ASPIRE

USER MANUAL

Version 4.0

22.11.2007

Antonija Mitrovic

Brent Martin

Pramuditha Suraweera

Nancy Milik

Jay Holland

Konstantin Zakharov

Intelligent Computer Tutoring Group

University of Canterbury

Christchurch, New Zealand

Contents

List of Figures

1. Introduction

ASPIRE is an authoring system that supports teachers in developing Intelligent Tutoring Systems (ITSs) for their courses. This document is a user manual; it is aimed at authors, i.e. domain experts who want to develop ITSs, and do not necessarily have experience in computer programming and/or ITSs. The document explains how to use different components of ASPIRE, but it will not teach the basic concepts underlying ITSs. For more information about the necessary steps in building ITSs, please consult our training material (*The Authoring Primer*).

Intelligent Tutoring Systems are knowledge-based, adaptive systems the goal of which is to simulate the behaviour of a good human teacher. These systems typically support the student while learning problem-solving skills in a particular instructional domain. An ITS tracks the student's behaviour, analyses the behavioural data and produces/maintains a model of the student's knowledge. This model is later used to adapt instructional sessions towards the needs, learning abilities and preferences of the student. ITSs are knowledge-based because they contain explicitly represented domain knowledge, which can be used to analyse students' solutions against, and/or to solve problems given to students. The generated student model is used to tailor pedagogical decisions, such as selecting/generating problems and feedback.

Numerous ITSs have been developed, but only a very small number of them are used in real classrooms. This problem comes from the fact that ITSs are complex systems, which require a lot of time, resources and knowledge to be developed. Some researchers estimate the time needed to develop one hour of instruction within an ITS to be around 300 hours. It is therefore not surprising that authoring systems are sorely needed in the area of ITSs. Authoring systems support and/or automate the process of ITS development, by making it possible for a non-computer specialist to develop ITSs.

The ASPIRE project is funded by the e-Learning Collaborative Development Fund grants 502 and 592. ASPIRE supports the process of developing ITSs by automating some tasks, and supporting the remaining tasks, thus making it possible for tertiary teachers with little background in programming and Artificial Intelligence to develop systems for their courses. The resulting educational systems will overcome the deficiencies of existing distance learning courses and support deep learning.

ASPIRE consists of ASPIRE-Author, the authoring server, and ASPIRE-Tutor, the tutoring server, which delivers the resulting ITSs to students (see Figure 1). ASPIRE-Author makes it possible for the human expert (the author) to describe the instructional domain and the tasks the students will be performing, as well as to specify problems and their solutions. Once an ITS has been specified in ASPIRE-Author, the tutoring server delivers the developed system to the student.

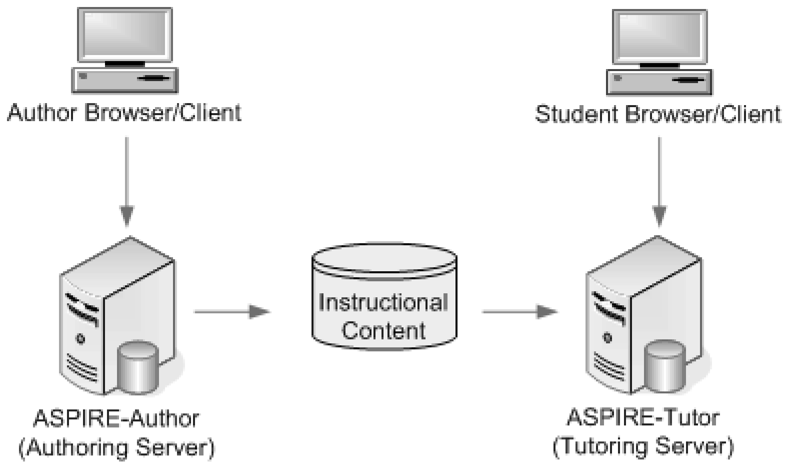


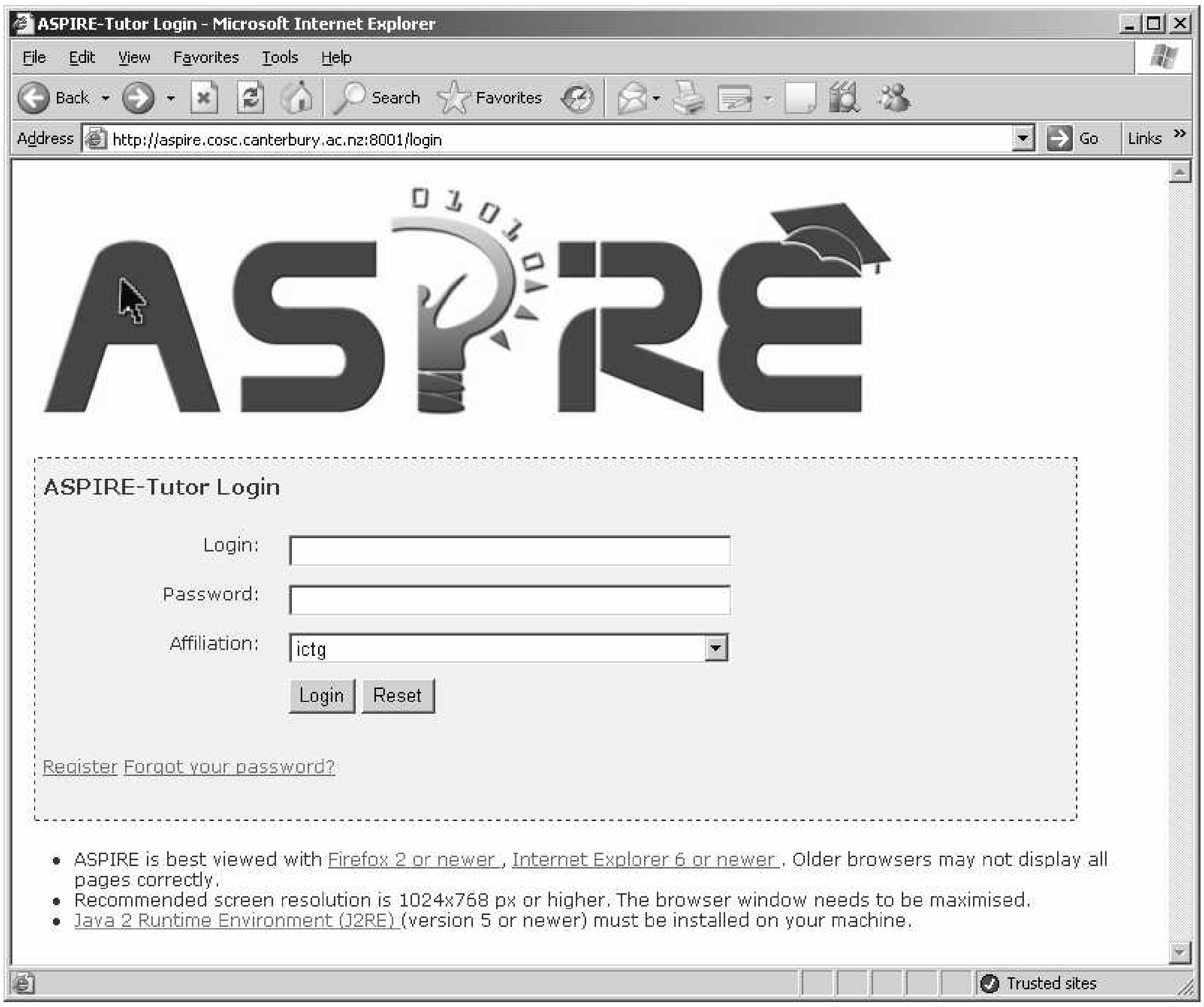
Figure 1. The Architecture of ASPIRE

This document describes the authoring side of ASPIRE first. We start by describing the login procedure in Section 2. Section 3 describes the architecture and functionality of ASPIRE-Author, followed by detailed instructions for authoring new intelligent tutoring systems. In Section 4, we briefly present the architecture of ASPIRE-Tutor, and then discuss the actions that can be performed by various types of users. Conclusions are given in Section 5, followed by a list of useful references.

2. Logging on to ASPIRE

ASPIRE is a Web-enabled system and therefore you will do all the work while developing an ITS within a web browser. We recommend Firefox (version 2 or above) or Internet Explorer (version 6 or above) as the web browser. Older versions of browsers may not display all pages correctly. The resolution of your display should be set at 1024x768 or higher. Lower resolutions are not supported. The browser window needs to maximized in order to enable the pages to be displayed properly. Java Runtime Environment (version 5 or above) should also be installed on the machine.

Figure 2 shows the Login page, which is available at http://aspire.cosc.canterbury.ac.nz:8001. If you do not have account for ASPIRE, click the *Register* link.


ASPIRE-Tutor Login - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address http://aspire.cosc.canterbury.ac.nz:8001/login
Go Links
Trusted sites

ASPIRE-Tutor Login

Login:

Password:

Affiliation: ictg

Login Reset

Register Forgot your password?

- ASPIRE is best viewed with Firefox 2 or newer , Internet Explorer 6 or newer . Older browsers may not display all pages correctly.
- Recommended screen resolution is 1024x768 px or higher. The browser window needs to be maximised.
- Java 2 Runtime Environment (J2RE) (version 5 or newer) must be installed on your machine.

Figure 2. The Login Page

To log in, you need to have a valid usercode, password and affiliation. If you cannot remember your password, the password will be emailed to you if you click the *Forgot your password* link and specify all the necessary information.

Once you have successfully logged on, you will see the *Home* page. If you have logged on with an author usercode, the home page will contain the *Jump to Authoring Tools* link, which will take you to ASPIRE-Author. The same can be achieved by switching to the *Authoring* tab. The other tabs are described in Section 4 of this document.

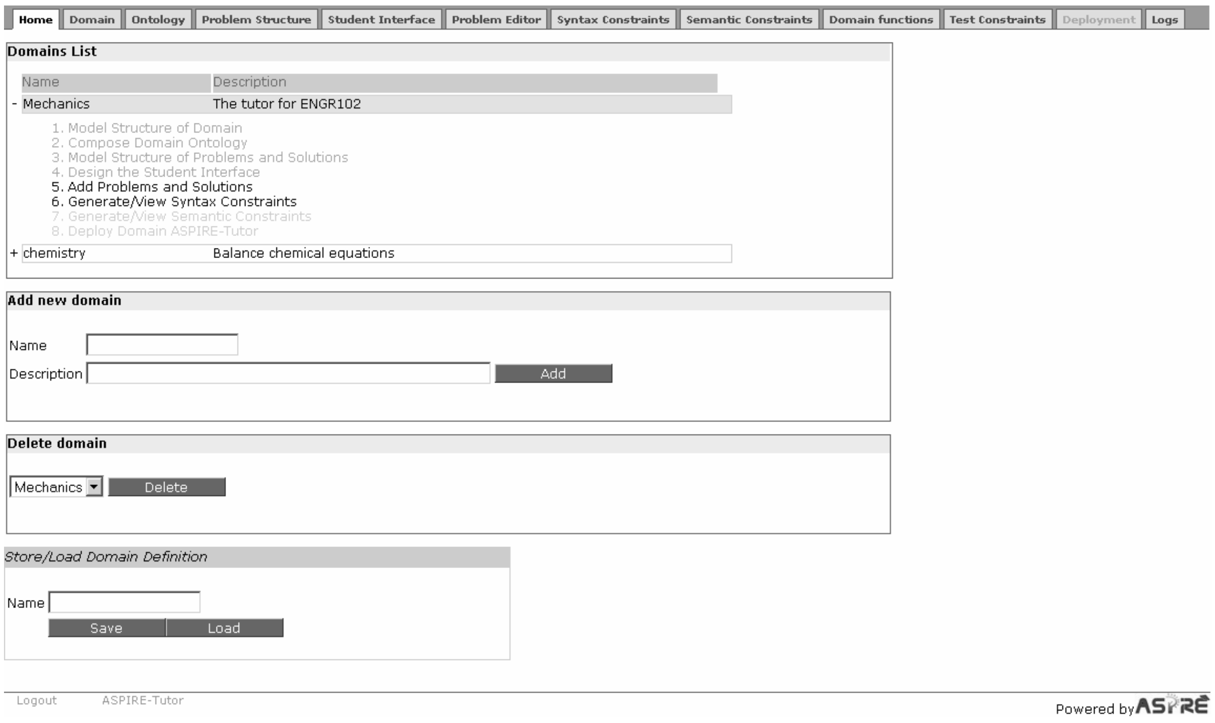
The *Home* page on ASPIRE-Author, shown in Figure 3, allows you to specify instructional domain (i.e. ITS) you want to work on. At the top of the page, there is a table showing names and descriptions of all domains you have previously worked on. If you are using ASPIRE for the first time, this table will be empty. In that case, you need to add a new domain. To add a new domain, you need to specify a unique name, a description, and then click the *Add* button. The newly created domain will be shown in the table at the top of the page. You can then open the domain.

To select a domain from the domain table, click on the wanted domain. In the screenshot[1] in Figure 3, the author has selected the *Fraction Addition* domain, and the page shows all steps of the authoring process. Clicking the step line takes you to the appropriate page to perform the task; note that this can also be achieved by clicking the appropriate tab. The steps that have been completed are shown in blue, while the steps that have not been attempted are shown in black. Some steps may be disabled (coloured grey) as they cannot be attempted until some other step is complete.

This page also allows you to delete an existing domain, by selecting it from the list of domains you have created. It is also possible to Save/Load a domain definition in XML format. Saving the domain has the result of storing the domain (in the XML representation) on the tutoring server. The *Load* function is available for loading pre-existing domains in to ASPIRE-Author. In other words, if the XML representation of a domain exists on the tutoring server, it can be loaded into ASPIRE-Tutor. Once a domain is loaded, it appears under the domains list of the Home page.

At the bottom of the page, there is the *Logout* link to use when you want to logout from ASPIRE-Author, which appears at the end of each page in the system.

[1] Please note that the type of account determines which tabs would be available (i.e. visible). The screenshots in this manual were generated using a developer account. For example, the *Test Constraint* and *Domain Functions* tabs are only available to developers.



Domains List

Name	Description
- Mechanics	The tutor for ENGR102
+ chemistry	Balance chemical equations

1. Model Structure of Domain
2. Compose Domain Ontology
3. Model Structure of Problems and Solutions
4. Design the Student Interface
5. Add Problems and Solutions
6. Generate/View Syntax Constraints
7. Generate/View Semantic Constraints
8. Deploy Domain ASPIRE-Tutor

Add new domain

Name

Description

Delete domain

Mechanics

Store/Load Domain Definition

Name

Figure 3. The Home Page on ASPIRE-Author

3. The Authoring Process

The development of the domain model is the most complex and time-consuming task in the development of an ITS. ASPIRE supports this process by automating some of the tasks required, and providing support for authors. The authoring process in ASPIRE consists of the following eight steps:

1. Modelling the domain structure;
2. Composing the domain ontology;
3. Modelling the problem and solution structures;
4. Designing the student interface;
5. Adding problems and solutions;
6. Generating syntax constraints;
7. Generating semantic constraints;
8. Deploying the domain.

The architecture of ASPIRE-Author is illustrated in Figure 4. This manual describes some of the functions of various components, but does not provide the details of implementation. The *Authoring Controller* is the central component which manages the authoring process and communication between the various components of ASPIRE-Author. The *Domain Structure Modeller* supports step 1 of the authoring process, by allowing the author to specify the general characteristics of the chosen instructional domain. This information is stored as the initial part of the domain model. The author then specifies the domain ontology using the *Ontology Workspace* (step 2). The *Problem/Solution Structure Modeller* allows the author to specify the structure of problems and solutions in the domain (step 3). The *Student Interface Builder* supports the author in specifying the initial version of the student interface (step 4), which will be used to communicate with students. The author uses the *Problem/Solution Editor* to provide examples of problems and their solutions (step 5).

On the basis of all specified information, the *Constraint Generator* develops the domain knowledge necessary for the ITS to be able to analyse students' solutions (step 6). This knowledge is represented in terms of constraints, which describe the syntax and the semantics of the instructional domain. The generated constraints are validated in step 7. The developed domain models are maintained by the *Domain Model Manager*.

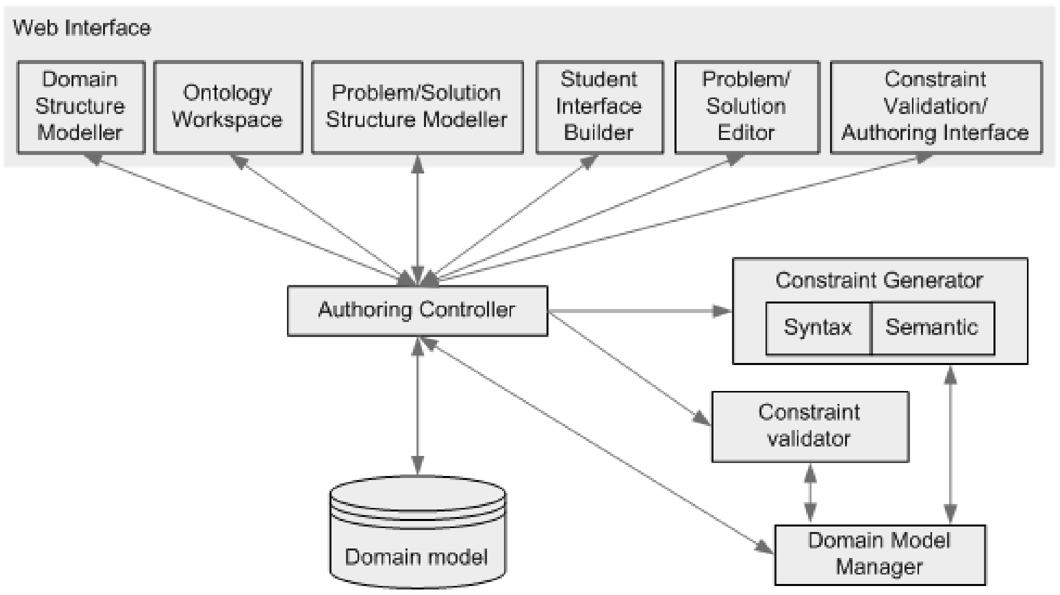


Figure 4. The Architecture of ASPIRE-Author

You will now learn how to perform each of these steps within ASPIRE-Author.

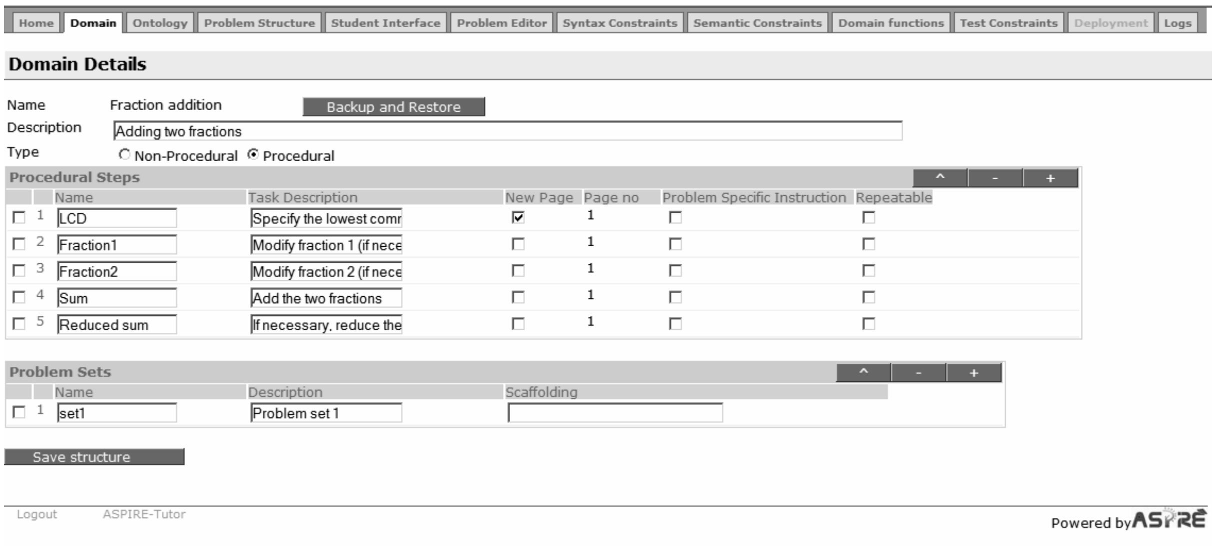
3.1. Modelling domain structure

The domain structure needs to be defined for each instructional domain. You can view/specify details of domain structure for the currently selected domain by clicking on the `Domain` tab of the authoring interface.

The *Domain* page illustrated in Figure 5 shows the details of the *Fractions* domain. The name and description of the selected domain are shown at the top of the page. This page also allows you to modify the description of the domain, without having to go back to the previous page. Note, however, that the domain name cannot be changed.

The *Type* of the domain refers to the nature of the task students will be performing in the ITS. ASPIRE offers two possibilities: procedural or non-procedural tasks. A procedural task contains a number of steps that need to be performed in a specified order. The *Fractions* domain contains procedural tasks: the student will need to specify the least common denominator for the two fractions to be added initially, and then (if necessary) modify the fractions. Then in step 4, the student needs to add the two fractions, and finally, in step 5, the resulting fraction might need to be simplified. Therefore, the task of adding fractions is a procedural one (notice that the *Procedural* option is selected in Figure 5). If tasks for the chosen domain do not require such a set of steps, but rather students can perform actions in any order, select a non-procedural task.

For a procedural task, it is necessary to add a set of steps. Once you specify that the task is procedural, ASPIRE will add a table showing steps, which is initially empty. To add a step, click on the + button, which will add a new row to the table. Each step has a unique number, generated automatically. The step name is what the students will see when they solve problems, as the label assigned to one of the interface components. Therefore it is important to give meaningful names to steps. The description of the step provides additional information to the student.



Domain Details

Name	Fraction addition
Description	Adding two fractions
Type	○ Non-Procedural [✓ Procedural]

Procedural Steps

		Name	Task Description	New Page	Page no	Problem Specific Instruction	Repeatable
☐	1	LCD	Specify the lowest comr	☑	1	☐	☐
☐	2	Fraction1	Modify fraction 1 (if nece	☐	1	☐	☐
☐	3	Fraction2	Modify fraction 2 (if nece	☐	1	☐	☐
☐	4	Sum	Add the two fractions	☐	1	☐	☐
☐	5	Reduced sum	If necessary, reduce the	☐	1	☐	☐

Problem Sets

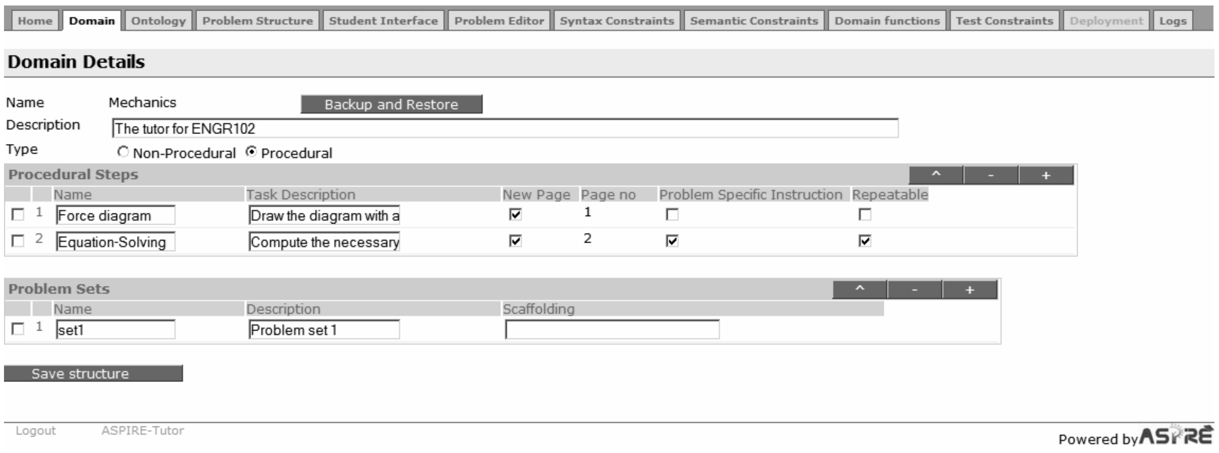
		Name	Description	Scaffolding
☐	1	set1	Problem set 1	

Figure 5. The Domain Details page

If the step is to be shown to the student on a new Web page, tick the *New Page* box. If this option is ticked, there will be a separate Web page for this step. Alternatively, you may specify several steps to be shown to the student on the same Web page. All steps defined for the Fractions domain in Figure 5 will be shown to the student on the same page.

To delete a step, select it first by ticking the box at the beginning of the corresponding row, and then click the – button. To change the order of the steps, tick the box at the beginning of a row that you want to bring upward, and then click the ^ button, as many times as necessary to bring the step at the desired position.

As specified previously, the task description will be shown to the student for each problem. If you want to have additional problem-specific instructions for one or more steps, tick the *Problem Specific Instruction* box for the appropriate steps.



Domain Details

Name	Mechanics
Description	The tutor for ENGR102
Type	○ Non-Procedural [✓ Procedural]

Procedural Steps

		Name	Task Description	New Page	Page no	Problem Specific Instruction	Repeatable
☐	1	Force diagram	Draw the diagram with a	☑	1	☐	☐
☐	2	Equation-Solving	Compute the necessary	☑	2	☑	☑

Problem Sets

		Name	Description	Scaffolding
☐	1	set1	Problem set 1	

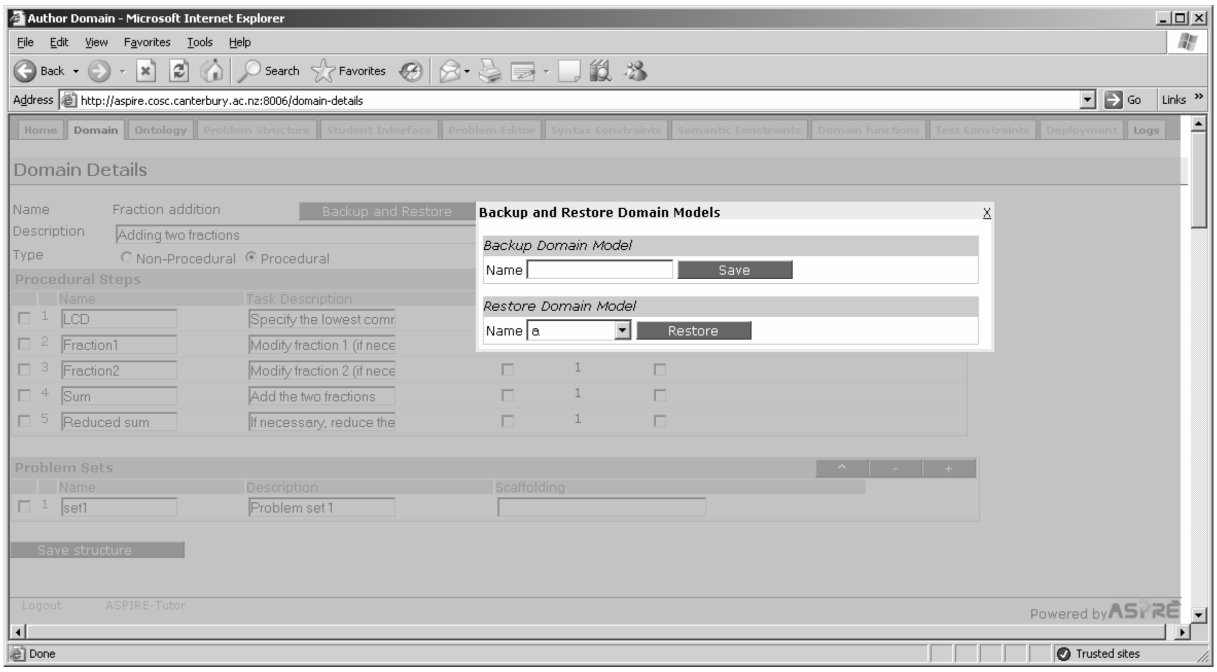
Figure 6. A domain with a repeatable step (Equation-Solving)

The *Repeatable* box allows the author to specify that the same step might be repeated several times. If this box is not selected, ASPIRE will assume that the step is to be done only once by a student. When the box is ticked, it will be possible to have several iterations of the same step. Figure 6. **A domain with a repeatable step (Equation-Solving)** illustrates a procedural domain with two steps, the second one being repeatable. Please note that a repeatable step is always the only step on a page.

The bottom part of the page is used to specify sets of problems for students to solve. In certain domains, problems can be classified into groups, containing problems that require the same skill. For example, in the domain of fractions, we might identify different sets of problems, dealing with fraction addition, fraction multiplication etc. In the domain of English, different problem sets might include turning verbs into nouns, adjective comparisons, past tense of verbs, work endings etc. Initially, ASPIRE-Author shows only one problem set. In Figure 5, there is only one set of problems. You can modify the name and description of the problem set. Please note that the scaffolding has not been implemented yet. To add/delete problem sets, use the +/- buttons.

Once you have finished describing the domain, steps and problem sets, click the *Save structure* button. This will lead you to the *Domain Ontology* page, which is described in the next section.

Finally, the *Backup and Restore* button facilitates experimenting with the domain. It allows a version of the domain to be saved under a name chosen by the author. When the button is clicked, a pop up window appears with two sections: back up and restore (see Figure 7). The back up section allows the current state of the domain to be saved. The author can always save the current state of the domain, and make modification to the domain. If the author wants to go back to the saved state of the domain, loosing all newer changes, the appropriate label from the restore drop down menu has to be selected and 'restore' should be clicked.


Author Domain - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address http://aspire.cosc.canterbury.ac.nz:8006/domain-details
Go Links
Home Domain Ontology Problem Structure Student Interface Problem Editor Syntax Constraints Semantic Constraints Domain functions Test Constraints Deployment Logs
Backup and Restore
Save structure
Logout ASPIRE-Tutor
Powered by ASPIRE
Done
Trusted sites

Domain Details

Name: Fraction addition
Description: Adding two fractions
Type: ( ) Non-Procedural (•) Procedural

Procedural Steps

		Name	Task Description			
☐	1	LCD	Specify the lowest comr			
☐	2	Fraction1	Modify fraction 1 (if nece			
☐	3	Fraction2	Modify fraction 2 (if nece	☐	1	☐
☐	4	Sum	Add the two fractions	☐	1	☐
☐	5	Reduced sum	If necessary, reduce the	☐	1	☐

Problem Sets

		Name	Description	Scaffolding
☐	1	set1	Problem set 1	

Backup and Restore Domain Models

Backup Domain Model
Name: [ ] Save

Restore Domain Model
Name: a Restore

Figure 7. Backup and Restore facility

3.2. Composing the domain ontology

An ontology describes the structure of the domain by showing the basic domain concepts, their properties and the relationships between concepts. A widely accepted definition is that an ontology is a specification of a formalisation (Gruber, 1993); in other words, it is an explicit, formal specification of the domain vocabulary which presents a common understanding of topics that can be communicated between users and applications. An ontology thus enables all the people involved to speak the same language, supporting knowledge sharing by applications and reuse. An ontology makes domain assumptions explicit, so that it is easier to change the domain description, as well as to understand and update existing data. An important feature of ontologies is that they separate domain knowledge from operational knowledge, in the same way in which a database schema is separated from the actual data stored in a database, thus introducing a high level of flexibility.

Initially, ontologies have been introduced within the field of Artificial Intelligence, but are now becoming widely used on the World Wide Web, as the foundation for the Semantic Web. In contrast to a large number of documents linked together, a Semantic Web is a huge network of machine-understandable and machine-processable human knowledge (Decker et al., 2000; Hendler, 2001, 2005). The WWW Consortium (W3C) has proposed several languages for encoding knowledge on the Web so that it can be used by intelligent agents to improve their performance. Key application areas of ontologies include e-commerce and search engines, among others.

In simple terms, an ontology represents a hierarchical organization of all important concepts in a given domain. Ontologies play a crucial role in ASPIRE. As the goal of ASPIRE is to make it possible for teachers to develop ITSs for their courses, the authoring process supported by ASPIRE relies on the domain ontology. Instead of asking the domain author (i.e. the teacher) to manually encode domain knowledge using a specific knowledge representation language, ASPIRE requires the author to describe the instructional domain by specifying the domain ontology. This is a much simpler requirement, as the author does not have to learn the knowledge representation language and the specifics of a particular approach to using domain models. Teachers are already aware of domain ontologies, even though they might have a simplified (i.e. informal) representation. In addition to specifying the domain ontology, the author is required to provide examples of problems and their solutions. ASPIRE-Author then analyses all three sources of knowledge (ontology, problems and solutions), and induces the domain model (represented in terms of a set of constraints, as discussed in Sections 3.6 and 3.7).

The ontology composition stage is the second phase of the authoring process. During this stage, the author develops the domain ontology using the Ontology Workspace, which is one of the components of ASPIRE-Author (as illustrated in Figure 4). If you are creating a new domain model (i.e. a new ITS), you would be taken to the Ontology Workspace after you specify the domain structure. Alternatively, you can get the Ontology Workspace by clicking the Ontology tab of the main interface.

In Section 3.2.1, we discuss the process of ontology development in general. Section 3.2.2 then discusses the Ontology Workspace, the component which supports the author while specifying the ontology. The following subsections describe the steps in developing an ontology, and illustrate how they are performed in the Ontology Workspace.

3.2.1. Ontology Development

There is no silver bullet when it comes to ontology development; similar to other design tasks, ontology development is under-specified and ambiguous. Therefore, there is neither one correct approach to ontology development, nor a single best ontology for a particular domain (Noy and McGuinness, 2001). In order to specify a domain ontology, the author needs to specify domain concepts, their properties and relationships between concepts. Each ontology will reflect the author's subjective view of the domain, and of the importance of domain concepts. The process is always iterative. Initially, it is necessary to decide on the scope of the ontology - how much of the domain will it cover? Generally, it is possible to reuse existing ontologies available on the Web, although there are a few ontologies available for educational domains that are directly applicable. In ASPIRE, we assume that authors will develop their own ontologies from scratch.

When developing an ontology, it is necessary to identify the important domain concepts. Roughly speaking, these concepts will include all types of entities appearing in the domain that students need to know about. When developing the ontology, it is useful to think about the interface the students will use to solve the problem; all the components appearing in the interface need to be described in the ontology as well.

Some domain concepts will be arranged into a taxonomy (i.e. a hierarchy, a tree), using the specialisation/generalisation relationship. This relationship is also commonly referred to as the *is-a* relationship, or the *a-kind-of* relationship. Taxonomy can be specified using a top-down or a bottom-up approach, or a combination of the two, which is probably most common. When using the top-down approach, the ontology is developed starting from the most general concepts, which are then refined into subclasses. The bottom-up approach, on the other hand, starts from specific concepts which are generalised into superclasses. Every concept in the ontology is important because of its properties and/or relationships to other concepts; therefore, properties and relationships need to be defined. The properties of a concept will be inherited by all of its subconcepts.

In this manual, we use fraction addition as an example instructional domain, to illustrate the various functions supported by ASPIRE. When learning about fraction addition, students should know about different kinds of numbers (whole numbers and fractions). Therefore, this suggests that the Number concept should be the root of the hierarchy, with whole numbers and fractions as specific subtypes. Furthermore, there are different kinds of fractions: proper fractions have numerators that are smaller than their denominators, while the opposite is true for improper fractions. Improper fractions can be further simplified. Based on this analysis, the ontology for this domain can be the one shown in Figure 8. Note that other possibilities exist for representing the same domain concepts. Also, we have not discussed properties and relationships between these concepts; they will be introduced in Section 3.2.2.

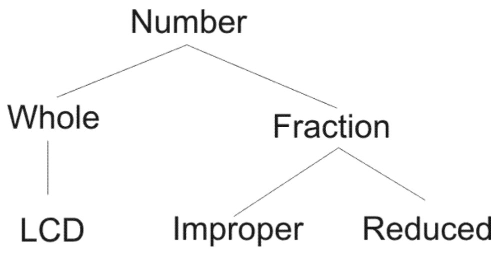


Figure 8. The ontology of the Fractions domain

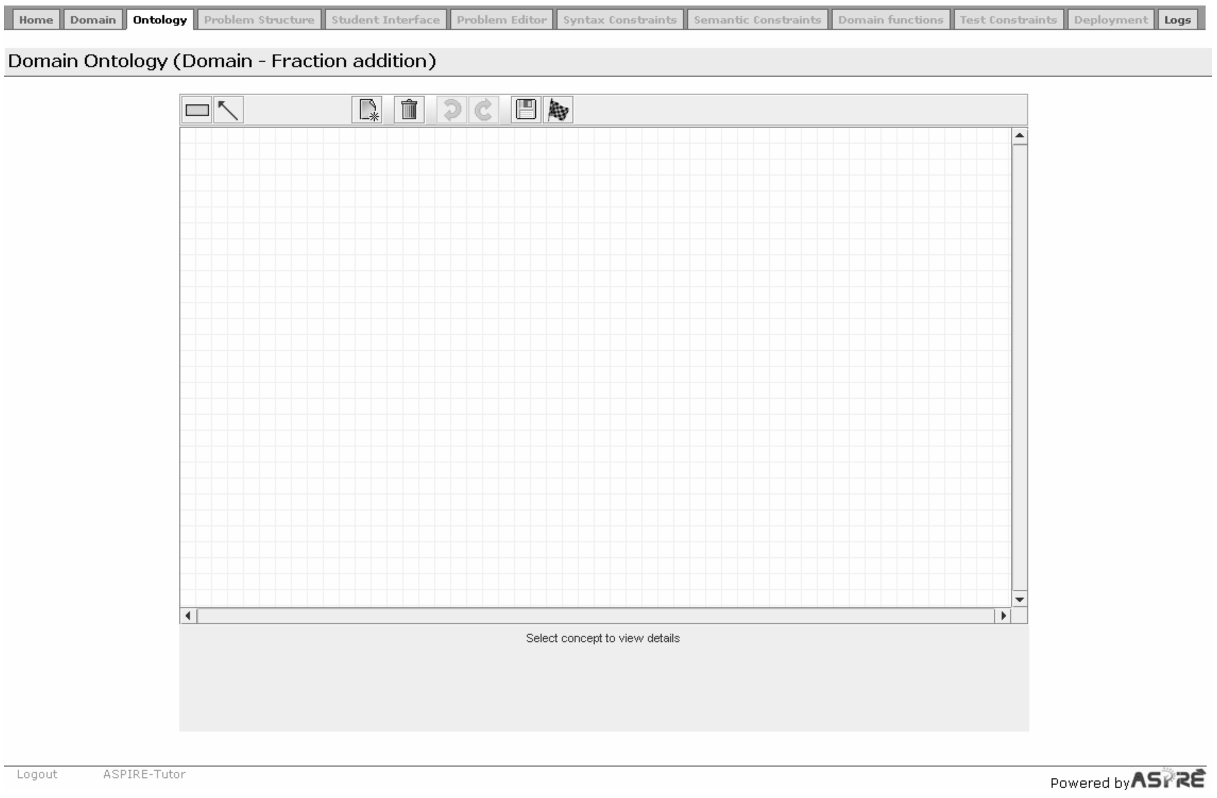
There are many ontology-development tools available, most popular of which are Protege (http://protege.stanford.edu/), Ontolingua (http://www.ksl.stanford.edu/software/ontolingua/) and OilEd (http://oiled.man.ac.uk/). These tools support different ontology languages and vary in terms of their expressiveness, reasoning abilities and support for users. ASPIRE contains an Ontology tool that will allow you to specify the ontology for your domain.

3.2.2. Ontology Workspace

After logging in, specifying the domain to work with and completing the definition of a domain, the author will be taken to the Ontology Workspace, which is initially empty (as in Figure 9). Note that the name of the instructional domain is shown at the top of the page. To show different features of Ontology Workspace, we will use the ontology shown in Figure 8.

In ASPIRE, ontologies are represented in terms as hierarchies, in which concepts are related via the 'is-a' relationship. The Ontology Workspace (Figure 9) is a graphical ontology-development tool, which supports a rich knowledge model. The taxonomy is represented as a set of concepts (rectangular boxes) connected with arrows, representing the *is-a* relationship. Note that the bar at the top of the drawing area contains a set of tools that can be used to draw the ontology and manage it. The rectangle and arrow tools are used to draw the hierarchy (i.e. to draw concepts and relationships between them). In addition to these two tools, there is also a tool for starting a new ontology (the empty page tool), the tool for deleting the currently select element of the ontology (the trash can tool), the tools for undoing/redoing the last action, the tool for saving the ontology (shown as diskette) and the finish tool, which has the effect of saving the ontology and leaving ASPIRE (shown as the finish flag). When you position the mouse over a tool without clicking on it, you will get a tool tip - a short text explaining what the tool does. Below this tool bar, there is a drawing pane, where the domain hierarchy can be drawn.

To define the ontology, it is necessary to create the identified domain concepts, and also to specify their properties and relationships. In our case, we start with the *Number* concept. To create a concept, select the box tool by clicking on it, and then click on the drawing pane. This creates a box; you can now type the name of the concept. Alternatively, you can click the rectangle tool, then click and drag on the drawing pane to create a box of the desired size. To name the concept, double click on the box. It is also possible to change the sizes of previously defined boxes, by selecting the handles (shown as crosses on boxes, which turn into large dots when a box is selected), and dragging the box. To delete a component, click on it and then press the Delete key, or alternatively, click the garbage can tool on the toolbar.



Domain Ontology (Domain - Fraction addition)

Select concept to view details

Figure 9. Ontology Workspace

The arrow graphically represent the *is-a* relationship. The direction of the arrow should be from the subconcept towards a concept. To draw the arrow, click on the tool in the toolbar, and then click on the concept and drag towards the subconcept. The Ontology Workspace assists in connecting concepts with arrows by automatically connecting endpoints of the arrow with a concept handle within a range of 5 pixels. Figure 10 shows the screenshot of the Ontology workspace after the whole hierarchy has been created.

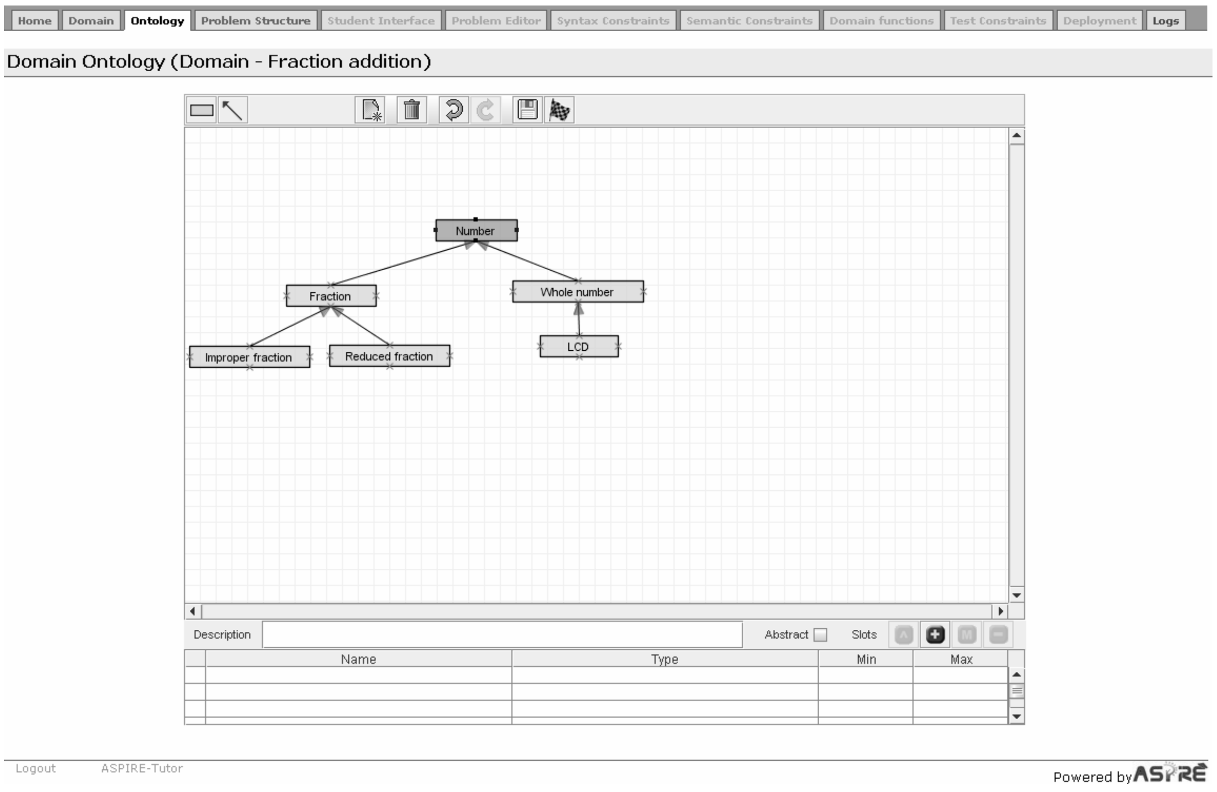


Figure 10. Ontology Workspace showing the Fractions ontology

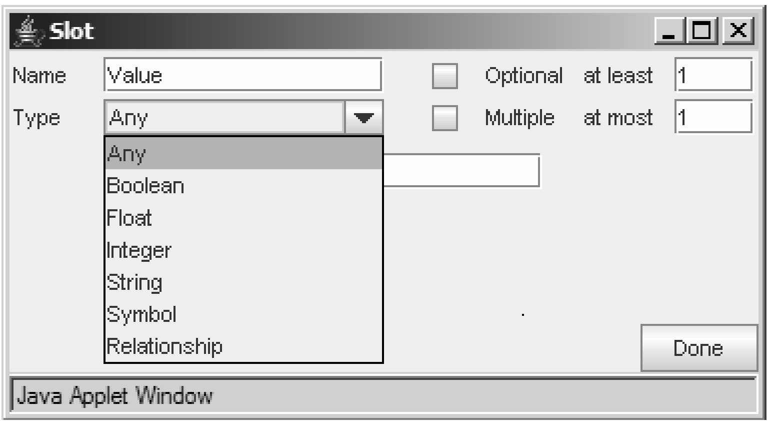
When the author selects a concept from the taxonomy, its details are shown in the bottom section of the Ontology Workspace. For example, Figure 10 shows a situation when the currently selected concept is *Number*. The details panel includes a text area for adding the description of the concept and a table that lists the concept's slots. The description of a concept allows the author to enter an explanation for the chosen concept. This description is only useful for the author him/herself: it is not used by ASPIRE, and may help the author to provide additional information.

The *Abstract* box allows the author to specify that the selected concept is an abstract concept. Abstract concepts are those that the student will not directly use in their solutions; such concepts might represent higher-level generalizations of the concepts that students will manipulate with directly. In other words, abstract concepts cannot be instantiated in solutions. As they do not appear in solutions, a domain model does not contain any constraints for such concepts. For example, the domain model would not check for the existence of items that are of an abstract concept type.

Slots can be either properties of a concept that describe that concept, or relationships with other concepts in the ontology. To add a slot, click the ⊕ button in the details panel. That results in a pop-up window, as one shown in Figure 11. The ⊖ button deletes the currently selected slot from the table, while the M button brings up the slot definition window, so that the author can modify it.

When adding a new slot, the name and type of a slot must be defined. The name of each slot must be unique. The type of the slot specifies whether it is a property or a relationship. A property describes one particular feature of the concept. On the other hand, a relationship is an association between the current concept and some other concept from the ontology.

In Figure 11, the author is specifying the Value property of the *Number* concept. A property may have values of type Boolean, integer (i.e. whole numbers), string, float, or symbol. To see the various options for Type, click on the icon on the right of *Any*, which displays the drop-down list. Select the appropriate value from the menu. When specifying the type of the *Value* property, the author selects *Float* (meaning the number can have a real value). However, the type of a property does not have to be specified (i.e. the *Any* option may be used).


Slot

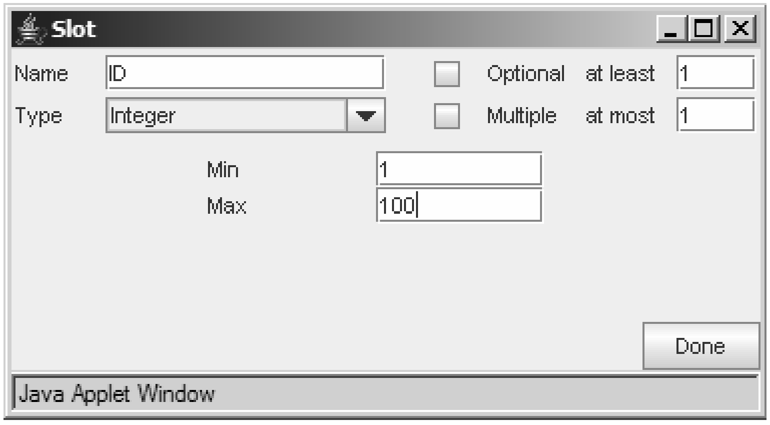
Name: Value
Type: Any
Optional: at least 1
Multiple: at most 1

Figure 11. Adding a new slot

The *Optional* checkbox allows the author to specify whether the slot is optional or mandatory (i.e., whether there must exist a value of the property for every instance of the concept). An optional property means that not all objects of this type will have a value for that property.

A property may have one or more values. For example, a number can have only one value. However, there are properties that can have more than one value simultaneously. For example, an object might have multiple colours. There are two ways to specify that a property might have multiple values. If the number of values is not known, but we know that there will be more than one value, it is enough to check the *Multiple* box. On the other hand, if we know that a certain property may have between one and three values. To do that, enter 1 into the "at least" box, and 3 into the "at most" box. Please note that the default value of the "at least" and "at most" box is 1 for all types of slots.

ASPIRE-Author also allows the author to specify a range of values that a property can take. Figure 12 illustrates a situation when the author is adding the ID property, which can take as its value any integer in the range of 1 to 100. To specify the range, the author enters 1 and 100 into the min and max boxes. The ranges can be specified for properties of the following types: integer, float and string.


Slot

Name: ID
Type: Integer
Optional: at least 1
Multiple: at most 1
Min: 1
Max: 100

Figure 12. Specifying a range of values

The *Boolean* type allows only two values: true and false. When specifying a property of type Boolean, instead of the *at least* and *at most* input boxes, the author specifies the default value, by selecting either *True* or *False* from the drop-down list, as shown in Figure 13.

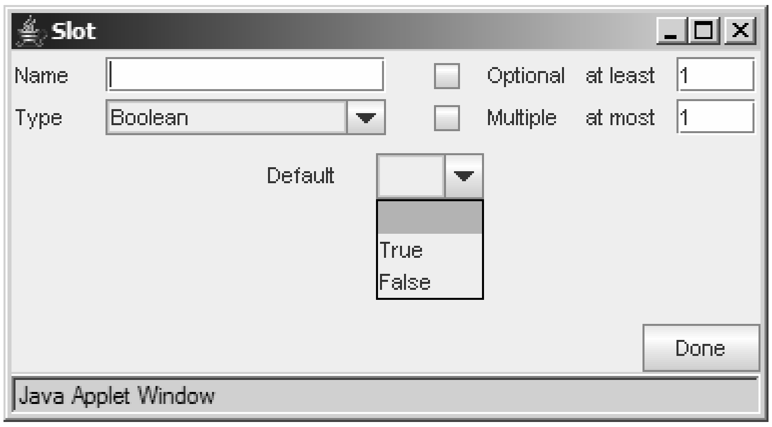


Figure 13. Adding a property of type Boolean

If the property is of type symbol, the author needs to enumerate the allowed values (Figure 14), by entering them one at a time and clicking the button. To delete a previously specified value, the author needs to select it and click the button.

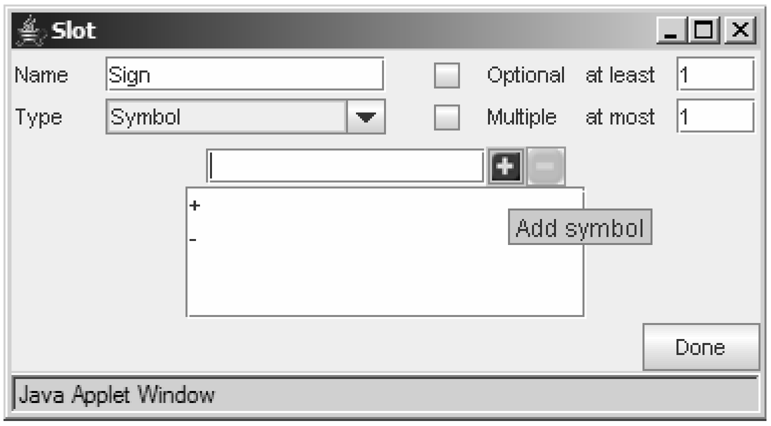
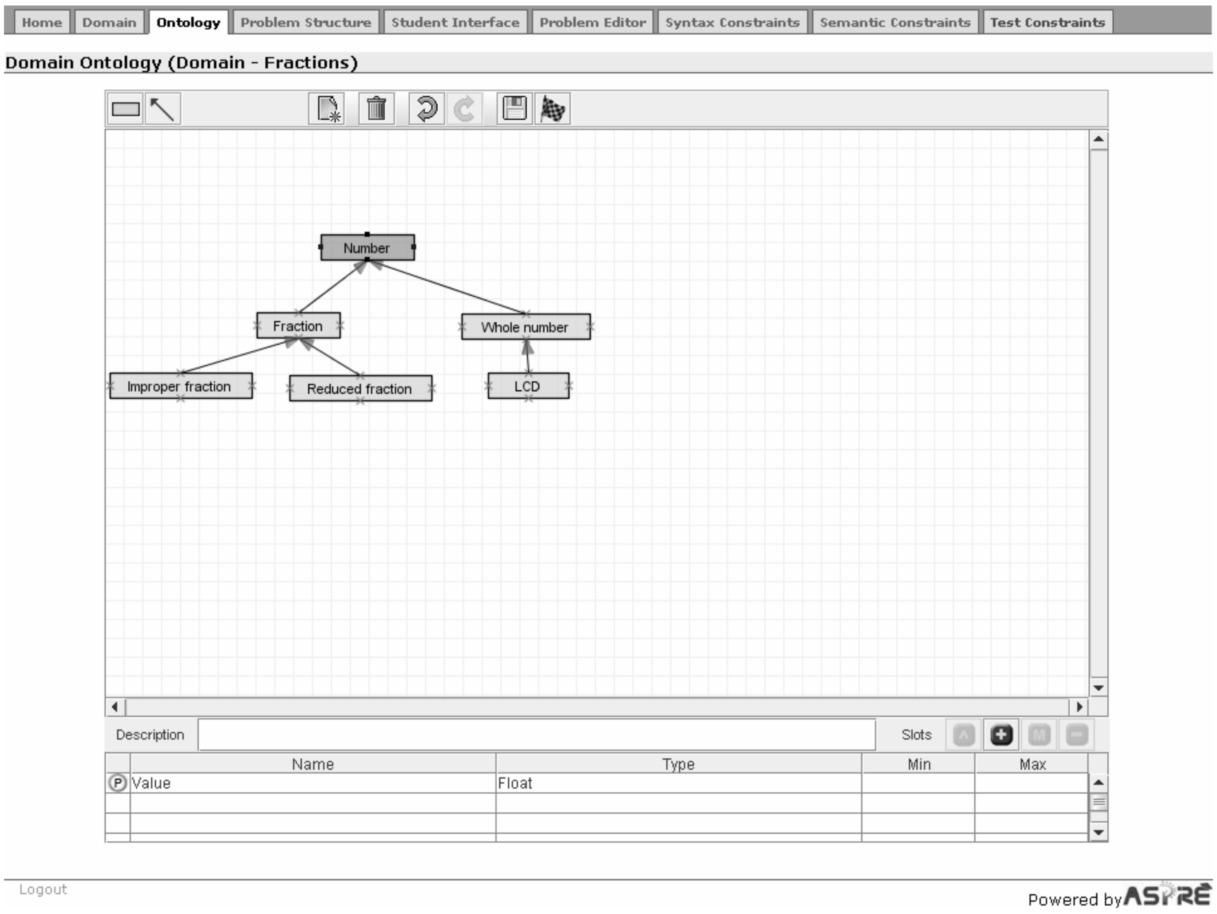


Figure 14. Adding a property of type symbol

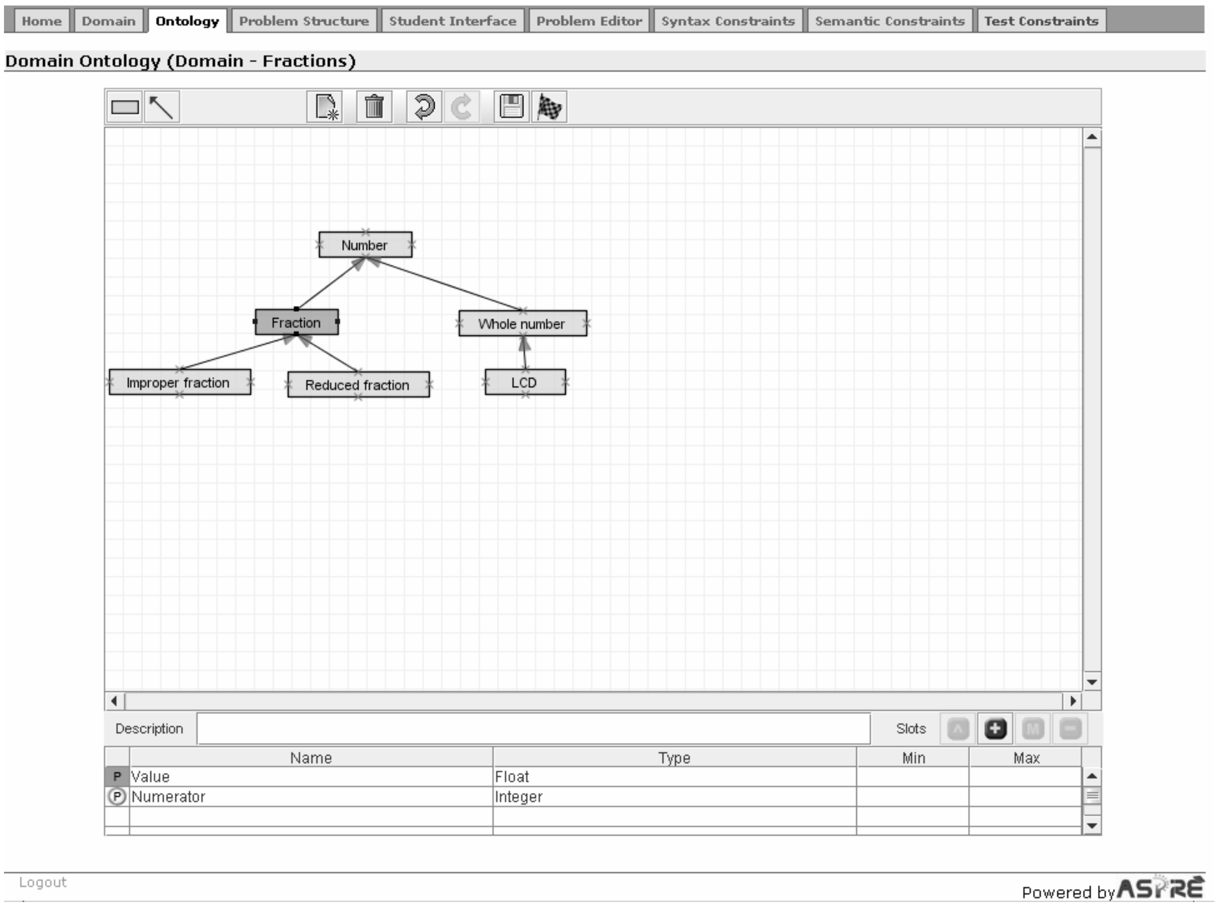
Figure 15. **Slots of the Number concept** shows the screenshot of the Ontology Workspace after one property of the *Number* concept has been specified. The icon in the first column of the slots table indicates whether the slot is a property or a relationship. Properties are identified by and relationships by . In Figure 15, *Value* is a property.



	Name	Type	Min	Max
Ⓟ	Value	Float		

Figure 15. Slots of the Number concept

All the properties defined for a concept would be inherited by its children (i.e. the concepts related to it by the 'is-a' relationship). Figure 16. **Inherited and local properties** shows the same ontology with *Fraction* as the selected concept. This concept inherits the *Value* property from the *Number* concept. To specify that the properties are inherited, the Ontology Workspace will put a different symbol in the appropriate row of the slot table. In Figure 16, the *Fraction* concept has one inherited property (*Value*) and one local property (*Numerator*).



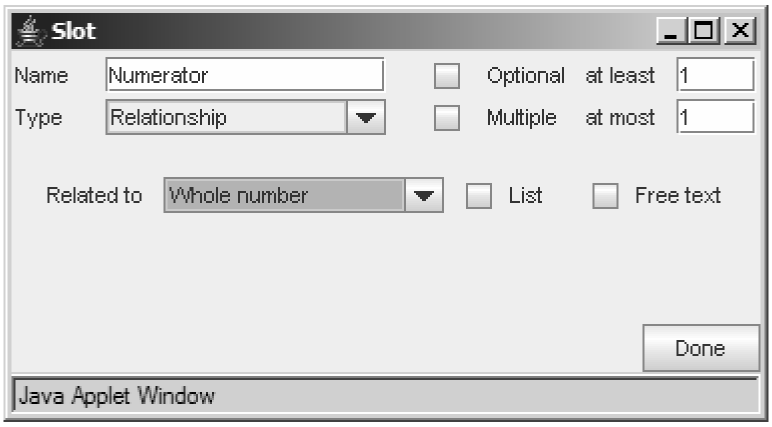
	Name	Type	Min	Max
P	Value	Float		
P	Numerator	Integer		

Figure 16. Inherited and local properties

Note: This paragraph explains planned functionality that has not been implemented yet. A property that is inherited from the parent concept may be modified in the current concept. For example, consider the *Value* property defined for the *Number* concept. Whole number would inherit this property from its parent; however, the value of a whole number is not float. To make this change, click the M button. This will bring the Slot definition window, in which you redefine the type of the property. Figure 3.1X shows the redefined *Value* property for the *Whole number* concept.

Figure 3.1X. Redefined property

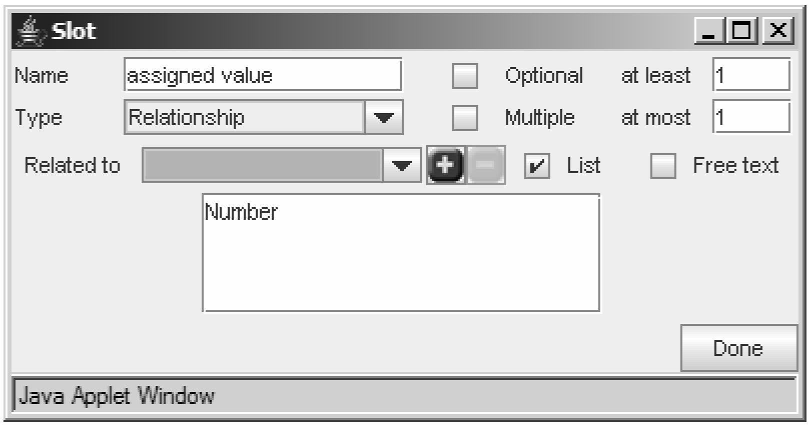
To specify a relationship between the currently selected concept and another concept from the ontology, click the + button in the Details frame, which brings up the slot definition window. After specifying the name of the slot, select *relationship* as its type. Figure 17 shows the *Numerator* relationship for the *Fraction* concept. Each fraction must have a numerator and a denominator, which are whole numbers. Whole numbers have previously been defined in the ontology. To specify the numerator as a component of a fraction, it is necessary to specify the relationship between *Fraction* and *Whole Number*. In order to specify the related concept, select an option from the drop-down list of concepts, which lists all the concepts from the current ontology.



Name: Numerator
Type: Relationship
at least: 1
at most: 1
Related to: Whole number

Figure 17. Adding a relationship

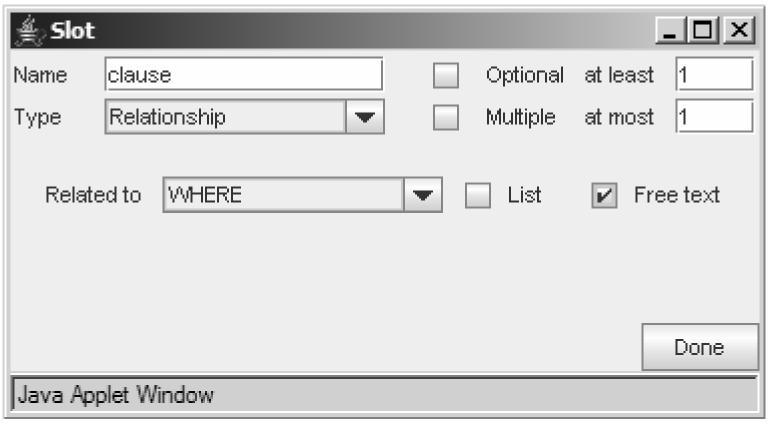
In some cases, a relationship may involve one of a set of concepts. For example, when specifying an assignment (a statement that assigns a value to a variable), the author may specify that the allowed concepts on the right-hand side are constants (e.g. "x = 1"), variables (e.g. "x = y"), functions (e.g. "x = max(a,b,c)") or arithmetic expressions (e.g. "x = y + 3"). To enumerate concepts that can participate in the relationship, tick the *List* tick box next to the concept list. ASPIRE-Author will show a table which will hold all selected concepts. Concepts can be added to the container by selecting the appropriate concept from the drop-down list, and clicking the + button. Figure 18 shows the *assigned value* relationship, when the first related concept has been added (i.e. the *Number* concept).



Name: assigned value
Type: Relationship
at least: 1
at most: 1
List: ✓
Number

Figure 18. Specifying a set of concepts for a relationship

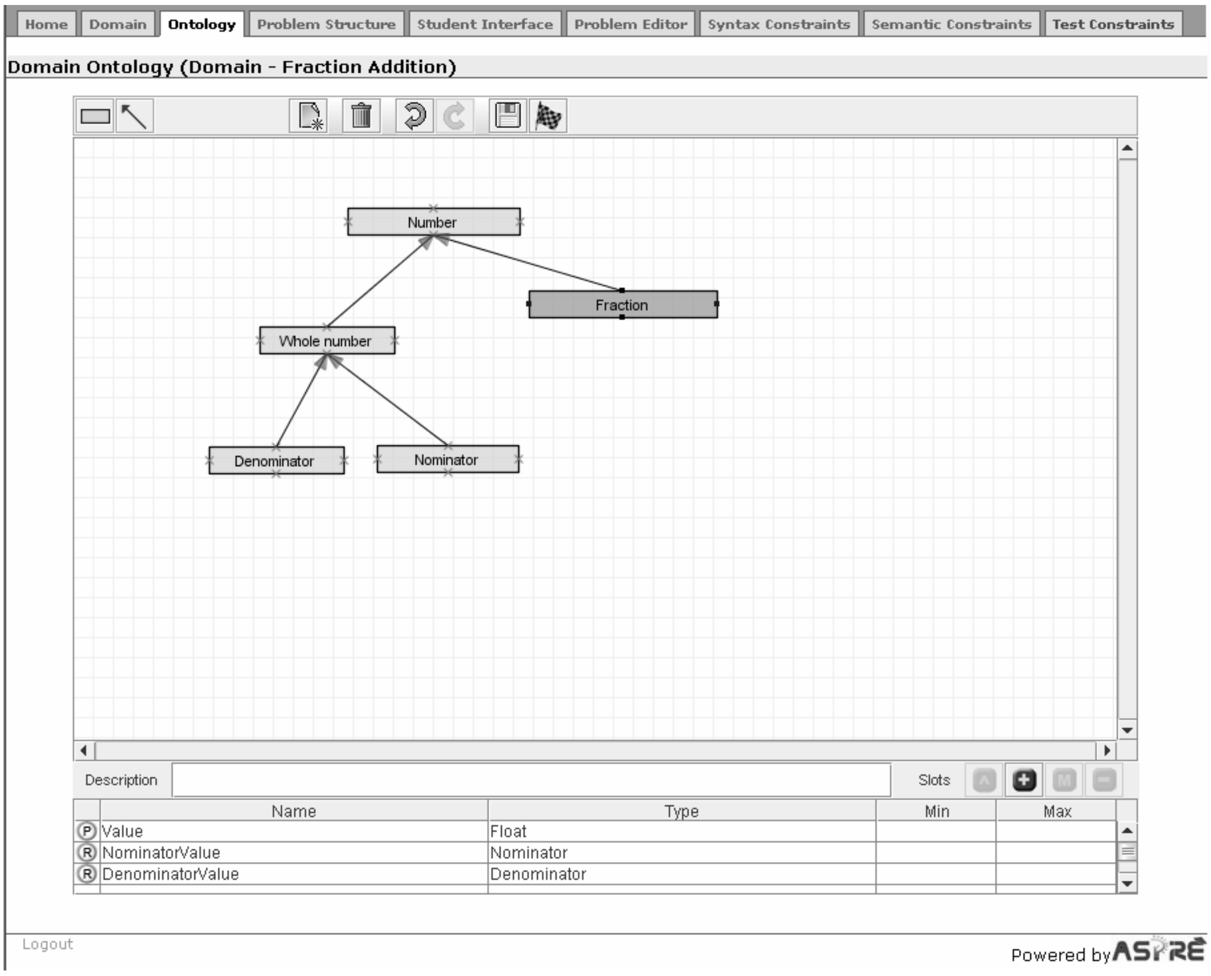
In certain domains, it may be the case that the student would be typing in a large part of the solution. In that case, it would be very complicated to specify the structure of that part of the solution directly in the ontology. In such cases, the author may specify that a slot of a certain concept is a relationship with another concept, the structure of which is not going to be described further. Figure 19 illustrates such a case. The *clause* relationship is related to the WHERE concept, and the author has ticked the *Free text* box to let ASPIRE-Author know that the student would be typing in the content of this concept.



Name: clause
Type: Relationship
[ ] Optional at least: 1
[ ] Multiple at most: 1
Related to: WHERE [ ] List [✓] Free text

Figure 19. Specifying a free-text relationship

Figure 20 shows the screenshot of the Ontology Workspace, with the currently selected concept having both properties and relationships. The slots table displays the type of the slot in the case of properties, but for relationships it shows related concepts.



Domain Ontology (Domain - Fraction Addition)

	Name	Type	Min	Max
P	Value	Float		
R	NominatorValue	Nominator		
R	DenominatorValue	Denominator		

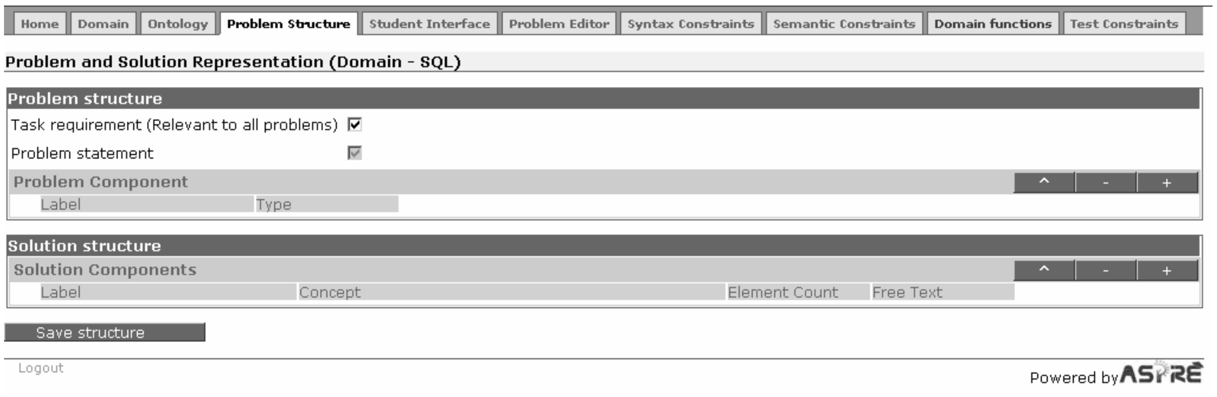
Figure 20. Properties and relationships of the Fraction concept

3.3. Modeling the problem/solution structures

Once the domain ontology is defined and a problem set is selected, it is necessary to specify the problem structure and the solution structure. To achieve this, click the *Problem Structure* tab. The *Problem and Solution Representation* page will be shown next. Figure 21 shows the initial state of the problem/solution structure for the SQL queries domain, while Figure 22 shows the initial state for the Fractions domain. The top portion of this page is used to specify the problem structure.

ASPIRE assumes that each problem will contain a problem statement. In addition to that, it is possible to specify the task requirement - this is the description of the task that will be given to students with each problem, giving them additional instructions.

For example, in the fraction addition domain, the task requirement may be "Add the following two fractions:", while the problem statement will specify the two fractions to be added (e.g. 1/5 + 2/3). As another example, let us take a look at a language tutor, which contains a set of problems dealing with turning verbs into nouns. All problems of this type would have the same task requirement entered just once by the author: "Turn the following verb into a noun", and then each verb would be entered separately as the problem statement.



Problem and Solution Representation (Domain - SQL)

Problem structure

Task requirement (Relevant to all problems) ✓

Problem statement ✓

Problem Component

Label	Type

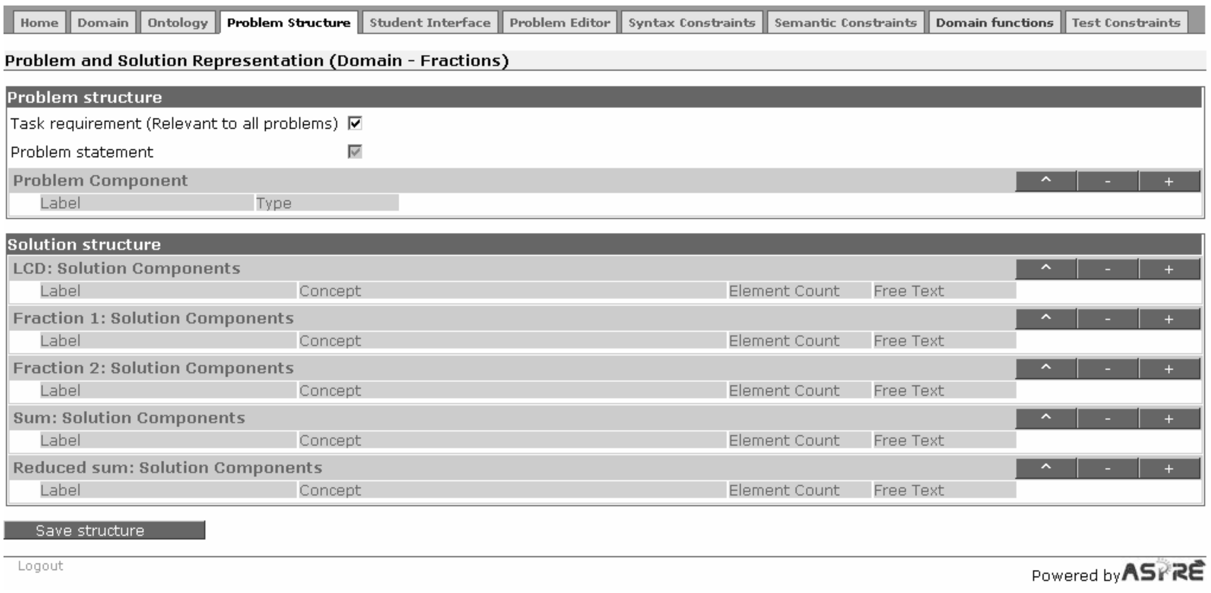
Solution structure

Solution Components

Label	Concept	Element Count	Free Text

Figure 21. Initial state of the problem/solution representation for the SQL queries domain

To specify that there is such a task requirement, use the tick box associated with the first element (called *Task requirement*) of the problem structure interface. The text for the task requirement can be added in the problem editor, when problems and their solutions are added (see Section 3.6). There is no need to do anything about the second element (called *Problem statement*), as it is assumed that every problem will have a specific problem statement. This element is included in the interface to make it obvious that a problem statement will always be a part of the problem specification.



Problem and Solution Representation (Domain - Fractions)

Problem structure

Task requirement (Relevant to all problems) ☑

Problem statement ☑

Problem Component

Label	Type

Solution structure

LCD: Solution Components

Label	Concept	Element Count	Free Text

Fraction 1: Solution Components

Label	Concept	Element Count	Free Text

Fraction 2: Solution Components

Label	Concept	Element Count	Free Text

Sum: Solution Components

Label	Concept	Element Count	Free Text

Reduced sum: Solution Components

Label	Concept	Element Count	Free Text

Figure 22. Initial state of the problem/solution representation for the Fractions domain

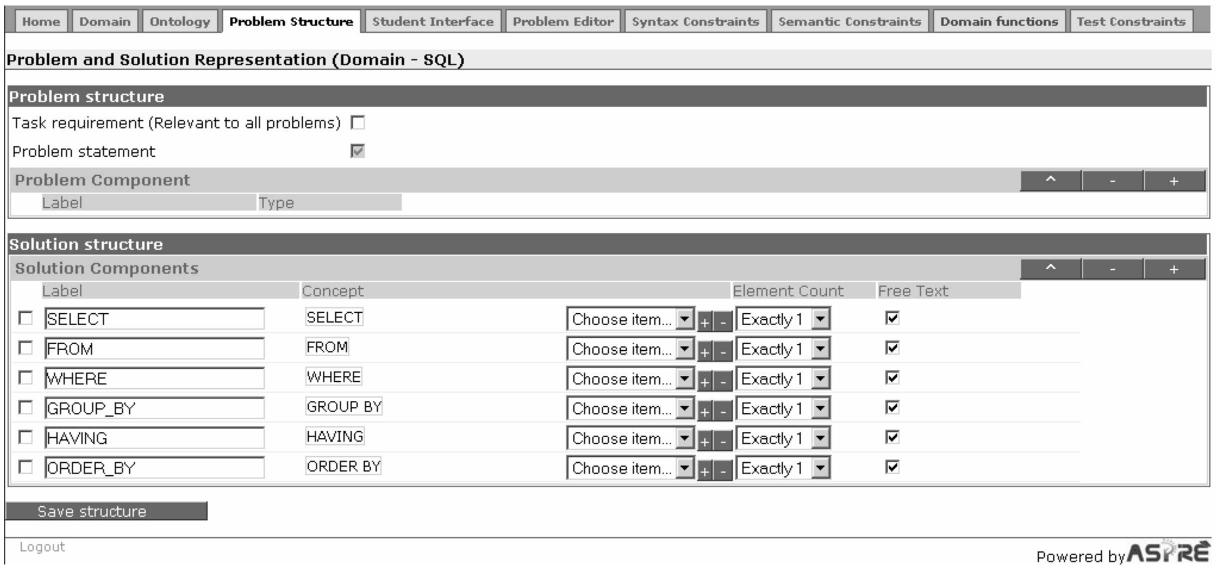
A problem may also contain a collection of sub-components that add more information to the problem statement. The problem components can be added into the problem components table by clicking the + button and removed using the – button. Clicking the + button results in a new row, which can be populated to add a new problem component. Problem components are described by their label and type. The label is displayed in the student problem solving interface next to the problem component. Each component can be either textual (*text*) or graphical (*image*). The components are problem-specific, and are therefore specified when the problem is defined, in the Problem Editor (see Section 3.6).

Next, it is necessary to specify the structure of the solution. The solution structure is different for declarative and procedural tasks.

3.3.1. Modeling the solution structure for non-procedural tasks

The initial state of the problem/solution structure for a non-procedural domain (SQL queries) is shown in Figure 21. As can be seen, there is nothing shown under *Solution structure*, as there are no steps defined for this task.

The solution structure for a non-procedural task consists of a list of solution components. The components can be added and removed in a manner similar to the addition and removal of problem components, by clicking the + and – buttons. Each solution component has a label, the type of elements it may hold (i.e. the concept from the domain ontology), and the number of elements it may hold (*Element Count*). Additionally, there is a *Free text* box for each component, which needs to be ticked if the student can freely type the content of the component. The free text type determines that the component should be displayed in the student problem solving interface as a text box. Figure 23 shows the structure of solutions in the SQL domain. Each solution contains six components, defined in terms of corresponding domain concepts. As can be seen from the figure, each component has exactly one element, and is of *free text* type, which means that the student will be asked to type the content of each component into a text box in the problem-solving interface.


Home | Domain | Ontology | Problem Structure | Student Interface | Problem Editor | Syntax Constraints | Semantic Constraints | Domain functions | Test Constraints

Problem and Solution Representation (Domain - SQL)

Problem structure

Task requirement (Relevant to all problems) ☐

Problem statement ☑

Problem Component ^ - +

Label	Type

Solution structure

Solution Components ^ - +

	Label	Concept		Element Count	Free Text
☐	SELECT	SELECT	Choose item... + -	Exactly 1	☑
☐	FROM	FROM	Choose item... + -	Exactly 1	☑
☐	WHERE	WHERE	Choose item... + -	Exactly 1	☑
☐	GROUP_BY	GROUP BY	Choose item... + -	Exactly 1	☑
☐	HAVING	HAVING	Choose item... + -	Exactly 1	☑
☐	ORDER_BY	ORDER BY	Choose item... + -	Exactly 1	☑

Save structure

Logout

Powered by ASPIRE

Figure 23. Specifying the solution structure for a non-procedural task

3.3.2. Modelling the solution structure for procedural tasks

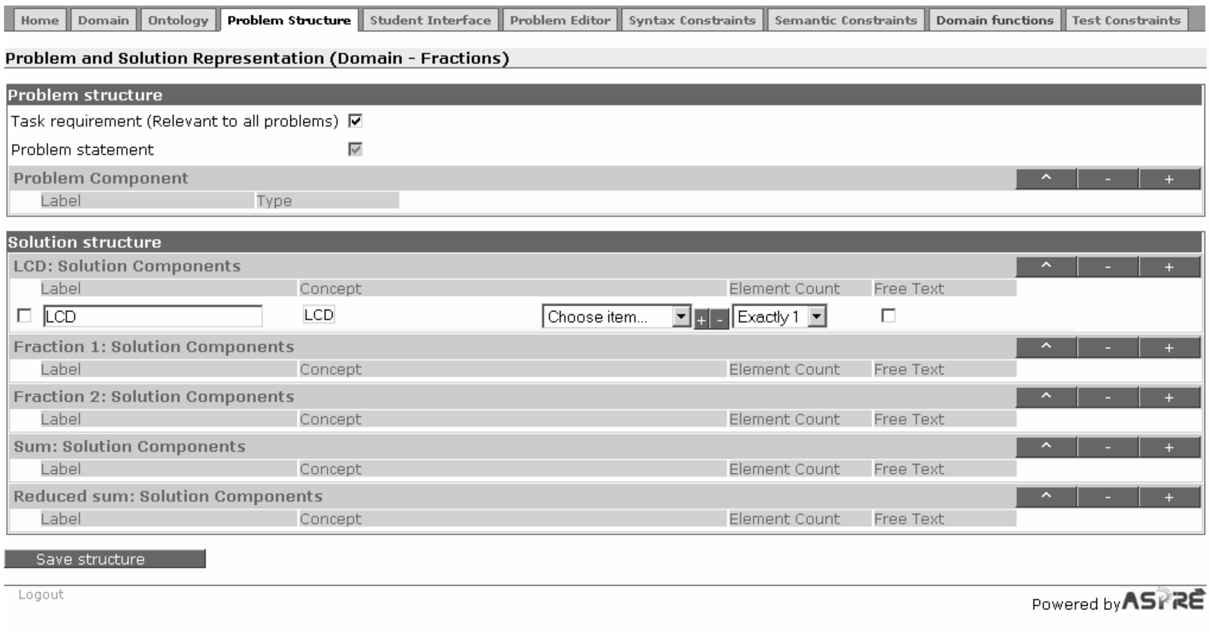
The interface for modelling the structure of solutions for procedural tasks (see Figure 22) is similar to the interface for declarative tasks. The main difference is the presentation of the solution structure. As each problem solving step requires a solution which may contain several parts, the composition of solutions for each step has to be modelled separately. Consequently, the solution structure for procedural domains consists of a collection of solution component lists, one for each problem solving step.

Initially, as can be seen in Figure 22, ASPIRE will list the steps that have been defined for the task of adding fractions. For each step, there might be one or more components of the solution that the student will need to specify. To add a component for a step, click the + button in the corresponding row of the table. Then specify the label for the component, the corresponding concept from the domain ontology and how many elements the student may enter. There are four options available for the *Element Count*:

- *exactly 1*, meaning that there is only one element which is mandatory;
- *at least 1*, meaning that there may be one or more elements of this type in the solution;
- *0 or 1*, meaning that the component is option. If it exists in the student's solution, there could be only a single element specified.
- *0 or more*, meaning that the component is optional, but may have multiple elements if specified.

For example, in the first step of adding fractions, the student needs to specify the lowest common denominator, which is a single number. To add this component, the author would specify the label the student will see (such as *Lowest Common Denominator*), and select the LCD concept from the options listed, and finally specify that there is only one number to be added (as in Figure 24).

The other components of solutions in the Fraction Addition domain (the two fractions, the sum of fractions and the reduced sum) are added in the same way. Figure 25 shows the complete solution structure. Once the solution structure is completed, click the *Save structure* button.



Problem and Solution Representation (Domain - Fractions)

Problem structure

Task requirement (Relevant to all problems) ☑

Problem statement ☑

Problem Component

Label	Type

Solution structure

LCD: Solution Components

	Label	Concept		Element Count	Free Text
☐	LCD	LCD	Choose item...	Exactly 1	☐

Fraction 1: Solution Components

Label	Concept	Element Count	Free Text

Fraction 2: Solution Components

Label	Concept	Element Count	Free Text

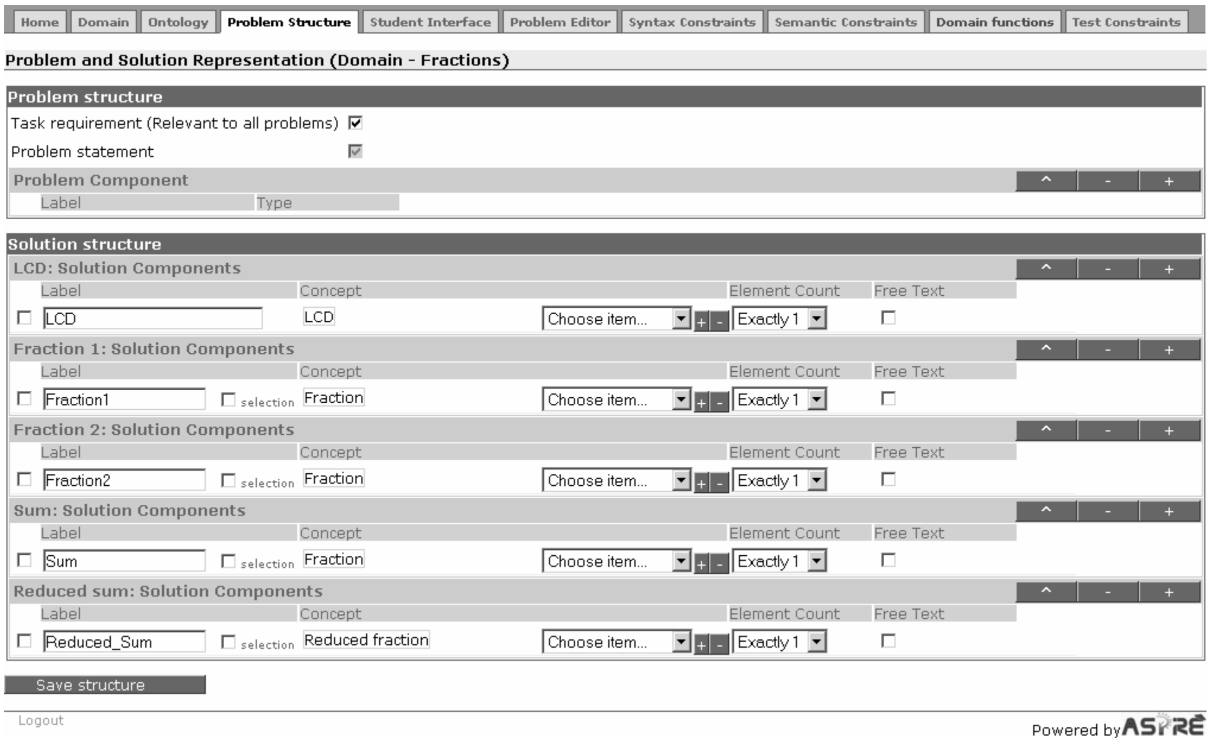
Sum: Solution Components

Label	Concept	Element Count	Free Text

Reduced sum: Solution Components

Label	Concept	Element Count	Free Text

Figure 24. The solution structure with the initial component specified



Problem and Solution Representation (Domain - Fractions)

Problem structure

Task requirement (Relevant to all problems) ☑

Problem statement ☑

Problem Component

Label	Type

Solution structure

LCD: Solution Components

	Label	Concept		Element Count	Free Text
☐	LCD	LCD	Choose item...	Exactly 1	☐

Fraction 1: Solution Components

	Label		Concept		Element Count	Free Text
☐	Fraction1	☐ selection	Fraction	Choose item...	Exactly 1	☐

Fraction 2: Solution Components

	Label		Concept		Element Count	Free Text
☐	Fraction2	☐ selection	Fraction	Choose item...	Exactly 1	☐

Sum: Solution Components

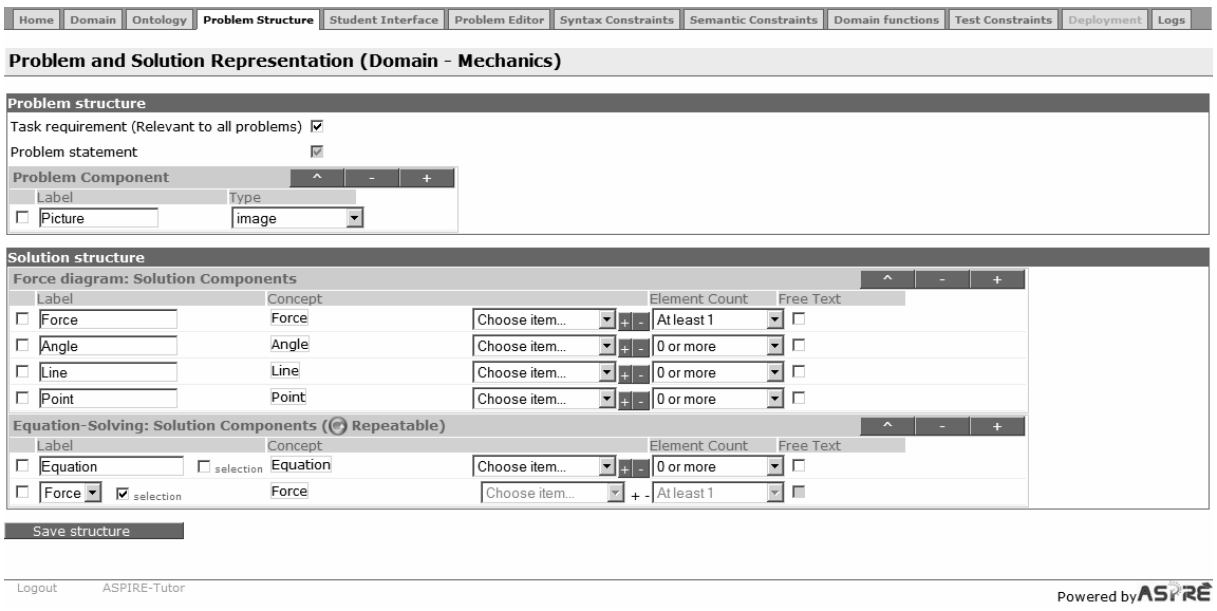
	Label		Concept		Element Count	Free Text
☐	Sum	☐ selection	Fraction	Choose item...	Exactly 1	☐

Reduced sum: Solution Components

	Label		Concept		Element Count	Free Text
☐	Reduced_Sum	☐ selection	Reduced fraction	Choose item...	Exactly 1	☐

Figure 25. The complete solution structure for fraction addition

In some domains, the student will be specifying some elements of components over two (or more) steps. It would therefore be convenient to allow the author to re-use components defined in a previous step in later steps. When the author wants to add a new component to a step which has been defined previously, it is necessary to tick the *selection* box of the corresponding row in the table. Figure 26 shows a screenshot of the Solution Structure editor, after the author has added a new component to the *Equation* step and ticked the selection box. The interface then displays a drop-down box from which the author can select a previously defined component (*Force*).



Problem and Solution Representation (Domain - Mechanics)

Problem structure

Task requirement (Relevant to all problems) ☑

Problem statement ☑

Problem Component

	Label	Type
☐	Picture	image

Solution structure

Force diagram: Solution Components

	Label	Concept		Element Count	Free Text
☐	Force	Force	Choose item...	At least 1	☐
☐	Angle	Angle	Choose item...	0 or more	☐
☐	Line	Line	Choose item...	0 or more	☐
☐	Point	Point	Choose item...	0 or more	☐

Equation-Solving: Solution Components (Repeatable)

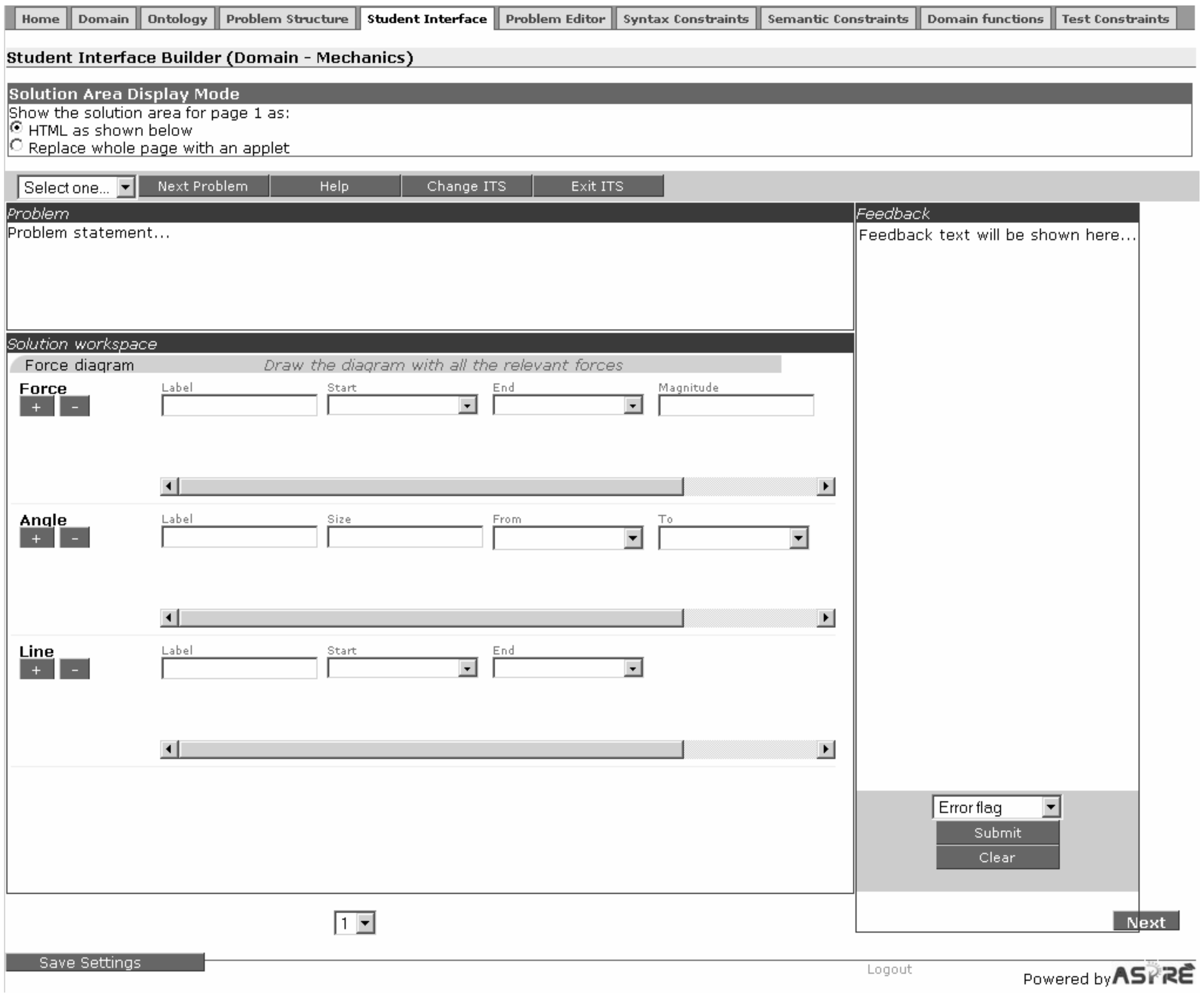
	Label		Concept		Element Count	Free Text
☐	Equation	☐ selection	Equation	Choose item...	0 or more	☐
☐	Force	☑ selection	Force	Choose item...	At least 1	☐

Figure 26. Reusing components from previous steps

Note that Figure 26. **Reusing components from previous steps** illustrates the solution structure for the domain previously discussed in Section 3.1. In this domain there are two steps, and the second step (Equation-Solving) is repeatable (see Figure 6. **A domain with a repeatable step (Equation-Solving)**). The repeatable step is clearly identified in the solution structure. All the components of this step will be repeated for each iteration of the step.

3.4. Designing the student interface

After specifying the problem/solution structures, ASPIRE will show the student interface builder. Figure 27 shows the initial state of the interface for the Mechanics domain, generated automatically from the information the author has specified. At the top of the page, the author is asked to supply information about the display mode. The default option is the HTML code as shown in this figure. If the author accepts this default option, the student will be given the HTML interface. At the top of this interface, there is a set of standard buttons that students may use to select problems, get help on how to use the system, change the system or log out. Under the buttons, the interface displays the problem area, including the general instruction and the problem statement.



Student Interface Builder (Domain - Mechanics)

Solution Area Display Mode
Show the solution area for page 1 as:
[✓ HTML as shown below]
Replace whole page with an applet

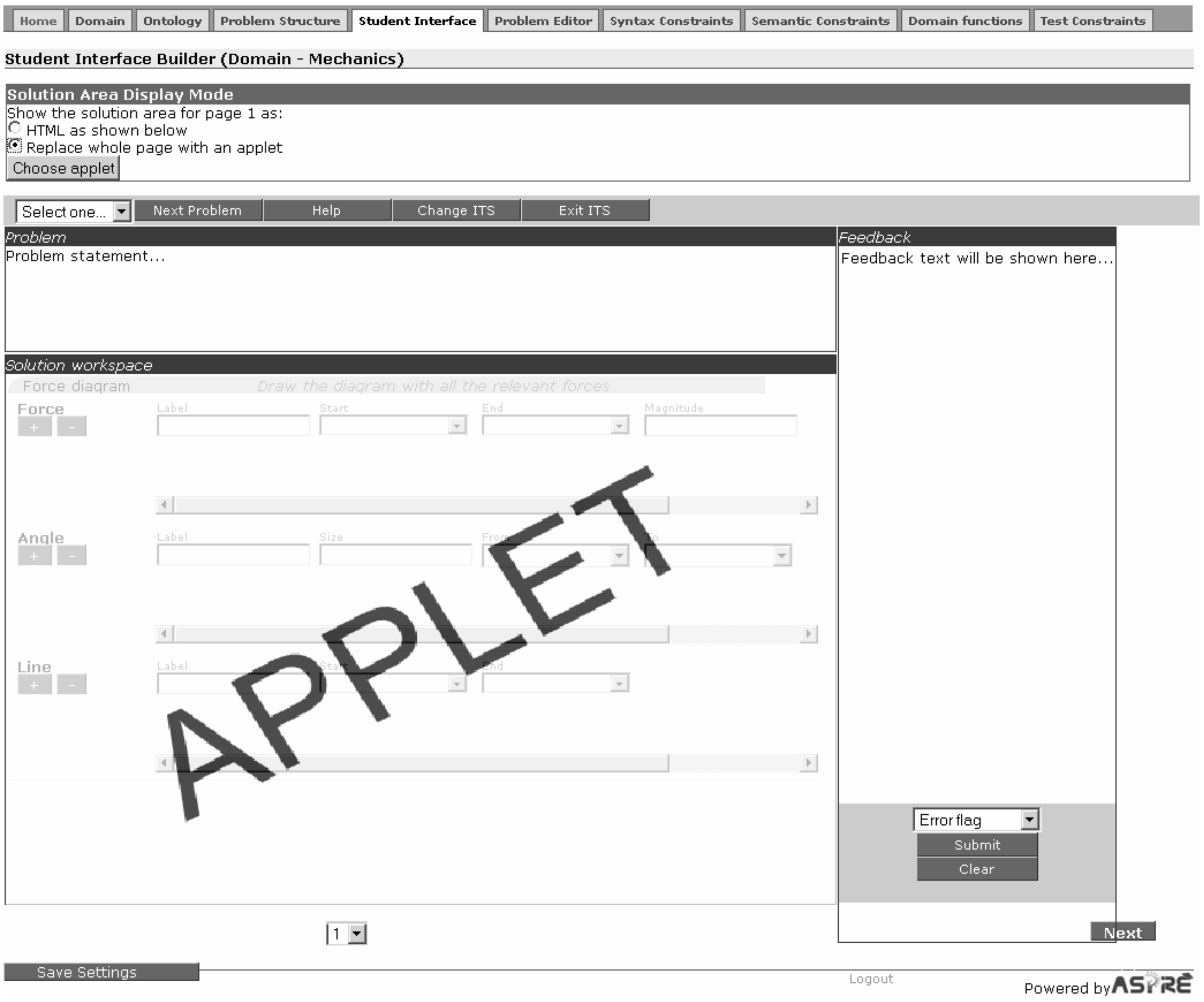
Problem statement...

Feedback text will be shown here...

Figure 27. The initial interface for the Mechanics domain

The default HTML interface expects the student to type in the components of the solution. However, in some domains this is not a realistic expectation. For example, in Mechanics, the student will be drawing a force diagram, and textual input is therefore not appropriate. In such cases, the author may provide a domain-specific applet, a component that will support the student in performing one or more steps. The author needs to specify that the student interface will contain such an applet.

If the author selects the second option, the applet will replace the HTML interface, and will include all solution components for that page. The state of the student interface builder for that case is shown in Figure 28.



Student Interface Builder (Domain - Mechanics)

Solution Area Display Mode

Show the solution area for page 1 as:

( ) HTML as shown below

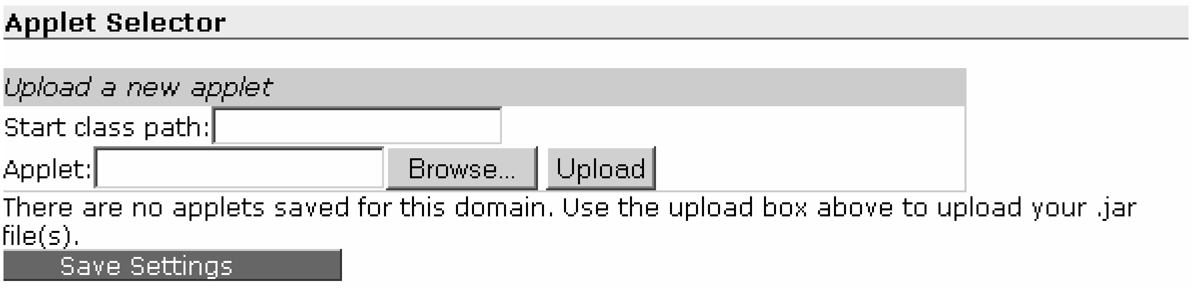
(•) Replace whole page with an applet

Problem statement...

Feedback text will be shown here...

Figure 28. Specifying an applet to replace the default interface

If the author specified that an applet will be used, he/she can then upload the applet by clicking the *Choose applet* button. This will bring a pop-up window, like the one shown in Figure 29. **Uploading applets**. The author needs to specify some applet-specific information (the start class path[2]), locate the applet on the local machine (via the *Browse* button) and upload it. Figure 29 shows a situation when there were no applets uploaded yet.


Applet Selector

Upload a new applet

Start class path:

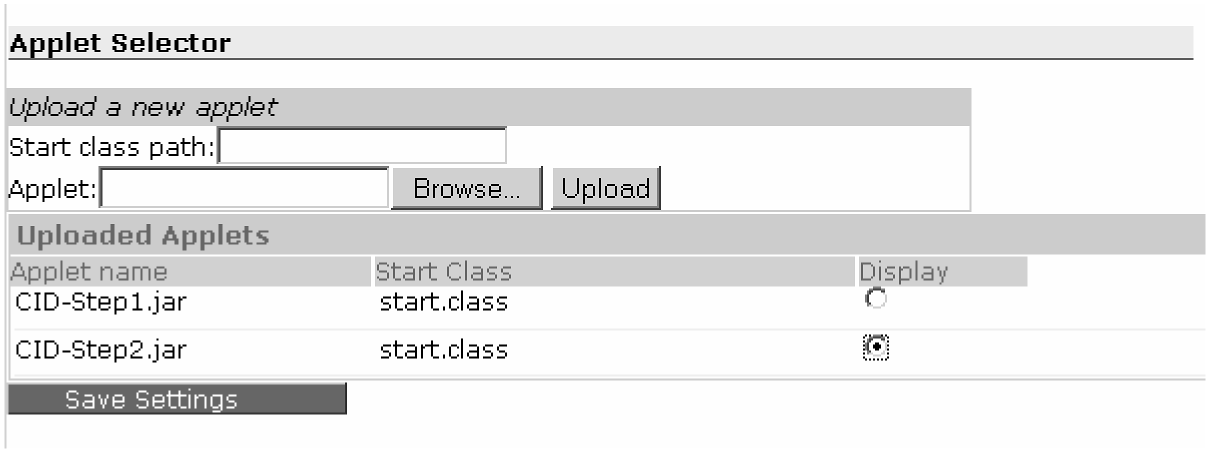
Applet:

There are no applets saved for this domain. Use the upload box above to upload your .jar file(s).

Figure 29. Uploading applets

[2] For more information about applets please see developers documentation

It is also possible to upload several applets, one after the other. The same applet can be used on more than one page. Figure 30. **Selecting applets**illustrates a situation when the author has already uploaded two applets. In order to use an existing applet, the author needs to select it (by selecting the appropriate radio button in the *Display* column). In Figure 30, the author has selected CID-Step2.jar to be used.


Applet Selector

Upload a new applet

Start class path:

Applet:

Uploaded Applets

Applet name	Start Class	Display
CID-Step1.jar	start.class	○
CID-Step2.jar	start.class	●

Figure 30. Selecting applets

The author does not have to upload the applets immediately. If the author asks for a domain to be deployed before uploading the applets, ASPIRE will show a warning message. In that case, the author can upload the applets, or continue without uploading them, in which case the default interface would be served to the student.

3.5. Problem/Solution Editor

The next step in the authoring process is to add examples of problems and their solutions. To do this, click the *Problem editor* tab. Figure 31 shows the initial state of the problem editor. There are no problems to show in the list, as none have been added yet for the domain. If some problems have been added previously, they would appear in the drop-down menu, and it would be possible to select a problem from that list in order to modify or view it.

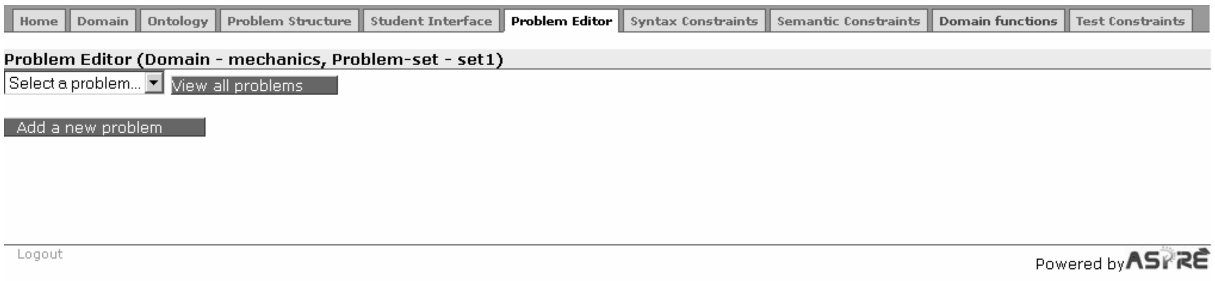
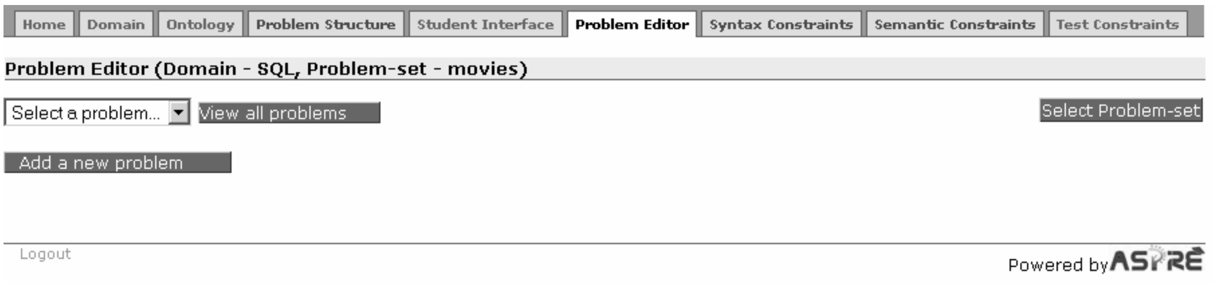

Problem Editor (Domain - mechanics, Problem-set - set1)

Figure 31. The initial state of the Problem Editor

3.5.1. Selecting a problem set

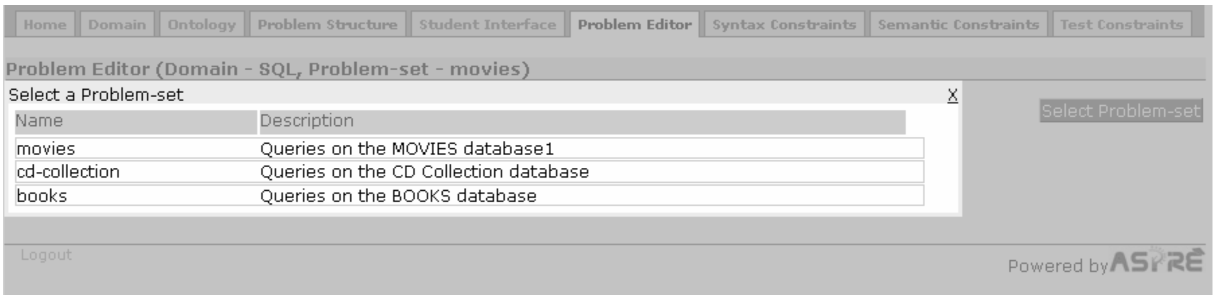
For the Mechanics domain shown in Figure 31, we defined only a single problem set; therefore all problems added will belong to the same problem set. In other situations, as discussed in Section 3.1, there might be several problems sets defined. In such a case, it is necessary to select a problem set before a problem can be added. Figure 32 shows a screen shot for a domain containing several problem sets.



Problem Editor (Domain - SQL, Problem-set - movies)

Figure 32. The initial state of the problem editor for the domain which contains several problem sets

The difference between Figure 31 and Figure 32 is that there are multiple problem sets in the case of the latter, and the interface contains an extra button at the top right (*Select Problem-set*). This button allows you to see all existing problem sets, and select the one to add new problems into. Figure 33 shows the state of the problem editor after this button is clicked. All defined problem sets are displayed in a table. To select a problem set, click the on the appropriate line in this table. After the selection, the Problem editor will show the name of the selected problem set at the top of the page.



Problem Editor (Domain - SQL, Problem-set - movies)

Select a Problem-set

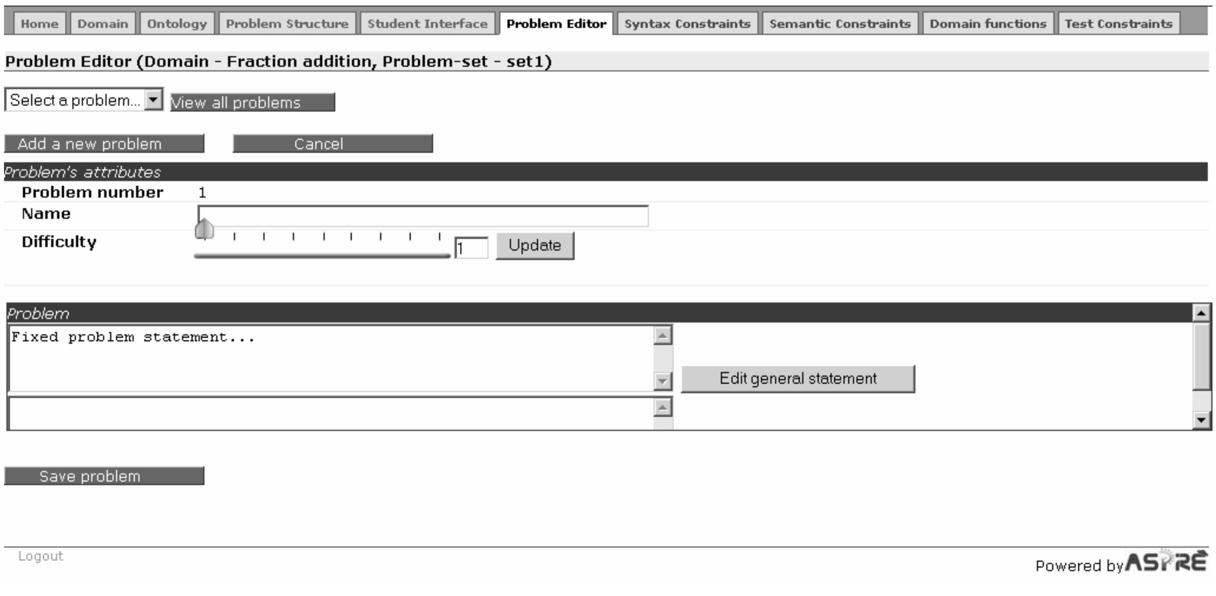
Name	Description
movies	Queries on the MOVIES database1
cd-collection	Queries on the CD Collection database
books	Queries on the BOOKS database

Figure 33. The selection of a problem set

3.5.2. Adding problems

The interface for entering problems and solutions is similar to the default student interface (i.e. HTML code generated by ASPIRE from the domain definition). To generate this interface, ASPIRE-Author uses the previously specified problem/solution structures. Therefore, when the author starts adding the first problem for the domain, the Problem Editor provides the author with the necessary interface widgets, based on the problem structure, and expects the author to populate them.

When the author clicks the *Add a new problem* button, ASPIRE will show the updated screen, as in Figure 34. There are several general problem features to specify, shown in the *Problem's attributes* area. The unique problem number is generated automatically by the system (1 in this case, as the author is adding the first problem for the chosen problem set). The author may specify a name for the problem, which is optional. If the problem name is specified, it will be shown to students together with the problem number; otherwise, students will only see the problem number.



Problem Editor (Domain - Fraction addition, Problem-set - set1)

Problem's attributes

Problem number	1
Name	
Difficulty	1

Problem

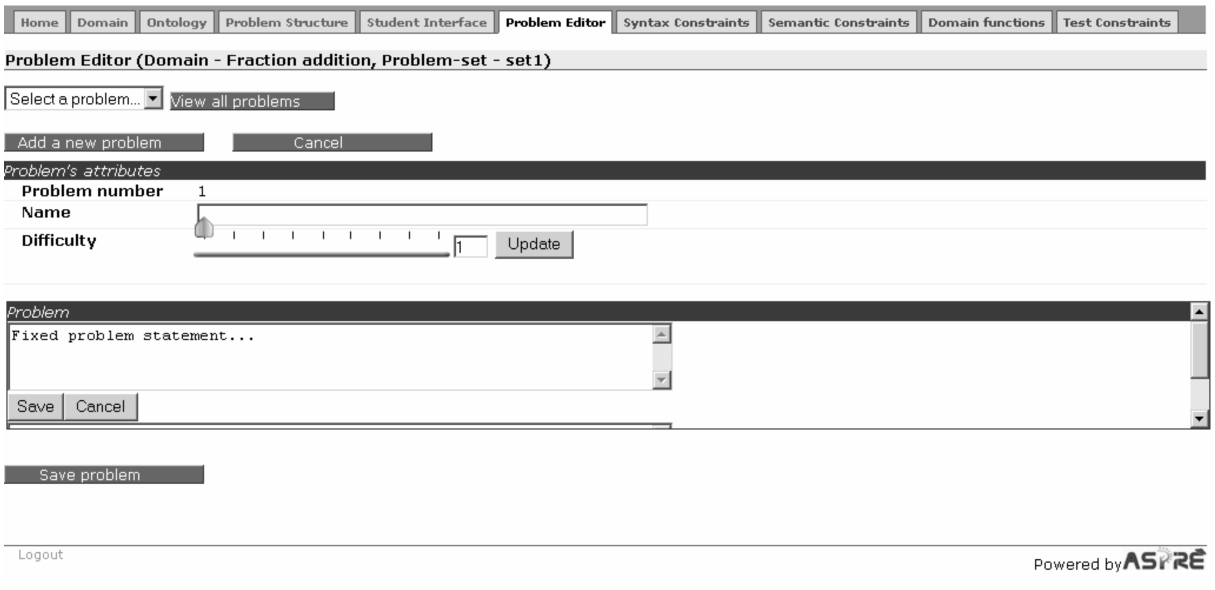
Fixed problem statement...

Figure 34. Adding a new problem

The author must specify the problem difficulty, which ranges from 1 (the simplest problems) to 9 (for most complex problems). To specify the difficulty level, the author can use the slider, or enter the desired number into the text box. If the text box is used to enter the difficulty number, the author needs to click the *Update* button for the difficulty to be updated on the slider shown.

Then the author needs to specify the task the student is to perform. As discussed in Section 3.4, in some domains all problems will have the same general description of what the task is: e.g., in a language tutor, this description might be: "Turn the following verb into a noun," and the student is given a series of verbs to work on. On the other hand, for certain problem sets, there is no such general description, and every problem will have distinct instructions. The author specifies whether there is such a general description for the problem set as well as any additional components for the problems (e.g. a diagram) in phase 3, and this information is available within the domain model.

To specify the problem statement, click the *Edit general statement* button. Figure 35 shows the modified interface. Type the problem statement, and then click the *Save* button. Once the problem statement is specified, it will appear automatically for all future problems in the same problem set.



Problem Editor (Domain - Fraction addition, Problem-set - set1)

Problem's attributes

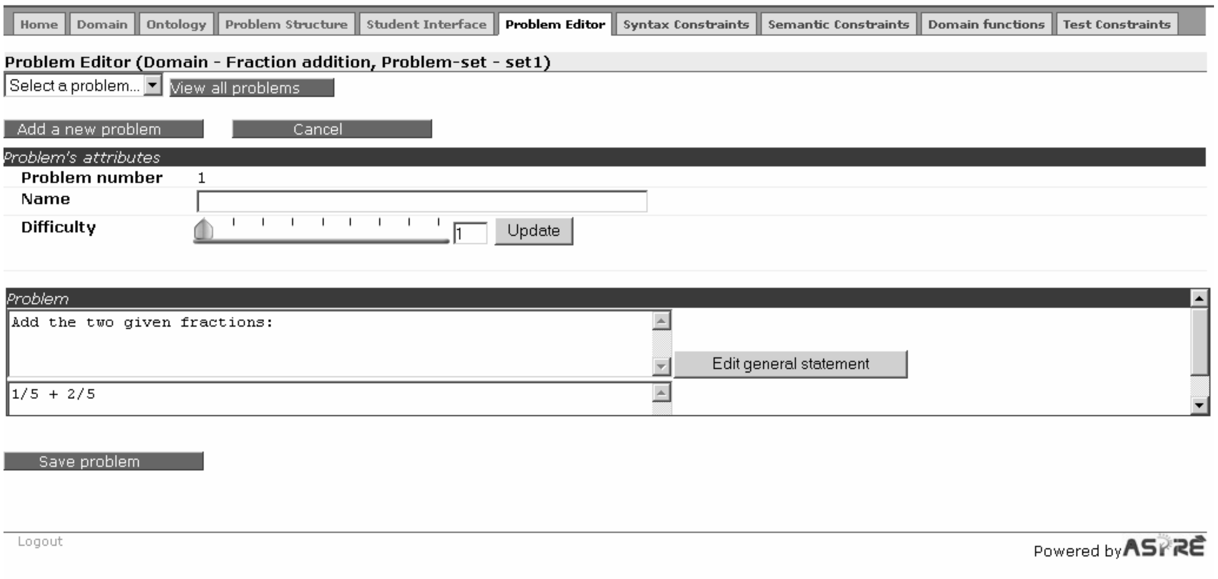
Problem number	1
Name	
Difficulty	1

Problem

Fixed problem statement...

Figure 35. Adding a problem statement

Next, it is necessary to add the problem itself. Figure 36 shows the author adding a problem to the *Fractions* domain.



Problem Editor (Domain - Fraction addition, Problem-set - set1)

Select a problem... | View all problems

Add a new problem | Cancel

Problem's attributes

Problem number	1
Name	
Difficulty	1

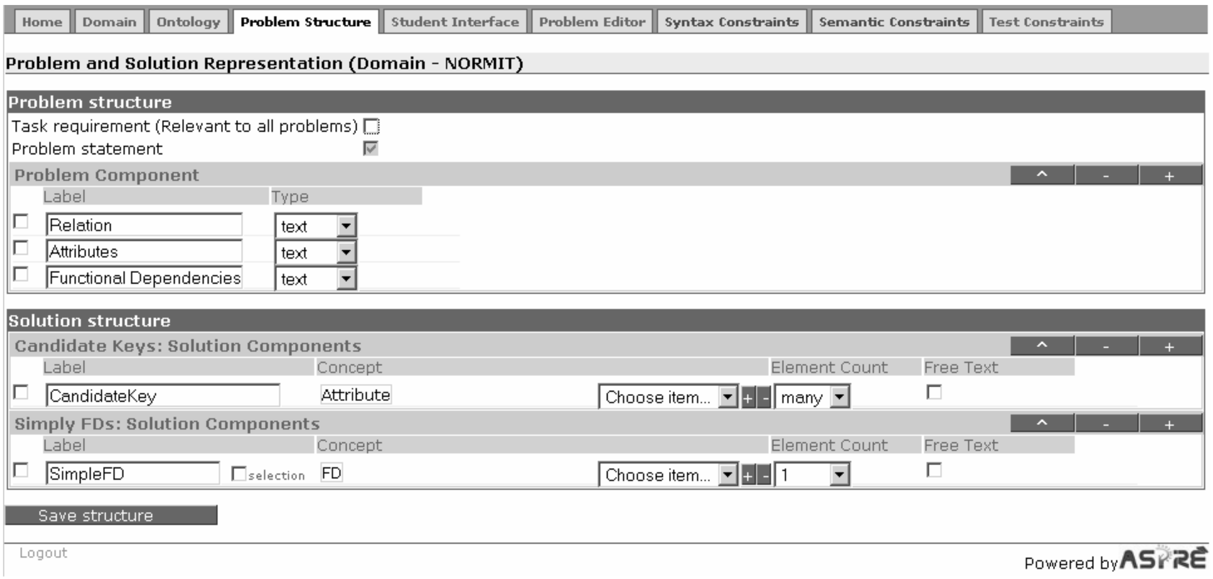
Problem

```
Add the two given fractions:
```

```
1/5 + 2/5
```

Figure 36. Adding a fraction addition problem

After specifying the problem text, the author also needs to specify the problem-specific components, if any exist. In the examples used so far in this manual, problems contained no components. Therefore, we now introduce an example instructional domain in which problems contain components. Figure 37 shows the screenshot of the Problem Structure Editor, showing the author defining the problem structure for the NORMIT domain. In this domain, students learn how to normalize relations. Each problem contains the task requirement, and a set of three components: the relation name, a list of attributes belonging to a relation, and a set of functional dependencies that exist in that relation.



Problem and Solution Representation (Domain - NORMIT)

Problem structure

Task requirement (Relevant to all problems) ☐
Problem statement ✓

Problem Component

Label	Type
Relation	text
Attributes	text
Functional Dependencies	text

Solution structure

Candidate Keys: Solution Components

Label	Concept		Element Count	Free Text
CandidateKey	Attribute	Choose item...	many	☐

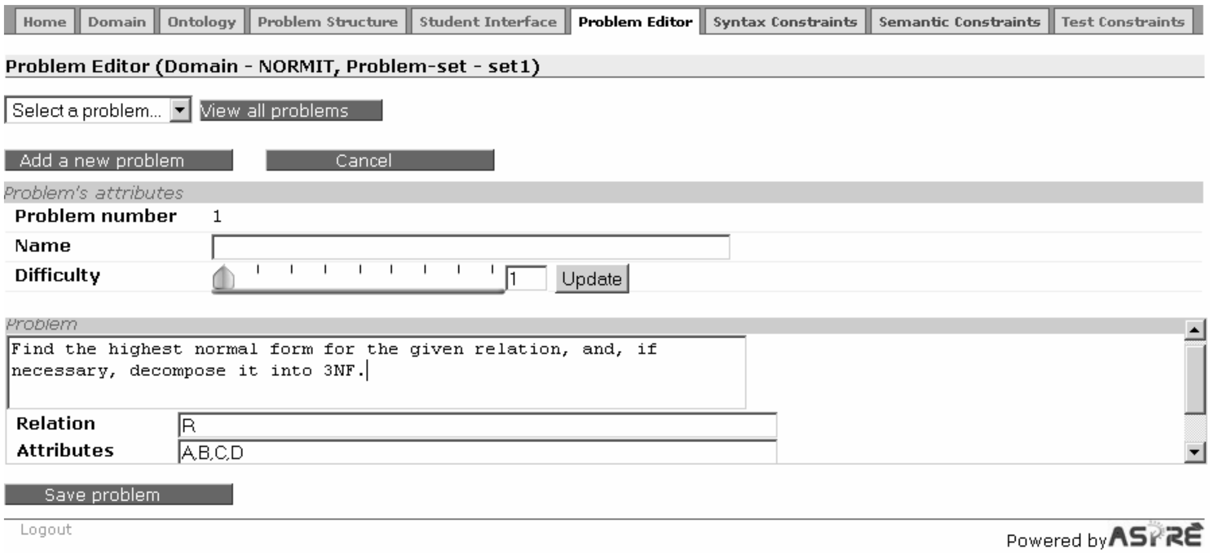
Simply FDs: Solution Components

Label	Concept		Element Count	Free Text
SimpleFD ☐selection	FD	Choose item...	1	☐

Figure 37. Specifying the structure of problems with three components

When the author starts adding problems for this domain, he/she will need to specify the mandatory elements of problems (problem name, difficulty and problem statement), and then additionally specify the problem-specific components, as illustrated in Figure 38. The author is able to type in the textual components in the text boxes provided. There are no graphical components in this instructional domain. To add a graphical component, the author needs to browse for or type in the image's URL and then press the *Upload* button, which will upload the image to the server to be stored along with the other specifications in the domain model.

Finally, it is necessary to click the *Save* button to store the problem.



Problem Editor (Domain - NORMIT, Problem-set - set1)

Problem's attributes

Problem number	1
Name	
Difficulty	1

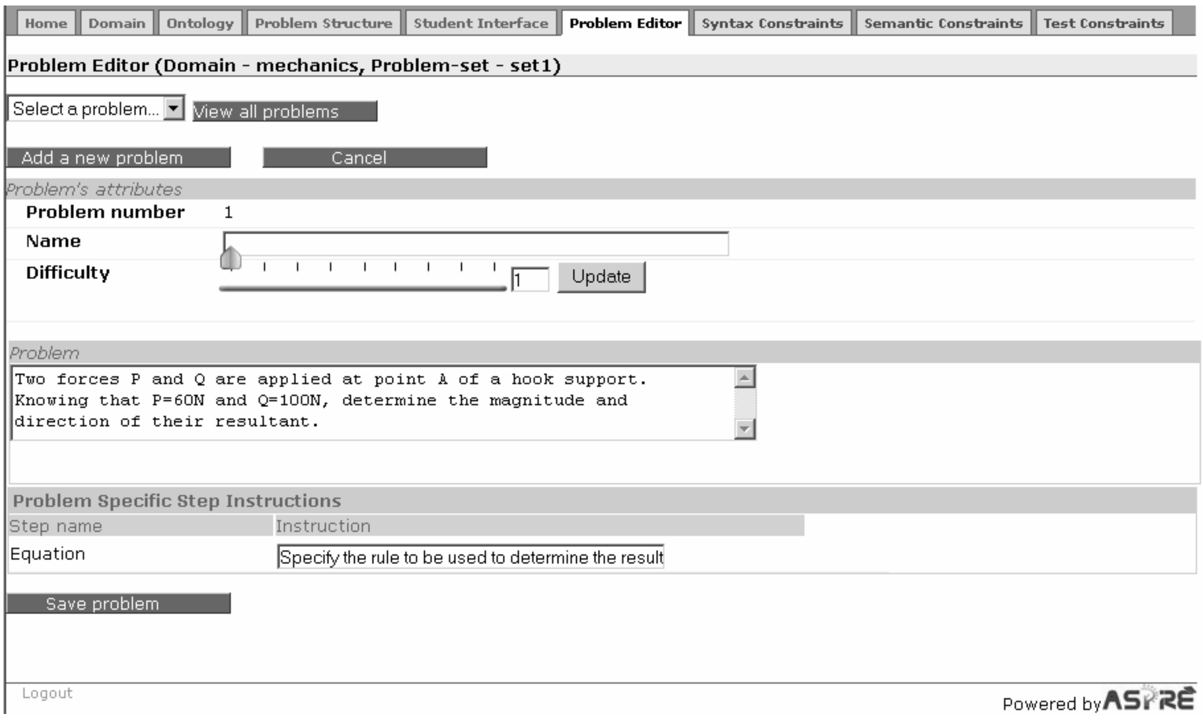
Problem

Find the highest normal form for the given relation, and, if necessary, decompose it into 3NF.

Relation	R
Attributes	A,B,C,D

Figure 38. Adding problem-specific components

As discussed in Section 3.1, in some instructional domains it might be necessary to specify problem-specific instructions for any problem-solving steps. When specifying the domain structure, the author may tick the *Problem Specific Instruction* box for the appropriate steps (see Figure 5). As the result of this action, the author would be asked to specify the problem-specific instructions for the selected steps, as illustrated in Figure 39. For the domain illustrated in this figure, the author requested problem-specific instruction for the *Equation* step. The author can add those instructions in the text box provided. The instruction will be given to the student when working on the appropriate step.



Problem Editor (Domain - mechanics, Problem-set - set1)

Problem's attributes

Problem number	1
Name	
Difficulty	1

Problem

Two forces P and Q are applied at point A of a hook support. Knowing that P=60N and Q=100N, determine the magnitude and direction of their resultant.

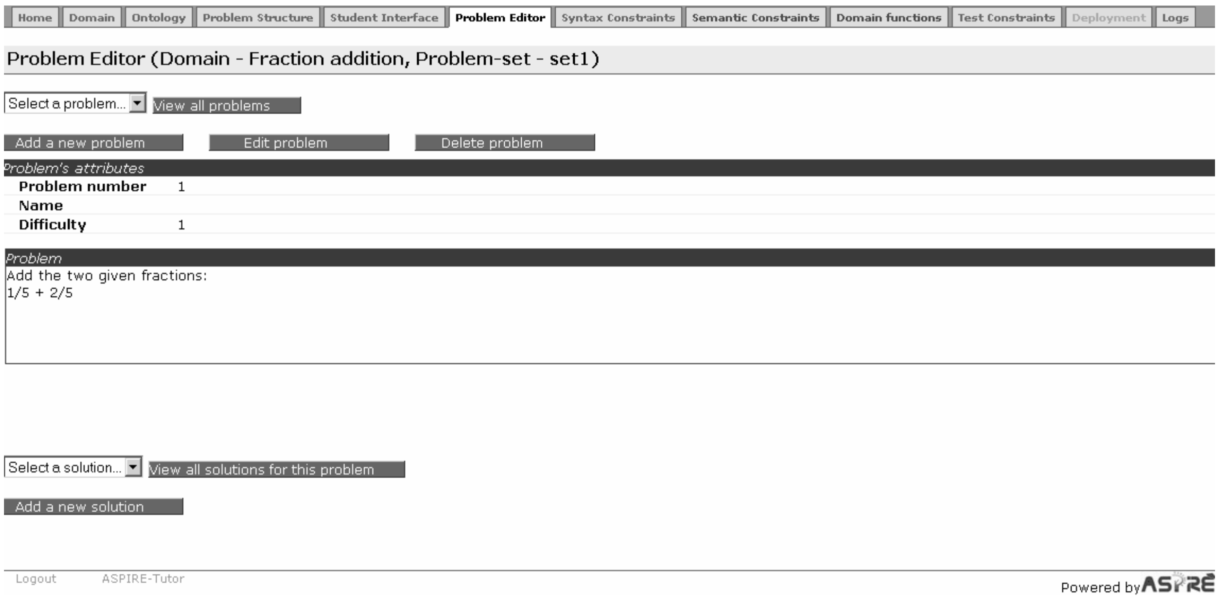
Problem Specific Step Instructions

Step name	Instruction
Equation	Specify the rule to be used to determine the result

Figure 39. Adding problem-specific instructions for a step

3.5.3. Adding solutions

After saving the problem, the author can add one or more solutions for it. Figure 40 shows the state of the interface after the problem has been saved. Similar to adding and selecting problems, the author can either ask for a new solution to be added, or select one of the previously specified solutions to modify. Solutions can be selected from the drop-down menu, to either view or modify.



Problem Editor (Domain - Fraction addition, Problem-set - set1)

Problem's attributes

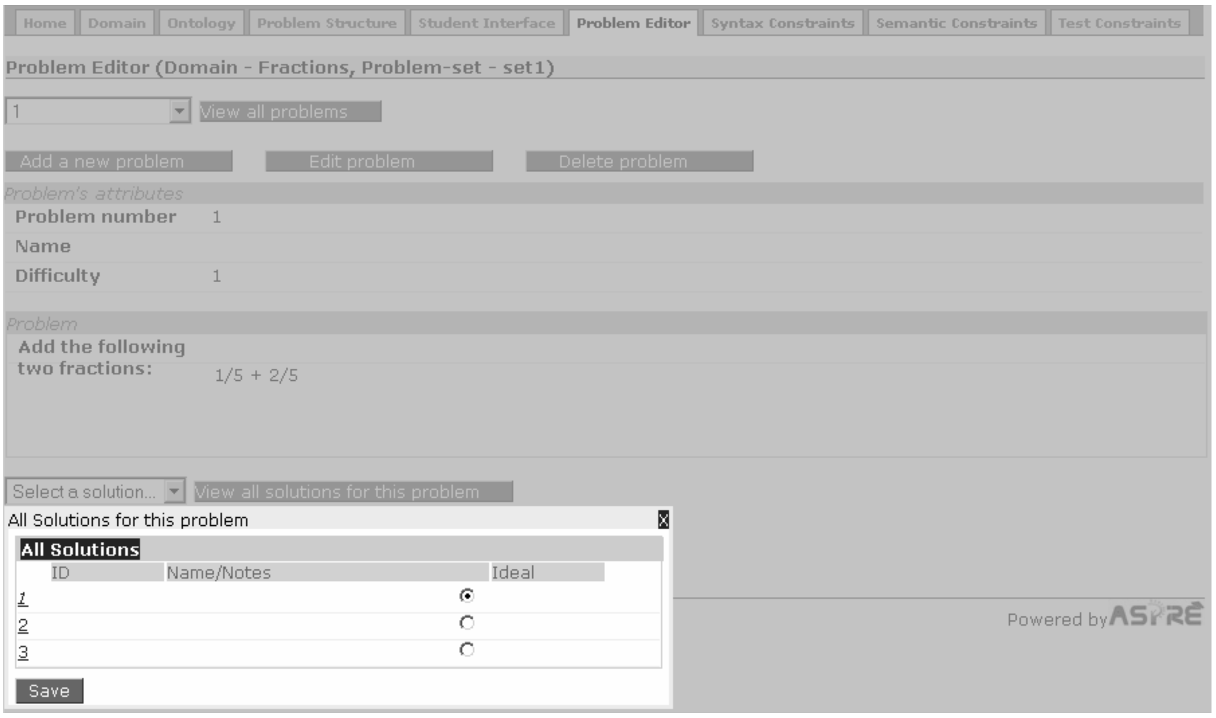
Problem number	1
Name	
Difficulty	1

Problem

Add the two given fractions:
1/5 + 2/5

Figure 40. Adding a solution

The *View all solutions for this problem* button will show a pop-up window with all the solutions. In the pop-up window, the author is able to nominate one solution to be the ideal solution (i.e. the preferred solution for the problem). The ideal solution is used by ASPIRE-Tutor when students request to see the *Full Solution* of a problem. By default, the first entered solution is the ideal solution. Figure 41 shows a situation when three solutions have been added for a problem, and the author can nominate one of them to be the ideal solution. (Please note that in the fractions domain there is only one ideal solution per problem. The goal of Figure 41 is to show how the ideal solution can be specified).



Problem Editor (Domain - Fractions, Problem-set - set1)

Problem's attributes

Problem number	1
Name	
Difficulty	1

Problem

Add the following two fractions: 1/5 + 2/5

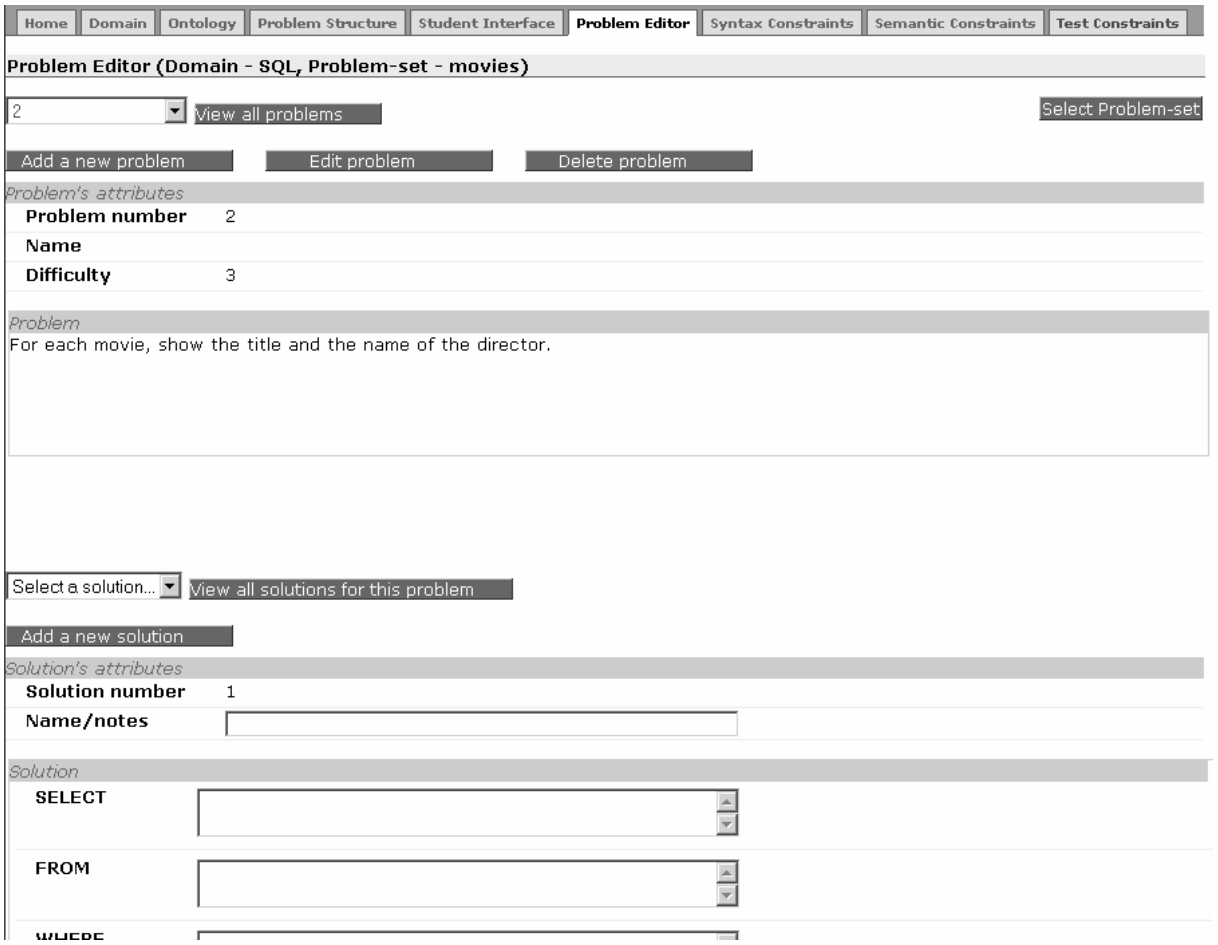
All Solutions for this problem

All Solutions

ID	Name/Notes	Ideal
1		●
2		○
3		○

Figure 41. Specifying the ideal solution

When the author clicks the *Add a new solution* button, the Problem Editor displays the interface for entering a new solution, as illustrated in Figure 42. For procedural domains, when there are multiple steps for solving a problem, the solution workspace allows the author to enter all the steps simultaneously, as opposed to navigating through the steps one at a time as the students would. This eliminates the navigation efforts needed between steps, making it easier for the author to add and inspect the full solution for a problem. Each step is displayed along with its name and description that the students would see, and are separated by borders to make a clear distinction between steps. The author needs to specify the solution components for each problem solving step. Once the author is satisfied with the solution, it can be saved by clicking on the *Save solution* button.



Problem Editor (Domain - SQL, Problem-set - movies)

2

Problem's attributes

Problem number	2
Name	
Difficulty	3

Problem

For each movie, show the title and the name of the director.

Solution's attributes

Solution number	1
Name/notes	

Solution

SELECT

FROM

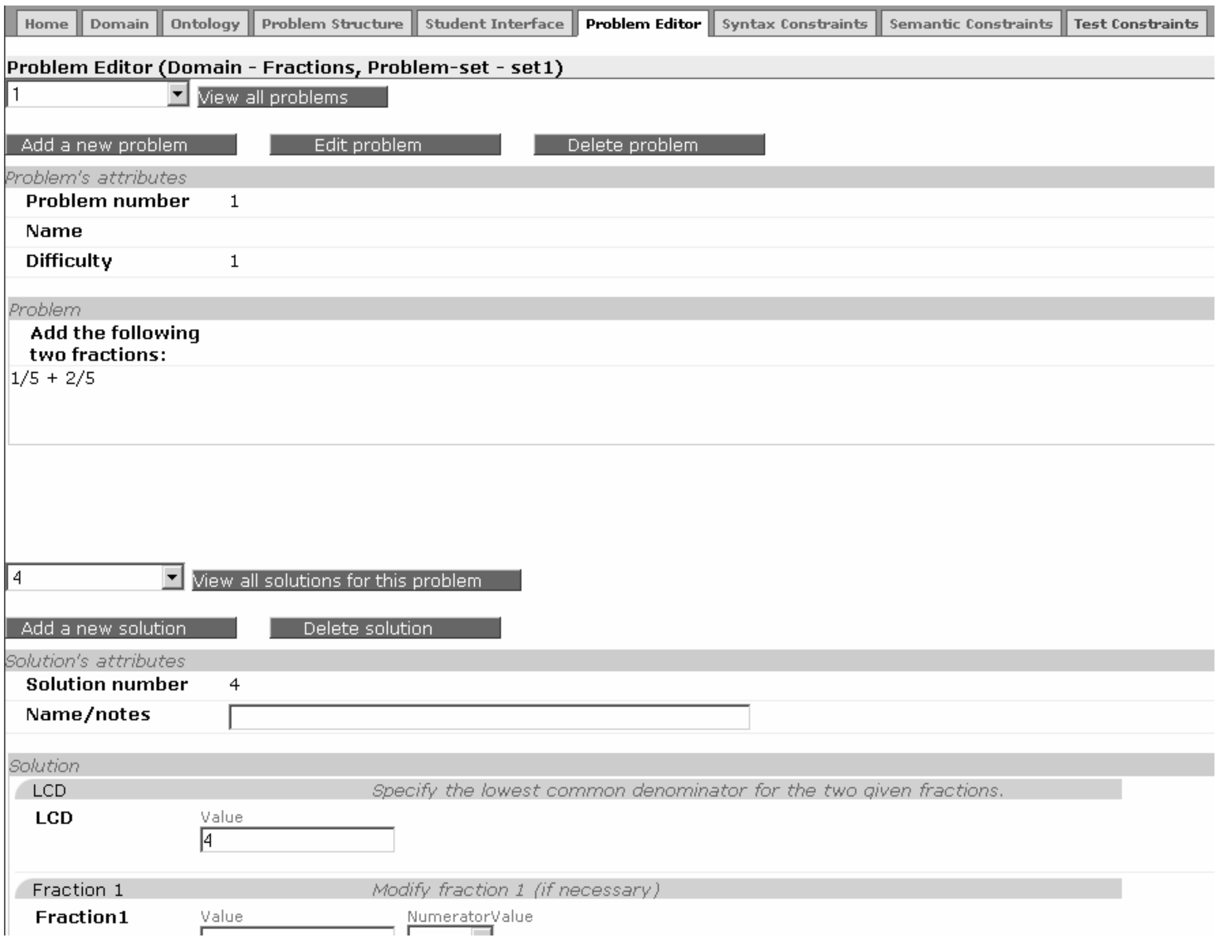
WHERE

Figure 42. Entering a solution to problem 2

The author may choose to enter problems first and then add their solutions at a later time. The Problem Editor shows the problem number in red if there are no solutions specified for that problem yet.

As shown in Figures 40-42, the author is also able to edit or delete any problem previously entered by clicking on the *Edit* or *Delete* buttons respectively. Deleting a problem, however, will also delete all of its solutions at the same time.

Solutions can also be deleted independently of the problem. Figure 43 shows a situation when the author has selected solution 4 for problem 1. To delete only this solution and keep the other solutions for the same problem, the author clicks the *Delete solution* button.


Home | Domain | Ontology | Problem Structure | Student Interface | Problem Editor | Syntax Constraints | Semantic Constraints | Test Constraints

Problem Editor (Domain - Fractions, Problem-set - set1)

1 | View all problems

Add a new problem | Edit problem | Delete problem

Problem's attributes

Problem number	1
Name	
Difficulty	1

Problem

Add the following two fractions:

1/5 + 2/5

4 | View all solutions for this problem

Add a new solution | Delete solution

Solution's attributes

Solution number	4
Name/notes	

Solution

LCD — *Specify the lowest common denominator for the two given fractions.*

LCD Value 4

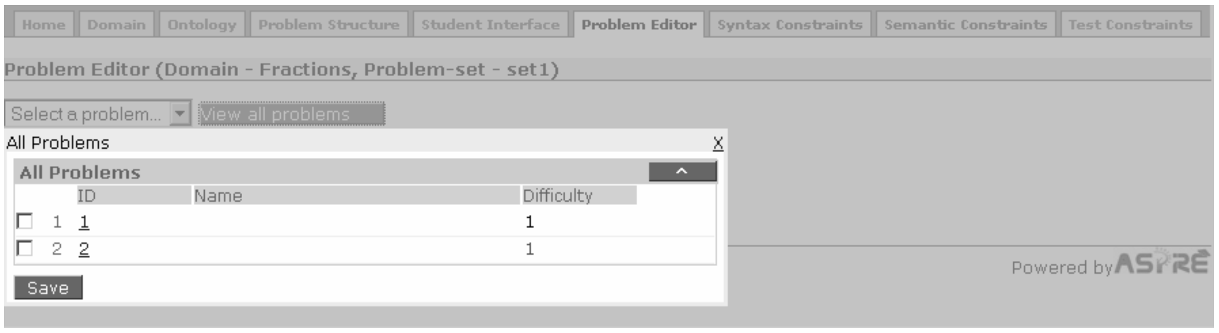
Fraction 1 — *Modify fraction 1 (if necessary)*

Fraction1 Value NumeratorValue

Figure 43. Deleting a solution

3.5.4. Re-arranging problems

The author can see all the problems in the current problem set by clicking the *View all problems* button, which opens a new window, as shown in Figure 44. As the results, the new pop-up window shows the ids, names and difficulty levels of all problems. Clicking on the problem id will display the problem in the main window along with the options to edit or delete the selected problem.


Home | Domain | Ontology | Problem Structure | Student Interface | Problem Editor | Syntax Constraints | Semantic Constraints | Test Constraints

Problem Editor (Domain - Fractions, Problem-set - set1)

Select a problem... | View all problems

All Problems X

All Problems

	ID	Name	Difficulty
☐ 1	1		1
☐ 2	2		1

Save

Powered by ASPIRE

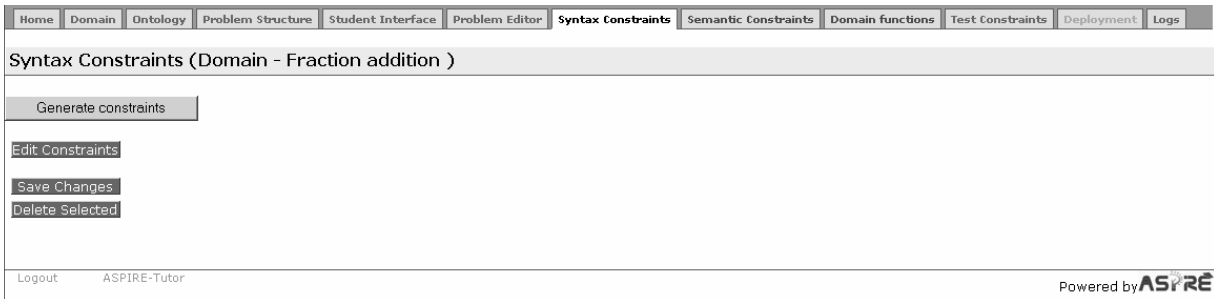
Figure 44. Viewing problems

The problems will be shown initially in the order in which they have been specified. However, the author can modify the order of problems to be presented to students. To change the order of problems, the author needs to select the problems to move upwards by ticking the boxes at the beginning of corresponding rows, and then click the ^ button as many times as necessary.

3.6. Generating syntactic constraints

Once when the author finishes entering problems and solutions, constraints can be generated. Syntactic constraints check the student's solution for syntax errors. These constraints are generated automatically by ASPIRE-Author from the domain ontology. All restrictions specified on domain concepts and their slots (such as minimal/maximal values, types of slots and restrictions on relationships) are translated into syntax constraints. Additionally, for procedural domains ASPIRE-Author also defines constraints that make sure that the student has completed all previous problem steps before attempting the current step. We do not provide explanation of how syntactic constraints are generated here - the interested reader is referred to [Suraweera, Mitrovic & Martin, 2005; Mitrovic et al., 2006, Suraweera, Mitrovic & Martin, 2007]. It is sufficient to know that if the student's solution violates a constraint, the intelligent tutoring system will inform the students that there is an error in their solution. In that case, the student might also be given some feedback, corresponding to a message that is attached to the constraint.

The *Syntax constraints* tab allows the author to ask for syntactic constraints to be generated. At the top of the page (Figure 45) there is a *Generate constraints* button. When the author clicks this button, ASPIRE-Author will analyze the domain ontology and generate the corresponding syntax constraints. To save the generated constraints, click the *Save* button at the bottom of the page.



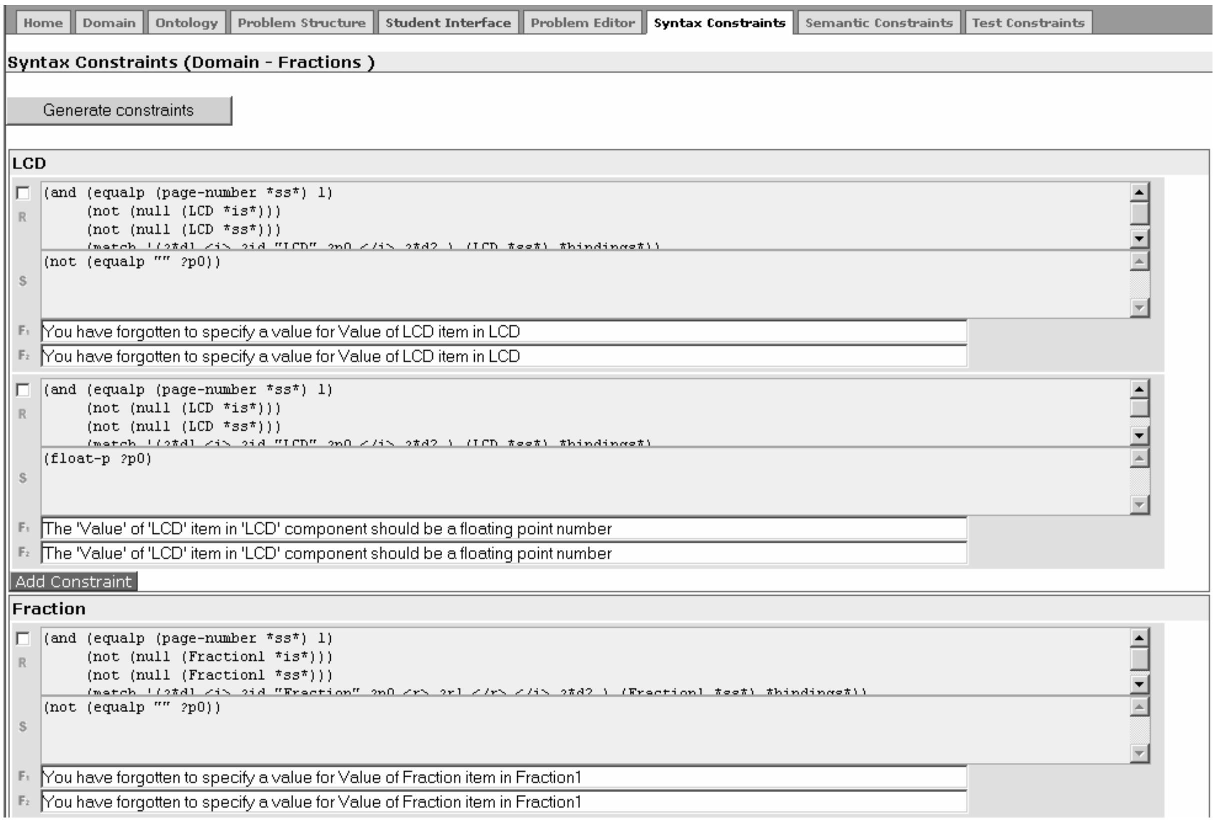
Syntax Constraints (Domain - Fraction addition)

Figure 45. The initial state of the Syntax Constraint page

Figure 46 illustrates syntactic constraints generated for the fractions domain. The constraints are arranged in groups corresponding to domain concepts they have been generated from. The figure shows two concepts: LCD and Fraction. LCD (the lowest common denominator) is shown first, as this is the first step the student has to specify the answer to. There are two constraints generated for this concept. Each constraint consists of two conditions (relevance and satisfaction condition) followed by two feedback messages. The two feedback messages will be shown to the student one at a time when the constraint is violated, and the student asks for more feedback. These feedback messages are automatically generated by ASPIRE; however, the author can modify them to make them more useful for the student.

The tick box at the beginning of each constraint allows the constraint to be selected. In order to delete one or more constraints, they need to be selected first, and then the *Delete* button at the bottom of the page needs to be clicked.

At the end of a group of constraints for each concept from the ontology, there is the *Add Constraint* button. This constraint allows developers to add new constraints, which are manually defined. Authors are not allowed to add/modify constraints.



Syntax Constraints (Domain - Fractions)

LCD

```
(and (equalp (page-number *ss*) 1)
     (not (null (LCD *is*)))
     (not (null (LCD *ss*)))
     (match '(?*d1 <i> ?id "LCD" ?p0 </i> ?*d2 ) (LCD *ss*) *bindings*))
```

```
(not (equalp "" ?p0))
```

F1: You have forgotten to specify a value for Value of LCD item in LCD

F2: You have forgotten to specify a value for Value of LCD item in LCD

```
(and (equalp (page-number *ss*) 1)
     (not (null (LCD *is*)))
     (not (null (LCD *ss*)))
     (match '(?*d1 <i> ?id "LCD" ?p0 </i> ?*d2 ) (LCD *ss*) *bindings*)
```

```
(float-p ?p0)
```

F1: The 'Value' of 'LCD' item in 'LCD' component should be a floating point number

F2: The 'Value' of 'LCD' item in 'LCD' component should be a floating point number

Fraction

```
(and (equalp (page-number *ss*) 1)
     (not (null (Fraction1 *is*)))
     (not (null (Fraction1 *ss*)))
     (match '(?*d1 <i> ?id "Fraction" ?p0 <r> ?r1 </r> </i> ?*d2 ) (Fraction1 *ss*) *bindings*))
```

```
(not (equalp "" ?p0))
```

F1: You have forgotten to specify a value for Value of Fraction item in Fraction1

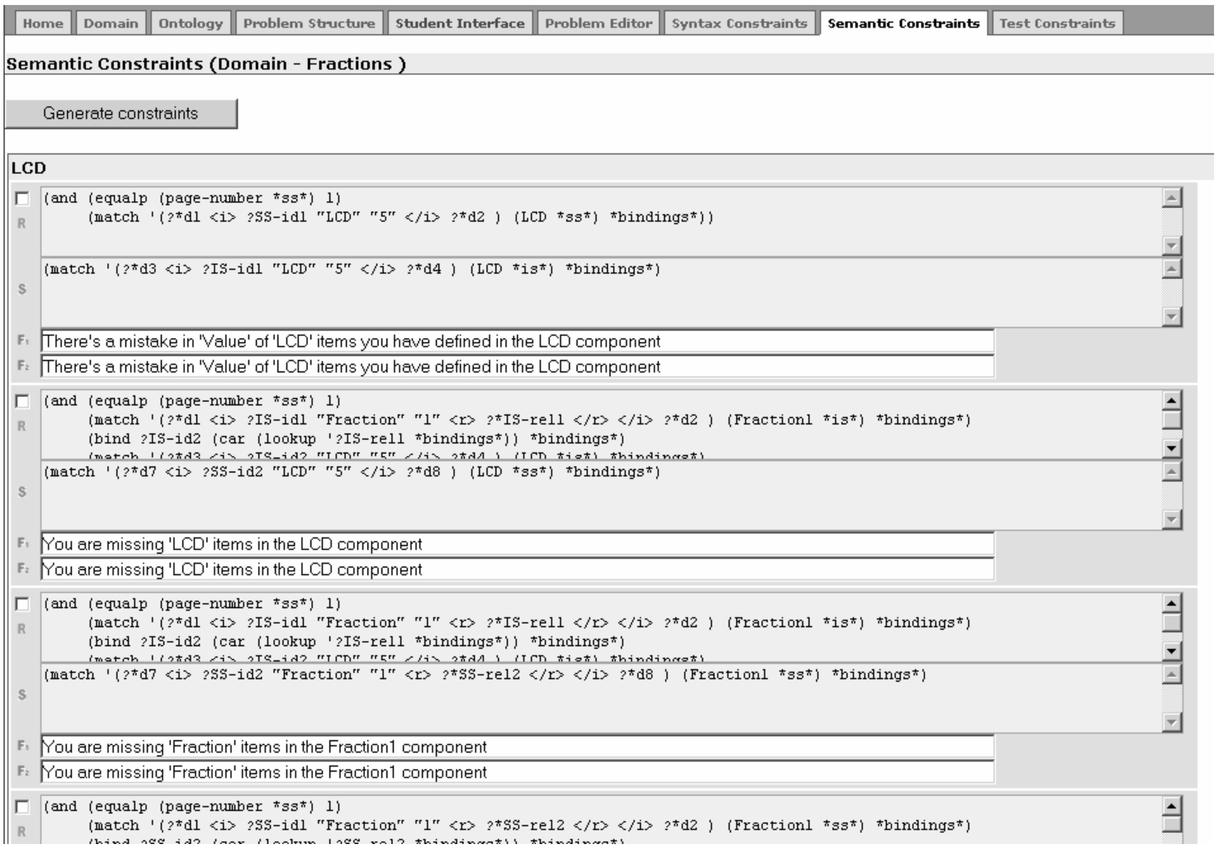
F2: You have forgotten to specify a value for Value of Fraction item in Fraction1

Figure 46. Generated syntactic constraints

3.7. Generating semantic constraints

Semantic constraints are generated on the basis of problems and their solutions that the author has specified. Please note that for valid semantic constraints to be generated, ASPIRE needs multiple problems and their solutions. Also, note that in domains where multiple solutions exist, they need to be provided by the author, to guarantee a good domain model. In contrast to syntactic constraints, these constraints do not check for syntax errors. Semantic constraints look for semantic errors in the student solution - these are errors that are specific to the problem the student is attempting to solve. We do not provide explanation of how semantic constraints are generated here - the interested reader is referred to [Suraweera, Mitrovic & Martin, 2005; Mitrovic et al., 2006, Suraweera, Mitrovic & Martin, 2007].

The *Semantic constraints* tab allows the author to ask for semantic constraints to be generated. This tab is only available when the author has specified some problems and their solutions. At the top of the page (Figure 47) there is the *Generate constraints* button. When the author clicks this button, ASPIRE-Author will analyze the specified problems and their solutions, and generate the corresponding semantic constraints. To save the generated constraints, click the *Save* button at the bottom of the page.



Semantic Constraints (Domain - Fractions)

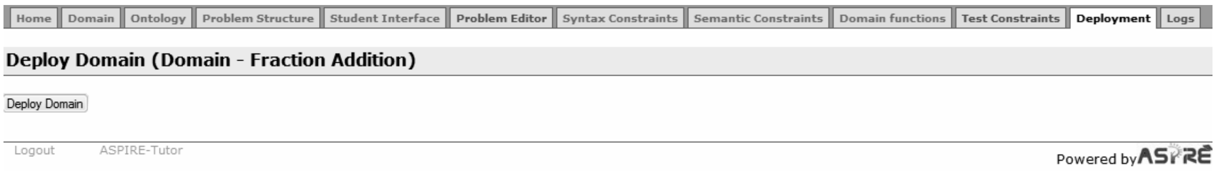
LCD

```
R  (and (equalp (page-number *ss*) 1)
        (match '(?*d1 <i> ?SS-id1 "LCD" "5" </i> ?*d2 ) (LCD *ss*) *bindings*))
S  (match '(?*d3 <i> ?IS-id1 "LCD" "5" </i> ?*d4 ) (LCD *is*) *bindings*)
F1 There's a mistake in 'Value' of 'LCD' items you have defined in the LCD component
F2 There's a mistake in 'Value' of 'LCD' items you have defined in the LCD component

R  (and (equalp (page-number *ss*) 1)
        (match '(?*d1 <i> ?IS-id1 "Fraction" "1" <r> ?*IS-rel1 </r> </i> ?*d2 ) (Fraction1 *is*) *bindings*)
        (bind ?IS-id2 (car (lookup '?IS-rel1 *bindings*)) *bindings*)
        (match '(?*d3 <i> ?IS-id2 "LCD" "5" </i> ?*d4 ) (LCD *is*) *bindings*)
S  (match '(?*d7 <i> ?SS-id2 "LCD" "5" </i> ?*d8 ) (LCD *ss*) *bindings*)
F1 You are missing 'LCD' items in the LCD component
F2 You are missing 'LCD' items in the LCD component

R  (and (equalp (page-number *ss*) 1)
        (match '(?*d1 <i> ?IS-id1 "Fraction" "1" <r> ?*IS-rel1 </r> </i> ?*d2 ) (Fraction1 *is*) *bindings*)
        (bind ?IS-id2 (car (lookup '?IS-rel1 *bindings*)) *bindings*)
        (match '(?*d3 <i> ?IS-id2 "LCD" "5" </i> ?*d4 ) (LCD *is*) *bindings*)
S  (match '(?*d7 <i> ?SS-id2 "Fraction" "1" <r> ?*SS-rel2 </r> </i> ?*d8 ) (Fraction1 *ss*) *bindings*)
F1 You are missing 'Fraction' items in the Fraction1 component
F2 You are missing 'Fraction' items in the Fraction1 component

R  (and (equalp (page-number *ss*) 1)
        (match '(?*d1 <i> ?SS-id1 "Fraction" "1" <r> ?*SS-rel2 </r> </i> ?*d2 ) (Fraction1 *ss*) *bindings*)
        (bind ?SS-id2 (car (lookup '?SS-rel2 *bindings*)) *bindings*)
```

Figure 47. Generating semantic constraints

3.8. Deploying the tutoring systems

Once the author has completed all the authoring steps, he/she may want to see the tutoring system running. This allows the author to interact with the tutoring system, solving problems and receiving feedback in a manner similar to students. The task of starting a tutoring system (to run on ASPIRE-TUTOR) is called *deployment*.

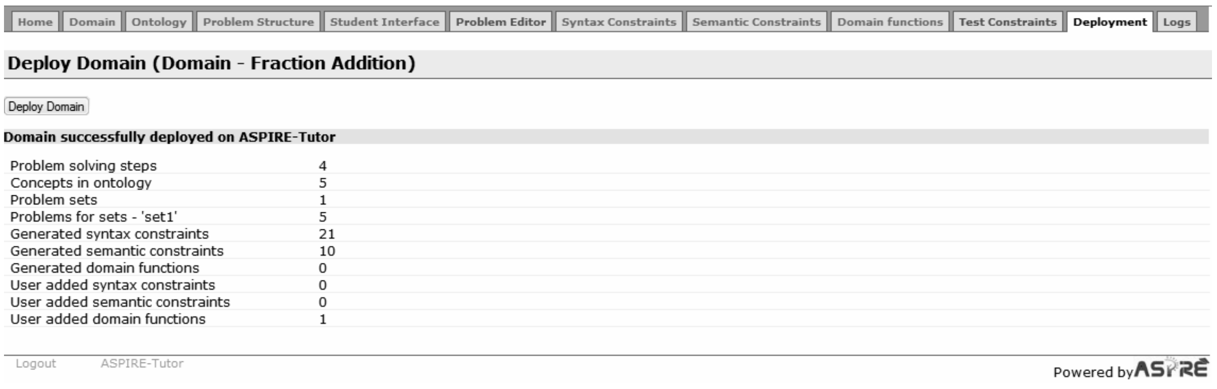


Deploy Domain (Domain - Fraction Addition)

Figure 48. Deploying a domain

Clicking the Deployment tab will initiate a number of checks on the domain to test whether the information supplied by the author is consistent and whether the domain model is complete. Figure 48 shows a screenshot of the deployment page of a domain where ASPIRE has not found any inconsistencies. In such cases, the author can simply click on the Deploy Domain button. After clicking this button, a success message is displayed if the domain was deployed successfully (Figure 49). A summary of the deployed domain is also shown.

The author can then try the tutoring system on ASPIRE-Tutor. Click the ASPIRE-Tutor link at the bottom of the page, which will take you to ASPIRE-Tutor. The new tutoring system will be listed on the My Domains tab.



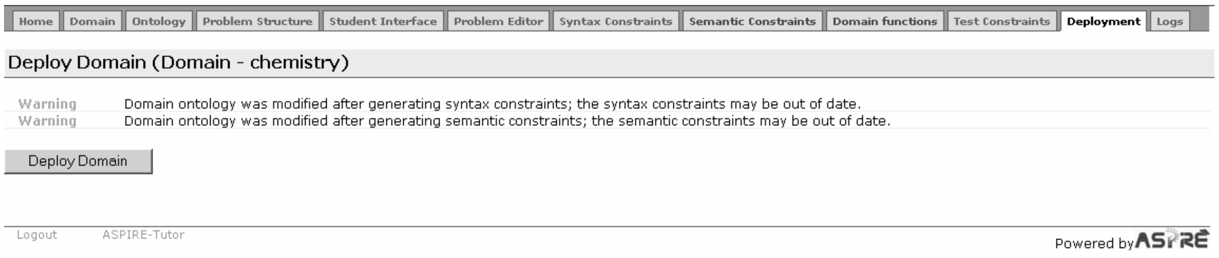
Deploy Domain (Domain - Fraction Addition)

Domain successfully deployed on ASPIRE-Tutor

Problem solving steps	4
Concepts in ontology	5
Problem sets	1
Problems for sets - 'set1'	5
Generated syntax constraints	21
Generated semantic constraints	10
Generated domain functions	0
User added syntax constraints	0
User added semantic constraints	0
User added domain functions	1

Figure 49. Successful deployment of a domain

Figure 50 shows an example of a domain where ASPIRE is warning the author that the domain ontology was modified after the constraints were generated. This indicates that the author has modified the ontology after generating constraints. The warnings should only be used as a guide. Therefore, the author has to decide whether to deploy the domain with the warnings or re-generate the constraints.

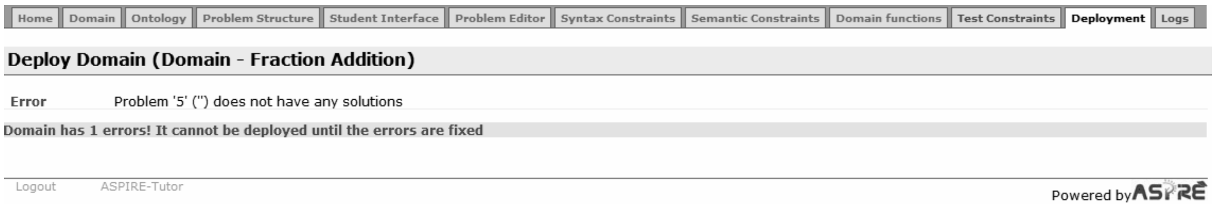


Deploy Domain (Domain - chemistry)

Warning	Domain ontology was modified after generating syntax constraints; the syntax constraints may be out of date.
Warning	Domain ontology was modified after generating semantic constraints; the semantic constraints may be out of date.

Figure 50. The Deployment page with warnings

During the domain testing procedure ASPIRE may identify errors in the domain that would result in an incomplete tutoring system. ASPIRE does not allow deployment of domains with errors. In such cases, ASPIRE will display an error message that says the domain cannot be deployed. Figure 51 contains an example of a domain that contains a problem with no solutions. ASPIRE does not allow such domains to be deployed.



Deploy Domain (Domain - Fraction Addition)

Error Problem '5' ('') does not have any solutions

Domain has 1 errors! It cannot be deployed until the errors are fixed

Figure 51. Domain with errors

3.9. Additional Pages for Developers

Several of the tabs in ASPIRE-Author (*Domain Functions*, *Test constraints* and *Logs*) are only available to developers. We do not describe them in this manual. Information applicable only to developers is available in a separate document[3] from the ICTG group. See Section 4.5 for a discussion of the developer user type in ASPIRE.

4. ASPIRE-Tutor

As explained previously (see Figure 1), ASPIRE-Tutor is the tutoring server. It delivers all the tutoring systems developed in ASPIRE-Author to users. The architecture of ASPIRE-Tutor is illustrated in Figure 52. ASPIRE-Tutor consists of a set of modules, with each module having specific responsibilities in the serving of intelligent tutoring systems. This document provides a very abstract discussion of the functionality provided by ASPIRE-Tutor as the understanding of internal operations is not required for authoring new systems in ASPIRE. For detailed discussion on the design and functionality of ASPIRE, please see [Mitrovic et al., 2006].

The student accesses an intelligent tutoring system served by ASPIRE through a Web browser. Every action performed by the student would be sent to the Session Manager, which passes the appropriate requests to the Pedagogical Module. The Session Manager thus manages the flow of control of the interaction.

The Pedagogical Module decides what actions to take to fulfill the request, and does so by sending appropriate requests to the other modules, i.e. any/all of the Diagnostic Module, Domain Manager, Student Modeller, Log Manager and User Manager. The Pedagogical Module thus manages the pedagogical decisions that determine what the response to each request will be.

Each request to a module results in a status and optional data being returned to the Pedagogical Module. In addition, the functional modules may access and/or update data objects, e.g. student model, domain model, logs, which are stored in the Allegro Cache database. The various components of the model may also be updated. The Pedagogical Module returns the final status and data to the Session Manager. The Session Manager organises the result to be returned to the interface, by packaging up a response and/or indicating what interface object should be presented next.

The Diagnostic Module analyses students' solutions, and identifies any mistakes students made. In order to be able to perform this task, the Diagnostic Module needs the services of the Domain Manager, the component that is in charge of all knowledge bases (shown as *domain models* in Figure 52) developed for various intelligent tutoring systems. On the basis of the diagnosis performed by the Diagnostic Module, the Student Modeller updates the student model, i.e. the system's view of the student's knowledge. The student model is used to adapt instructional actions to meet the needs and abilities of each individual student.

All actions students perform in ASPIRE are logged, and the Log Manager is responsible for maintaining the logs. Finally the User Manager is the component which maintains user information, and makes sure that only authorized people can access ASPIRE and various intelligent tutoring systems defined within it. There are several types of users in ASPIRE: students, teachers, administrators, developers and authors. Each group of users has specific privileges and rights in the

[3] Please email Tanja.mitrovic@canterbury.ac.nz in order to get developer's documentation

system, and can access different parts of the system. User Manager makes sure that users can access the part of ASPIRE they need.

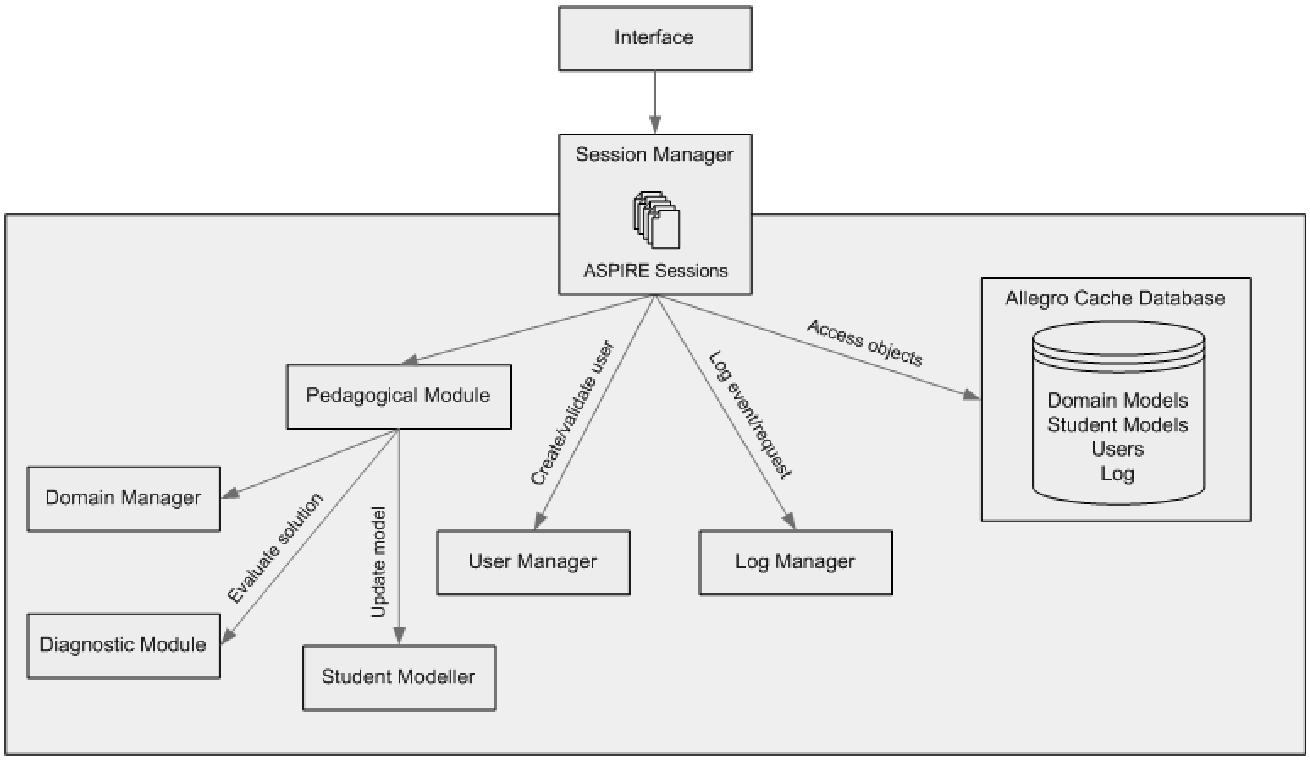


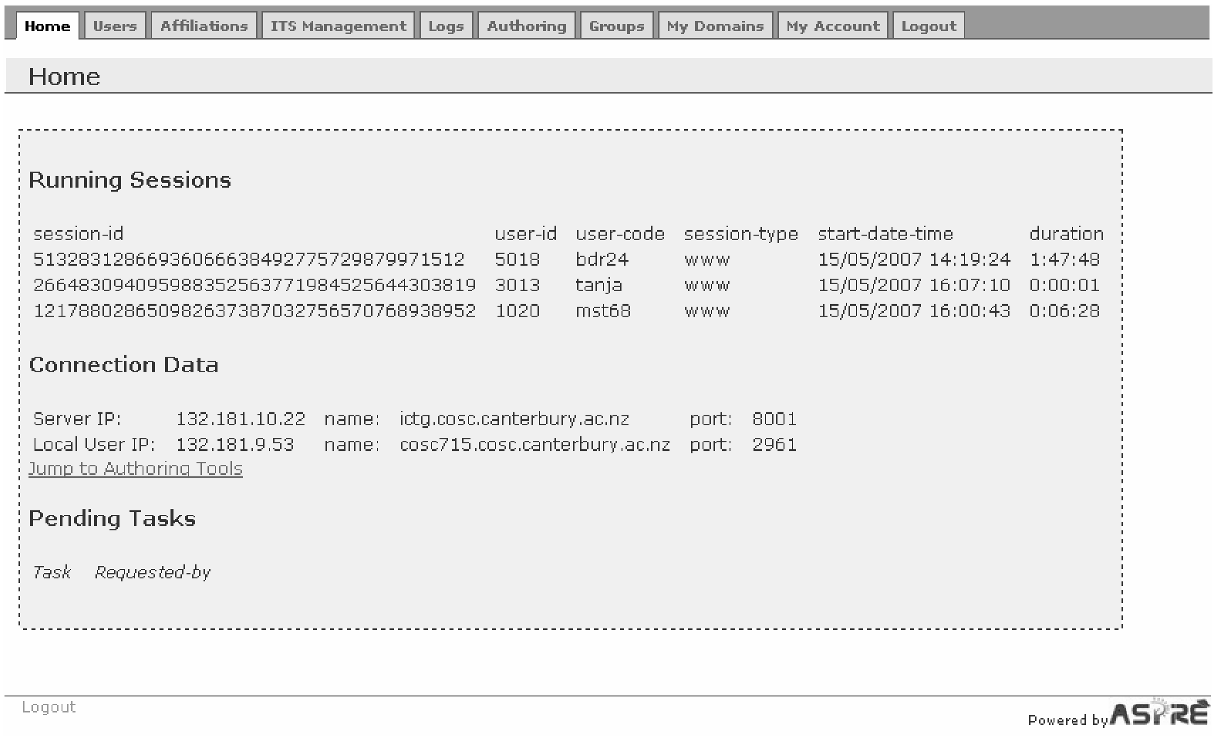
Figure 52. The architecture of ASPIRE-Tutor

4.1. Logging in to ASPIRE-Tutor

As discussed previously (see Section 2), to be able to log into ASPIRE, you need to have a valid user account. Once when the user specifies all the necessary information, the home page will be shown. There are various types of users in ASPIRE, and the home page displayed after logging in will depend on the type of the account used. The role of authors has already been discussed in detail in Section 3 of this document. The following sections explain the roles and functionality available to administrators, teachers and students.

4.2. Administrator

An administrator is a person who is responsible for maintaining ASPIRE, controlling the deployment of new tutoring systems, and maintaining user accounts. Once when an administrator logs in, he/she will be taken to the administrator home page, shown in Figure 53. The home page provides information about all sessions currently running in ASPIRE-Tutor. Please note that Figure 53 shows three running sessions. For each session, the administrator will be shown the session id, the id and the user code of the corresponding user, the type of the session, starting time and duration. Information listed under *Connection data* includes the IP number of the server, and the IP number of the machine the administrator is using.



Home

Running Sessions

session-id	user-id	user-code	session-type	start-date-time	duration
51328312866936066638492775729879971512	5018	bdr24	www	15/05/2007 14:19:24	1:47:48
266483094095988352563771984525644303819	3013	tanja	www	15/05/2007 16:07:10	0:00:01
121788028650982637387032756570768938952	1020	mst68	www	15/05/2007 16:00:43	0:06:28

Connection Data

Server IP:	132.181.10.22	name:	ictg.cosc.canterbury.ac.nz	port:	8001
Local User IP:	132.181.9.53	name:	cosc715.cosc.canterbury.ac.nz	port:	2961

Jump to Authoring Tools

Pending Tasks

Task *Requested-by*

Logout

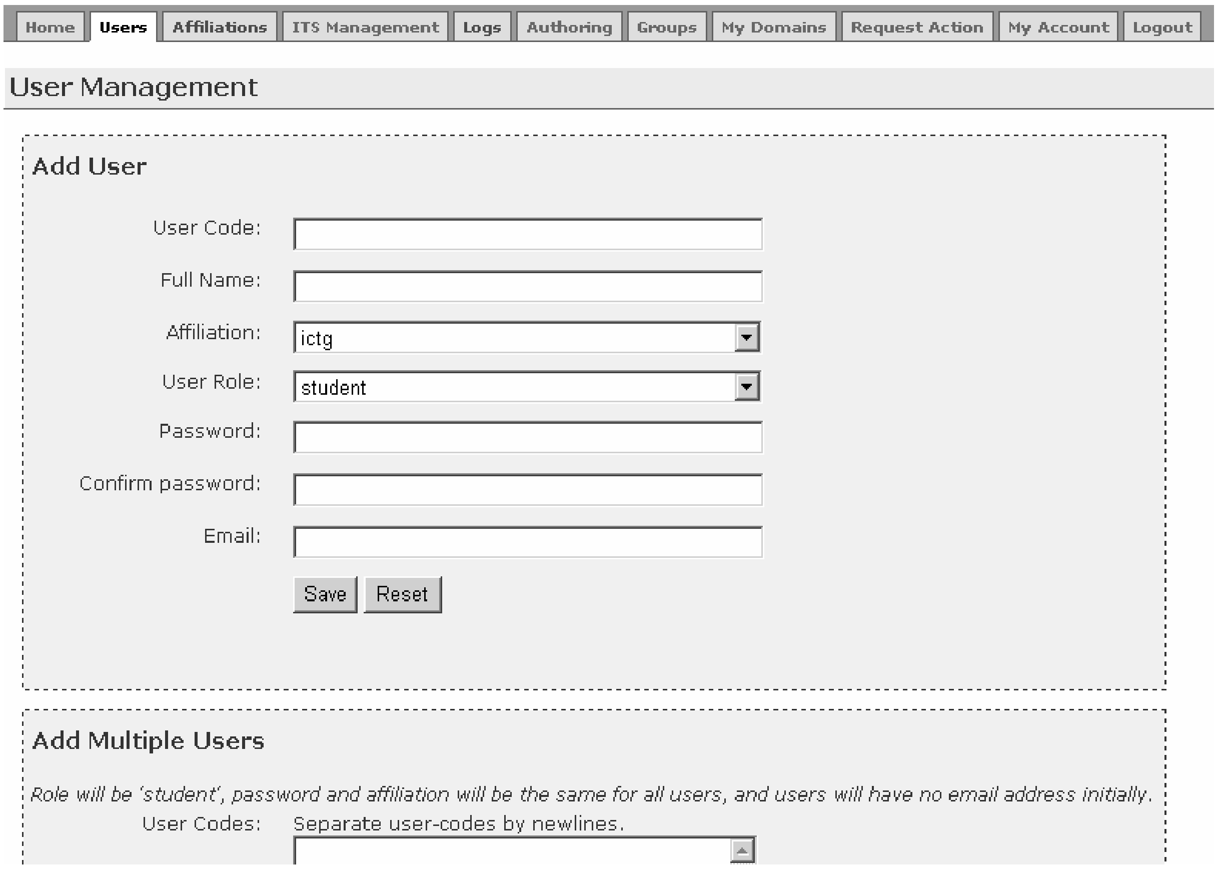
Powered by ASPIRE

Figure 53. The administrator home page

At the top of the home page, there is a set of tabs for various functions that the administrator can perform. The same tabs appear on all pages that are available to administrators. The *Home* tab brings the administrator back to this home page.

The *Users* tab takes us to the *User Management* page, which consists of three parts. The top part of the page (shown in Figure 54) allows the administrator to add new users (one at a time), by specifying the information about the user. The middle part (Figure 55) allows the administrator to add multiple users at once, so that all of them will have student accounts, and will have the same initial password and the same affiliation. The bottom part allows the administrator to retrieve information about existing accounts. The administrator can search for all accounts for a specific affiliation, or search for a specific account. Figure 56 shows that part of the page, after the administrator has searched for a specific user. The administrator can change the information about a particular user by clicking the *edit* link, or delete the user account. The page also shows the total number of accounts created in ASPIRE.

In order to create a new user, the administrator has to provide a user code, the full name of the person, his/her affiliation (selected from the list of options), the role of the user (also selected from the drop-down list). Then it is necessary to specify a password for the user (and confirm it by entering the same password for the second time), followed by the user's email. A password must be longer than 4 characters. Each user code must be unique within the affiliation it is related to.

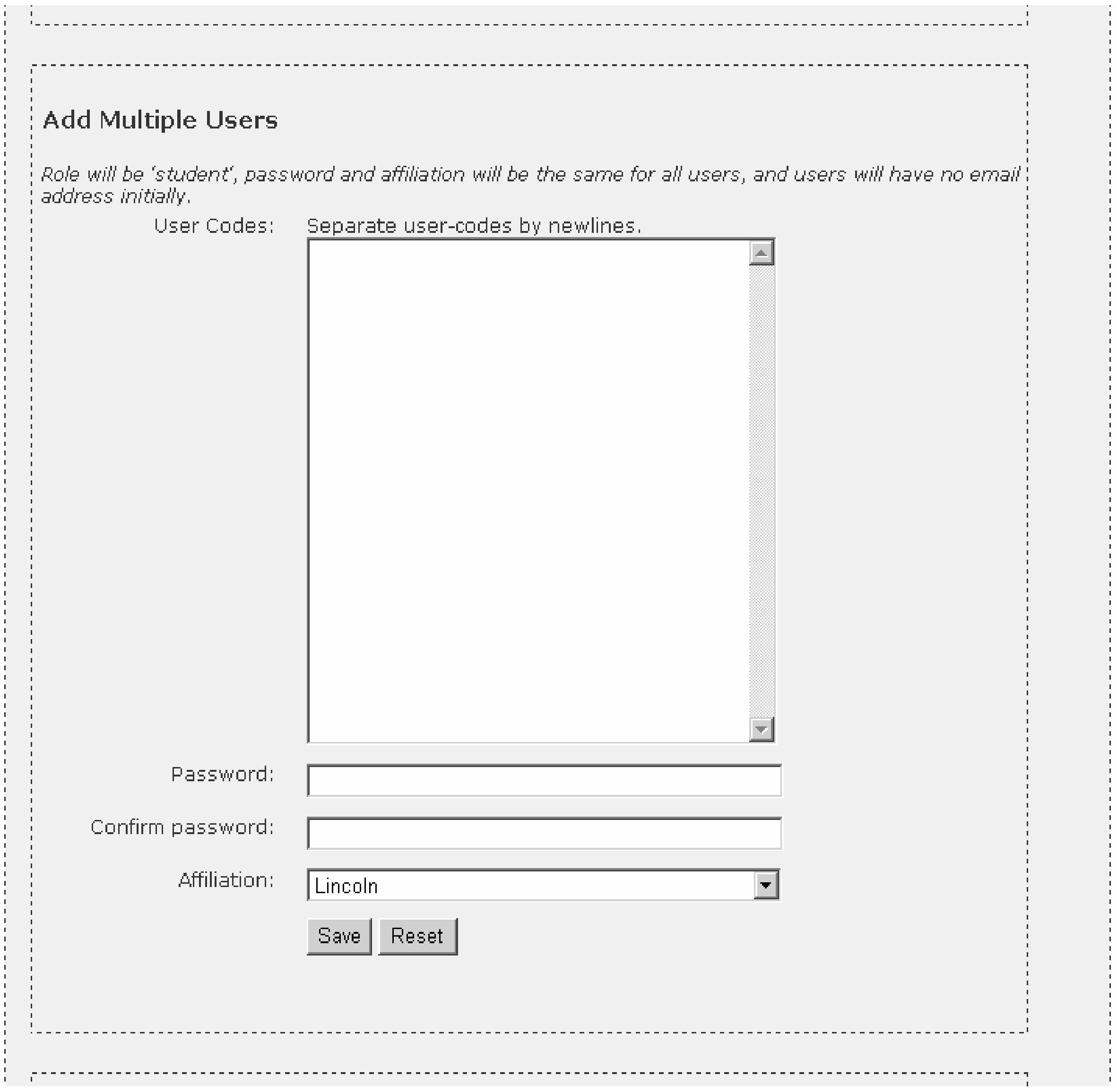


User Code:	
Full Name:	
Affiliation:	ictg
User Role:	student
Password:	
Confirm password:	
Email:	

Role will be 'student', password and affiliation will be the same for all users, and users will have no email address initially.

User Codes: Separate user-codes by newlines.

Figure 54. The User Management page

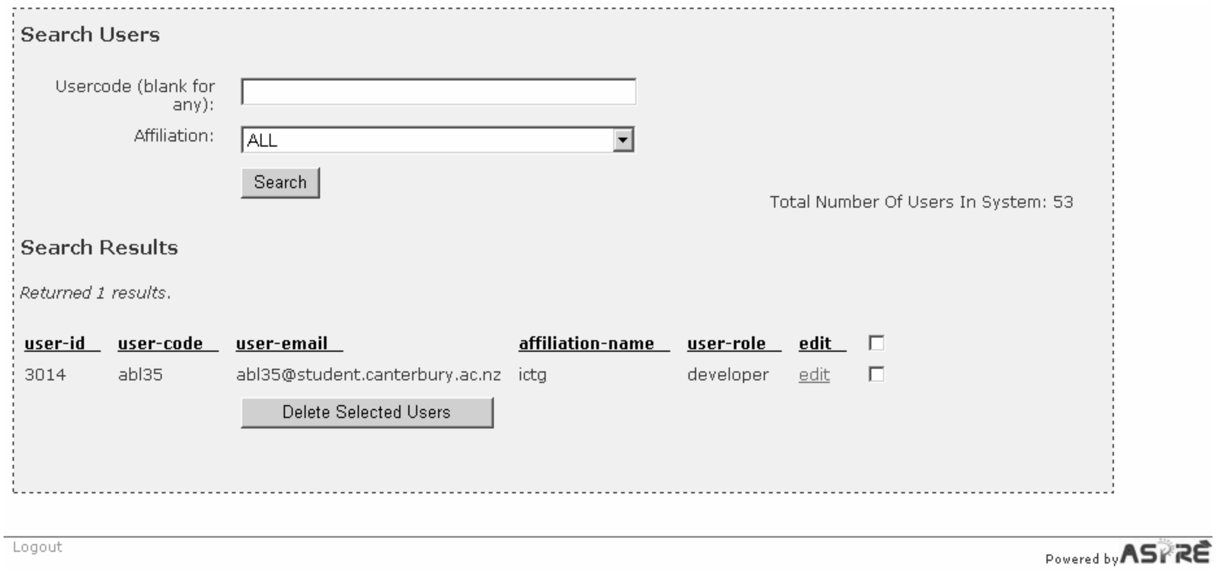

Add Multiple Users

Role will be 'student', password and affiliation will be the same for all users, and users will have no email address initially.

User Codes:	Separate user-codes by newlines.
Password:	
Confirm password:	
Affiliation:	Lincoln

Figure 55. Adding multiple users at once

The *Affiliations* tab allows the administrator to manage affiliations. Figure 57 shows the screenshot of this page. The administrator can add a new affiliation (and its description), and view all affiliations previously defined.



Search Users

Usercode (blank for any):	
Affiliation:	ALL

Search

Total Number Of Users In System: 53

Search Results

Returned 1 results.

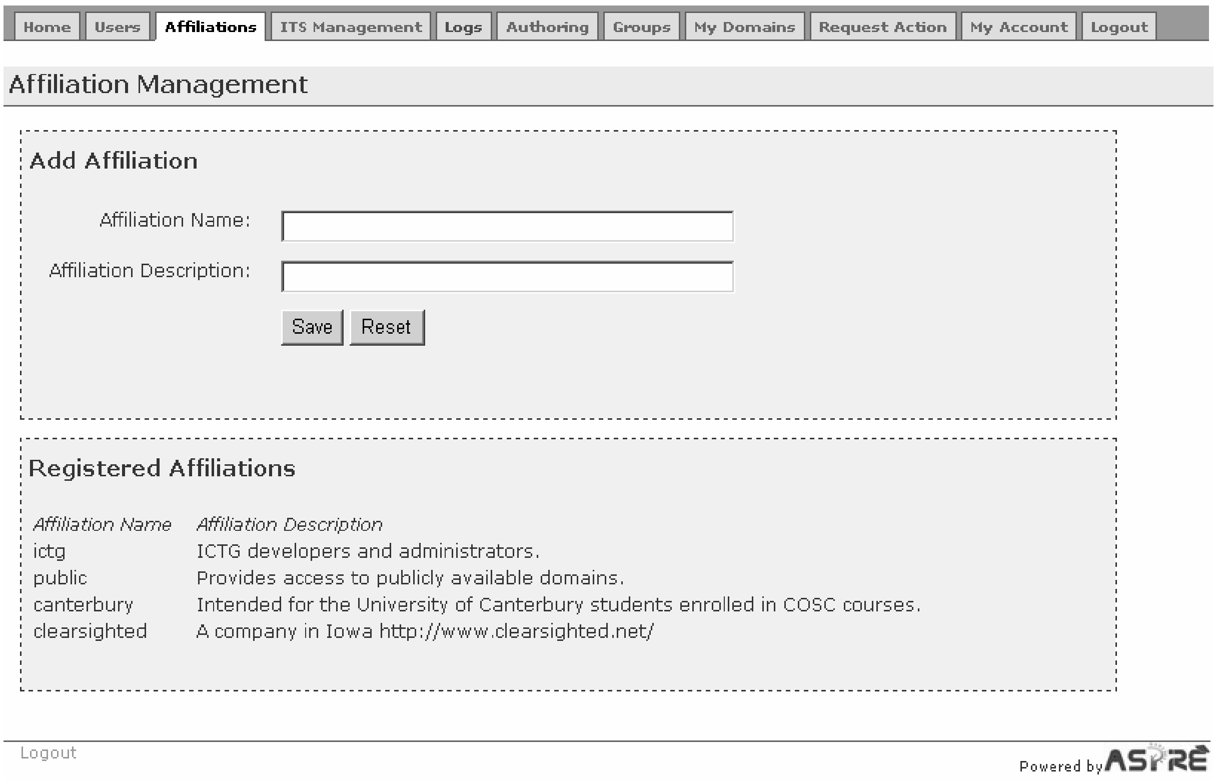
user-id	user-code	user-email	affiliation-name	user-role	edit	
3014	abl35	abl35@student.canterbury.ac.nz	ictg	developer	edit	☐

Delete Selected Users

Logout

Powered by ASPIRE

Figure 56. Retrieving information about existing users



Affiliation Management

Add Affiliation

Affiliation Name:	
Affiliation Description:	

Save Reset

Registered Affiliations

Affiliation Name	*Affiliation Description*
ictg	ICTG developers and administrators.
public	Provides access to publicly available domains.
canterbury	Intended for the University of Canterbury students enrolled in COSC courses.
clearsighted	A company in Iowa http://www.clearsighted.net/

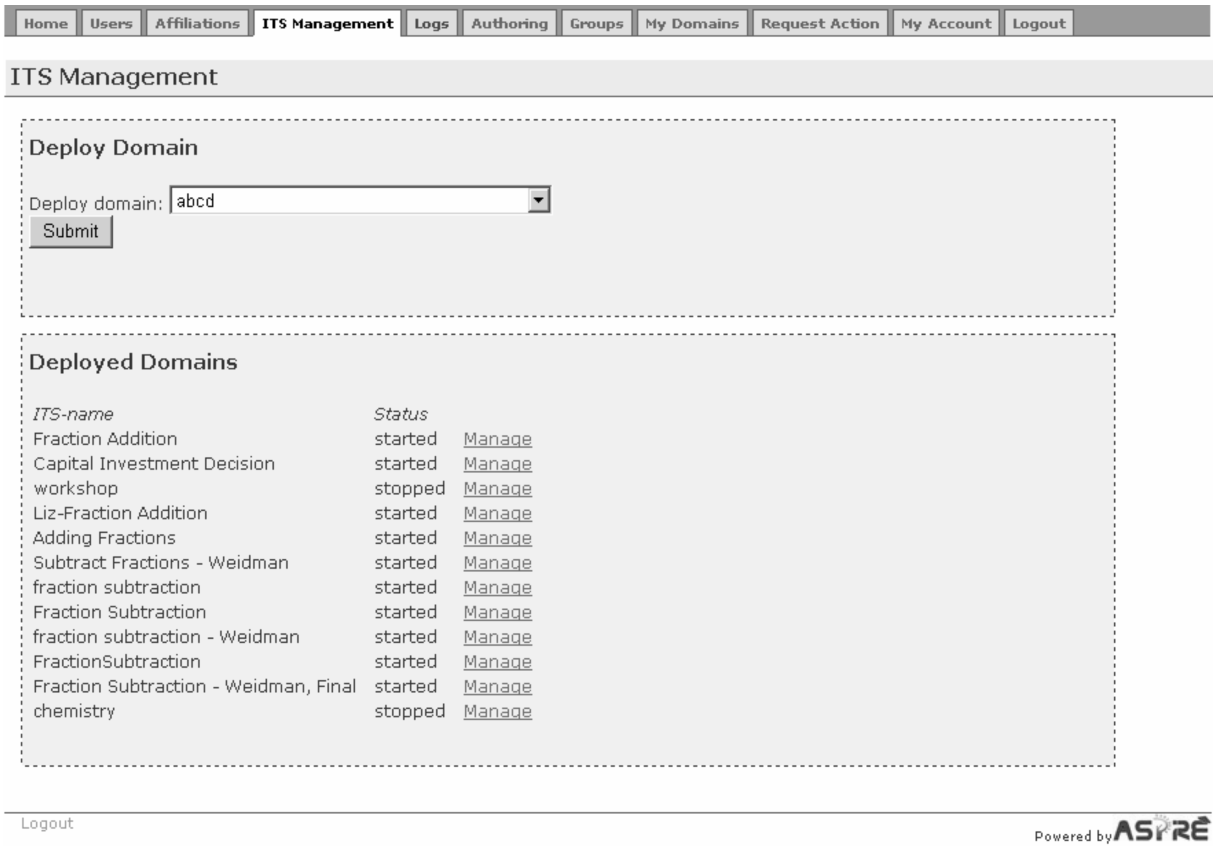
Logout

Powered by ASPIRE

Figure 57. The Affiliations page

Figure 58 shows the *ITS management* page, which allows the administrator to manage the tutoring systems served by ASPIRE-Tutor. The top part of this page allows the administrator to deploy a new domain, by selecting it from the drop-down menu.

The bottom part of this page shows the list of deployed domains (i.e. the intelligent tutoring systems that are currently available in ASPIRE). For each domain (identified by its name), the page shows the status. In Figure 58, there are several domains whose status is *started*, which means that those systems are available to students. When a domain is initially uploaded to ASPIRE-Tutor, its status is *stopped*, which means that the domain is available, but has not been used by any students yet. The *Manage* link provides the administrator with more information about the domain, such as the number of users currently using that domain. The administrator can stop the tutor using the Manage link.

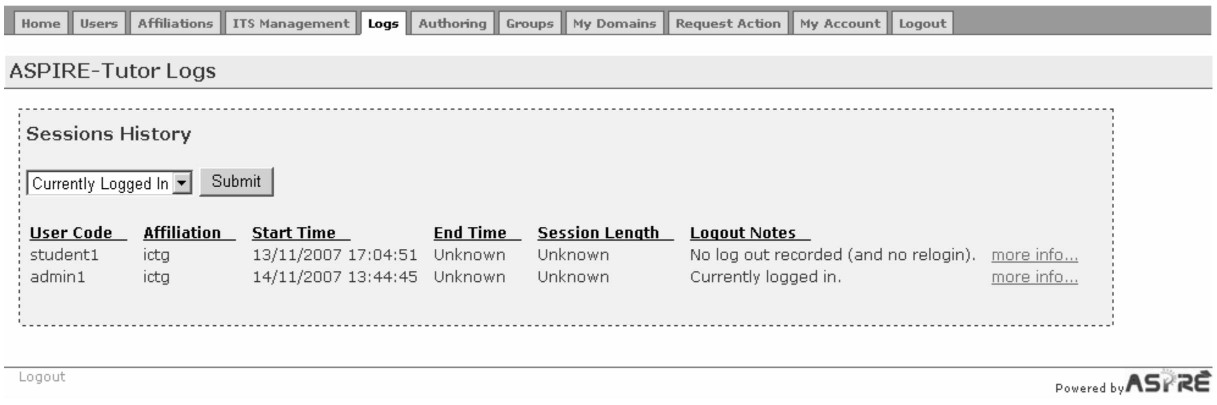


Deploy domain: abcd

ITS-name	*Status*	
Fraction Addition	started	Manage
Capital Investment Decision	started	Manage
workshop	stopped	Manage
Liz-Fraction Addition	started	Manage
Adding Fractions	started	Manage
Subtract Fractions - Weidman	started	Manage
fraction subtraction	started	Manage
Fraction Subtraction	started	Manage
fraction subtraction - Weidman	started	Manage
FractionSubtraction	started	Manage
Fraction Subtraction - Weidman, Final	started	Manage
chemistry	stopped	Manage

Figure 58. The ITS Management page

The *Logs* page (Figure 59) shows the information about the user sessions. The administrator may specify the types of sessions he/she is interested in: all sessions, or just sessions of users who are currently logged in. For each selected user, the page shows the user code and affiliation, as well as the start/end times, the session length and whether the user is currently logged in. The administrator can also get more information about a session by clicking the *more info* link.



ASPIRE-Tutor Logs

Sessions History

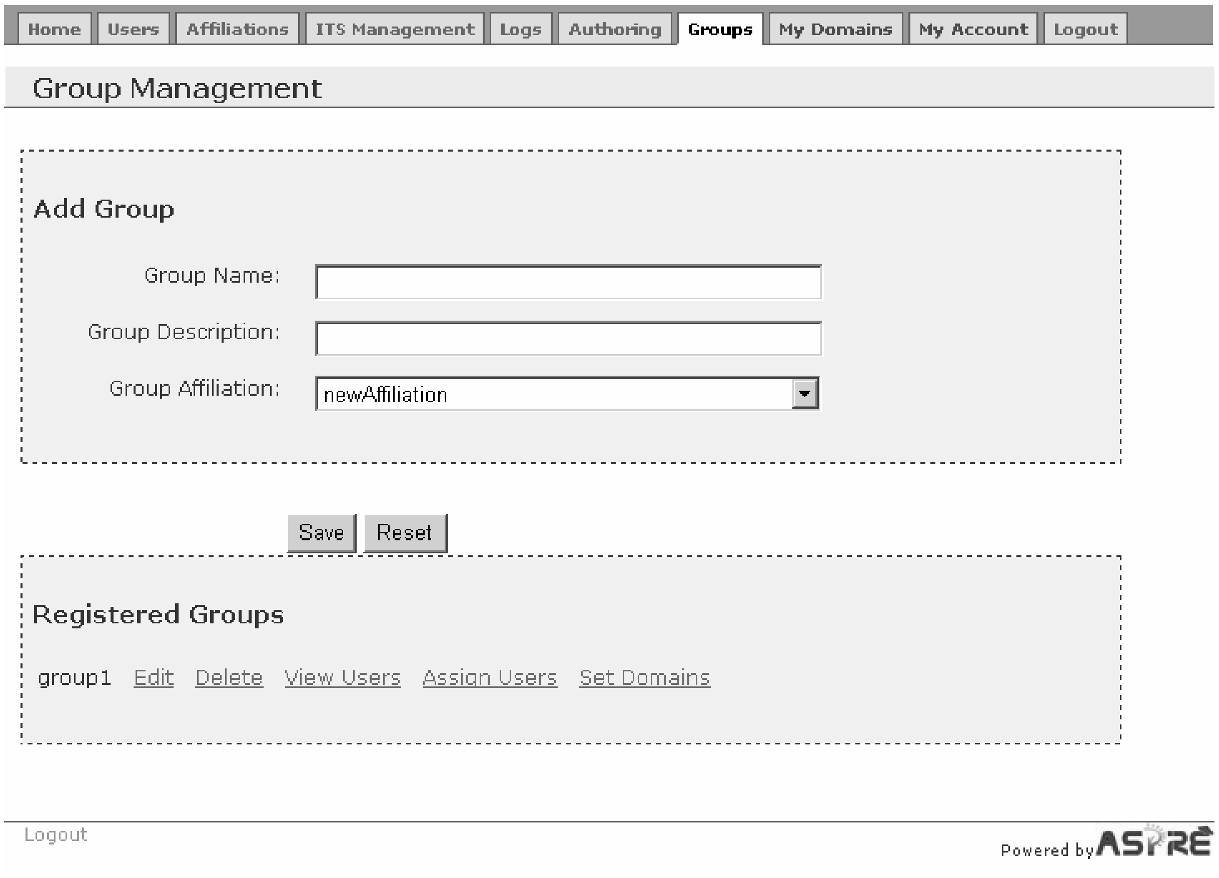
Currently Logged In | Submit

User Code	Affiliation	Start Time	End Time	Session Length	Logout Notes	
student1	ictg	13/11/2007 17:04:51	Unknown	Unknown	No log out recorded (and no relogin).	more info...
admin1	ictg	14/11/2007 13:44:45	Unknown	Unknown	Currently logged in.	more info...

Figure 59. The Logs page

The *Authoring* tab allows the administrator to switch to ASPIRE-Author (discussed in Section 3).

The *Groups* page, shown in Figure 60, allows the administrator to add a group, and modify existing groups. This functionality is primarily the responsibility of teachers, but administrators also have the privileges to work with groups. A group is a collection of student accounts (i.e. a class) who are allowed to use one or more instructional domain. To add a new group, the administrator has to give it a unique name, add an optional description, and specify the affiliation. The administrator can also modify existing groups, by adding/deleting users, adding/deleting domains, and view information about users belonging to the group. See Section 4.3.1 for more information.



Group Management

Add Group

Group Name:	
Group Description:	
Group Affiliation:	newAffiliation

Registered Groups

group1 Edit Delete View Users Assign Users Set Domains

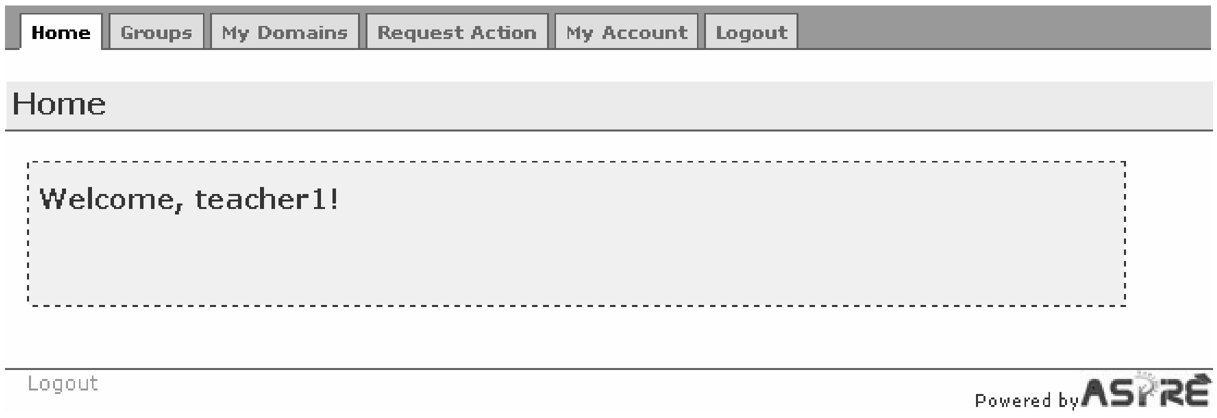
Figure 60. The Group Management page

The *My Domains* page provides information about available domains. The *My Account* page allows the administrator to modify his/her profile. The administrator's own user code and affiliation will be shown. The administrator can change these two, as well as change the password and email.

See Section 4.3.5 for more information about the *Request Action* tab. To log out from ASPIRE, click the *Logout* link at the bottom of the page, or the *Logout* tab.

4.3. Teacher

The role of the teacher is to set up access to the tutoring systems for various groups of students. The teacher may set up various parameters, defining how the student will interact with the target instructional system, and will also specify groups of students. A group of students would have exactly the same experience while working with the system.



Welcome, teacher1!

Figure 61. The Teacher Home Page

Once when a teacher logs in, he/she will be taken to the home page shown in Figure 61. The six tabs at the top of this page are shown on all pages available to teachers. The *My Account* tab brings up a page which allows the teacher to modify his/her own profile, and the *My Domains* tab allows the teachers to see the instructional domain he/she has access to (as discussed in the previous section for administrator). The *Home* tab brings you back to this page.

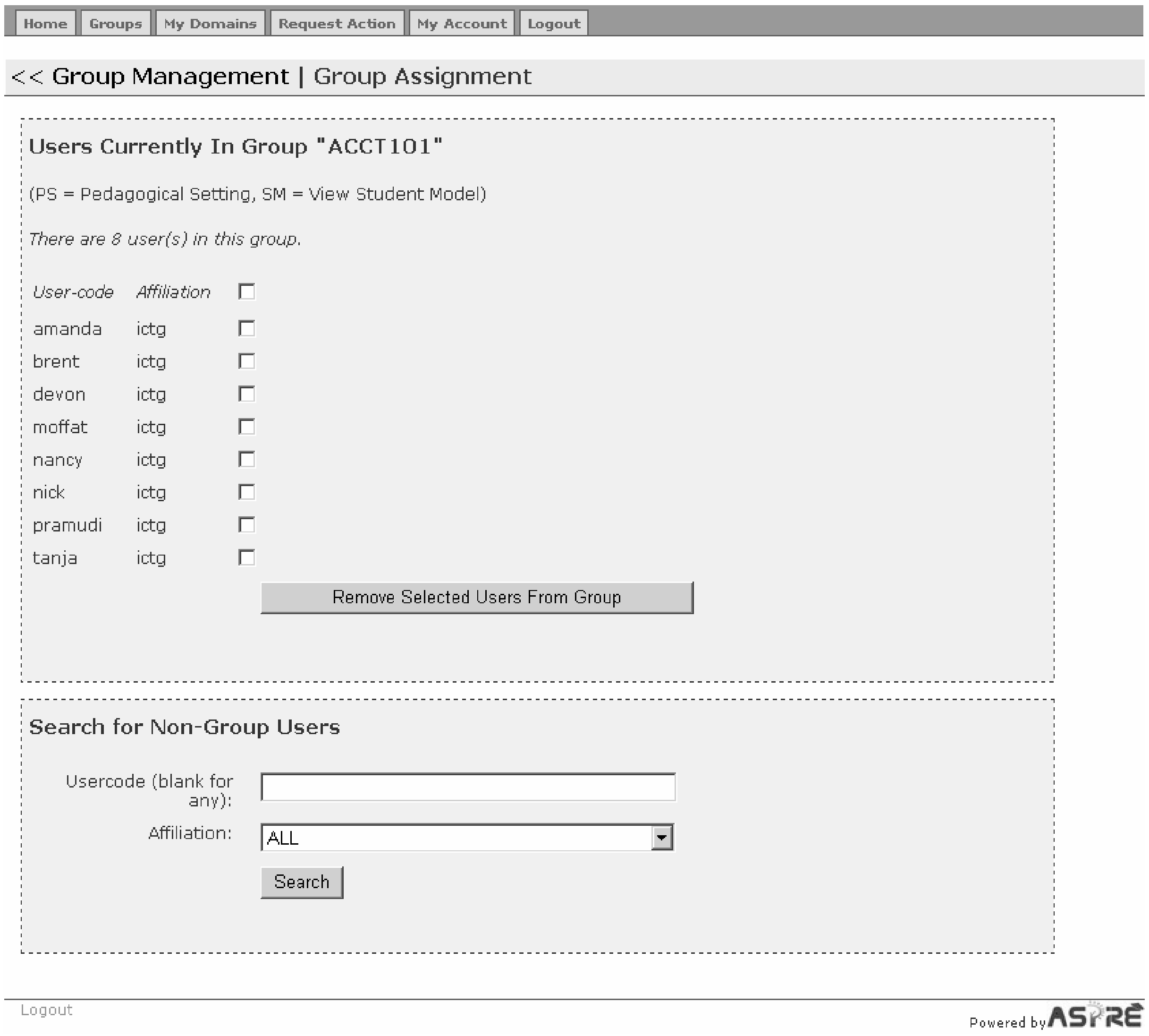
4.3.1. Group Management

The *Groups* tab displays the page shown in Figure 60. As specified in Section 4.2, the *Groups* page allows the teacher to add a group, and modify existing groups. A group is a collection of student accounts (i.e. a class) who are allowed to use one or more instructional domains.

To add a new group, the teacher needs to specify the group name, description and affiliation (selected from the drop-down list). The group name must be unique within affiliation, and must be at least five characters long. When done, click the *Save* button. The *Reset* button clears all the fields.

Existing groups are shown in the bottom part of the page. For each group, there is a set of links that can be used to edit/delete the group, view users assigned to the group, assign students to the group or set instructional domains for a group.

Figure 62 shows the page for assigning users to groups. The top part of this page shows users who are currently assigned to the chosen group. Besides each user code, there is a tick box which can be used to select the account to me removed from the group (by clicking the *Remove Selected Users From Group* button).



<< Group Management | Group Assignment

Users Currently In Group "ACCT101"

(PS = Pedagogical Setting, SM = View Student Model)

There are 8 user(s) in this group.

User-code	*Affiliation*	☐
amanda	ictg	☐
brent	ictg	☐
devon	ictg	☐
moffat	ictg	☐
nancy	ictg	☐
nick	ictg	☐
pramudi	ictg	☐
tanja	ictg	☐

Search for Non-Group Users

Usercode (blank for any):

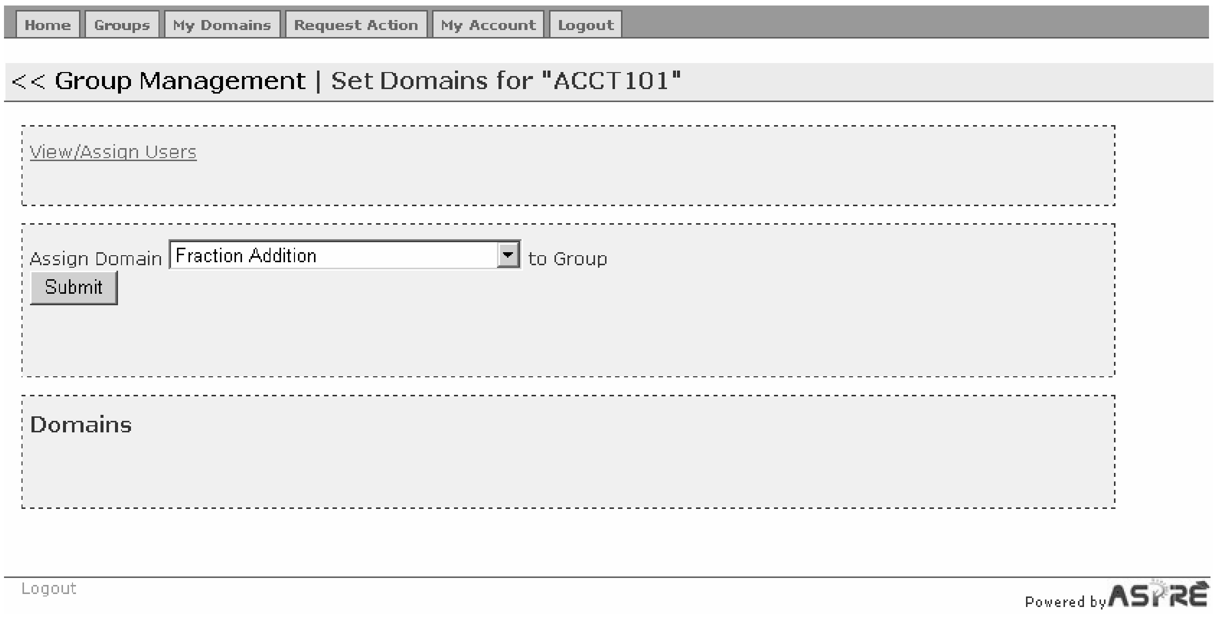
Affiliation: ALL

Figure 62. The Group Assignment Page

The bottom part of this page allows the teacher to add new students to the group. To do that, the teacher needs to search for the account by specifying the user code, or to ask for all accounts within a specific affiliation. The page then shows all the retrieved accounts, and the teacher can ask for one or more of them to be added to the group.

4.3.2. Assigning Domains to Groups

When a group is created, the teacher needs to specify the tutoring system(s) the group will have access to. To do that, click the *Set Domains* link from the Group Management page (Figure 60). ASPIRE will then show the Set Domains page (Figure 63). At the top of the page, there is a *View/Assign Users* link, which allows the teacher to view the students in the group. Next, the page shows a menu containing all domains, from which the teacher can select a new domain and assign it to the group.



<< Group Management | Set Domains for "ACCT101"

View/Assign Users

Assign Domain Fraction Addition to Group

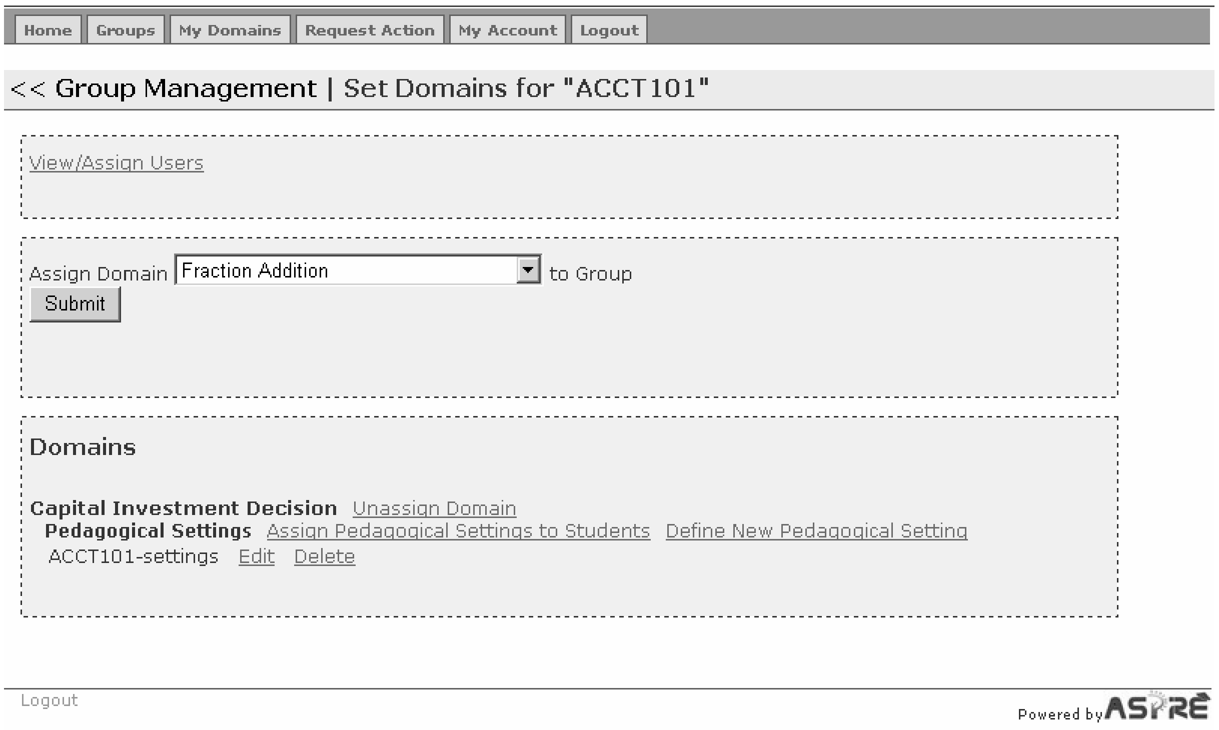
Submit

Domains

Figure 63. The Set Domains Page

The bottom part of the page shows all domains have been allocated to the group. The group illustrated in Figure 63 has no allocated domains. The teacher then selects the name of the domain (e.g. *Capital Investment Decision*), which then requires the teacher to specify how the chosen domain will be used. In ASPIRE, this specification is known as the *pedagogical settings*, and is explained in the following section.

After allocation the *Capital Investment Decision* domain to this group, the resulting state of the Set domains page is shown in Figure 64. The bottom part of the page now shows the domain, with two a related *Unassign Domain* link, which can be used to remove the domain from the group; after that, the students in the group will not be able to access that domain. The *Assign Pedagogical Settings to Students* link (discussed in Section 4.3.4) allows the teacher to specify the pedagogical settings, that is, to tune the behaviour of the tutoring system for the specific group. The teacher can also define a new pedagogical setting by clicking on the corresponding link. The definition of pedagogical setting is specified in the following section. Finally, ASPIRE shows the pedagogical settings the teacher defined for this domain previously (*ACCT101-settings*), with the Edit and Delete links.



<< Group Management | Set Domains for "ACCT101"

View/Assign Users

Assign Domain Fraction Addition to Group
Submit

Domains

Capital Investment Decision Unassign Domain
Pedagogical Settings Assign Pedagogical Settings to Students Define New Pedagogical Setting
ACCT101-settings Edit Delete

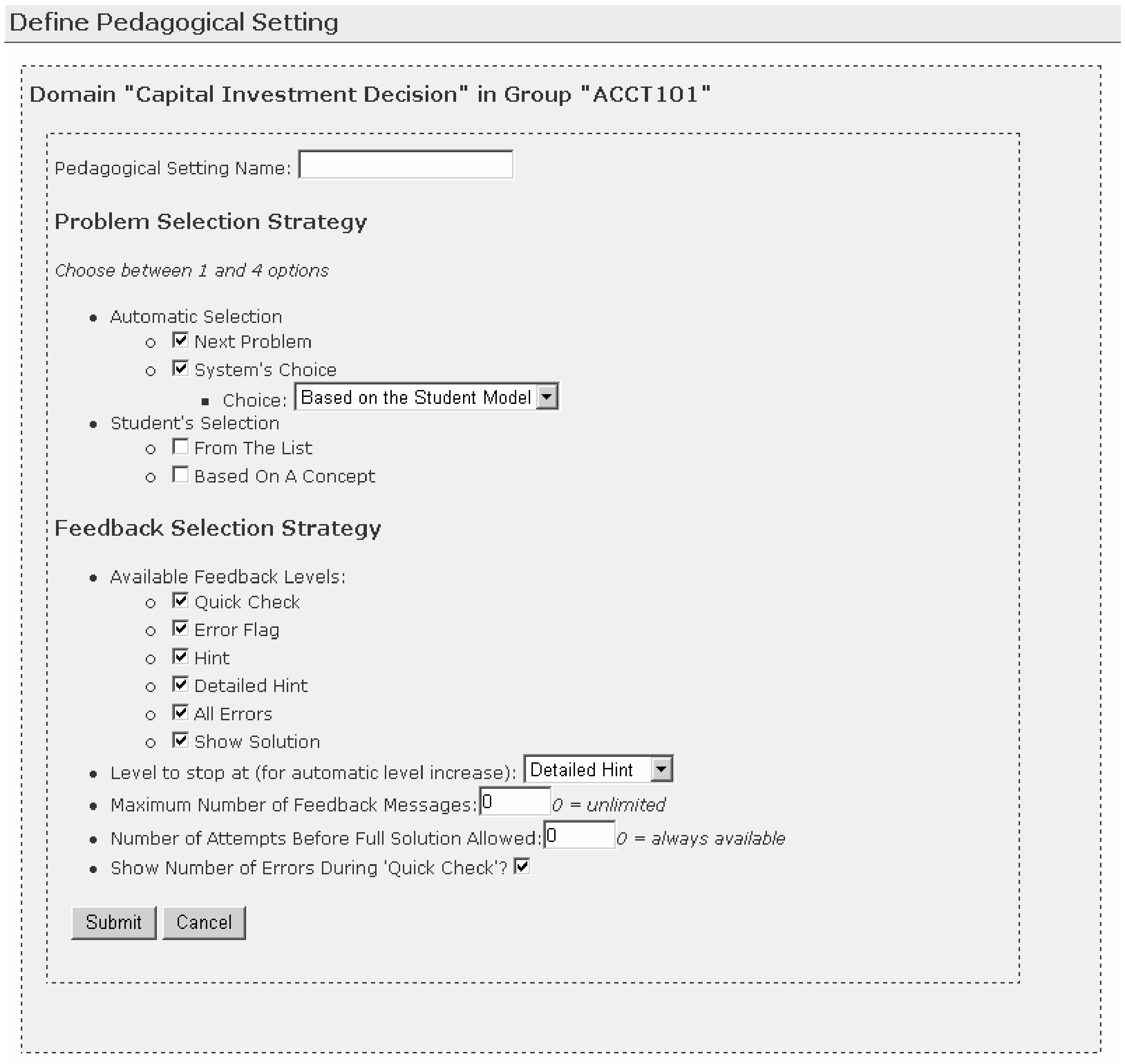
Figure 64. The modified Set Domains Page

4.3.3. Specifying Pedagogical Settings

The teacher needs to specify the pedagogical settings for each domain assigned to a group. The pedagogical settings page will appear automatically after a domain has been added to a group (as discussed in the previous section), or can be defined/modifed at a later time (by clicking the Define New Pedagogical Setting link shown in Figure 64). ASPIRE-Tutor will then open a new page, illustrated in Figure 65.

The Pedagogical Setting (PS) is a set of parameters used to specify fine details of the options students will have when they work in the corresponding domain. For each domain, the teacher needs to specify at least one set of pedagogical settings.

The Define Pedagogical Setting page (Figure 65) requires the teacher to specify the name for the PS first (note that there might be multiple PSs for the same domain for one group). Each PS must have a unique name. The teacher needs to specify the problem selection strategies that will be available to students. There needs to be at least one problem selection strategy specified. ASPIRE-Tutor starts with some default options, which can be modified. There are two groups of problem-selection strategies. In the first group (*Automatic Selection*), there are two options. The *Next Problem* option means that the student will get as a new problem the problem that immediately follows the current problem (please note that the order of problems is specified by the author). The *System's Choice* option is a problem-selection strategy which uses the student model: ASPIRE will analyze the student's knowledge at the time, and select a problem at the appropriate level of complexity. At the moment, there is only one problem-selection strategy available of this kind, but in the future others will be added.



Define Pedagogical Setting

Domain "Capital Investment Decision" in Group "ACCT101"

Pedagogical Setting Name:

Problem Selection Strategy

Choose between 1 and 4 options

- Automatic Selection
  - [✓] Next Problem
  - [✓] System's Choice
    - Choice: Based on the Student Model
- Student's Selection
  - [ ] From The List
  - [ ] Based On A Concept

Feedback Selection Strategy

- Available Feedback Levels:
  - [✓] Quick Check
  - [✓] Error Flag
  - [✓] Hint
  - [✓] Detailed Hint
  - [✓] All Errors
  - [✓] Show Solution
- Level to stop at (for automatic level increase): Detailed Hint
- Maximum Number of Feedback Messages: 0 *0 = unlimited*
- Number of Attempts Before Full Solution Allowed: 0 *0 = always available*
- Show Number of Errors During 'Quick Check'? [✓]

Figure 65. Specifying pedagogical settings

The second group of problem-selection strategies involves the student in choosing a problem to work on next. The *From The List* option means that the student will be given a list of problems to select from . The *Based On A Concept* option means that the student will be given a list of domain concepts, and he/she will select a concept to practise. ASPIRE will then select a problem of the appropriate complexity based on the chosen domain concept.

Next, the teacher needs to specify the levels of feedback given to students. There are seven feedback levels to choose from, and by default, they are all available:

- *Quick Check*: this level of feedback provides minimal information to the student. Once when the student submits a solution (and this level of feedback is selected), the student will only be told whether his/her solution is correct or not.
- *Error Flag*: at this feedback level, the student is told, for incorrect solutions, what part of the solution is wrong;
- *Hint*: for this level, the student is given feedback on the first violated constraint;

- *Detailed Hint*: the student is given the detailed message associated with the first violated constraint;
- *All Errors*: the student is given hint messages for all violated constraints;
- *Show Solution*: the complete solution to the current problem is shown to the student.

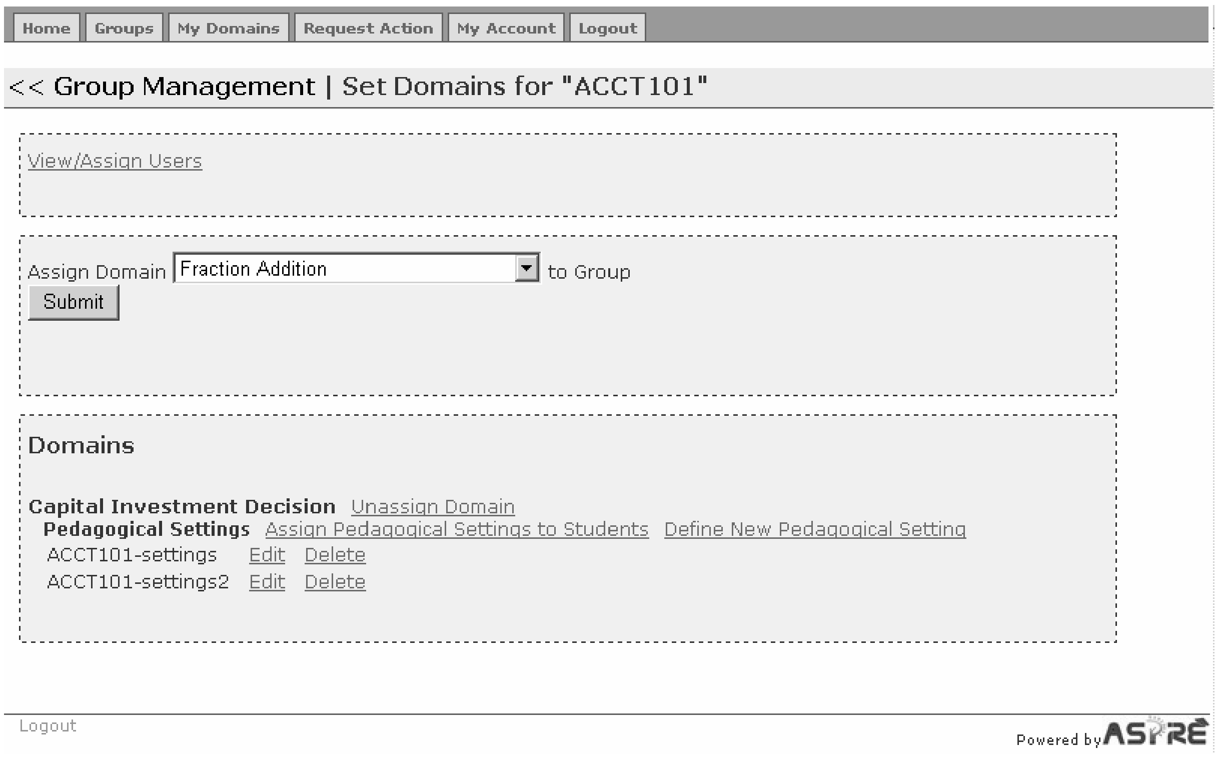
The default feedback-presentation strategy in ASPIRE is to start from *Positive/Negative* on the first submission, and then increase the level for each subsequent submission until the *Detailed Hint* level is reached, and then stay at that level for all later submissions. However, the student has the option to ask for a specific level of feedback on each submission.

The teacher can disable some feedback levels, if appropriate. The teacher is also given an opportunity to modify the default feedback-presentation strategy. There is a drop-down box labelled *Level to stop at (for automatic level increase*); the teacher can modify the default level (*Detailed Hint*) by selecting another level from the list. The teacher can also specify the maximum number of feedback messages to show to the student at the *All Errors* level. The default value for this option is zero, which means that all feedback messages will be shown, i.e. as many hints as there are errors (violated constraints). The default option for full solution is that the student can request it whenever they want (default value of zero). The teacher can, however, specify the minimal number of submissions the student must make before he/she will be allowed to see the full solution.

When showing Positive/Negative Feedback, ASPIRE will, by default, show the total number of errors. The teacher can modify this default behaviour by requiring that the student be only told that there are some errors in his/her solution.

4.3.4. Assigning Pedagogical Settings to Students

If there is more than one PS defined for a domain, it is necessary to assign them to students within the group. Figure 66 shows two pedagogical settings defined for the *Capital Investment Decision* domain.



<< Group Management | Set Domains for "ACCT101"

View/Assign Users

Assign Domain Fraction Addition to Group

Domains

Capital Investment Decision Unassign Domain

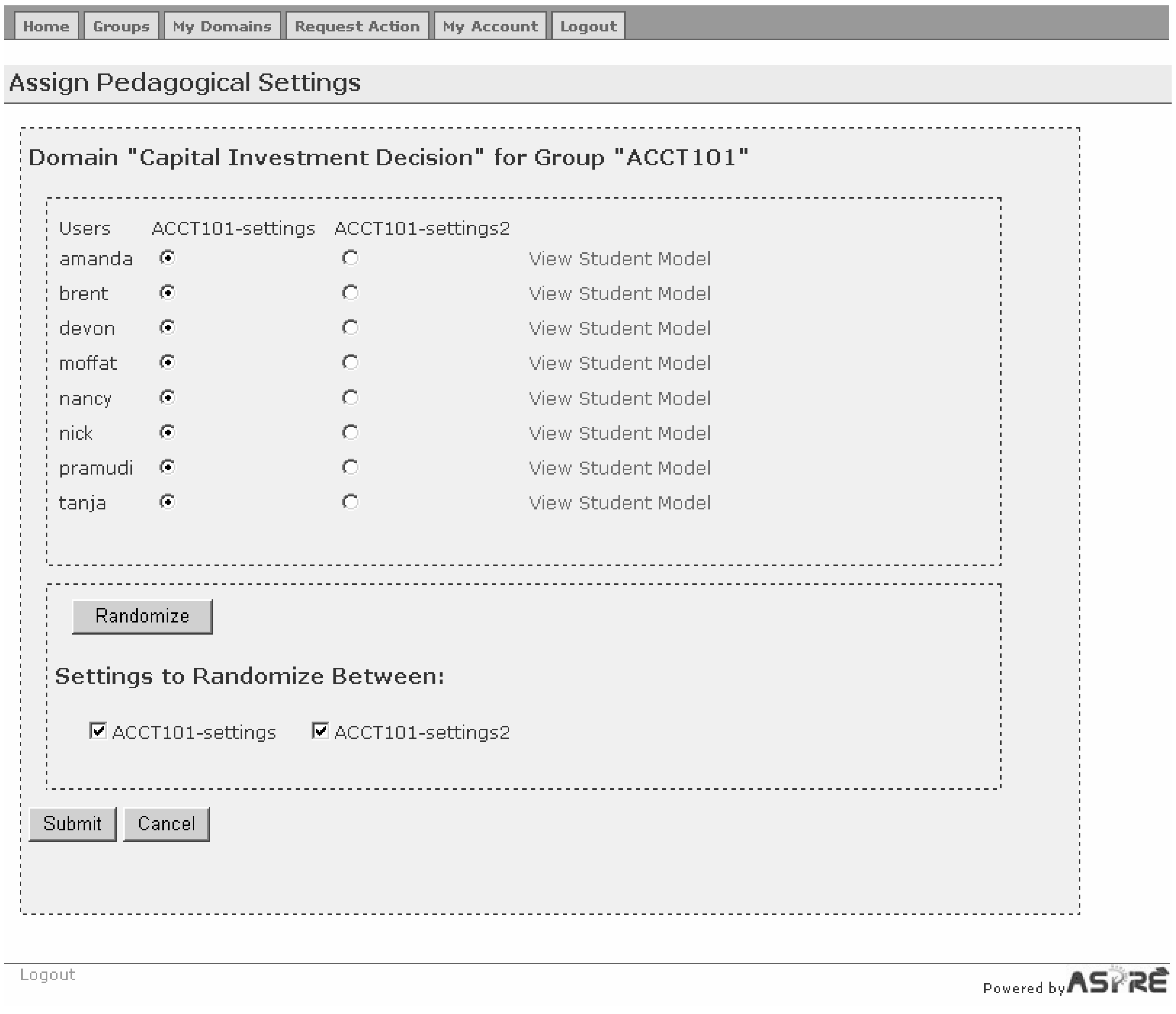
Pedagogical Settings Assign Pedagogical Settings to Students Define New Pedagogical Setting

ACCT101-settings Edit Delete

ACCT101-settings2 Edit Delete

Figure 66. A group with two sets of pedagogical settings

The teacher then needs to allocate pedagogical setting to each student in the group. To achieve that, the teacher clicks the *Assign Pedagogical Settings to Students* link, which shows the page in Figure 67.



Assign Pedagogical Settings

Domain "Capital Investment Decision" for Group "ACCT101"

Users	ACCT101-settings	ACCT101-settings2	
amanda	●	○	View Student Model
brent	●	○	View Student Model
devon	●	○	View Student Model
moffat	●	○	View Student Model
nancy	●	○	View Student Model
nick	●	○	View Student Model
pramudi	●	○	View Student Model
tanja	●	○	View Student Model

Randomize

Settings to Randomize Between:

☑ ACCT101-settings ☑ ACCT101-settings2

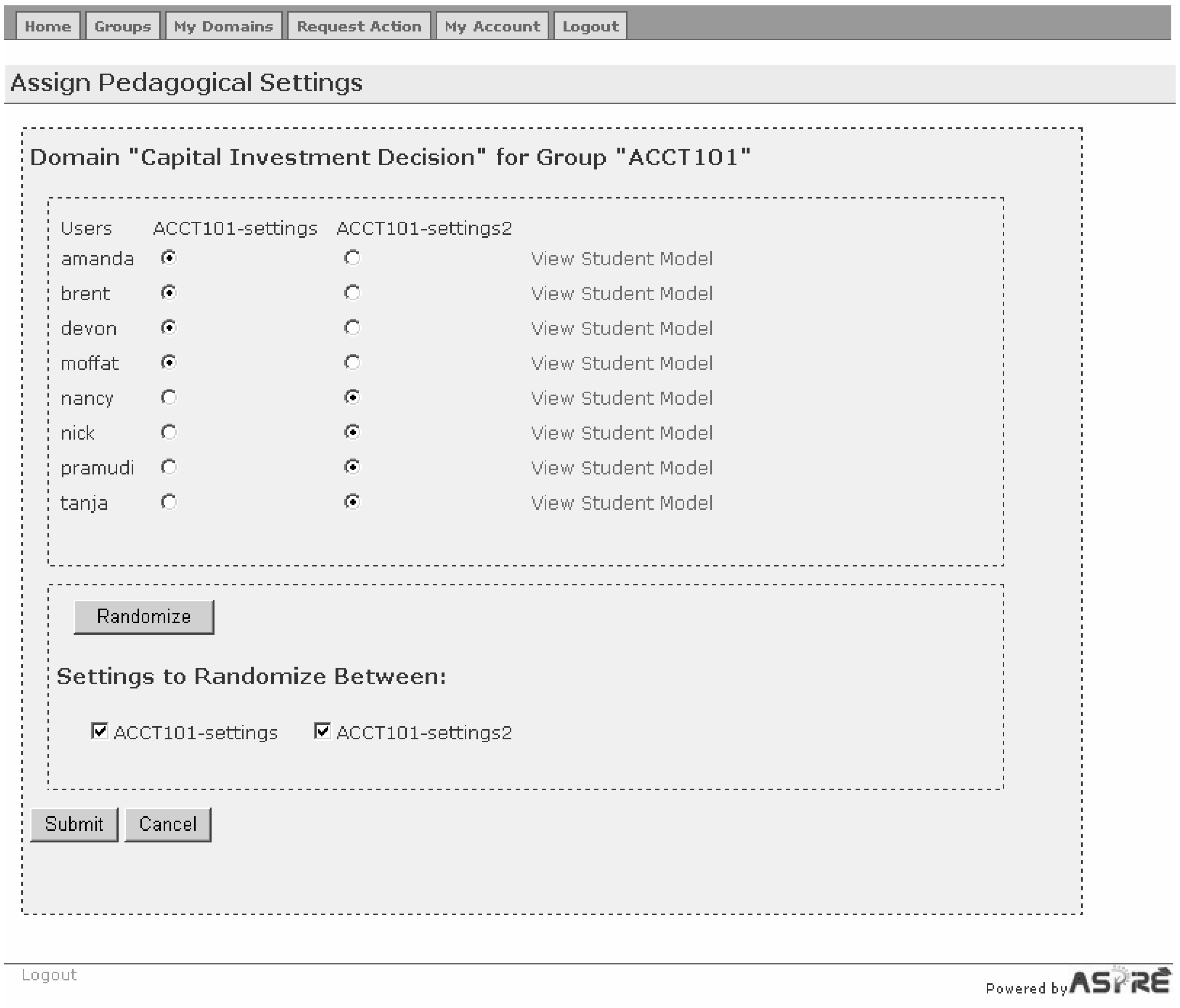
Submit Cancel

Logout

Powered by ASPIRE

Figure 67. Assigning pedagogical settings to students

The assignment can be done manually, in which case the teacher needs to select a PS for each individual student by selecting one of the radio buttons. The assignment can also be done in a random fashion - to achieve that, the teacher click the *Randomize* button, the effect of which is shown in Figure 68.



Assign Pedagogical Settings

Domain "Capital Investment Decision" for Group "ACCT101"

Users	ACCT101-settings	ACCT101-settings2	
amanda	●	○	View Student Model
brent	●	○	View Student Model
devon	●	○	View Student Model
moffat	●	○	View Student Model
nancy	○	●	View Student Model
nick	○	●	View Student Model
pramudi	○	●	View Student Model
tanja	○	●	View Student Model

Randomize

Settings to Randomize Between:

☑ ACCT101-settings ☑ ACCT101-settings2

Figure 68. Random assignment of pedagogical settings to students

The *View Student Model* link allows the teacher to see statistics about the student's knowledge of the instructional domain. The teacher will see the list of problems the students has solved correctly, and also statistics about constraint use.

4.3.5. Requesting action

The teacher (and other types of users) may request actions to be taken within ASPIRE. This is supported through the Request Action tab, shown in Figure 69. The user may type the request, which will be emailed to administrators.

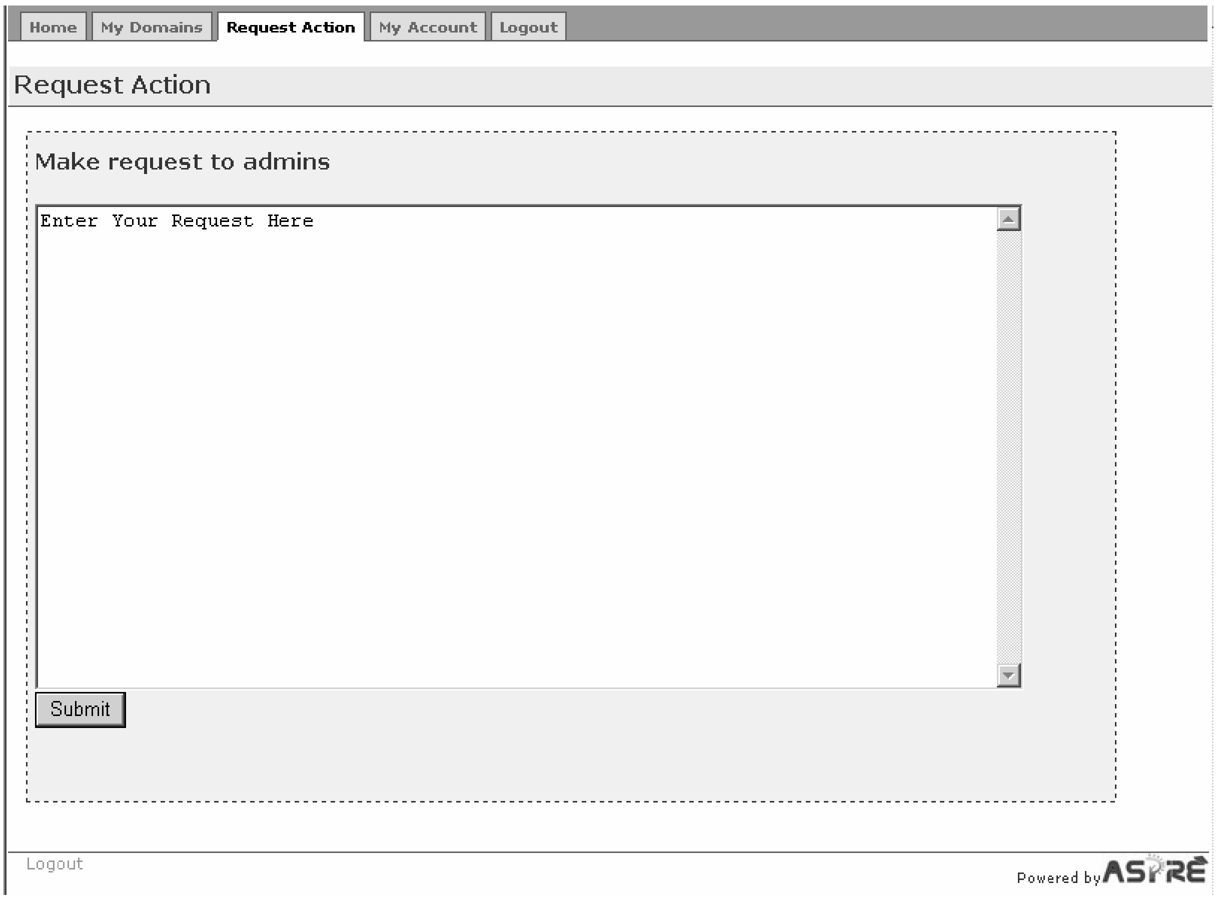


Figure 69. The Request Action page

4.4. Student

When a student logs into ASPIRE, he/she will be taken to the student home page. Figure 70 shows an example of a student home page. The student who this page was created for has been granted access to one instructional domain. If there are several instructional domains a student has the right to access, the home page will show multiple links.

At the top of the page, there are five tabs, which are available on all pages available to students. The *Home* tab brings the student back to this page. The *My Domains* tab allows the student to see all the domains he/she has access to. The *Request Action* tab was already discussed in Section 4.3.5. The *My Account* page allows the student to modify his/her profile. Use the *Logout* tab/link to terminate the session with ASPIRE.

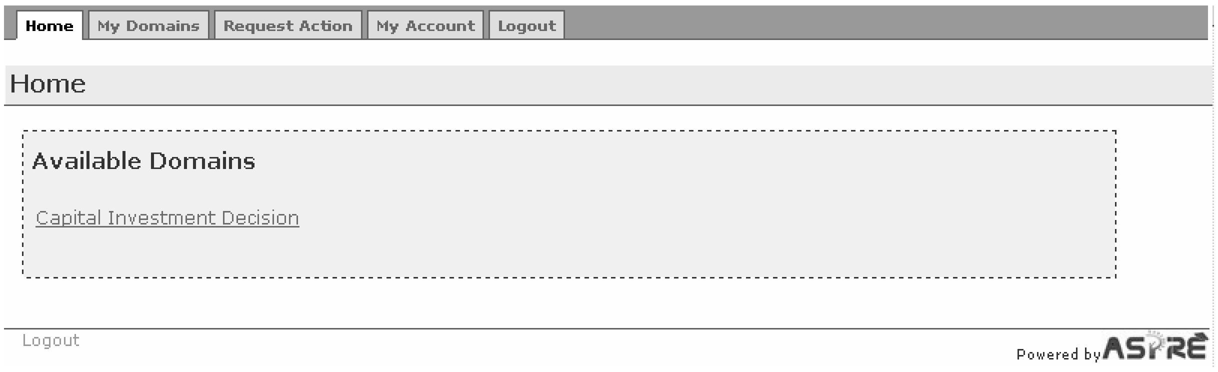
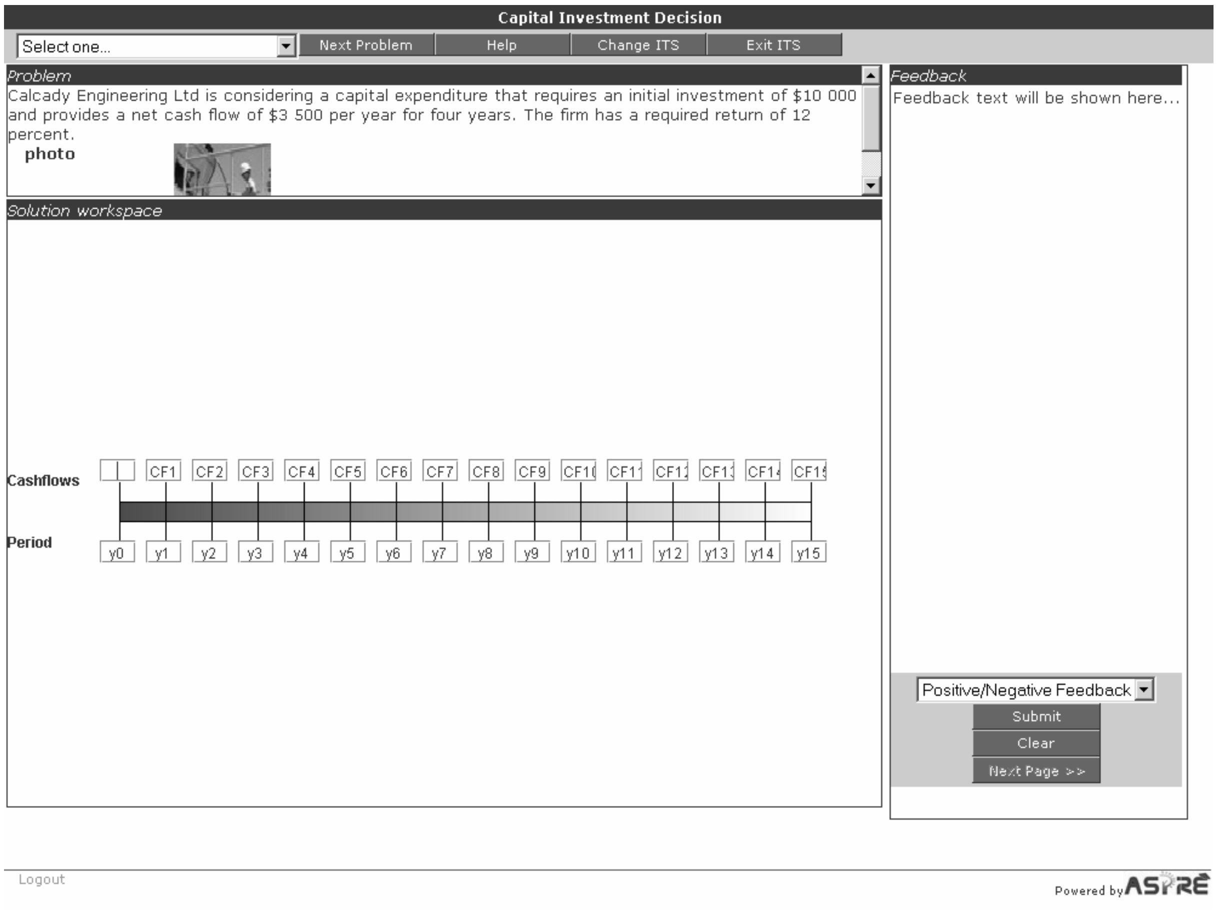


Figure 70. The Student Home Page

When a student selects a domain (by clicking the appropriate link on the home page or on the My Domains page), he/she will be given the problem-solving interface for the chosen domain. Figure 71 shows the problem-solving interface for the domain the student has selected.


Capital Investment Decision
Select one...
Next Problem
Help
Change ITS
Exit ITS
Problem
photo
Solution workspace
Cashflows
CF1 CF2 CF3 CF4 CF5 CF6 CF7 CF8 CF9 CF10 CF11 CF12 CF13 CF14 CF15
Period
y0 y1 y2 y3 y4 y5 y6 y7 y8 y9 y10 y11 y12 y13 y14 y15
Feedback
Feedback text will be shown here...
Positive/Negative Feedback
Submit
Clear
Next Page >>
Logout
Powered by ASPIRE

Calcady Engineering Ltd is considering a capital expenditure that requires an initial investment of $10 000 and provides a net cash flow of $3 500 per year for four years. The firm has a required return of 12 percent.

Figure 71. The problem-solving interface for an instructional domain

4.5. Developer

Developers have all the privileges of authors, with the additional privilege of being able to make changes to the domain model. A developer can add new constraints, as well as delete or modify constraints generated by ASPIRE (in the Syntax/Semantic Constraints tabs). The developer can also test the constraints (in the *Test Constraints* tab), as well as add/modify/delete domain-specific functions (in the *Domain Function* tab).

The process of developing constraints is similar to programming; the developer is a person with significant experience in programming and knowledge engineering. The developer understands the constraint language used by ASPIRE, and must also have an excellent grasp of constraint evaluation within ASPIRE and Lisp, which is the programming language ASPIRE is based on. Within this manual, we do not present the details of constraint evaluation, constraint language and Lisp. The interested reader is referred to the developer documentation[4].

5. Conclusions

This document illustrated authoring support provided by ASPIRE-Author, as well as the functionality of ASPIRE-Tutor, the deployment environment. ASPIRE is available freely on the Web[4]. The ASPIRE team would be grateful for any feedback on ASPIRE and this user manual.

6. References

1. Decker, S., Melnik, S., van Harmelen, F., Fensel, D., Klein, M., Broekstra, J., Erdmann, M., Horrocks, I. (2000) The Semantic Web: The Roles of XML and RDF. IEEE Internet Computing, 4(5), 63-74.
2. Gruber, T.R. (1993) A Translation Approach to Portable Ontology Specification. Knowledge Acquisition, 5, 199-200.
3. Hendler, J. (2001) Agents and the Semantic Web. IEEE Intelligent Systems, 16(2), 30-37.
4. Hendler, J. (2005) Knowledge is Power: a View from the Semantic Web. AI Magazine, 26(4), winter 2005, 76-84.
5. Mitrovic, A., Suraweera, P., Martin, B., Zakharov, K., Milik, N., Holland, J. (2006) Authoring constraint-based tutors in ASPIRE. M. Ikeda, K. Ashley, and T.-W. Chan (Eds.) Proc. 8th Int. Conf. on Intelligent Tutoring Systems ITS 2006, LNCS 4053, pp. 41-50.
6. Noy, N., McGuinness, D. (2001) Ontology Development 101: a Guide to Creating your First Ontology. Stanford Knowledge Systems Laboratory Technical Report KSL-01-05 and Stanford Medical Informatics Technical Report SMI-2001-0880.
7. Suraweera, P., Mitrovic, A., Martin, B. (2005) A knowledge acquisition system for constraint-based intelligent tutoring systems. In: C-K Looi, G. McCalla, B. Bredeweg, J. Breuker (eds) Proc. Artificial Intelligence in Education AIED 2005, IOS Press, pp. 638-645.
8. Suraweera, P., Mitrovic, A., Martin, B. (2007) Constraint Authoring System: an empirical evaluation. R. Luckin, K. Koedinger, J. Greer (eds) Proc. *13th Int. Conf. Artificial Intelligence in Education AIED 2007*, Los Angeles, 451-458.

[4] Please email Tanja.mitrovic@canterbury.ac.nz to obtain a user account for ASPIRE, and additional information.

INDEX