Sharing engineering design knowledge in a distributed environment

Zdenek Zdrahal, Paul Mulholland, John Domingue and Marek Hatala

Knowledge Media Institute

The Open University

Milton Keynes, MK7 6AA, UK

http://kmi.open.ac.uk/

{ Z.Zdrahal, P.Mulholland, J.B.Domingue, M.Hatala}@open.ac.uk

+44 1908 65-3800 (Fax: 3169)

Engineering design is a complex activity, relying heavily on know-how gained from personal experience. Competitive pressures and new technology are making further demands on the skills and experience of designers, as effective knowledge reuse in design is seen as increasingly vital, and the work of design teams is often a collaborative and distributed activity. University students with a thorough knowledge of the engineering domain can be ill prepared for professional practice, with its increasing reliance on skills and know-how as well as knowledge of theory. Our approach aims to better prepare students for professional practice, through hands-on experience of design reuse, participation in distributed collaboration, and the development of presentation and documentation skills. Our case-study in the domain of modelling engineering systems, in which the course materials themselves are evolving and distributed, has ramifications for the publication model of educational materials, and the way students should be prepared for working life.

1.   Introduction

There has always been something of a gap between the knowledge gained at school and university and the way knowledge is accumulated and applied within professional practice. This is true across a range of professional disciplines including engineering, medicine, law and architecture (Schön, 1983). In these professions and others, the transition from school to work can be daunting, as a new set of skills and a new way of working and learning has to be rapidly adopted. It is widely accepted that learning must now be a lifelong experience. Universities cannot provide students with all the knowledge they will need throughout their working life, but universities can do more to arm students with the skills they will need in order to be a lifelong learner.

Recent economic and technological developments are if anything widening this gap between school and work. Across many industrial areas, there is increasing pressure for workers to reuse previous knowledge of products and processes when developing new solutions. Work is also increasingly recognized as a social and collaborative activity. Advances in internet technology is also leading to collaborative activities to be distributed over space and time. This tendency will no doubt continue, and educational institutions need to respond to the changing demands placed on their graduates.

Our approach is to develop new methods for teaching, currently within the field of modelling and simulation of dynamical systems in engineering, which respond to the above shortfalls. Motivated by the gap between university and professional status, we have developed a distributed environment for promoting the skills of professional practice. Our course, called RichODL (Rich Open and Distance Learning) (RichODL, 1999) gives students hands-on experience of combining the domain knowledge accumulated in

lectures and from textbooks with the skills of professional practice, including distributed collaboration, design reuse, and the construction of documentation and rationale. We believe this will better prepare students for the transition to work.

The rest of this paper is structured as follows. Section two describes the practices of professional designers, which we aimed to integrate into the new course. Section three outlines how our approach prepares students for the workplace. The rest of the paper describes our distributed design environment covering the presentation of course materials, the development of the knowledge model, ontology driven search of the educational resources and the underlying technical architecture. Section seven discusses implications for the traditional publishing model of educational materials.

2.   The nature of professional engineering practice

Engineering design work has a number of important characteristics that have to be considered when developing a course to better promote the skills of professional practice. In particular:

- Design work is often characterized as an optimized search of a number of design alternatives.

- Strategic design shortcuts are common among professional designers.

- Past cases are frequently used during the design process.

- Design is an increasingly distributed and collaborative activity.

Engineering design is a complex activity. The design process is often characterized as a top-down breadth-first search of the space of possible solutions (Ball, Evans, Dennis and Ormerod, 1997). This figures strongly in prescriptions of how the design process should proceed (Cross, 1989). This demanding process ensures an optimal solution is constructed to the initial problem specification. This involves maintaining and elaborating numerous candidate solutions in parallel. *Designers need to be adept at generating and evaluating a range of candidate solutions to a problem.*

Simon (1969) used the term 'satisficing' to describe how experts will sometimes limit their search of the solution space. This is done possibly in response to cognitive limitations, or to reduce the time taken to reach a solution. There are differing accounts as to the extent to which designers opportunistically deviate from an optimal strategy. Ball et al (1997) found satisficing to be a relatively rare occurrence, but Ullman (1988) in their empirically motivated model of design, argue that generally design involves early commitment to, and refining of, a suboptimal solution. It is clear that satisficing is often advantageous due to reduced cost, or where only a satisfactory rather than an optimal design is required. Satisficing, is but one example of an approach to solving the problem that relies heavily on strategic knowledge and can only be developed though trial and error. *Much of what the designer knows is gained through feedback on experience.*

It is also well established that designers make use of past cases when developing a solution. Schön (1983) describes how designers, like many other professional groups, use past cases to frame a problem, and view the problem in a new light, as a crucial part of the problem solving process. Gero (1990) in his schematic model of design, describes the role of past experience in both routine and non-routine design tasks. Reuse in design was traditionally limited to reuse of personal experience, with a reluctance to reuse solutions of other designers. There is however evidence that companies are encouraging more extensive design reuse (Ormerod, Mariani, Ball and Lambell, 1999). This is no doubt driven by market competitiveness and the rate of innovation within design companies. Reuse is seen as offering a method for ensuring that innovation is captured and disseminated quickly, therefore helping the company to remain at the leading edge. *Designers need to be skilled in reusing design knowledge and preparing their own design solutions to facilitate reuse.*

The affordances of new technology and demands of the workplace provide greater opportunity and motivation for design to be conducted, at least in part, as a collaborative, distributed activity. A large amount of current research is concerned with developing tools and methodologies to support design teams separated by space and time (Szykman, Bochenek, Racz, Senfaute and Sriram, to appear). *Designers need to possess the skills that allow them to work effectively in a distributed, collaborative environment.*

3. Preparing engineering design students for the workplace

The above discussion has illustrated some of the demands of professional engineering design practice. Preparing students for such a working environment is much more than providing them with the necessary domain knowledge. Students could have all the necessary theoretical knowledge but be ill prepared for the workplace. Our approach will promote the development of skills required in professional practice, in particular:

- Learning by gaining feedback on experience.

- Evaluating alternative approaches to the same problem.

- Reusing design knowledge contained in past cases, and facilitating the reuse of solutions.

- Working in a collaborative and distributed setting.

Learning by gaining feedback on experience. Our approach to on-line teaching is not a matter of taking a traditional course in engineering design, and putting it on the web. There is also an important shift in the kind of knowledge and skills we aim to instill through this course. Much of the knowledge used in professional engineering design is tacit, and what Schön (1983) refers to as knowledge-in-action. This is the "know how" of how to put knowledge into practice (Brown and Duguid, 1998). This "know how" cannot be taught in the abstract, it can only be developed through hands on experience. In order to accumulate knowledge from experience, there is a need to gain feedback on the design work undertaken to effectively reflect on activities. Interestingly, in the architectural profession, there is a less marked change in the nature of design judgements from graduate to professional. It is argued that this is due to the lack of feedback architects receive on their designs (Rambow and Bromme, 1995). One possible source of feedback as to the quality of an engineering design would be for each design to be immediately built and tested for real. This is obviously impractical within an on-line course, or in professional practice. An alternative is for the design to be expressed in a model that can be constructed, run and modified. In the dynamical systems area, the domain of our course, the ability to construct and run simulations is even more important due to the complexity of the resulting behaviour. All of the examples within the RichODL course contain executable and modifiable components.

In addition to giving feedback, promoting the accumulation of hands-on knowledge through experience, modelling adds further challenges to the design process. When building a model, the student needs to decide what are the important features that must be captured in the model, what can be assumed or left out, and how the results should be interpreted. For example, the student needs to decide whether friction can be ignored for sake of simplicity, or whether it is crucial to the model, fundamentally affecting the outcome. Constructing the model is therefore not only a tool for learning and exploration but also an intricate design activity in its own right. One source of support for constructing faithful models is the underlying ontology which represents concepts relating to how models can be constructed. An ontology is an explicit representation of important concepts and relations within the domain. This will be described later.

Evaluating alternative approaches to the same problem. Professional design involves constructing, elaborating and evaluating a range of candidate solutions. This is a strong feature of the RichODL course. The basic structure of the on-line component of the course is a large set of solutions to different dynamical system problems. The solution description includes an initial problem description, some guidelines on how the problem was approached, and a solution description containing an executable, modifiable model. Each example illustrates one or more concepts, techniques or skills that may be applicable to the students' current problem. Solving a problem may involve evaluating and exploring a number alternative approaches as contained in the set of cases. The nature of the problems solved by the students, and the range of ideas contained in the cases, promotes the view that there is no single right answer to a problem, and the design process involves interpreting claims made in solution cases, assessing different approaches to the same problem, and making trade-offs between them. These same issues recur when designing the executable models. The students need to asses whether the model is faithful to their interpretation of the design problem, and evaluate the explicit and implicit assumptions that have been made.

Reusing design knowledge contained in past cases, and facilitating the reuse of solutions. Students evaluate a range of available solution cases, in order that they may reuse knowledge and techniques captured within the examples. Being able to reuse design knowledge does not develop automatically from a good

understanding of the domain. Reuse in design requires explicit training and experience (Woodfield, Embley and Scott, 1987). Maiden and Sutcliffe (1990) found that when inexperienced designers are encouraged to reuse, their strategies for selecting cases for reuse is ineffective, as the selection process is based on surface similarity to the problem, rather than on deeper analogies. The difficulty students have in selecting examples for reuse can be thought of as part of the larger problem of how students often fail to make effective use of analogous examples when learning. Roberston (1994) in his studies of students learning from textbook examples, found students often have to be explicitly shown the mapping between their current problem and the example. Within the RichODL course, students are helped in developing effective reuse skills, through teacher guidance, and also through the search mechanisms derived from the underlying ontology. Searching for examples does not focus on finding examples with similar surface characteristics, but rather deeper connections, such as having the same modelling assumption.

The students' work within the course also gives them first hand experience of how examples have to be presented to facilitate effective reuse, both by observing the structure of the examples, and the documentation of their solution to the problem. Reuse with poorly documented designs can often be detrimental, leading to the perpetuation of suboptimal designs (Vidgen and Hepworth, 1990). Presentation and documentation skills are therefore an essential part of the reuse process.

Working in a collaborative and distributed setting. Communication and team working skills are seen as increasingly important in technical professions, and attempts are being made to encourage the development of these skills among students (Dawson, Newsham and Kerridge, 1992). Our environment is designed to support and encourage collaborative and distributed working. The course is web based, dynamic models can be run remotely, and discussion spaces promote coordination and the sharing of ideas within distributed groups. The learning environment is further supported by the nature of the surrounding curriculum, the credit students are given for participation, and the pervading culture that the kinds of skills the course aims to teach are not an optional extra, but indispensable for an engineering design professional. Brown and Duguid (1991) perceive working and learning, which is what we want to prepare the students for, as situated and improvisational, where models of the situation and possible solutions are collaboratively built on an *ad hoc* basis. The models constructed during collaboration are not only shared mental models of the problem, but also robust dynamic models that act as "holding environments" for a range of ideas that could otherwise be difficult to fully articulate and share (Schön, 1988).

4.   Scenario: An on-line course in dynamical systems

For explication purposes in this section we will assume that a student Simone is currently working on a collaborative project with three other students. For the project the students have to create a preliminary design of the body of a car. The students have constructed a number of prototypes and they would now like to investigate their aerodynamic qualities, specifically, how the shapes would affect the performance of the car.

Simone decides to see if she can find any relevant examples on the RichODL site. To do this she opens her web browser and goes to the RichODL query tool, called Lois (Domingue and Motta, in press), shown in figure 1.

The Lois interface is created automatically from the RichODL ontology. The search is focused around key pedagogic concepts in the ontology. For RichODL the key concepts are:

- Part,
- Model,
- Model mapping,
- Coding technique, and
- Example.

These key concepts become the starting points for the construction of queries, and form the five buttons in the `Concepts` column of figure 1. A detailed explanation of the composition of the RichODL ontology is contained in section 6.

Simone starts constructing her query using Lois. First she selects the `Example` button. This causes the attributes of the `Example` concept to be displayed in the window under the `Attributes` label. Simone would like to find an example in the mechanical domain (as opposed to the electrical domain, for example) as the car problem is also mechanical. She begins to specify this. She selects `has-application-domain` in the `Attributes` window. The types of value that can fill this attribute are displayed in the window under the `Types` label. She selects `mechanical` from this window. Simone has now specified the first row of her query and she adds it to the bottom area of Lois by selecting `Add Row`.

Simone elects to further constrain her search by specifying that the example should be linked to the `part air-resistance`. This becomes the second row of her query.

Simone clicks on the `All Solutions` button and her display changes to figure 2. A single solution, called `tra6`, has been found to the query. Within the ontology are generic rules that link formal knowledge structures to relevant resources. In figure 2 an associated simulation example "Parachutist's fall" has been linked to the knowledge model component `tra6`. Simone can now either examine the structure of `tra6` to gain insight on the rationale for why it was chosen or she can go straight to the associated simulation example.

Simone decides to go straight to the Parachutist's Fall example (Mann, 1999a) by selecting "Parachutist's fall". The web page, shown in figure 3, has three main areas. On the left is a window which allows users to navigate, search or change the layout of the examples on the RichODL site. The central part of the frame contains the original example as written by a simulation modelling expert. The right part contains a discussion space for the example. The discussion space allows students and tutors to engage in dialogue about the example.

Simone starts to read the example. She clicks on the "Hide" button in the navigation window to temporarily hide the discussion space and give more room to the description of the parachute example. After reading the informal problem description she scrolls down to look at the solution. As can be seen in figure 4 the solution is described using two formalisms. The diagram at the top of figure 4 shows a solution to the problem using a multipole diagram (Mann, 1999b). Multipole diagrams were developed to provide a high level easy to read description language for simulation models. The solution is also represented using Dynast code (Mann, 1991), an executable formalism similar to that used in standard simulation and modelling systems such as MatLab™. The Dynast code is editable allowing students to experiment with given solutions. Simone wonders what would happen if the mass of the parachutist was halved, so she changes the mass to 40 and clicks on the "Submit to Dynast" button. The code is run on the Dynast server, in this case at the Czech Technical University in Prague, and a graph of the results is returned. The graph is shown in figure 5. The x-axis of the graph shows time and the y-axis shows velocity (the downward velocity is represented as a negative number). Simone observes from the graph that the downward velocity increases until just after 6 seconds when the parachute opened.

After reading the problem description and solutions and experimenting with the Dynast code Simone feels that she fully understands the example. She, however, is not sure how generic the equation $R = ky_A^2$ is for the force developed by the air resistance (see the middle window figure 6). Can she use this equation as part of her model of the aerodynamics of car bodies? She decides to ask for help by posting a comment. She selects the "Show" link in the navigation window and after a moment's thought decides to add a comment under the "Modelling Technique" heading. The comment form is shown in figure 6. Simone adds her comment, indicating that it is a question by selecting the "? Question" radio button. A tutor later gives Simone an answer to her question, which immediately becomes available to the other students in Simone's project group.

The enhanced Parachutist's Fall example, including the navigation facilities and integrated discussion space, was created, from a plain HTML document, using an extended version of the Digital Document Discourse Environment (D3E) (Hatala, 1999). D3E (Sumner and Buckingham Shum 1998) supports the publication of web-based documents with integrated discourse facilities and interactive components.

The RichODL site also facilitates the creation of additional examples. A tutor might for instance create an example for a group of students who had the same misconception. To create a new example a tutor simply has to fill in a simple web based form indicating where the HTML for the example can be found. Submitted

examples become available to students visiting the RichODL site, once approved by an editor. The editorial process is described later.
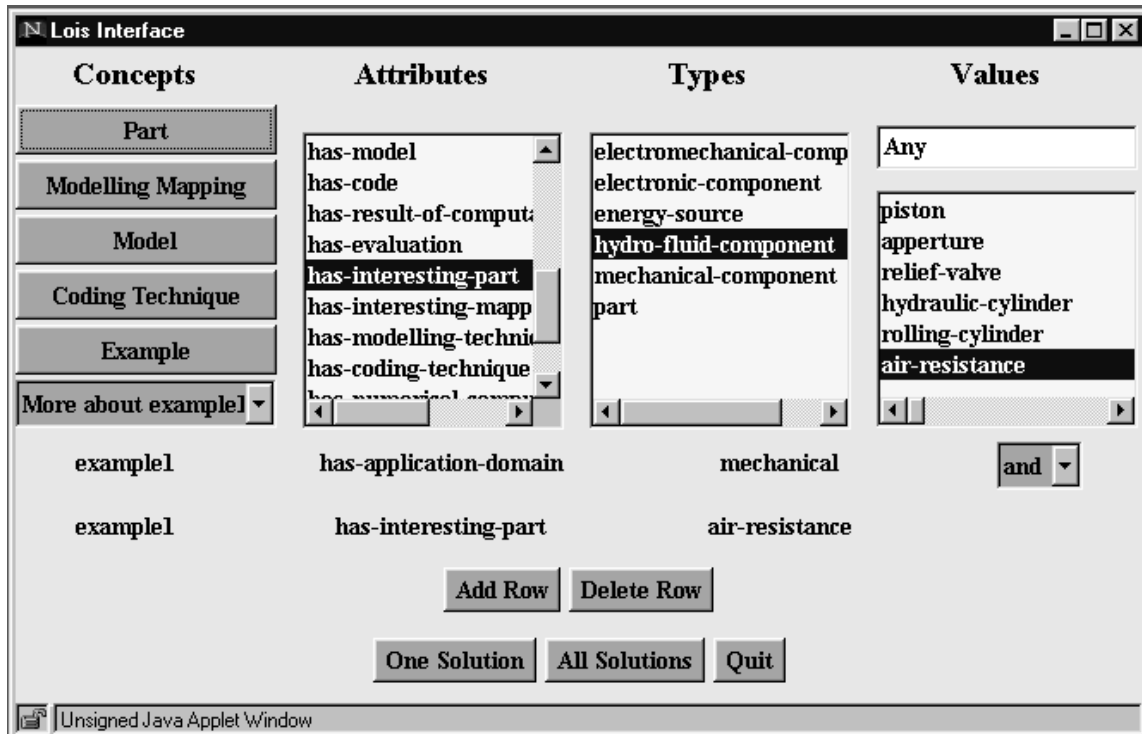

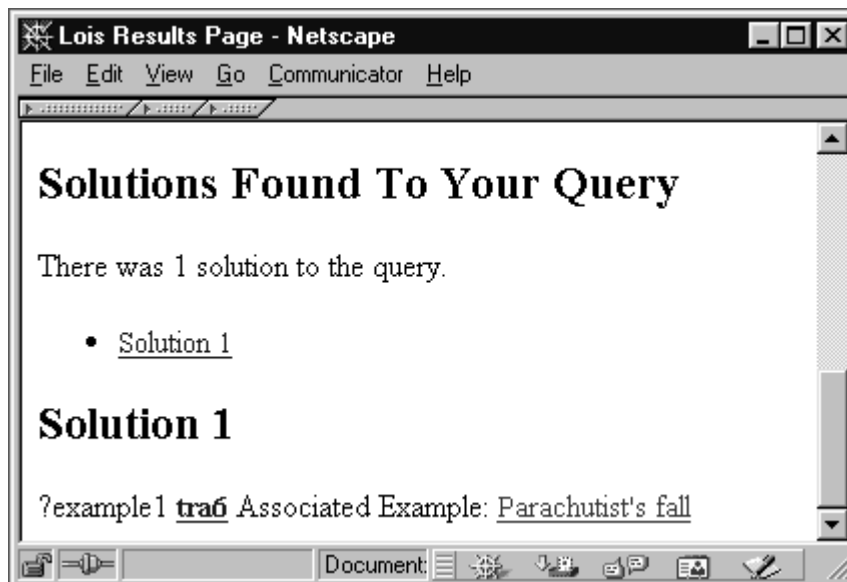**Figure 1. A screen snapshot of Simone's query in the Lois tool.**

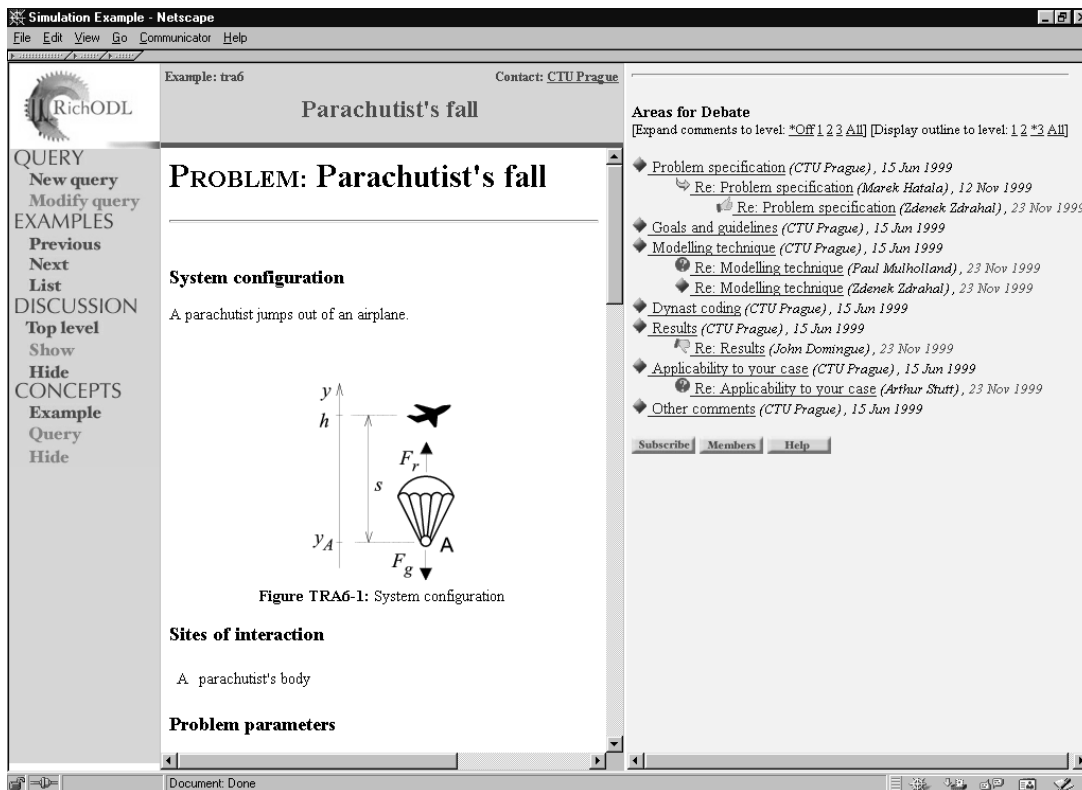**Figure 2. A screen snapshot showing the web page displayed after the query in figure 1 is posed.**

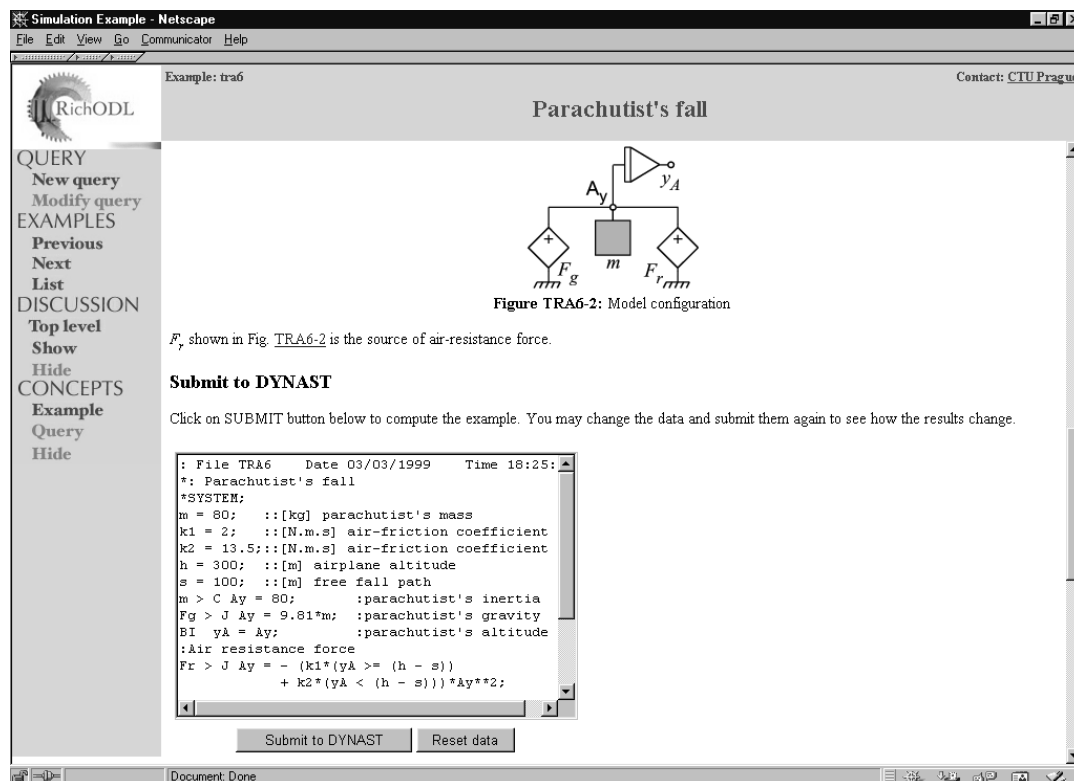**Figure 3. A screen snapshot showing the Parachutist's Fall example (Mann, 1999a).**

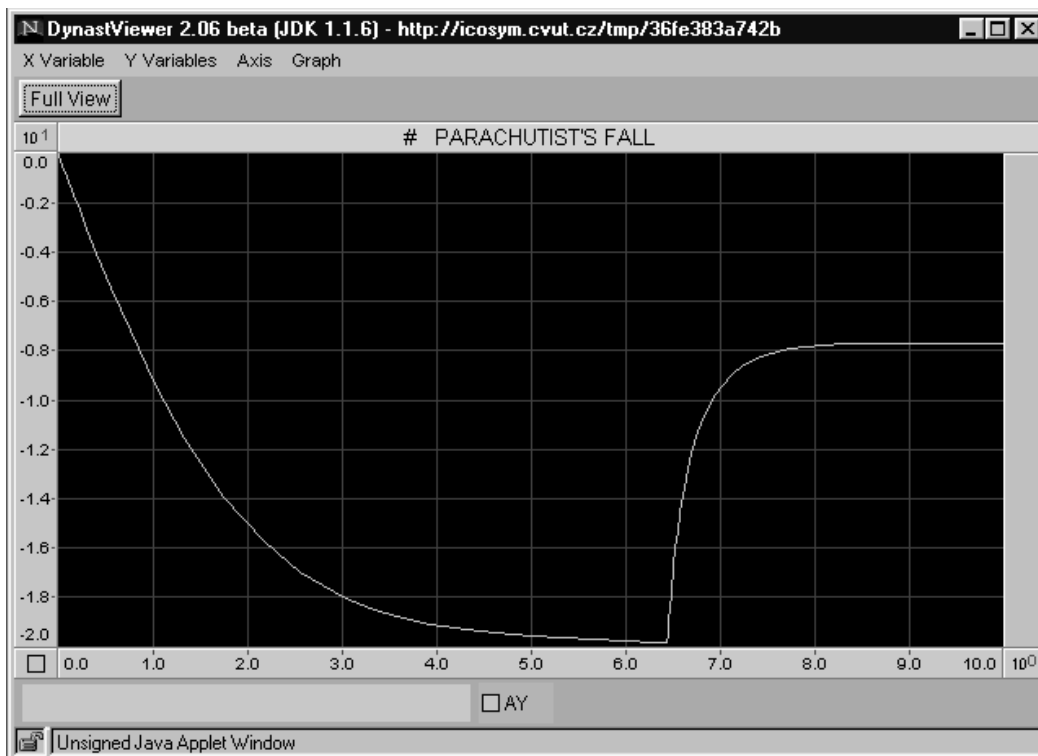**Figure 4. The solution to the Parachutist's Fall problem described as a multipole diagram and Dynast code.**

Figure 5. A screen snapshot showing the output of running the Dynast code representing the solution to the Parachutist's Fall problem. The x-axis represents time and the y-axis velocity.

Figure 6. A screen snapshot showing Simone adding a comment under the "Modelling Technique" heading.

5. Architecture

The RichODL architecture was designed to facilitate the sharing of engineering design knowledge in a distributed environment. From our own experiences at the Open University and from the literature we knew that students could learn from examples so long they were scaffolded with appropriate support material. To this end we decided to embed the examples in threaded discussion spaces. When students found a relevant simulation example they would also have access to the experiences of previous students attempting to learn from the same example and to the wisdom of the course tutors, past and present.

Enabling students to edit example solutions, to run them and get immediate feedback through high level visualizations has been proven to be useful here at the Open University in a number of technical domains (see, for example (Mulholland and Eisenstadt, 1998)). The examples in the RichODL site contain modifiable, executable solutions.

We decided to integrate formal knowledge representations into the architecture as these would:

- Enable students to search for examples using semantic based queries,

- Provide a high level course map which can be used to spot inconsistencies and missing components,

- Facilitate the construction of intelligent agents able to pro-actively notify students and tutors about interesting events, for example, that examples pertaining to certain course concepts were heavily used.
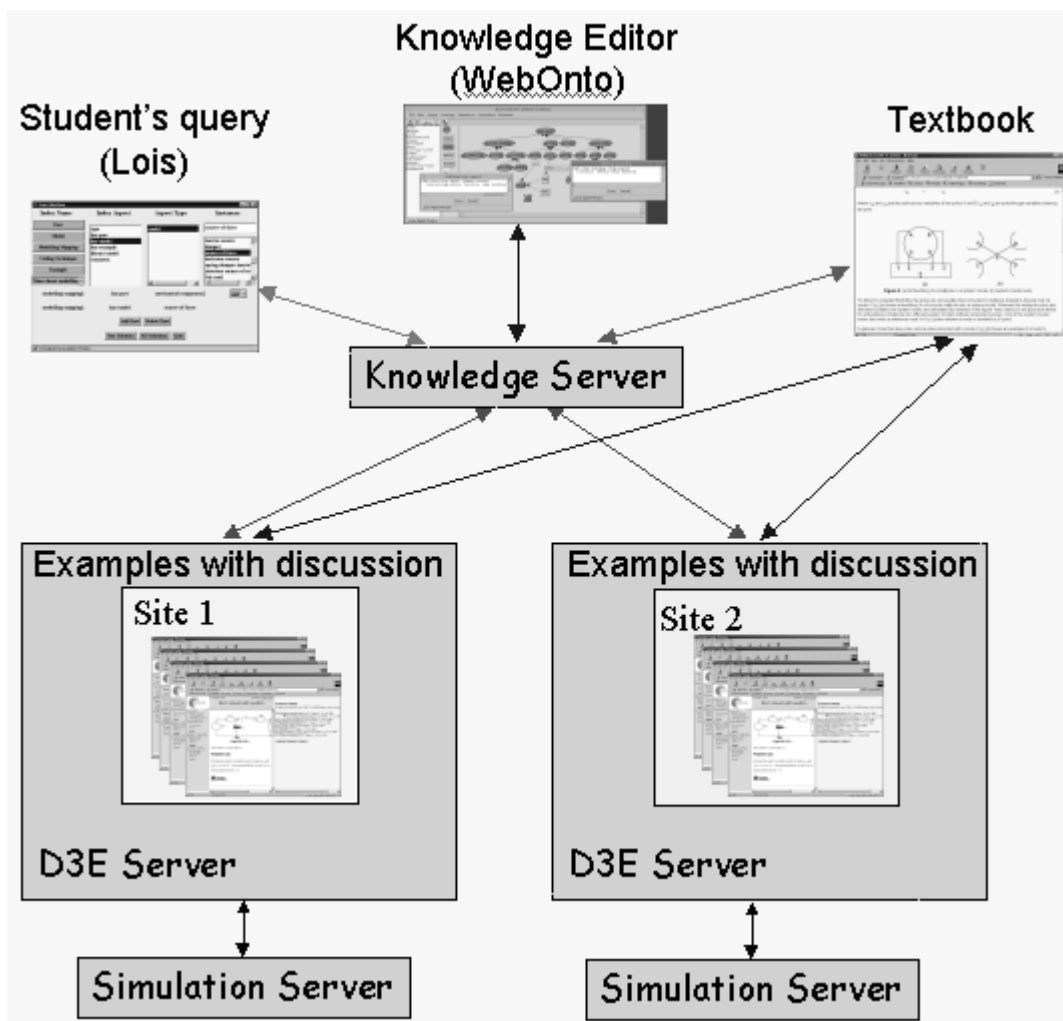

Figure 7. The RichODL architecture.

As we can see in figure 7 students use Lois to search for relevant resources on the RichODL site. The Lois queries are sent to the knowledge server where they are interpreted. As we described in the last section generic rules on the knowledge server link the query solutions to appropriate resources such as examples on one of the D3E sites or text book examples stored on a generic web server. It is worth emphasizing that this search is not string based. As all web surfers know using string based search techniques for finding documents often results in crucial documents (that do not contain the given string) being overlooked or irrelevant documents (that use the given string but with inappropriate semantics) being retrieved. Our approach is based on carrying out semantic based inference where some of the knowledge concepts used are linked to web based resources.

It is important the RichODL archive evolves with use, otherwise the site would quickly become out of date. Growth is supported in three ways:

- Students and tutors can add to the example discussion spaces (see figure 6),

- Tutors can submit new examples using a web based form, and

- Knowledge editors can edit the formal knowledge structures.

We shall now describe the architecture from a technical point of view. We can see from figure 7 that the RichODL architecture is based on three types of server:

- Knowledge Server – this contains a formal representation of the knowledge contained in the examples. This server contains specific knowledge about each example as well as generic knowledge about the domain.

- D3E Server – this contains the example descriptions and associated discussion spaces as shown in figures 3, 4 and 6.

- Simulation server – a simulation server can execute representations of example solutions and present the results back in a graphical form. Within RichODL we use a number of different types of simulation server including a Dynast server for Dynast code and a VRML server for interactive three dimensional displays of mechanical devices.

The knowledge server is composed of three parts:

- LispWeb (Riva and Ramoni, 1996) – a customised web server which allows web pages to be created on-the-fly from web browser requests. LispWeb provides the basic web server infrastructure for our knowledge manipulation tools.

- OCML (Motta, 1999) – an operational knowledge modelling language. All the formal knowledge concepts are represented in OCML.

- WebOnto server (Domingue, 1998) – the server side of our web based knowledge modelling tool. WebOnto also incorporates a Java applet which allows knowledge engineers to browse and edit OCML code using a direct manipulation interface. A screen snapshot of WebOnto displaying the `part` and `coding-technique` taxonomies is shown in figure 8.

The D3E server complements the D3E toolkit described in the previous section. The D3E server keeps track of additions to the archive such as comments posted in a discussion space or the submission of new examples.
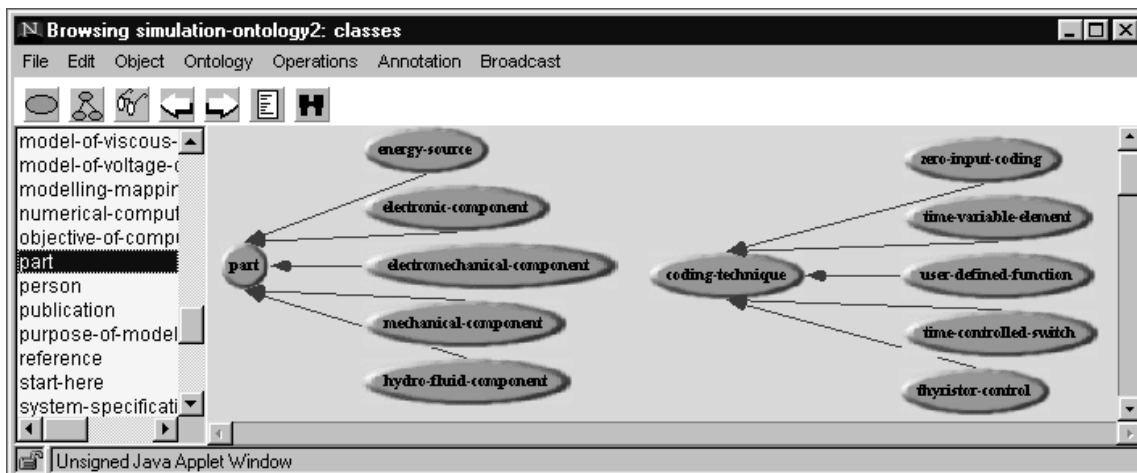
Figure 8. A screen snapshot showing the `part` and coding-technique class hierarchies of the RichODL formal knowledge model.

6.   A knowledge model of engineering design learning

Standard courses of engineering design teach students how to apply some design methodology to an appropriate engineering domain. Students design for example electrical circuits, mechanical machines, electromechanical, hydraulic, fluid or similar engineering devices. The result of the design process is a blueprint for making the artefact. The quality of the artefact is the only real measure of the design knowledge possessed by the student. Such a measure has great value as a feedback to the learning process. However, manufacturing and testing a real product is usually inconvenient, expensive, time consuming or even dangerous. The next best way of testing the design results is by modelling the artefact and evaluating its simulation model. Modelling can be viewed as a mapping from the engineering domain into the domain of dynamic systems, simulation is then building executable versions of models. In RichODL we assume that the models are dynamic systems. In engineering this assumption is almost always true.

In order to support the development and hands-on testing of simulation models the RichODL pedagogical framework integrates three types of formal knowledge:

- **Domain knowledge**
  The design and implementation of simulation models consists of two steps: First, the blueprint and its components are represented by an appropriate model, which is in then converted into the code of some simulation language. Three different domains take part in this process: the domain of the design results, denoted in this paper as the domain of `objects and parts`, the domain of `models` of objects and parts and finally, the domain of `simulation code`. Examples of object and parts are electronic circuits and their components, mechanical machines and machine parts, hydraulic and pneumatic devices, pipes, valves etc. Models of dynamical systems might be differential equations, transfer functions, multipoles, block and Bond diagrams etc. The simulation code is a program written in an appropriate simulation language such as Matlab, Simulink or Dynast. Domain knowledge defines existing domain concepts in terms of their properties and their relationship to other concepts and individual instances of concepts which may occur in the aforementioned domains. This knowledge is usually static and decontextualised and therefore easy to teach.

- **Process knowledge**
  Transitions between domains i.e. transition from parts to models and from models to simulation code, are described as process knowledge. In general, process knowledge characterises various design strategies. Some of these strategies are described as modelling (design) methodologies but experienced practitioners have, in addition, a large repertoire of modelling and programming tricks. For example, the same part can be modelled in a number of ways each of them having advantages and disadvantages depending on the context. Similarly, the simulation code for the same model may include various implementation tricks. In order to have broad applicability, methodologies are decontextualised and

therefore easier to teach but are correspondingly more difficult to use. On the other hand, experience-based process knowledge is more difficult to acquire because it is only applicable in certain contexts. Process knowledge is situated with the context defined by the application and has a dynamic character.

- **Media knowledge**
  The carriers of informal knowledge used by the RichODL framework are characterised by media knowledge. It includes URLs of relevant www pages, discussion spaces textbooks, lectures etc. Media knowledge allows us to associate formal domain and process knowledge with informal knowledge sources.

In RichODL, formal knowledge is expressed as knowledge models represented in OCML. Knowledge modelling is a standard technique for representing and studying properties of knowledge, similarly as modelling a pendulum serves to investigate its properties. Obviously, the representation languages and techniques are different. Knowledge models consist of ontologies and knowledge bases. Ontologies formally define the 'vocabulary' and the meaning of concepts used in the modelling process. Knowledge bases introduce individual instances of these concepts. Examples of the concept `part` from the `part` ontology and its instance `air-resistance` are shown in table 1. Since the concept `part` is intended to satisfy a wide range of different objects its 'definition' cannot be too restrictive. We characterise the part just by its name, possible application domain and controllability. For explanatory purposes, we include synonyms of terms within the engineering domain.

Class `part`

Slot Name	Documentation	Slot values
has-application-domain	The application domain where the part can be used. Application domains are e.g. mechanical, electronic, power-electronic, hydraulic, fluid, electro-mechanical etc.	The slot value must be of the type application-domain.
has synonyms	Synonyms and alternative spelling	The slot value must be of the type string.
is-controlled	'Yes' if some variables control the others. Otherwise 'no'.	The value of the slot be either 'yes' or 'no'. The default value is 'no'.

Instance `air-resistance` of the class `part`

Slot Name	Slot value	Comment
has-application-domain	Mechanical, fluid	
has synonyms	Wind resistance, air resistance	
is-controlled		Takes the default 'no' value from the class.

Table 1. The definition of class `part` and its instance `air-resistance`.

Examples are associated with the `example` ontology, which serves as a 'glue' connecting together part, model and code ontologies, textbook knowledge and informal discourse spaces. The simplified definition of the `example` concept from the example ontology is shown in Figure 9. In this figure the OCML code for the `example` class is shown instead of the tabular description shown above.

The class `example` is a subclass of the class `publication`, and therefore contains slots for author, title and the date of publication. Each instance of the example ontology includes an explicitly articulated

problem definition, theoretical solution, simulation code, experience-based knowledge (the lesson learned) and references to relevant textbooks. The context for the solution is defined by grounding the example in domain ontologies. Domain knowledge is described as follows: Interesting parts are described in the `Object and part` ontology, properties of the model used are defined in the `Model` ontology and the properties of simulation code are characterised in the `Simulation code` ontology. Not all parts, model components or code properties used in the example are formally represented in the ontology. Only those, which carry the 'message of the example' are referenced. This approach allows the student to focus on relevant issues and keeps the size of the RichODL ontologies and knowledge bases under control. After all, additional knowledge is captured in the example and could be formalised if needed.

Process knowledge includes the `Modelling mapping` ontology and the `Coding technique` ontology. These ontologies specify available techniques for defining models of parts and for coding selected models. For example, in the parachute problem the air resistance can be modelled as an external force acting on the parachutist's body proportional to the square of the velocity. This is a modelling trick, not a part of the general methodology. The corresponding knowledge is represented in table 2.
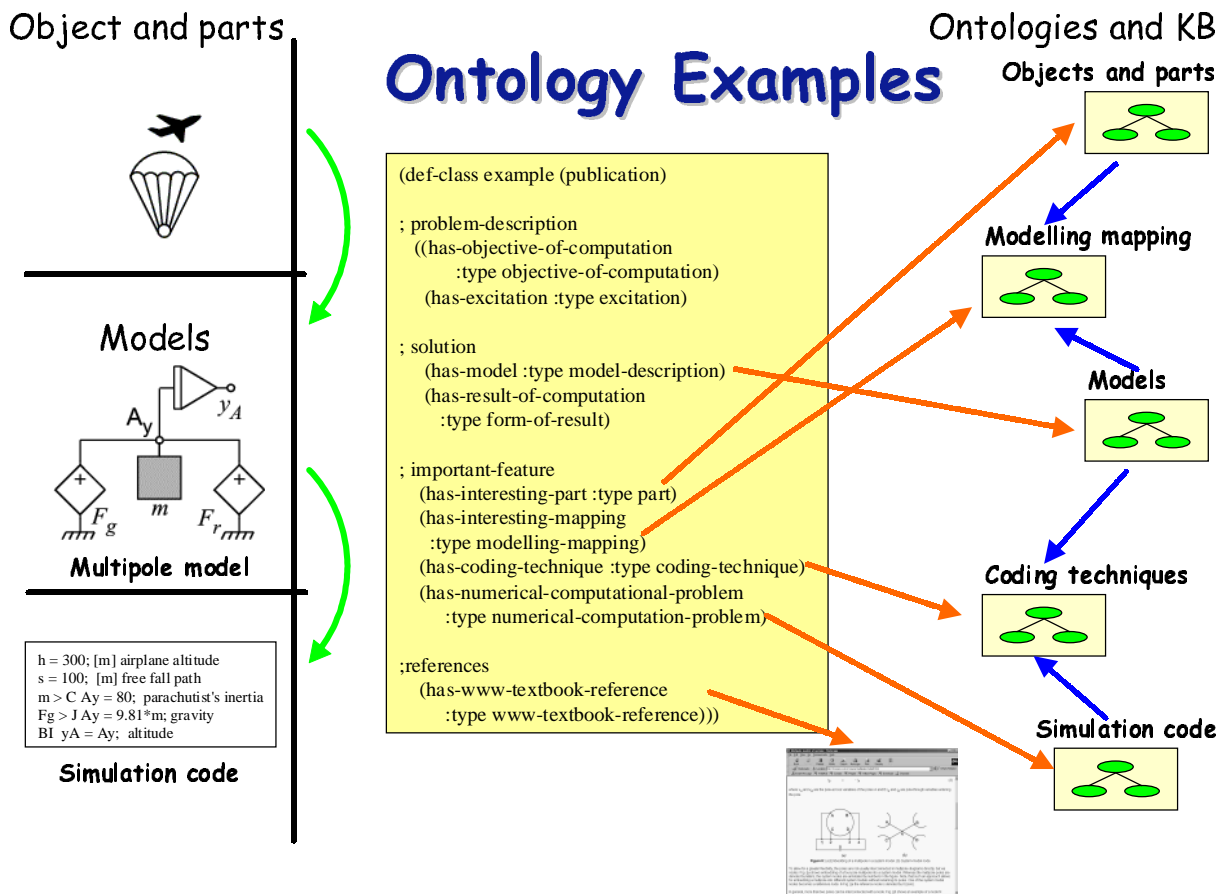

Figure 9. The `Example` ontology and its relation to process, domain and media knowledge.

The structure of the `Coding technique` ontology is shown in table 3. Both modelling mapping and coding techniques are a part of reusable, experience-based knowledge. They can be retrieved from examples where they are used by a Lois query as shown earlier. As shown in figure 9, the formal structure of the `Example` ontology also references media knowledge, which connects the ontology to web based resources.

Class `modelling-mapping`

Slot Name	Documentation	Slot Values
has-part	The part to be modelled.	The slot value must be an object from the Objects and part ontology
has-model	The model used for the part.	The slot value must be an object from the Model ontology.
has-example	Reference backwards to all examples in which this modelling mapping has been	The slot value must be an object from the Example ontology.

Instance `air-resistance-to-source-of-force` of the class `modelling-mapping`

Slot Name	Slot value	Comment
has-part	air-resistance	air-resistance is a part represented in the Objects and part ontology.
has-model	source-of-force	source-of-force is a model represented in the Model ontology.
has-example	tra6	Reference to the parachute example.

Table 2. The definition of `modelling-mapping` and `air-resistance-to-source-of-force`.

Class `coding-technique`

Slot Name	Documentation	Slot Values
has-effect	Effect demonstrated by the piece of code.	String
has-description	More detailed information	String
from-line	Beginning of the coding trick	Integer (line number)
to-line	End of the coding trick	Integer (line number)

Instance `controlled-source-of-force` of the class `coding-technique`

Slot Name	Slot value	Comment
has-effect	"non-linear force"	
has-description	"Source of force depends on the square of a control variable (velocity) and a multiplicative coefficient"	"Implementation comment: The variable in line 7 is raised to the power which is the value of the variable in line 8".
from-line	6	
to-line	32	

Table 3. The definition of `coding-technique` and `controlled-source-of-force`.

7. An evolutionary publication model for a distributed course

The RichODL approach has a number of ramifications for the nature of university learning and its relation to the skills and learning appropriate to the workplace. Further consequences ensue from the distributed evolving nature of the course, where numerous educators have authorship and ownership for various components. The RichODL course therefore requires a new model encompassing the nature, evolution and ownership of, and responsibility for, educational material, spanning more than one educational institution.

As described earlier, preparing and publishing and indexing new examples, is straightforward and supported by a suite of tools. There is though an editorial process ensuring the quality and relevance of submitted material. The role of the editor in this scenario is closer to the role of the editor of an academic journal than the editor of a book. The editorial team define the educational scope of RichODL, provide guidelines for submitting new examples, and coordinate the review of submissions. As in the journal, as opposed to the book editorial model, they define the academic field within which new examples can evolve, but they do not explicitly map out what examples are needed, and who should be their author. In the current model, submitted examples are prepared using the D3E publishing toolkit, but are stored in a separate area, not accessed by students, until cleared for publication. Great efforts have to be taken to ensure a swift turnaround of submitted examples so as not to create a bottleneck which would discourage educators from taking part. As the rate of submission of new examples is relatively low, this is achievable with a small editorial team.

Course maintenance is an equally important process within the RichODL model. The course is an evolving distributed resource, and requires regular maintenance. This process consists of far more than "checking links". As the examples are used, the discussion space will grow, as distributed students collaborate via on-line discussion, and teachers use it as a forum for providing advice and guidance. Periodically, this discussion space needs to be "weeded", to remove discussion threads judged of little benefit to future students. Some of the dialogue will be retained if it illustrates important characteristics of an example and how it can be used. Leaving some discussions in place blurs the distinction between the students' own work and the educational materials, in the same that a lecturer, after a discussion with a student, may find a new way to explain a concept that they will use in future years. Monitoring the discussion space can also lead to changes to the ontology, as pedagogically significant connections between examples are uncovered. The analysis of the ontology itself, can lead to the identification of areas within the academic domain that are under-represented in the current course. This information can be passed to the editorial board who can encourage the submission of new examples in this area, similar to how a journal editorial board may promote a special issue on a specific subject of interest.

8. Conclusions

The RichODL course and pedagogic model gives students a taste of working and learning practices appropriate to the workplace, in particular, knowledge reuse and collaboration. Developing the RichODL course was far more than putting existing educational resources on the web. Students accumulate knowledge collaboratively from feedback on experience in a distributed environment. This supplements rather than replaces conventional forms of learning from textbooks and lectures. The approach also required reassessment of the accepted model of educational materials, their ownership and evolution. The RichODL approach is currently being used and tested within a European wide initiative into teaching and learning dynamical systems engineering, and we eagerly await the outcome.

References

Ball, L. J., Evans, J. St. B. T., Dennis, I., Ormerod, T. C. (1997). Problem-solving strategies and expertise in engineering design. *Thinking and Reasoning*, 3 (4), 247-270.

Brown, J. S. and Duguid, P. (1991). Organizational Learning and Communities of Practice: Towards a Unified View of Working, Learning, and Innovation. *Organization Science* ,2 (1), 40-57.

Brown, J. S. and Duguid, P. (1998). Organizing knowledge. *California Management Review*, 40 (3), 90-111.

Cross, N. (1989). *Engineering design methods: Strategies for product design*. Chichester, Sussex, UK: John Wiley and Sons.

Dawson, R. J., Newsham, R. W. and Kerridge, R. S. (1992). Introducing new software engineering graduates to the 'real world' at the GPT company. *Software Engineering Journal*, 7, 171-176.

Domingue, J. (1998) *Tadzebao and WebOnto: Discussing, Browsing, and Editing Ontologies on the Web*. 11th Knowledge Acquisition for Knowledge-Based Systems Workshop, April 18th-23rd. Banff, Canada. WebOnto is now available as service at http://webonto.open.ac.uk/.

Domingue, J. and Motta, E. (in press) Planet-Onto: From News Publishing to Integrated Knowledge Management Support. *IEEE Intelligent Systems Special Issue on Knowledge Management and Knowledge Distribution over the Internet*.

Gero, J. S. (1990). Design prototypes: A knowledge representation schema for design. *AI Magazine*, 11, 26-36.

Hatala, M. (1999) *Simulation Examples: The User's Guide*. RichODL report, KMi, The Open University. June.

Maiden, N. and Sutcliffe, A. (1992). Analogously based reusability. *Behaviour and Information Technology*, 11 (2), 79-98.

Mann, H. (1991) Comparison 1 - DYNAST. EUROSIM Simulation News Europe, Nov. 1991, p.32.

Mann, H. (1999a) The Parachutist's Fall Example. RichODL Example TRA6, http://rich-odl.open.ac.uk/simulation-examples/example38/example-t.html.

Mann, H. (1999b) Physical modeling with multipoles. 1999 IEEE Symposium on Computer-Aided Control System Design, Hawaii Island 1999.

Motta, E. (1999) *Reusable Components for Knowledge Modelling,* Amsterdam: IOS Press.

Mulholland, P. and Eisenstadt, M. (1998) Using Software to Teach Computer Programming: Past, Present and Future. In J. Stasko, J. Domingue, M. Brown, and B. Price (Eds.) *Software Visualization: Programming as a Multi-Media Experience*. Cambridge, MA: MIT Press.

Ormerod, T. C., Mariani, J., Ball, L. J. B., Lambell, N. (1999). Desperado: Three-in-one indexing for innovative design. *Proceedings of INTERACT '99*, September, Edinburgh.

Rambow, R. and Bromme, R. (1995). Implicit psychological concepts in architects' knowledge - How large is a large room? *Learning and Instruction*, 5, 337-355.

RichODL (1999). http://rich-odl.open.ac.uk.

Riva, A. and Ramoni, M. (1996) LispWeb: a Specialized HTTP Server for Distributed AI Applications, Computer Networks and ISDN Systems, 28 (7-11), 953-961. (also available at http://kmi.open.ac.uk/~marco/papers/www96/www96.html).

Robertson, S. I. (1994). *Problem solving from textbook examples*. Unpublished Ph.D. Thesis, Human Cognition Research Laboratory, The Open University, UK.

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

Schön, D. A. (1988). Designing: rules, types and worlds. *Design Studies*, 9 (3), 181-190.

Simon, H. (1969). *Sciences of the Artificial*. Cambridge, MA: MIT Press.

Sumner, T. and Buckingham Shum, S. (1998). From Documents to Discourse: Shifting Conceptions of Scholarly Publishing. *Human Factors in Computing Systems (CHI '98)*, Los Angeles (April 18-23).

Szykman, S., Bochenek, C., Racz, J. W., Senfaute, J. and Sriram, R. D. (to appear), Design Repositories: Next-Generation Engineering Design Databases. *IEEE Intelligent Systems*.

Ullman, D. G., Dietterich, T. G., Stauffer, L. A. (1988). A model of the mechanical design process based on empirical data. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, 2 (1), 33-52.

Vidgen, G. and Hepworth, J. (1990). Yesterday's philosophy. *British Journal of Healthcare Computing*, 7, 23-34.

Woodfield, S. N., Embley, D. W. and Scott, D. T. (1987). Can programmers reuse software? *IEEE Software*, July, 52-60.