments*. Paper presented at the International Conference on the Learning Sciences, Evanston, IL.

Fischer, G. (1994). *Domain-Oriented Design Environments*. Automated Software Engineering, 1(2), 177-203.

Fischer, G. (1998). *Beyond 'Couch Potatoes': From Consumers to Designers*. Paper presented at the 3rd Asia Pacific Computer Human Interaction Conference, Kanagawa, Japan. <http://www.cs.colorado.edu/~gerhard/papers/apchi98>

Fischer, G., Girgensohn, A., Lemke, A., McCall, R., and Morch, A. (1990a). *Conceptual Frameworks and Innovative System Designs for Participatory Design*. Conference on Participatory Design, PDC'90, (pp. 59-81). P.O. Box 717, Palo Alto, CA 94301.

Fischer, G., Grudin, J., Lemke, A. C., McCall, R., Ostwald, J., Reeves, B. N., and Shipman, F. (1992). *Supporting Indirect, Collaborative Design with Integrated Knowledge-Based Design Environments*. Human Computer Interaction, Special Issue on Computer Supported Cooperative Work, 7(3), 281-314.

Fischer, G., Henninger, S. R., and Redmiles, D. F. (1991). *Cognitive Tools for Locating and Comprehending Software Objects for Reuse*, Thirteenth International Conference on Software Engineering (Austin, TX), (pp. 318-328). Los Alamitos, CA: IEEE Computer Society Press.

<http://www.cs.colorado.edu/~gerhard/papers/se91>

Fischer, G., McCall, R., Ostwald, J., Reeves, B., and Shipman, F. (1994). *Seeding, Evolutionary Growth and Reseeding: Supporting Incremental Development of Design Environments*. Paper presented at the Human Factors in Computing Systems (CHI'94), Boston, MA. <http://www.cs.colorado.edu/~gerhard/papers/chi94>

Fischer, G., and Nakakoji, K. (1994). *Amplifying Designers' Creativity with Domain-Oriented Design Environments*. In T. Dartnall (Ed.), Artificial Intelligence and Creativity, (pp. 343-364). The Netherlands: Kluwer Academic Publishers.

Fischer, G., Nakakoji, K., Ostwald, J., Stahl, G., and Sumner, T. (1993). *Embedding Critics in Design Environments*. The Knowledge Engineering Review Journal, 8(4), 285-307.

Furnas, G. W., Landauer, T. K., Gomez, L. M., and Dumais, S. T. (1987). *The Vocabulary Problem in Human-System Communication*. Communications of the ACM, 30, pp. 964-971.

Girgensohn, A. (1992). *End-User Modifiability in Knowledge-Based Design Environments*. (Ph.D. Dissertation ). Dept. Computer Science, University of Colorado, Boulder, CO, U.S.A..

Harel, I., and Papert, S. (1991). *Constructionism*. Norwood, NJ: Ablex Publishing Corporation.

Henderson, A., and Kyng, M. (1991). *There's No Place Like Home: Continuing Design in Use*. In J. Greenbaum and M. Kyng (Eds.), Design at Work: Cooperative Design of Computer Systems, (pp. 219-240). Hillsdale, NJ: Lawrence Erlbaum Associates, Inc.

Hirsch, E. D. (1996). *The Schools We Need And Why We Don't Have Them*. New York: Doubleday.

Illich, I. (1971). *Deschooling Society*. New York: Harper and Row.

Kintsch, W. (1998). *Comprehension—A Paradigm for Cognition*. Cambridge, England: Cambridge University Press.

Landauer, T. K. (1995). *The Trouble with Computers*. Cambridge, MA: MIT Press.

Landauer, T. K., and Dumais, S. T. (1997). *A Solution to Plato's Problem: The Latent Semantic Analysis Theory of Acquisition, Induction, and Representation of Knowledge*.

*Psychological Review*, 104(2), 211-240.

Lave, J., and Wenger, E. (1991). *Situated Learning: Legitimate Peripheral Participation*. Cambridge, UK: Cambridge University Press.

Nardi, B. A. (1993). *A Small Matter of Programming*. Cambridge, MA: The MIT Press.

Norman, D. A. (1993). *Things That Make Us Smart*. Reading, MA: Addison-Wesley Publishing Company.

Papert, S. (1980). *Mindstorms: Children, Computers and Powerful Ideas*. New York: Basic Books.

Popper, K. R. (1965). *Conjectures and Refutations*. New York, Hagerstown, San Francisco, London: Harper and Row.

Raymond, E. S. (1998). *The Cathedral and the Bazaar*. <http://earthspace.net/~esr/writings/cathedral-bazaar/cathedral-bazaar.html>

Repenning, A., and Ambach, J. (1997). *The Agentsheets Behavior Exchange: Supporting Social Behavior Processing*. Proc. CHI 97, Conference on Human Factors in Computing Systems, Extended Abstracts, (Atlanta, Georgia), 26-27. ACM Press: NY.

Repenning, A., Ioannidou, A. and Ambach, J. (1998). *Learn to Communicate and Communicate to Learn*. Journal of Interactive Media in Education, 98 (7). <http://www-jime.open.ac.uk/98/7>

Resnick, L. B. (1989). (Ed.), *Knowing, Learning, and Instruction: Essays in Honor of Robert Glaser*. Hillsdale, NJ: Lawrence Erlbaum Associates, Inc.

Resnick, M. (1994). *Turtles, Termites, and Traffic Jams*. Cambridge, MA: The MIT Press.

Resnick, M. (1996). *Distributed Constructionism*. Proc. International Conference of the Learning Sciences, Chicago, IL.

Rittel, H. (1984). *Second-Generation Design Methods*. In N. Cross (Ed.), Developments in Design Methodology, (pp. 317-327). New York: John Wiley and Sons.

Ritter, S., Anderson, J., Medvedeva, O., and Cytrynowitz, M. (1998). *Authoring Content in the*

*PAT Algebra Tutor.* Journal of Interactive Media in Education, 98 (9). <http://wwwjime.open.ac.uk/98/9>

Roschelle, J., Kaput, J. Stroup, W. and Kahn, T. M. (1998). *Scaleable Integration of Educational Software: Exploring The Promise of Component Architectures*. Journal of Interactive Media in Education, 98 (6). <http://www-jime.open.ac.uk/98/6>

Schön, D. A. (1983). *The Reflective Practitioner: How Professionals Think in Action*. New York: Basic Books.

Simon, H. A. (1996). *The Sciences of the Artificial, 3rd edition*. Cambridge, MA: The MIT Press.

Spohrer, J., Sumner, T. and Buckingham Shum, S. (1998). *Educational Authoring Tools and the Educational Object Economy: Introduction to this Special Issue from the East/West Group*. Journal of Interactive Media in Education, 98 (10). <http://www-jime.open.ac.uk/98/10>

Stefik, M. (1995). *Introduction to Knowledge Systems*. San Francisco, Calif: Morgan Kaufmann Publishers.

Suchman, L. A. (1987). *Plans and Situated Actions*. Cambridge, UK: Cambridge University Press.

Sumner, T. (1995). *Designers and Their Tools: Computer Support for Domain Construction*. Unpublished Ph.D., Department of Computer Science, University of Colorado at Boulder, Boulder, CO.

Sumner, T., Bonnardel, N., and Kallak, B. H. (1997). *The Cognitive Ergonomics of Knowledge-Based Design Support Systems*. Proc. CHI 97, Conference on Human Factors in Computing Systems, (Atlanta, Georgia). ACM Press: NY.

Wenger, E. (1987). *Artificial Intelligence and Tutoring Systems: Computational and Cognitive Approaches to the Communication of Knowledge*. Los Altos, CA: Morgan Kaufmann Publishers, Inc.