DOI 10.1007/s10115-002-0088-z
Springer-Verlag London Ltd. © 2003
Knowledge and Information Systems (2003) 5: 263–287

Knowledge and Information Systems

Web Site Synthesis Based on Computational Logic

João M. B. Cavalcanti and David Robertson

Centre for Intelligent Systems and their Applications, Division of Informatics,
University of Edinburgh, Edinburgh, UK

Abstract. Web site design and maintenance has become a challenging problem due to the increase in volume and complexity of information presented in this way. Much attention has been given to the deployment of Web sites but little thought has been given to methods for their design and maintenance. Web site applications can also benefit from systematic approaches to development that make design more methodical and maintenance less time consuming. One way to tackle this problem is via automated synthesis, automatically deriving a Web site from a high-level application description. Computational logic is well suited to this problem because of its support of a uniform view of data and computation, allowing reasoning with both specification and program via meta-programming.

Keywords: Automated synthesis; Computational logic; Web site application

1. Introduction

Web site design and maintenance has become a challenging problem due to the increase in volume and complexity of information presented in that way. It often involves access to databases, complex cross-referencing between information within the site and sophisticated user interaction. This is particularly true for data-intensive Web sites. These sites are subject to constant updates, raising the cost of site maintenance. As this cost is usually recurrent, a design method that facilitates maintenance is valuable.

Depending on the approach used, the cost of designing and maintaining a Web site can vary. Without a systematic approach to Web site construction, a site developer performs this task by writing HTML files by hand (possibly using a structure editor). The effort to produce the site in this way increases with the size and complexity of the application. The main sink of effort is maintenance, which can be very time consuming and tedious because information content and presentational details are mingled. It also involves direct manipulation of source files or program code, which requires technical skills.

Received 14 June 2001
Revised 21 Dec 2001
Accepted 22 April 2002

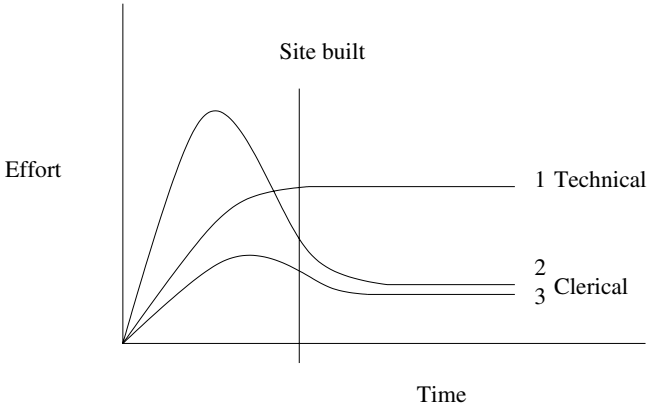


Fig. 1. Cost to design and maintain Web sites.

An alternative approach is to develop programs which automatically synthesize a Web site given a specification of an application. This approach reduces maintenance cost and in some cases it is possible to allow non-technical personnel to update the content of the site via an appropriate interface. However, changes in the navigation structure or visualization can be hard as this requires modifications in the program that generates the Web site. If changes of this nature are frequent, this approach shifts the problem from Web page maintenance to synthesizer maintenance.

Declarative specifications can form a basis to automated synthesis changing the focus from the mechanics to produce Web sites to specification of applications (Florescu et al., 1998; Cavalcanti and Robertson, 2000). Providing a high-level description of a problem independent from any particular implementation allows the designer to concentrate on the application description rather than on the mechanics for producing the Web site.

We have been using a domain-specific synthesizer for our research group Web site at Edinburgh (http://www.dai.ed.ac.uk/groups/ssp/index.html) for about 4 years (Robertson and Agustí, 1999). Maintenance is done simply by updating its declarative specification. Although this has proved cost-effective it is limited to the generation of a particular visualization and navigation structure. Although there is a declarative specification of the site, any substantial change to the visualization style requires modification in the synthesizer program.

We want a design method that produces a Web site consistent with a specification and facilitates maintenance more generally. Updating data should become a clerical task and modifications in navigation structure or visualization should not involve further programming, although inevitably it still requires technical expertise. As a result, the effort to produce a Web site is reduced and maintenance cost is low. Figure 1 shows a comparison of cost between the traditional Web site maintenance (line 1), the automated method used for the last 4 years (line 2) and the proposed approach (line 3).

Our approach applies computational logic to support a uniform view of data and computation, allowing reasoning with both specification and program via meta-programming. Each Web site application is described in three separate components (Schwabe and Rossi, 1995; Florescu et al., 1998): information content, navigation structure and visualization. Information content refers to data to be displayed in the Web pages. Navigation structure defines the organization of the site and how items of information are related to each other. Finally, visualization concerns how the information will be presented in

the Web pages comprising the site. These separate descriptions allow each component to be changed independently.

An interesting issue is how such a method can be generalized and still be useful. If the synthesizer is very general it is likely to require specialist expertise to control it. On the other hand, if the synthesizer is very specific it will only be able to produce Web sites for a narrow range of applications. We want to find a balance between these two extremes.

The following section presents a discussion about related works. Section 2 presents a general discussion of the approach and architecture of the system. Section 3 discusses related work. In Section 4 the formalism used for application descriptions is discussed. Sections 5 to 7 present the details of the Web site synthesis approach. Section 8 discusses an example of Web site synthesis. In Section 9 an evaluation of the approach is discussed, followed by some concluding remarks.

2. A Three-Level Approach to Web Site Synthesis

The main idea behind the proposed approach is to derive a Web site automatically from an application and a related visualization description. This requires the use of an appropriate formalism and we use a logic for this task. However, few people feel comfortable using a general-purpose logic directly. One way to deal with this issue is to offer a suitable interface to the designer based on a task- or domain-specific formalism and translate the resulting description into logic expressions. As a result, the synthesizer is organized into three different levels. The architecture of our synthesizer is illustrated in Fig. 2.

We begin with a high-level description of an application. From the application description an intermediate representation is automatically derived. Finally, this intermediate representation is combined with a visualization description to generate a corresponding Web site code automatically.

The intermediate representation defines the structure of the Web site as a graph, in which each node is a set of pieces of information and edges corresponding to links between pieces of information. It defines navigational paths between all pieces of information that should be presented in the site. This second level allows the definition of a more flexible Web site generation process because it is independent of any particular implementation.

Navigation structure is automatically derived by the system from the application specification. It is described in the intermediate representation (Level 2) by logic expressions which we call transition rules. Section 5.2 explain transition rules in detail.

A particular visualization for the site is also automatically derived by the system. The visualization description (Level 2) provides templates for the Web pages and specific visualizations for individual pieces of information. Visualization descriptions are detailed in Section 7.

Independence between application and visualization specifications gives us the ability to produce different Web sites by combining the same application specification with different visualization descriptions. Formally, given a transition rule set T_i and a visualization specification V_j, a Web site W_{ij} can be automatically synthesized. The ideal situation, depicted in Fig. 3, is that every transition rule set is compatible with any visualization specification. This means that a particular visualization specification can be reused in different Web sites. For instance, from three transition rules sets and three visualization specifications it is possible to produce nine different Web sites for the same application.

However, this is not always the case as some visualization specifications may not be compatible with a transition rule set, so there is an empirical issue of how close we get to the ideal of Fig. 3 in practice. We return to this topic in Section 9.

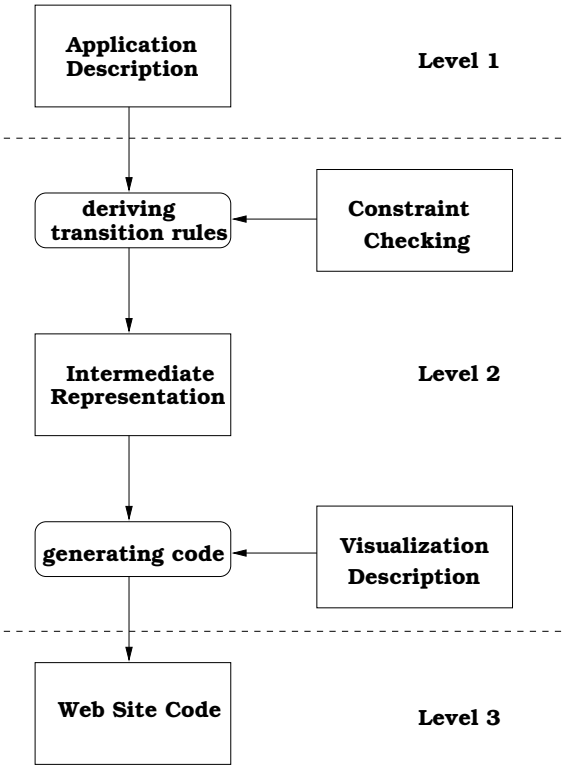


Fig. 2. The three-level approach.

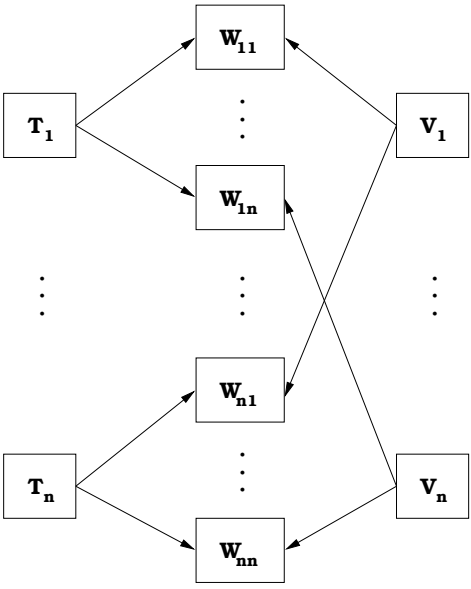


Fig. 3. Ideal transition rule set and visualization specification compatibility.

3. Related Work

Various approaches to Web site design and maintenance have been proposed in recent years. There is general agreement amongst these on some core concepts:

- separation between information content, navigation structure and visualization;
- declarative specifications, by means of high-level conceptual data models or declarative languages;
- automated or semi-automated generation of Web site by means of CASE tools.

The main differences between the different proposals are in the emphasis given to particular aspects of the process. Most studies focus on modeling aspects: Araneus (Atzeni et al., 1998), Strudel (Fernández et al., 1998), AutoWeb (Fraternali and Paolini, 1998), OOHDM (Schwabe and Rossi, 1995), WebML (Ceri et al., 1999a); some are data-driven: Torii (Ceri et al., 1999b); and others are based on semantic descriptions: OntoWebber (Jin et al., 2001), SEAL (Maedche et al., 2001). Our approach is a data-driven approach, which focuses on the generation of different visualizations and on associated Web site maintenance.

Where the research is driven by modeling most approaches are based on traditional conceptual data models, such as the entity-relationship model (ER) and its extensions or object-oriented data models. In this category we can include Araneus, Torii, WebML, AutoWeb and OOHDM. WebML proposes a structural model compatible with ER, ODMG object-oriented data model and UML class diagrams. AutoWeb is based on the HDM-lite model, which is a Web-specific version of HDM. OOHDM is also an object-oriented extension to HDM. Strudel models a Web site as graphs. OntoWebber and SEAL are based on DAML + OIL and RDF, respectively, which are used to define ontologies describing the application domain. Our approach can support different conceptual data models. We advocate the idea of using existing data models, providing the appropriate mapping procedures to our intermediate representation.

Support for heterogeneous data sources is offered by OntoWebber and Strudel. Some limited support (for relational database systems only) is offered by Torii. This issue is not very well exploited by SEAL, AutoWeb or OOHDM.

Generation of different visualizations for the same specification and personalization of Web sites is supported by most approaches. Query languages and templates are the main tools used for this purpose. OntoWebber, SEAL, Torii, WebML, AutoWeb and Strudel all support this feature. Our approach provides support for combining declarative descriptions of alternative visualizations with templates in different target languages.

Another interesting feature is the support for integrity constraints, which is given by OntoWebber, Torii and Strudel.

Maintenance is not explored deeply by most approaches, although many claim support or some degree of automation, such as OntoWebber and Araneus.

Our work is distinct by offering a framework for Web site construction based on computational logic. This feature makes it more convenient to introduce reasoning capabilities, supporting definition of integrity constraints (am issue also addressed by OntoWebber, Torii and Strudel) and rules for both navigation and visualization. Our approach is also highly extensible, either supporting different data sources or different target languages for generating Web pages.

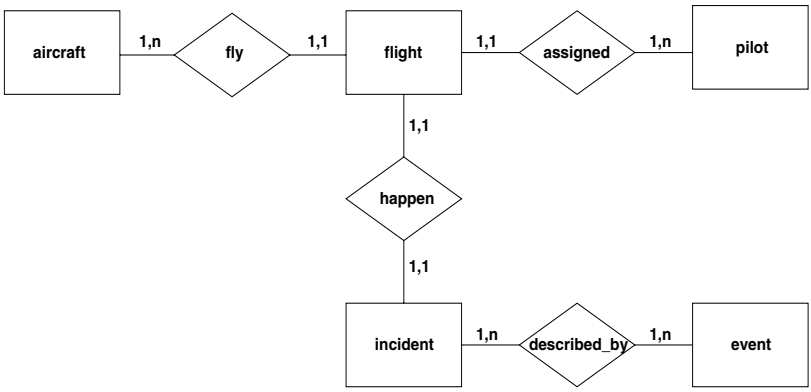


Fig. 4. Accident-reporting ER diagram.

4. Application Description using an Entity-Relationship Model

As we noted in Section 2, we require as a precursor to Web site generation a model of the task or domain for which the site is to be generated. The language in which this model is described should be as general as possible, so it may be widely applied, but it must also be accessible to application-specific engineers, which requires some commitments to accepted engineering notation. One way of reconciling this tension between generality and specificity is to follow a standard view of a class of Web design applications. One such standard view is to provide an information system built around one or more databases with an accompanying navigation structure. This has many similarities with traditional non-Web systems and allows us to use for our domain a standard notation for data modeling. We have chosen the Entity-Relationship model (ER) due to its popularity and widespread use in application modeling, although other conceptual data models are possible. The Entity-Relationship model originally proposed by Chen (1976) has undergone many extensions, leading to a large number of slightly different data models. We follow the concepts defined in the Enhanced Entity-Relationship model (Elmasri and Navathe, 1999). This adds the concepts of specialization, generalization and categories.

Throughout this paper we use as a working example a Web site for an accident-reporting application. The Web is an attractive medium for an accident-reporting application domain because it allows easy access to information both to experts and the general public. It is also a data-intensive Web site, as new reports or updates on existing reports are very frequent. Figure 4 shows an ER diagram to an accident-reporting application.

The application concerns incidents that have happened during scheduled flights (cargo or passenger). Some information about the aircraft and the pilot is also of interest. Incidents are related to events which can be of three different types: descriptions (weather conditions, flight level, etc.), nature of problem (mechanical failure, collision, fire, etc.) or actions (by the crew or instructions by air traffic control).

Given that existing techniques for mapping an ER diagram into a corresponding relational schema are available (Elmasri and Navathe, 1999), we assume that a relational database is created. As an example, the corresponding relational database schema to entities aircraft, flight and the relationship fly (including attributes and data types) is

presented below. It also includes indication of foreign key attributes where appropriate. Note that relationship fly is represented by attribute aircraft_registration in table flight.

aircraft

Attribute	Data Type	Key	Foreign Key
model	string		
registration	string	√	
no_of_engines	integer		
type_of_engine	string		
year_of_manufacture	integer		

flight

Attribute	Data Type	Key	Foreign Key
number	integer	√	
date	date	√	
type_of_flight	string		
crew	integer		
passengers	integer		
aircraft_registration	string		√
commanders_license_no	integer		√
incident_no	integer		√

The ER schema presented in Fig. 4 is the starting point to the Web site synthesis and all examples in the next sections will refer to the data items defined here.

5. The Intermediate Representation

In this section we concentrate on the intermediate representation, which corresponds to the second level presented in Fig. 2.

5.1. Pieces of Information

A piece of information can directly correspond to a database item or it can be derived from the database as a result of queries. These are represented as additional facts or simple rules.

A piece of information has the format Label(Type, Info), where

- Label is used as additional information about the data item giving a specific context to the information. It usually corresponds to an attribute name.
- Type is a predefined data type.
- Info describes a cluster of information associated with the label.

This notation is very useful to identify pieces of information individually and relate them to other components such as pages in which they appear and a particular presentation. We have defined the following data types: integer, float, string, list, tuple, table, image. Type list includes elements of a same type. tuple is a list of primitive elements of different types. Table is a list of tuples.

```
aircraft_model(string, Model) ← aircraft(Model, _, _, _, _)

aircraft_instance(tuple, [Model, Reg, NoEngines, TypeEngine, Year]) ←
                    aircraft(Model, Reg, NoEngines, TypeEngine, Year)

all_aircraft(table, T) ← T = {[Model, Reg, NoEngines, TypeEngine, Year] |
                    aircraft(Model, Reg, NoEngines, TypeEngine, Year)}
```

Fig. 5. Definition of pieces of information.

Figure 5 presents definitions of pieces of information corresponding respectively to an aircraft model, an instance of an aircraft and a table with all instances of aircraft. In this example constants start with a lower-case letter and variables start with a capital letter.

Pieces of information are the building blocks of our approach to Web site construction. They also represent access between pieces of information which will be achieved in the synthesized Web site by constructing hyperlinks to other pages. Instantiation of pieces of information is performed within transition rules, as we explain in Section 5.2.

Dynamic Web sites have the ability to present information generated 'on the fly'. One way to do this is to present an HTML form and trigger an operation after user input. The proposed approach also supports the specification and automated generation of operations. Operations are implemented as CGI programs written in Prolog which are associated to HTML forms defined in the Web pages. Due to scope and space in this paper we omit the details of operation synthesis. These can be found in Vasconcelos et al. (2000).

5.2. Transition Rules

A general view of site navigation is a sequence of actions, where each action is the display of a set of pieces of information in a unit of display, possibly followed by a transition to another set of pieces of information. Transition rules thus represent the navigation structure of the site.

The predicate display is defined for describing the action of presenting information in a particular unit of display. The predicate display has the form:

```
display(InfoList)
```

where InfoList is a list of pieces of information.

The binary operator $\Rightarrow$ specifies transitions from one page to another. Using this operator together with the predicate display we can specify all transitions of an application. A transition expression has the form:

$$\begin{aligned} &\mathsf{display}([\mathsf{Info}_0]) \Rightarrow \mathsf{display}([\mathsf{Info}_1, \ldots, \mathsf{Info}_n]) \leftarrow \\ &\qquad \mathsf{p}_0(\mathsf{Info}_0) \wedge \\ &\qquad \mathsf{p}_1(\mathsf{Info}_1) \wedge \\ &\qquad \vdots \\ &\qquad \mathsf{p}_n(\mathsf{Info}_n) \end{aligned}$$

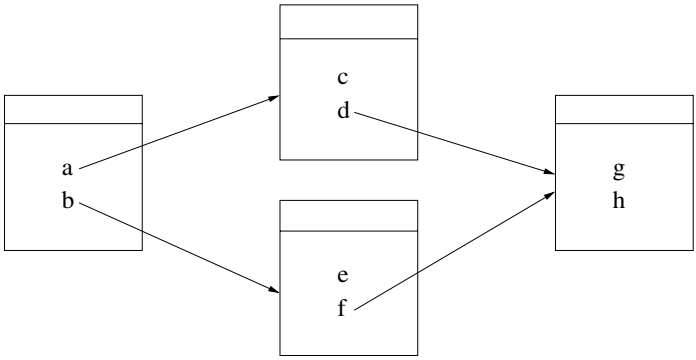


Fig. 6. A Web site example.

This rule states that after Info_0 is displayed the information set $[\mathsf{Info}_1, \ldots, \mathsf{Info}_n]$ can be displayed. Predicate p_i corresponds to a predicate that instantiates each corresponding piece of information Info_i. The piece of information Info_0 defines a link to the set $[\mathsf{Info}_1, \ldots, \mathsf{Info}_n]$.

Note that the display expression on the left-hand side of the rule usually includes only one piece of information. This piece of information must be a subset of another set of pieces of information appearing on the right side in another transition rule. By inspecting subset relations between pieces of information on the left and sets of information on the right side of transition rules, the navigation structure of a Web site is defined.

For example, the following transition rules define the Web site of Fig. 6. Pieces of information are represented as atoms for simplicity, although they are normally more complex structures as we see later.

$$\mathsf{display}([a,b]) \leftarrow \\ \quad \mathsf{p}_a(a) \wedge \mathsf{p}_b(b)$$

$$\mathsf{display}([a]) \Rightarrow \mathsf{display}([c,d]) \leftarrow \\ \quad \mathsf{p}_c(c) \wedge \mathsf{p}_d(d)$$

$$\mathsf{display}([b]) \Rightarrow \mathsf{display}([e,f]) \leftarrow \\ \quad \mathsf{p}_e(e) \wedge \mathsf{p}_f(f)$$

$$\mathsf{display}([d]) \Rightarrow \mathsf{display}([g,h]) \leftarrow \\ \quad \mathsf{p}_g(g) \wedge \mathsf{p}_h(h)$$

$$\mathsf{display}([f]) \Rightarrow \mathsf{display}([g,h]) \leftarrow \\ \quad \mathsf{p}_g(g) \wedge \mathsf{p}_h(h)$$

The initial page, also known as the homepage, needs a different rule because there are no links to it. The homepage is defined by a rule of the form:

$$\mathsf{display}([\mathsf{Info}_1, \ldots, \mathsf{Info}_n]) \leftarrow \\ \quad \mathsf{p}_1(\mathsf{Info}_1) \wedge \\ \quad \vdots \\ \quad \mathsf{p}_n(\mathsf{Info}_n)$$

```
display([number(integer, N), date(date, D), aircraft_reg(string, AR)]) ←
        flight(N, D, _, _, _, AR, _)

display([aircraft_reg(string, AR)]) ⇒
display([aircraft_model(string, AM), aircraft_reg(string, AR), year(integer, Y)]) ←
        flight(_, _, _, _, _, AR, _) ∧
        aircraft(AM, AR, _, _, Y)
```

Fig. 7. Example of transition rules.

An example of transition rules for aircraft and flight pages is given by Fig. 7.

In this example, each instance of aircraft is displayed in a Web page. From an aircraft page there is a link to a flight page via the piece of information aircraft_reg. This link corresponds to the relationship between entities aircraft and flight as defined in the ER diagram presented in Fig. 4.

A complete intermediate representation for an application includes a set of transition rules and pieces of information. A set of pieces of information defines the content of a Web page. Transition rules define how to navigate between those pages.

5.3. Constraints on Paths

Constraints can be used to enforce an order of information presentation. A very common constraint of this sort appears in electronic commerce Web sites, where information about the purchase and the total amount must be displayed *before* the customer provides the payment information. Similarly, a confirmation of payment must be displayed *after* checkout. With a complex navigational structure constraints are important to prevent errors.

We use two concepts from Transaction Logic (TR) (Bonner and Kifer, 1995), serial conjunction and path, that were adapted to represent the sort of constraints we need. Serial conjunction is used to represent a sequence of actions. This is written in the form $\mathsf{a} \otimes \mathsf{b}$ to define a path formed of action a followed by action b.

In a Web site context a path is simply a sequence of information displays. Hence constraints on a Web site can be expressed in terms of valid/invalid paths. Paths can be derived from the site graph, where nodes correspond to pages and edges correspond to transitions. The site graph is easily built by inspecting the transition rules. We assume that a finite number of acyclic paths (non-looping paths) can be extracted from the transition definitions. This is not a constraint on the generation of paths, but constraint checking only considers acyclic paths.

The simplest path contains a single element which is a set of pieces of information, as defined in Section 5.1. Hence path expressions are of the form:

$$\mathsf{display}(\mathsf{InfoSet}_1) \otimes \mathsf{display}(\mathsf{InfoSet}_2) \otimes \cdots \otimes \mathsf{display}(\mathsf{InfoSet}_n)$$

Another useful concept taken from TR is a special symbol path which corresponds to a sequence of actions of any length. This concept allows us to write simplified expressions. For example, the expression:

$$\mathsf{path} \otimes \mathsf{display}(\mathsf{InfoSet}_1) \otimes \mathsf{display}(\mathsf{InfoSet}_2) \otimes \mathsf{path}$$

denotes any path that displays pieces of information in $\mathsf{InfoSet}_1$ which is immediately followed by the display of $\mathsf{InfoSet}_2$. From the transition rule defined in Fig. 7 the following path expression can be derived:

display([aircraft_reg]) ⊗ display([aircraft_model, aircraft_reg, year])

Constraint expressions are similar to path expressions, basically imposing an order to the presentation of information. The following table presents some common constraint expressions:

Expression	Interpretation
¬ (**path** ⊗ ¬ display($\mathsf{InfoSet}_1$) ⊗ **path** ⊗ display($\mathsf{InfoSet}_2$) ⊗ **path**)	Information in $\mathsf{InfoSet}_1$ must be displayed before information in $\mathsf{InfoSet}_2$.
¬ (**path** ⊗ ¬ display($\mathsf{InfoSet}_1$) ⊗ display($\mathsf{InfoSet}_2$) ⊗ **path**)	Information in $\mathsf{InfoSet}_1$ must be displayed immediately before information in $\mathsf{InfoSet}_2$.
¬ (**path** ⊗ display($\mathsf{InfoSet}_1$) ⊗ **path** ⊗ ¬ display($\mathsf{InfoSet}_2$) ⊗ **path**)	Information in $\mathsf{InfoSet}_2$ must be displayed after information in $\mathsf{InfoSet}_1$.
¬ (**path** ⊗ display($\mathsf{InfoSet}_1$) ⊗ ¬ display($\mathsf{InfoSet}_2$) ⊗ **path**)	Information in $\mathsf{InfoSet}_2$ must be displayed immediately after information in $\mathsf{InfoSet}_1$.

Data types and values of pieces of information are omitted in path and constraint expressions because they play no role in constraint checking. As a result constraint and path expressions are simplified, including only information labels.

The designer is the one responsible for entering the appropriate constraints. A simple user interface can be constructed to help in building constraint expressions as only information labels are needed. Standard expressions such as those presented in the table above can also be used to guide the definition of constraints.

As an example, if we want information about an aircraft only to be reached after viewing the related flight page, the following constraint may be defined:

¬ (**path** ⊗ ¬ display([number, date, aircraft_reg]) ⊗ **path** ⊗ display([aircraft_model, aircraft_reg, year]) ⊗ **path**)

A site graph can easily be derived from transition rules, in which sets of pieces of information correspond to nodes and links correspond to edges. From the site graph paths can be derived. Finally, matching path expressions with constraint expressions we can check if constraints hold. This ensures that all paths are valid ones.

6. Deriving Transition Rules from ER Schemata

Given that an ER schema is provided by the Web site designer our task is automatically to define a corresponding intermediate representation. A fully automated process at this step is necessary because the details of the intermediate representation are hidden from the Web site designer.

It is possible to derive different sets of transition rules from one ER diagram, allowing the construction of different Web sites for the same application. This is an interesting solution for rapid application prototyping as the only interaction with the site designer is the ER diagram input and style choices for pieces of information and Web pages.

Fig. 8. Transition rule corresponding to entities.

A standard navigation structure and a layout for visualization are also automatically produced.

Here we suggest two different transition rule sets, which are called entity-based and instance-based. The first defines a Web page for each entity, placing all instances of that entity together. The second option defines a Web page for each instance of an entity. Links between pages are derived from relationships between entities. The mapping procedures for these two approaches are described in the next two sections.

There are some additional auxiliary predicates used in the transition rules defined in the next two sections. Some of these predicates correspond to the ER model concepts and they are self-explanatory. Other predicates are used to instantiate values. These predicates are:

- instantiate_all_instances(E, Info): Info is instantiated to a structure including all instances of entity E. A relationship can also be used in place of an entity.
  For example, the result for instantiate_all_instances(aircraft, Info) is

  Info = [[model(string, 'Douglas C47 Dakota 4'), reg(string,'G-AMPZ'), engine_no(integer, 2), engine_type(string, 'P&W R1830-92 piston engines'), year_manufacture(integer, 1944)], [model(string, 'Boeing 777-200B', reg(string,'N784UA'), engine_no(integer, 2), engine_type(string, 'P&W PW 4090 turbofan engines'), year_manufacture(integer, 1997)], …].

- instantiate_instance(E, K, Info): Info is instantiated to one instance of entity E, given that key attribute K is already instantiated. A relationship can also be used. instantiate_instance(aircraft, ['N784UA'], Info) would result in:

  Info = [model(string, 'Boeing 777-200B'), reg(string,'N784UA'), engine_no(integer, 2), engine_type(string, 'P&W PW 4090 turbofan engines'), year_manufacture(integer, 1997)]

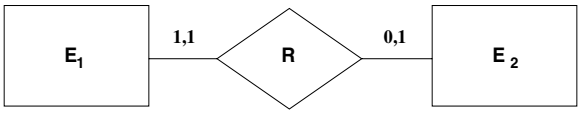
- instantiate_val(E, Attr, Var): instantiate the variable Var with an instance value of attribute Attr belonging to entity E. A relationship can also be used in place of an entity. For example, in instantiate_val(aircraft, [model], Var) the result can be: Var = 'Boeing 777-200B'.

6.1. Entity-based Transition Rules

For each entity a transition rule is created to instantiate all instances in a page as described by Fig. 8.

Info$_E$ is a piece of information composed of all instances of that entity. The entity name is also used as the label for each piece of information.

Transition rules related to relationships depend on the cardinality constraint defined. Relationships N : N require the construction of separate Web pages whereas 1 : 1 and 1 : N relationships are represented by links in the related entities Web pages. Figure 9



display(Info$_{E1}$) ⇒ display(Info$_{E2}$) ←
relationship(_, E1, _, 1, E2, _, _, _) ∧
instantiate_all_instances(E2, Info$_{E2}$)
display(Info$_{E1}$) ⇒ display(Info$_{E2}$) ←
relationship(R, E2, _, _, E1, _, 1, _) ∧
instantiate_all_instances(E2, Info$_{E2}$)

Fig. 9. Binary relationship 1:1 and its transition rules.

shows the transition rules derived from 1 : 1 and 1 : N relationships. Rules for N : N and n-ary relationships are presented in Fig. 10.

Transition rules corresponding to ternary and n-ary relationships are similar to those related to binary N:N relationships.

For hierarchies of generalization and specialization, the corresponding rule is defined in Fig. 11.

Applying these rules in the ER diagram described in Fig. 4, the resulting structure is illustrated by Fig. 12:

6.2. Instance-Based Transition Rules

In this transition rule set, a page is created for each instance of an entity. Transitions now are defined in terms of relationship between instances as described in Fig. 13. They follow the same pattern of entity-based transition rules, but for each entity or relationship instance there will be an individual page. Key attributes are used as pages identifiers.

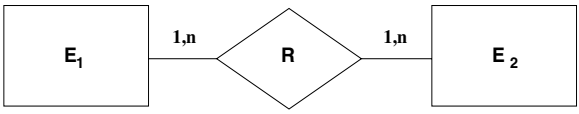
The labels and data types of each piece of information depend on the attributes, names and types of each entity or relationship. As a result of transition rules, a similar information structure to the entity-based one is created. In this case, each information set corresponds to a particular instance of an entity or relationship. Figure 14 illustrates the information structure resulting from the transition rule related to the relationship between aircraft and flight.

Given these two choices for generating transition rules, the last task is to define a visualization for those pieces of information. This issue is discussed in the next section.

7. Visualization

One of the main ideas of our approach is separation between the application information content from visualization features. We have described so far how to model and develop a Web application from the data and navigation point of view. Now we should explain how to relate those to an appropriate visualization.

In order to connect information content with a particular visualization we use data types as defined in Section 5.1. Similarly a set of visualization styles is defined. Examples of such styles include text, table, enumerated_list, itemised_list. By associating visualization styles to data types, visualization description becomes completely independent from any particular application.



display(Info$_R$) ←
relationship(R, _, _, _, _, _, _, _) ∧
instantiate_all_instances(R, Info$_R$)

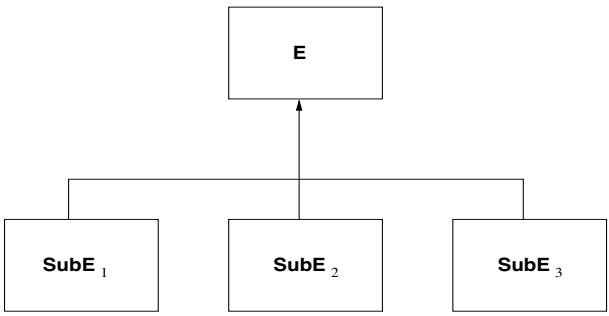
display(Info$_{E1}$) ⇒ display(Info$_R$) ←
relationship(R, E1, _, n, _, _, n, _) ∧
instantiate_all_instances(R, Info$_R$) ∧
instantiate_all_instances(E1, Info$_{E1}$)

display(Info$_{E2}$) ⇒ display(Info$_R$) ←
relationship(R, _, _, n, E2, _, n, _) ∧
instantiate_all_instances(R, Info$_R$) ∧
instantiate_all_instances(E1, Info$_{E2}$)

display(Info$_R$) ⇒ display(Info$_{E1}$) ←
relationship(R, E1, _, n, _, _, n, _) ∧
instantiate_all_instances(R, Info$_R$) ∧
instantiate_all_instances(E1, Info$_{E1}$)

display(Info$_R$) ⇒ display(Info$_{E2}$) ←
relationship(R, _, _, n, E2, _, n, _) ∧
instantiate_all_instances(R, Info$_R$) ∧
instantiate_all_instances(E2, Info$_{E2}$)

Fig. 10. Binary relationship N:N and its transition rules.



display(Info$_E$) ⇒ display(Info$_{SubE}$) ←
specialisation(E, ListofSubEntities) ∧
SubE ∈ ListofSubEntities ∧
instantiate_all_instances(SubE, Info$_{SubE}$) ∧
instantiate_all_instances(E, Info$_E$)

Fig. 11. Transition rule derived from specialization hierarchies.

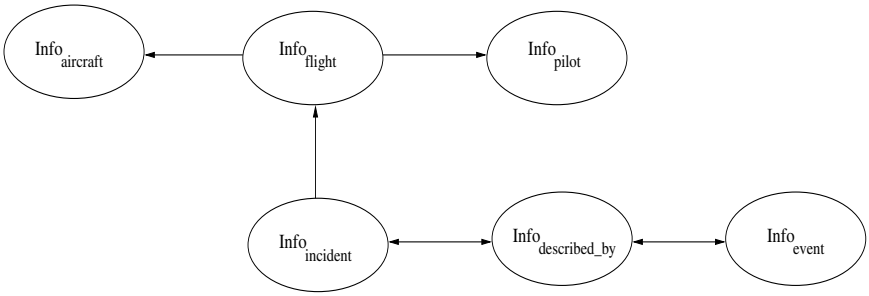


Fig. 12. Information structure – Entity-based.

display([KeyInfo, OtherInfo]) ←
entity(E) ∧
key(E, K) ∧
instantiate_val(E, K, Keyinfo) ∧
instantiate_instance(E, K, OtherInfo)

display([$KeyInfo_1$, $KeyInfo_2$, OtherInfo]) ←
relationship(R, E_1, _, n, E_2, _, n, _) ∧
key(E_1, K_1) ∧
key(E_2, K_2) ∧
instantiate_val(R, [K_1, K_2], [$KeyInfo_1$, $KeyInfo_2$]) ∧
instantiate_instance(R, [$KeyInfo_1$, $KeyInfo_2$], OtherInfo)

display([$KeyInfo_1$]) ⇒ display([$Keyinfo_2$, $OtherAttribs_2$]) ←
relationship(R, E_1, _, 1, E_2, _, _, _) ∧
key(E_1, K_1) ∧
key(E_2, K_2) ∧
instantiate_val(E_1, K_1, $KeyInfo_1$) ∧
instantiate_val(E_2, K_2, $KeyInfo_2$) ∧
instantiate_instance(E_2, K_2, $OtherAttribs_2$)

display([$KeyVal_1$]) ⇒ display([$KeyVal_2$, $OtherAttribs_2$]) ←
specialisation(E, ListofSubentities) ∧
SubE ∈ ListofSubentities ∧
key(E, K_1) ∧
key(SubE, K_2) ∧
instantiate_val(E, K_1, $KeyVal_1$) ∧
instantiate_val(SubE, K_2, $KeyVal_2$) ∧
instantiate_instance(SubE, K_2, $OtherAttribs_2$)

Fig. 13. Transition rules for instance-based visualization.

A specific visualization can be associated to data types using the following expression:

style(Type, Style)

where Style is a predefined visualization style and Type is a data type.

Using this specification, pieces of information can be translated into corresponding HTML code, which is our target language.

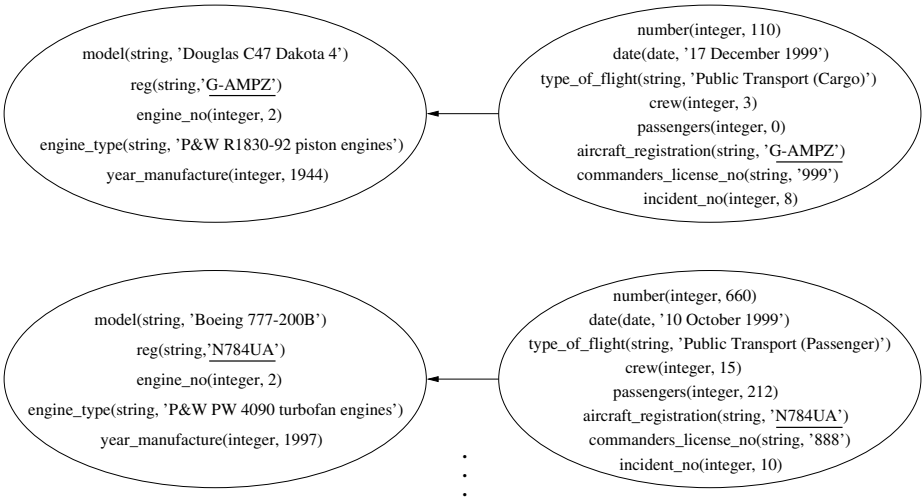


Fig. 14. Information structure – instance-based.

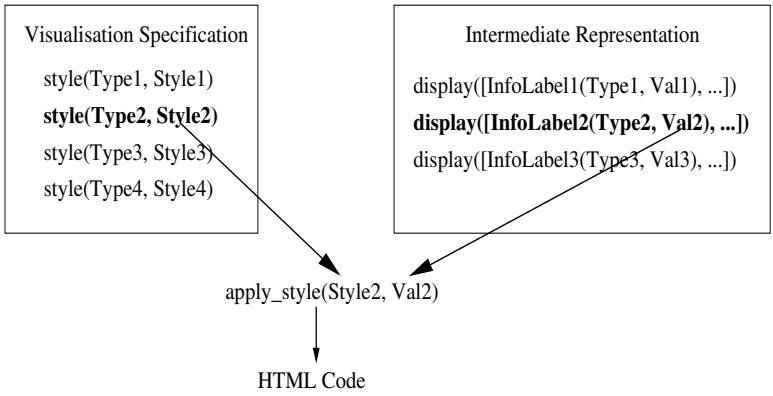


Fig. 15. Applying styles to pieces of information.

Figure 15 illustrates how pieces of information presented in Fig. 5 can be related to a visualization style. The data type of a piece of information is used to find the related visualization style, then the selected style is applied to the instantiated information and finally transformed into HTML code. We make use of the Pillow library (Cabeza and Hermenegildo, 1997), which provides facilities for translating Prolog terms into HTML terms.

Additional details such as font type, font size, colors and text alignment are defined in a CSS style sheet (W3C, 1999). A suitable interface should be offered to the designer in order to input all necessary parameters to the style sheet. Currently, a standard CSS style sheet is automatically generated including definitions for each style.

The reason to make use of style sheets is to keep the representation for our visualization styles simple. Without a style sheet, details such as colors and fonts should be included as arguments to the mapping procedure to translate a visualization style to HTML.

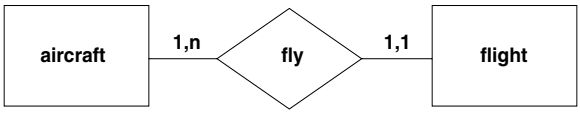


Fig. 16. Partial ER diagram for accident-reporting application.

The use of CSS style sheets provides an additional flexibility for defining visualizations for the site as it can be changed without any impact on other specification levels. An example of a style sheet is given in Section 8.

Given that information content and styles are separately defined it is easy to change the visualization of a piece of information. This can be done in different ways:

1. Changing the style associated with the piece of information data type. In this case all pieces of information of that type will also have their style changed.
2. Changing the piece of information data type. As a result, the information is displayed according to the new data type style.
3. Changing the CSS style sheet. This allows modifying colors, font types and sizes and margin alignments.

8. Example of Web Site Synthesis: An Accident-Reporting Web Site

In this section we present an example of Web page synthesis from the initial specification until the final code using the application discussed so far, an accident-report Web site. The example presented here uses data from the United Kingdom Air Accidents Investigation Branch (AAIB: http://www.open.gov.uk/aaib/).

8.1. Mapping the ER Schema to a Corresponding Set of Transition Rules

From the ER diagram presented in Fig. 4 transition rules are automatically derived. Entity-based rules instantiate pieces of information as structured lists including all instances of an entity or relationship. Instance-based rules instantiate each instance of an entity as an individual piece of information.

In order to save space, we concentrate on the generation of the aircraft Web page. Figure 16 shows the part of the ER diagram of the accident-reporting application used in this example.

The example uses the following data about aircraft as it would be found in the application database:

Model	Registration	Engines	Type of engine	Year
Douglas C47 Dakota 4	G-AMPZ	2	P&W R1830-92 piston engines	1944
Boeing 777-200B	N784UA	2	P&W PW 4090 turbofan engines	1997
Airbus A310-304	5YBFT	2	GE CF6-80C2 turbofan engines	1989
Cessna 208B Caravan	LN-PBB	1	P&W PT6A-114 turboprop engine	1992
Boeing 747-136	G-AWNF	4	P&W JT9D-7 turbofan engines	1970
McDonnell Douglas MD11	N1756	3	GE CF6-80 turbofan engines	1992

```
display(Flight_Info) ←
        entity(flight) ∧
        instantiate_all_instances(flight, Flight_Info)
display(Flight_Info) ⇒ display(Aircraft_Info) ←
        relationship(fly, flight, 1, 1, aircraft, 1, n, [ ]) ∧
        instantiate_all_instances(aircraft, Aircraft_Info)

where:

Flight_Info = [all_flights(table, FlightData)]
Aircraft_Info = [all_aircraft(table, AircraftData)]
```

Fig. 17. Transition rules for defining aircraft page.

8.1.1. Entity-Based Transition Rules

The rules presented in Fig. 17 define a Web page for entity aircraft, including all instances of that entity. It also shows the definition of a page for entity flight which includes a link to the aircraft page.

Note that the piece of information all_aircraft is defined by the following expression:

```
all_aircraft(table, AircraftData) ←
        AircraftData = {[Model, Reg, NoEngines, TypeEngine, Year] |
        aircraft(Model, Reg, NoEngines, TypeEngine, Year)}
```

Once we have the transition rules defined, we need a visualization specification such as:

```
style(table, table_style)
```

which defines that data type table is to be displayed using style table_style. This means that any piece of information of type table will be rendered as an HTML table. The resulting Web page is presented in Fig. 18.

Changing the visualization can be done by simply changing the visualization specification, as for example,

```
style(table, text_style)
```

This definition would generate an alternative presentation for the same aircraft page, using the same transition rule set.

8.1.2. Instance-Based Transition Rules

Instance-based rules need to deal with the database instance level. This means that access to actual values is necessary in order to define the relationship between instances. Figure 19 presents the transition rules corresponding to the same ER diagram presented in Fig. 16.

The styles used for this example are:

```
style(integer, labelled_info)
style(string, labelled_info)
style(date, date_style)
```

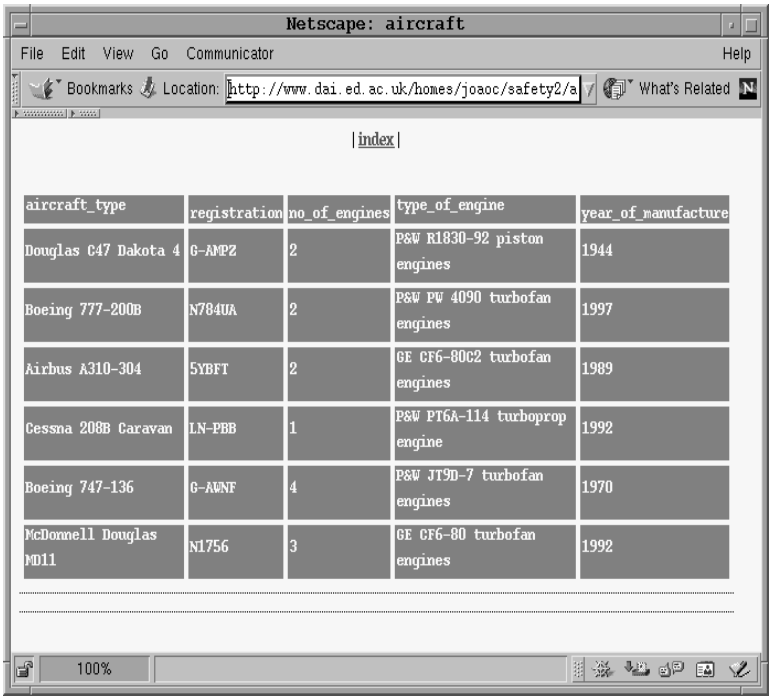


Fig. 18. Aircraft page: entity-based and table visualization.

Figure 20 present the resulting page for one instance of aircraft. For all other instances similar pages are created.

8.2. Deriving a Site Navigator

In almost all Web site applications there is a set of pages which should be referenced by all the others. In most cases it is desirable that links to these pages are grouped together and presented in a standard form in every page of the site. We call this set of links 'navigator'.

The navigator can be derived automatically by inspecting the topological features of the Web site graph. Alternatively, it can be explicitly defined by the designer. A specific visualization for the navigator is defined in the same way as any other piece of information using predicate style:

style(navigator, navigator_style)

where navigator_style defines a specific presentation style for the navigator.

One way to find out which pages should belong to a site navigator is by inspecting the site graph. The simplest case is that of a page which is linked from all other pages of the site. The cardinality of its neighbor's set and intersections between other pages' neighbours set can help in deciding if a page should be included in the navigator.

Once a navigator is defined it is also treated as a piece of information and will be presented in all pages generated. The visualization of the navigator depends on the style defined for it and its place in a page depends on the layout of the page template.

```
display([FlightKeyVal, FlightOtherVal]) ←
        entity(flight) ∧
        key(flight, FlightKey) ∧
        instantiate_val(flight, FlightKey, FlightKeyVal) ∧
        instantiate_instance(flight, FlightKeyVal, FlightOtherVal)

display(FlightKeyVal) ⇒ display([AircraftKeyVal, AircraftOtherVal]) ←
        relationship(fly, flight, _ , 1, aircraft, _ , _ , _) ∧
        key(flight, FlightKey) ∧
        key(aircraft, AircraftKey) ∧
        instantiate_val(flight, FlightKey, FlightKeyVal) ∧
        instantiate_val(flight, AircraftKey, FlightForeignKeyVal) ∧
        instantiate_instance(aircraft, FlightForeignKeyVal, AircraftOtherVal)

where:

FlightKey = [number, date]

FlightKeyVal = [number(integer, 110), date(date, '17 December 1999']

FlightOtherVal = [type_of_flight(string, 'Public Transport (Cargo)'),
        crew(integer, 3), passengers(integer, 0),
        aircraft_registration(string, 'G-AMPZ'), incident_no(integer, 8),
        commanders_license_no(integer, 1111)]
```

Fig. 19. Transition rules for defining aircraft page.

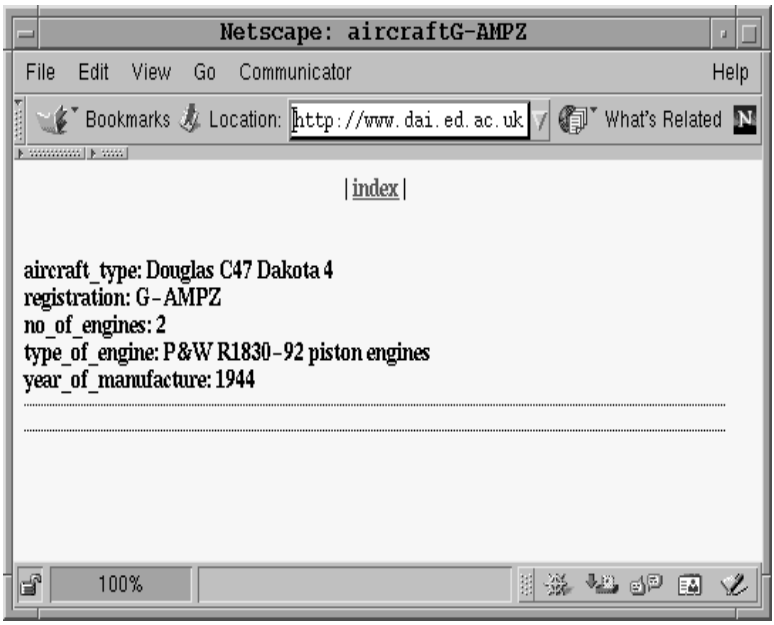


Fig. 20. Aircraft page: instance-based and text visualization.

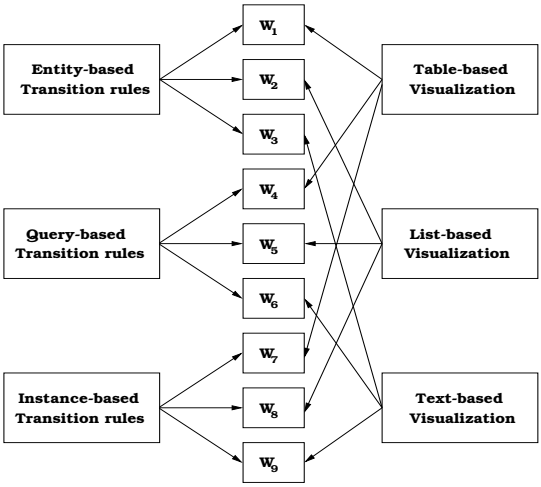


Fig. 21. Combination of transition rules and visualizations.

9. Evaluation

The example introduced in the previous section shows that four different Web sites can be generated for this application from the same entity-relationship schema. We have defined two transition rule sets (entity-based and instance-based) and two visualization specifications (table-based and text-based). If more transition rule sets and visualization specifications are defined, the number of different Web sites that can be produced will be greater, the number of Web sites being the product between the number of transition rule sets and visualization specifications. Figure 21 illustrates the combination of transition rules and visualizations.

The advantage of this approach is the greater number of different Web sites produced using fewer specifications. Traditional Web site synthesis requires the same number of specifications as the number of Web sites produced. The reason is the lack of separation between application and visualization specifications.

For example, from three transition rules sets and three visualizations (six specifications), nine Web sites can be constructed in total (Fig. 21). From four transition rules sets and five visualizations (nine specifications), 20 different Web sites can be constructed. Traditional approaches would require nine and 20 specifications respectively, to construct the same number of sites.

The ability of combining a Web site application specification (including content and navigation) with different visualization descriptions dramatically reduces maintenance effort. This is clear even in the small example presented in the previous section. For example, changing the visualization of the page presented in Fig. 18 can be done simply by changing the data type of each piece of information corresponding to an entity, from table to list. This new visualization is presented in Fig. 22. Similarly, Fig. 23 illustrates another visualization for the same Web page by adjusting parameters related to the table style in the CSS style sheet.

Although it is possible to combine a set of transition rules with different visualization definitions, there are visualizations more appropriate to each kind of application. However, this issue is subjective, making it very difficult to decide automatically which visualization should be applied. In the example presented in Section 8 the table-based

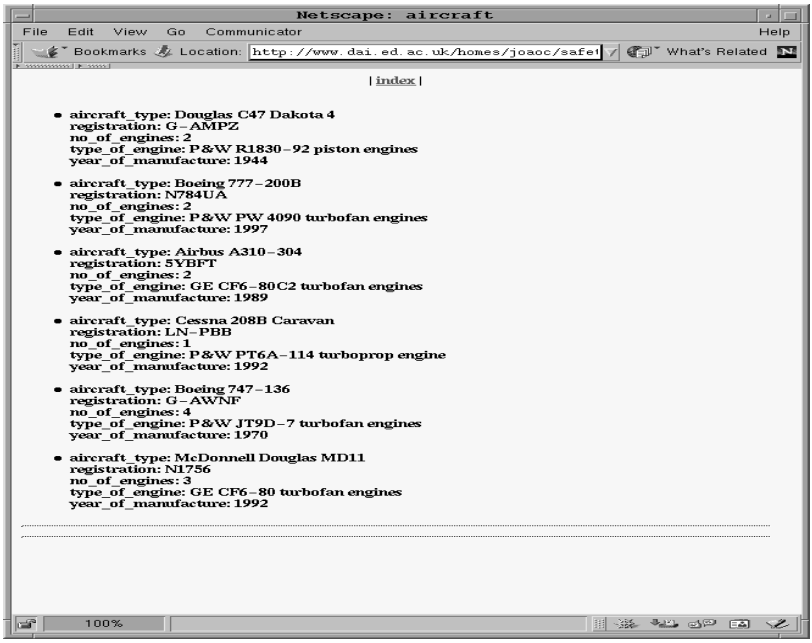


Fig. 22. Aircraft page: entity-based and list visualization.

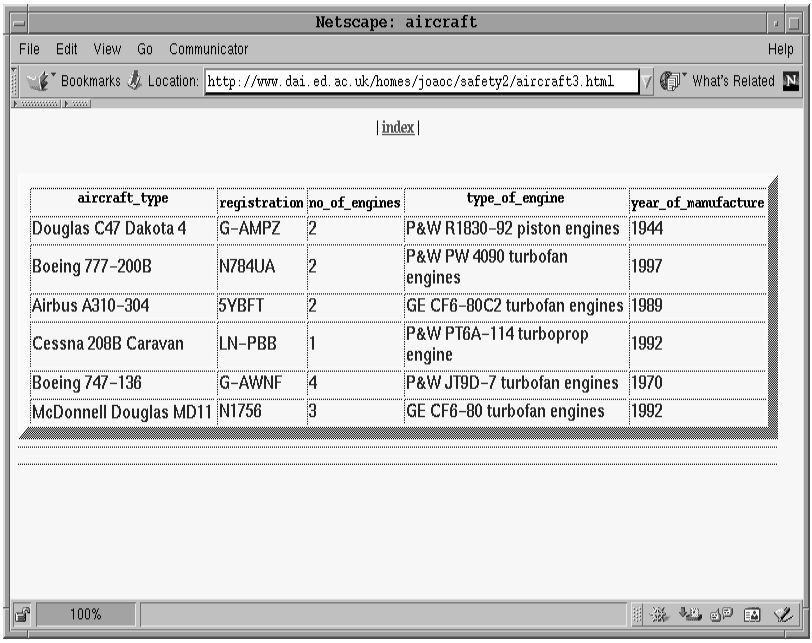


Fig. 23. Aircraft page: entity-based and new table visualization.

visualization is more appropriate to entity-based rules than text-based presentation. For instance-based rules both visualizations are appropriate.

The example shows that an appropriate relation between data types and styles easily supports the generation of alternative visualizations. A practical application of this feature is defining specific visualization for different classes of users.

10. Concluding Remarks

Declarative specifications of Web site applications offer possibilities to apply automated synthesis techniques to reduce the effort of design and maintenance. We presented an approach to design and maintenance of Web sites based on high-level declarative descriptions of applications. An entity-relationship model is used to describe applications and the resulting diagram is the input to the Web site synthesizer.

This approach has the following main features:

1. declarative specification of an application;
2. separation between information content, navigation structure and visualization specification;
3. automated generation of Web site code from a standard high-level specification based on ER schema;
4. generation of alternative visualizations for the site.

The separation between information content, navigation structure and visualization specification allows separate reasoning in each component:

- Different data models could be used instead of the entity-relationship model to describe the application domain. Many different mapping procedures from a conceptual data model to the intermediate representation can be defined.
- Pieces of information are derived from the conceptual data model and its content extract from the application database.
- From the navigation structure description, a site graph and a navigator can be derived.
- Constraints in the order of information presentation can be enforced by inspecting paths derived from the site graph.
- Visualization styles related to data types support a more flexible approach for defining Web site presentation, as it is independent from the problem domain.
- Fine tuning of visualization of pieces of information is done via CSS style sheets. This allows a simpler description of visualization styles and supports changes in the style sheet without necessarily changing the style.
- The target language, currently HTML/CSS, can also be changed (for example to XML/XSL). Only the mapping procedures from one level to another must be defined, making the approach very flexible and adaptable to new technologies.

Although visualization styles are defined for each piece of information, a general page template is still needed, restricting control over page presentation. A visualization specification language can overcome the need for predefined templates, allowing more customizable and flexible visualization specifications.

The main advantage of this approach is its support for:

- changing the visualization style of a type, which in turn changes the visualization of all pieces of information of that type;
- changing the type of a piece of information, which changes the visualization for that particular piece of information;
- updates on the database without any impact on the items above;
- changes in the general style of the site without necessarily changing the presentational form of pieces of information and vice-versa;
- reuse of visualization descriptions in different application domains.

The benefits of the approach are as follows:

- Complexity is reduced because fewer specifications are needed to produce different Web sites as the same visualization styles can be applied to different transition rule sets.
- The designer concentrates on the application description rather than on the mechanics for producing the Web site. Details of the synthesis process are completely separated from the conceptual level.
- Only the specification of the site should be changed in order to change the Web site, either its structure, visualization or both. There is no need to change the generator program.
- Information content (usually defined in a database) can be altered without any impact on visualization or navigation structure. Via an appropriate interface this task requires little or no technical expertise.

Acknowledgements. The first author is supported by the Brazilian Government through CAPES grant no. 1991/97-3. This work also is supported under the Advanced Knowledge Technologies (AKT) Interdisciplinary Research Collaboration (IRC), which is sponsored by the UK Engineering and Physical Sciences Research Council under grant no. GR/N15764/01. The AKT IRC comprises the universities of Aberdeen, Edinburgh, Sheffield, Southampton and the Open University.

References

Atzeni P, Mecca G, Merialdo P (1998) Design and maintenance of data-intensive web sites. In proceedings of the international conference on extending database technology (EDBT), Valencia, Spain

Bonner AJ, Kifer M (1995) Transaction logic programming. Technical report CSRI-323, Computer Systems Research Institute, University of Toronto, November

Cabeza D, Hermenegildo A (1997) WWW programming using computational logic systems (and the PiLLoW/CIAO library). Technical report, Computer Science Department, Technical University of Madrid. Online at http://www.clip.dia.fi.upm.es/miscdocs/pillow/pillow.html

Cavalcanti J, Robertson D (2000) Synthesis of web sites from high level descriptions. In third international workshop on web engineering, WWW9 conference, Amsterdam, The Netherlands, May. Lecture Notes in Computer Science 2016, Springer, Berlin

Ceri S, Fraternali P, Bongio A (1999a) Web modeling language (WebML): a modeling language for designing web sites. In proceedings of WWW9, Toronto, Canada, May

Ceri S, Fraternali P, Paraboschi S (1999b) Data-driven, one-to-one web site generation for data-intensive applications. In proceedings of the 25th international conference on very large data bases (VLDB'99), 7–10 September, Edinburgh, UK, pp 615–626

Chen PP (1976) The entity-relationship model: toward a unified view of data. ACM Transactions on Database Systems 1(1), pp 9–36

Elmasri R, Navathe S (1999) Fundamentals of database systems, 3rd edn. Benjamin Cummings, Redwood city, CA

Fernández M, Florescu D, Kang J, Levy A, Suciu D (1998) Catching the boat with strudel: experience with a web-site management system. In SIGMOD conference on management of data, Seattle, WA

Florescu D, Levy A, Mendelzon A (1998) Database techniques for the World-Wide Web: a survey. SIGMOD Record 27(3), pp 59–74

Fraternali P, Paolini P (1998) A conceptual model and a tool environment for developing more scalable, dynamic and customizable web applications. In proceedings of the international conference on extending database technology (EDBT), Valencia, Spain

Jin Y, Decker S, Wiederhold G (2001) OntoWebber: model-driven ontology-based web site management. In proceedings of the first international semantic web working symposium (SWWS'01), Stanford University, Stanford, CA, 29 July–1 August

Maedche A, Staab S, Stojanovic N, Studer R, Sure Y (2001) SEAL: a framework for developing SEmantic Web PortALs. In 18th British national conference on databases (BNCOD 2001), Oxford, 9–11 July. LNCS, Springer, Berlin

Robertson D, Agustí J (1999) Software blueprints: lightweight uses of logic in conceptual modelling, ACM Press/Addison-Wesley/Longman

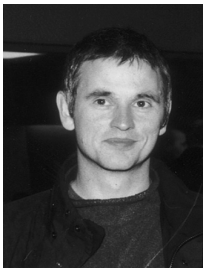
Schwabe D, Rossi G (1995) The object-oriented hypermedia design model. Communications of the ACM 38(8), pp 45–46

Vasconcelos W, Schwitter R, Molla D, Cavalcanti J (2000) Implementing Prolog-run WWW sites. In 13th international conference on applications of Prolog, inap 2000, Waseda University, Tokyo, Japan, October

W3C (1999) Cascading style sheets, level 1. W3C recommendation December 1996, revised January 1999. Online at http://www.w3.org/TR/REC-CSS1

Author Biographies

João Cavalcanti is a Ph.D. student at the Centre for Intelligent Systems and their Applications, part of Informatics at the University of Edinburgh. His research is on computational logic methods for automated Web site synthesis. He is also a lecturer at the Department of Computer Science at the University of Amazonas, Brazil.



David Robertson is the Director of the Centre for Intelligent Systems and their Applications, part of Informatics at the University of Edinburgh. His research is on lightweight uses of formal methods to the design and analysis of complex artefacts such as Web sites or large multi-agent systems. His research group (www.dai.ed.ac.uk/groups/ssp) tackles problems at the intersection of artificial intelligence and software engineering.

Correspondence and offprint requests to: João Cavalcanti, Centre for Intelligent Systems and their Applications (CISA), Division of Informatics, University of Edinburgh, 80 South Bridge, Edinburgh EH1 1HN, UK. Email: joaoc@dai.ed.ac.uk