Chapter 4

OVERVIEW OF DESIGN ISSUES FOR WEB APPLICATIONS DEVELOPMENT

Gustavo Rossi,[1] Daniel Schwabe,[2] Luis Olsina,[3] Oscar Pastor[4]
[1]*LIFIA, Facultad de Informatica, Universidad Nacional de La Plata (also at CONICET) Argentina,* gustavo@lifia.info.unlp.edu.ar
[2]*Departamento de Informática, PUC-Rio, Rio de Janeiro, Brazil,* dschwabe@inf.pucrio.br
[3]*GIDIS_Web, Engineering School, Universidad Nacional de La Pampa, Calle 9 y 110, (6360) General Pico, LP, Argentina,* olsinal@ing.unlpam.edu.ar
[4]*DSIC, Valencia University of Technology, Valencia, Spain,* opastor@dsic.upv.es

4.1 INTRODUCTION

In this part of the book we will focus on design issues in Web applications development. To make the book useful both for practitioners and for researchers, we decided (following the successful style of the IWWOST series) to use a common example throughout the rest of the chapters in which each design method is presented. In the following two sections we summarize the most important aspects that development approaches should consider, and then we briefly present the example that will be solved subsequently in each chapter. The aspects we introduce in Sections 4.2 and 4.3 will be refined in each subsequent chapter of the book.

4.2 ISSUES FOR DESIGN METHODS

For discussion purposes, we have grouped the issues under four headings; in Section 4.7 we discuss some additional issues.

1. data/information, with issues relating to data representation
2. navigation, with issues relating to navigation structure and behavior

3. functionality, with issues relating to application functionality beyond navigation
4. presentation/interface, with issues relating to interface and presentation design

This order reflects to some extent the historical evolution of Web applications over time—from simple "read-only" applications to full-fledged information systems—and also reflects the nature of the methods themselves.

This is in contrast with a more software engineering-centered view, which might place "functionality" first, since it is an aspect of applications that all traditional software design methods have taken into account, followed by navigation, which was the "new" abstraction introduced by hypermedia and the Web.

The issues presented here address different aspects of Web applications, which exist to a greater or lesser extent in any such application. Any method purporting to help design Web applications should, therefore, address a significant number of such issues, if not all.

4.3 DATA/INFORMATION/CONTENT—WHAT IS THE SUBJECT MATTER OF THE APPLICATION?

Web applications, like any application, must deal with information items that constitute the problem domain, i.e., the subject matter of the application. The first aspect that a method must define is how to characterize such information items, providing a data model.

This is not a new problem; it has been dealt with in software development as well as in database design methods. Among the most popular are the entity–relationship (E/R) model and the relational model, which are clear examples of widely used extended data models. Also, the data counterparts of popular object-oriented (OO) models, such as UML and ORM, address this same problem. Although differing in details, they all provide for the definition of information items as composed of attributes, which characterize individual properties of such items. Most methods provide means to describe sets of items with the same attributes, usually through concepts such as "classes" or "entities." In addition, generalization and specialization hierarchies can typically be defined among classes or entities.

While Web-based applications do not present additional requirements for the expressiveness of such models, some methods offer additional abstraction mechanisms that allow one to deal with specific issues of the

contents of Web applications, such as multimedia data types, or with multiple representations for the same information item.

Besides the information items, relations must also be represented. While some methods allow a single relation type, others provide specialized types such as aggregation and composition relations. In addition, cardinality constraints must also be expressed.

When applications must deal with extensive domains, model specification may become very large. Some methods provide modularization primitives that allow the specification to be broken into smaller parts.

The application itself may be represented in many ways. In some models, it is the union of all items; in others, there is a special element that stands for the application itself. For example, in object-oriented methods, it is sometimes called an "application singleton" since it is a single object instance of its class.

4.4 NAVIGATION

Although it might seem obvious, given the widespread experience people have using the Web, many authors do not distinguish navigation from other application functionalities or, frequently, equate it with any event that causes changes in the interface. It is clear that navigation is a salient feature of Web applications, but such authors don't consider it worthy of particular attention during application design; it is simply "another application functionality," and the notion of linking is simply ignored.

On the other hand, most current Web application design methods recognize navigation as an outstanding feature of Web applications and provide models and notations to specify it. Briefly stated, characterizing navigation aspects of a Web application entails defining the "things" being navigated and specifying how the navigation space is structured—what items are reachable from any given item. Since the semantics of navigation is better understood, it justifies providing specialized notations that help users describe this functionality on an appropriate level of abstraction.

Much in the same tradition as in the database world, design method proposers have realized that the items being navigated (variously called *nodes*, *objects*, etc.) are different from the conceptual items that make up the problem domain. Whereas conceptual items represent the information in a task- and user-independent way, navigation items are defined taking into account user requirements, providing a view over conceptual items. The idea is that this abstraction mechanism allows the hiding of unwanted details as well as the grouping of interrelated items, with respect to the user profile and

the task being supported. This is analogous to the "external views" as related to "conceptual views" in traditional database design, where users actually manipulate (external) views over the conceptual objects.

Once navigation items have been characterized, methods must also allow the definition of the navigation space. For any reasonably sized application, defining the navigation space topology directly in terms of only "nodes-and-links" is too restrictive, since the amount of information quickly overwhelms both designers and users. Furthermore, descriptions at this low level miss the opportunity to exploit typically occurring regularities in the navigation space topology.

Many methods introduce higher-level abstractions that allow the navigation space to be defined in a more concise way, such as sets or (indexed, ordered) lists, navigation chains, etc. Such abstractions play an analogous role with respect to navigation topology as classes do with respect to object instances—they allow one to refer to the navigation properties of a large number of nodes (respectively, the structure and behavior properties of objects) with a single primitive (respectively, classes).

Such specifications will, in most cases, translate straightforwardly into implementation mechanisms for dynamic Web sites, where pages are not stored explicitly but are generated dynamically, on demand, combining HTML or XML templates with the appropriate data directly retrieved or computed from application data stored in databases.

The initial focus of most novice designers is on the navigation items that will be made available to users. More experienced designers have realized that equally important are the access structures that will lead users to the desired information items—there is no point in having elaborate, detailed information if the user can't reach it!

Consequently, defining the navigation space of Web applications necessarily entails defining its access structures. Once again, higher-level abstractions are necessary, such as (ordered) lists, guided tours, etc. For example, in our exemplar application described at the end of the chapter, we find indexes for accessing films by each of its specific features, such as genres or actors, or film directors, etc. More opportunistic indexes such as top-selling DVDs or today's recommendations are increasingly being used as well. Additionally, some methods also provide the means to specify items that are always accessible, i.e., items that can be reached from anywhere in the navigation space. For example, easy access to each key functionality is usually provided, for example, television movies, DVDs, access to the user's account or shopping cart, and so on.

4.5 FUNCTIONALITY (BEYOND NAVIGATION) AND TASKS

When the first applications were deployed in the WWW, they were mostly "read-only," meaning they allowed users to browse information (hence the name "Web browsers"!), but not to create or change information items. With the increased sophistication of the run-time environments of Web servers, applications have become increasingly more complex, reaching the current stage where browsers are really interfaces to full-fledged applications allowing the creation and modification of information items, often in a distributed, asynchronous fashion.

Since navigation is recognized as a special kind of functionality, most methods must allow the characterization of navigation states, i.e., the dynamic behavior of the application as the user navigates from node to node. The original browse-only applications had, therefore, only navigation states.

As additional functionality was added, the need arose to deal with its associated states and state changes. Stateful applications were already the norm outside the Web, and design methods proposed a variety of mechanisms to specify them. Web application design methods must allow the specification of such applications and integrate the application states with the navigation states. For example, the check-out operation in electronic stores represents a task that the user must fulfill; it is represented as a set of subtasks, although finally it consists of filling in a set of forms accessed as a sequence of nodes.

The added complexity arises from the combination of this functionality with navigation operations and the sometimes subtle interplay between them. For example, some application functions may only be accessed in certain navigation states, or, more generally, accessible functions may change depending on the navigation state. Conversely, certain navigation alternatives may only be available in certain functionality states; for example, personal data may only be accessed after the user has logged in.

The inclusion of states, coupled with the distributed nature of the WWW, naturally leads to the notion of non-atomic processes and transactions. Many applications allow functionality to be accessed as a sequence of steps, which may sometimes be interspersed with navigation operations. Conversely, some applications have the notion of transactions, which must be implemented over an essentially stateless run-time environment. Design methods must allow the specification of both types of run-time behavior, preferably independently of the run-time environment.

Another source of complexity is the inclusion of multimedia data, which are often combined with elaborate timing and synchronization requirements that must be integrated with the other functionalities.

To deal with these aspects, some methods propose new mechanisms, but many rely on integrating with or extending existing methods that have already been successfully applied to such aspects of application design.

4.6 INTERFACE AND PRESENTATION

Web applications are, obviously, interactive applications. Users access the application functionality, including navigation, through the interface. For some authors, the interface directly presents the conceptual information items and exposes the application functionality. In fact, applications typically react to some interface event (such as a link or button being clicked), triggering the corresponding functions, which in turn cause the interface elements to change in some way. Even for non-Web environments, many design methods already decouple interface design from functionality design as a way to modularize and reduce the complexity of the design task.

In contrast, for many design methods, a distinction should be made between interface transformations and navigation operations in Web applications. In other words, some interface events that trigger application functionality do not correspond to any navigation operation, even if the interface changes in some way. This is true even if there is access to the server as a result of the interface operation. A simple example is an update operation to an order being made, changing, for instance, the quantity of some item in the order. Even though this causes an access to the server, which triggers some script that updates the internal data structure representing the order (and possibly reflecting it in a database), there is no associated navigation—the item being "navigated" is still the same order.

Consequently, Web design methods must provide a way for the designer to specify the interface—which elements compose it and how it reacts to all possible events. The interface behavior must necessarily be tied to the application functionality, including navigation.

Web application interface design must deal with another dimension, namely graphic design. In contrast with non-Web applications, where the complete design is carried out by software engineers, the interface appearance of Web applications is mostly defined by graphic (or, in current parlance, user experience) designers, who determine the actual "look and feel" of the application. Design methods must allow the clear separation between the so-called abstract interface design, where interface functionality is defined, and the concrete design, where layout and graphical appearance are defined.

In addition, the existence of an abstract interface design is also useful because of the rapidly evolving technological platforms upon which Web

applications are implemented. New standards are issued and new versions of Web browsers are released almost every six months. Having an abstract interface design allows a more stable part of the design that remains unchanged by such technological evolution to be encapsulated. Besides this type of evolution, market reasons tend to pressure many Web applications to periodically completely change their graphical appearance; the abstract interface designs also help to cope with this requirement.

The rapid introduction of new devices used to access the Web, such as palmtops and cell phones, presents yet another dimension of requirements. Since such devices provide radically different run-time environments, with more limited display and interaction capabilities, some design methods strive to identify a "device-independent" portion of the interface design that remains unchanged regardless of the device being used to access it.

4.7 FURTHER ISSUES

4.7.1 Design Process

The discussion so far has focused mostly on the representational needs of Web application design methods. Beyond that, methods must also address the design process itself. The Web environment presents additional demands on the development process, caused by factors such as

1. the specific target environment, which is a hypermedia, distributed client-server environment
2. the rapid evolution and constant change in the implementation environment
3. the accelerated design cycle
4. the broader and sometimes harder-to-characterize audience (or intended target user categories)
5. the multidisciplinary nature of the development team, involving other professional skills such as communicators, content experts, graphics designers, etc.

Some methods provide additional tools to capture or model requirements that are better suited for this environment; typically, they are also more focused on user (or stakeholder) needs, as opposed to focusing on the designer or contractor alone.

Although a very large number of applications are available on the WWW (basically, most moderately sized Web sites can be considered Web applications of some sort), it is also true that several of their characteristics

can be found repeatedly across many applications. In other words, many subproblems recur, thereby presenting an opportunity for design reuse. After all, if a certain subproblem has been faced and resolved, why not reuse its solution, perhaps adapting it to the situation at hand?

There are several approaches to deal with this, some directly incorporated into the design methods themselves, others complementary to them. In this latter category, mechanisms such as design patterns are employed, allowing known problems and their solutions to be characterized in an easily used format.

On a higher abstraction level, it is also possible to recognize that certain types of applications, in given domains, also exhibit recurring structures. For example, most institutional Web sites are similar, as are many online stores. Some methods can be extended to allow the characterization of families of similar applications, effectively forming Web application design frameworks. Starting with such frameworks, it is possible to rapidly instantiate specific applications in the given domain, by appropriately instantiating the framework's hotspots.

As the number of different applications being deployed on the WWW increases, software engineers have identified a number of recurring functionalities. In addition to the various forms mentioned above, another approach to leverage this accumulated experience is to encapsulate these functionalities in components that can be composed to form a more complex application. More generally, complex applications can eventually be defined as the composition of simpler ones. In some methods, the notion of a component is available as a primitive, together with language specifying how a component can be composed.

4.7.2 Model Representation

An integral part of any method is the definition of the notation used to describe its models. Such a notation has many uses, as it must support the communication between

1. customers and designers
2. designers and end users
3. designers and other designers
4. designers and implementor
5. implementor and end users

and so on. Each of these communications poses different requirements on the notation. For example, the client–designer communication must allow the client to express himself as closely as possible to his own world and

vocabulary; the designer–implementor communication must be precise and unambiguous to ensure that the implementation adheres to the specified application. Notations may be graphical, textual, or both.

Most methods propose a new notation, extend some existing notation, or use existing notation directly. The advantages of directly using or extending existing notation are that, in most cases, users do not have to relearn entirely new conventions, and existing tools may be used directly or extended as needed. On the other hand, if the existing notation is too limited in its expressive power, extending it may require so much that the additions offset these advantages, and it may be more effective to use an entirely new notation.

Since many methods propose different models for describing different aspects of the application, different notations are used, and the relations between the models must also be represented.

Other considerations for notations regard their adequacy for automated tool support and legibility in printed form.

4.7.3 Implementation

Designing and implementing applications using methods produces several documents and, as with any larger software development, is best supported by computer-aided software engineering (CASE) tools. Such tools may support only the specification and help manage the documentation but may also include the generation of the final running implementation. This may be achieved in a completely automatic fashion or may be semi-automated, requiring the designer to manually fill in implementation details that cannot be automatically determined by the CASE environment.

Most complex Web applications involve dynamic processing of information, which in turn requires extensions to the server. The CASE environment may only generate the application targeted at a specific run-time environment, for example, Apache server with PhP scripting, or may be configured to generate the application for various such environments. In some cases, run-time environments are part of a larger framework, such as J2EE.

In addition, it may be further customized to also take into account the various access devices possible, such as cellular phones or handhelds, and provide environment information to support ubiquitous applications.

Dynamic Web applications rely heavily on databases to store and manage their data. An important part of the generated application is its interface to the DBMS. Typically, this involves establishing a mapping between the information items in the application and the data items stored in the database. Once more, this interface may be automatically generated and

managed by the CASE environment, or it may require the manual intervention of the designer. In some cases, the CASE environment may also provide support for performance tuning and for maintenance and evolution of the application.

However, the task of implementing Web applications is increasing in complexity day by day due to the continuous emergence of new platforms and technologies. Specific needs are also arising for the development of several kinds of Web applications: Web data systems, interactive systems, transactional systems, workflow-based systems, collaborative systems, site-oriented systems, social systems, ubiquitous systems, or Semantic Web applications.

In this context, Web Engineering methods are evolving to be properly adapted to the continuous evolution of Web system requirements and technology. Web Engineering is a domain where model-driven software development (MDSD) principles can be used to address the evolution and adaptation of Web software to new platforms and technologies in order to achieve technological independence.

The Model-Driven Architecture (MDA®) initiative from the Object Management Group (OMG™) proposes defining the software building process based on a set of models. Depending on the level of abstraction, these models are dependent or independent of technological issues. One of the major benefits that introduce this separation of concerns is that system definitions can be reused for generating the system implementation in different technologies. On the other camp, Software Factories promote the systematic reuse, the application of the product lines philosophy, the model-driven development, the definition and use of domain-specific languages, the development of frameworks, and the generation of incremental code.

Recent Web Engineering approaches have made real advances in the prospects that are offered by model-driven software development (MDSD). This becomes more evident if we consider that some Web Engineering methods have successfully adopted the MDSD principles. They address different concerns using separate models (navigation, presentation, data, etc.) and are supported by model compilers that produce most of the application's Web pages (using PHP, JSP, ASP.NET, etc.). Moreover, they also take into account the possibility of accessing these systems via different devices, such as cellular phones or handhelds, and business logic (using COM+, EJB, J2EE, Web services) based on the models. This may be achieved either completely automatically or semi-automatically, requiring the designer to manually fill in implementation details that the tool cannot automatically determine.

Several signs point out that the use of the MDSD approach is going to rapidly increase. First, MDSD has received significant support from both the

MDA and the Software Factories. Second, the proliferation of CASE tools that support MDSD-based approaches that claim to be "MDA-compliant" is widespread. Third, technologies and tools for developing "your own" DSDM tools (graphical editors, model transformers, code generators, etc.) have also become abundant. In this category of tools we can find a set of projects developed under the Eclipse Modeling Project (EMF, GMF, GMT, etc.) and the DSL tools that are integrated with the MS Visual Studio 2005 Team System Edition. In this context, companies and research groups are considering the development of their own CASE tool for supporting their own Web Engineering method (following the MDA, Software Factories, Product Lines, Generative Programming of whatever other, more specific proposal) using one of these tools.

Although current Web Engineering methods still present some limitations when it comes to modeling further concerns, such as architectural styles or distribution, the adoption of MDSD principles can help achieve a real technological independence. In this way, methods are ready to be adapted to the second (Web 2.0) or third (Web 3.0) generation of Web applications, giving support to AJAX-based (Asynchronous Javascript and XML) Rich Client applications, Mashups, folksonomies, REST or XML Web Services to integrate current Web applications with third-party services, portals, as well as legacy systems.

4.7.4 Evolution and Maintenance

The dynamic nature of the current Web environment, and of the Internet in general, means that applications evolve very rapidly, as does the environment in which they run. Some methods provide direct support for the evolution of Web applications or provide support for tracing design decisions at various levels, easing the maintenance problem.

The resilience of applications designed with such methods with respect to changes is in good part determined by the abstraction levels supported by the method. If the adequate abstraction levels have been provided, the magnitude of changes in the application should be directly proportional to the magnitude of the changes in their requirements or of their run-time environments.

4.7.5 Role of Standards

Perhaps the main reason for the success of the Web was its establishment and adherence to standards. Following this tradition, some methods adopt some of the more recent standard notations such as UML at the design level or XML at the data level, some with direct support from CASE tools. In such

cases, adopting such standards may also affect the target run-time environments, since several of them already provide direct support for these standards. The adoption of these standards may also facilitate model interchange between tools supporting different methods, such as XMI for UML-based notations.

Although standards such as XML address the syntactical aspect of model specifications, it may also be possible to use other standards that advance further into the semantic realm, such as RDF, RDFS, or OWL.

4.7.6 Personalization and Adaptation

Personalization has become a very important issue in the Internet, as a consequence of the increasing sophistication of Web sites, driven by the unabated competition between sites to attract viewers. Even though almost from the beginning browsers allowed personalization of presentation features, the current ubiquity of the World Wide Web, together with the myriad of platforms that support some kind of browsing, has reshaped this problem toward building applications customized to the individual.

Designing personalized Internet applications may mean building different interfaces, customized to a particular device; providing different navigation topologies to different persons; recommending specific products according to the user's preferences; implementing different pricing policies, and so on. All these facets of personalization share the need of modeling the user and his preferences, building profiles, finding algorithms for best linking options, etc., and integrating them in a cohesive design.

Several types of personalization must be accommodated:

1. role-based personalization, where the user sees different items and has different options depending on his role
2. link personalization, where the actual navigation topology depends on the individual and her access rights
3. content personalization, where actual contents of information items change depending on the person accessing the content
4. behavior or functionality personalization, where the functions the user can activate, and their behavior, change depending on the user
5. structure personalization, where the entire application may be customized according to the user's preferences or profile
6. presentation personalization, where the appearance (look and feel) of the content is adapted to the user, not the content itself

Many methods provide primitives that directly support personalization design, whereas others provide only guidelines.

Personalization can be seen as a special case of a more general behavior, adaptation. This behavior allows Web applications to alter some of their own characteristics as a function of various possible parameters. Personalization is really adaptation in which the input parameter is the user's identity and role. Other possible parameters are geographic location, available bandwidth, input device, past browsing history, etc.

Adaptation is paramount in the case of ubiquitous Web applications, i.e., applications that can be accessed anywhere, anytime. This means that they must be context-aware, in the broadest sense—not only must the logical context be taken into account, but also the physical and geographical environments as well. Such awareness may be directly expressible by some primitives in certain methods or implemented by lower-level primitives in others.

4.7.7 Quality Assurance Issues

Current Web applications can be very complex products and critical assets for an organization, so their quality can, to some extent, determine the organization's success or failure.

Quality assurance is a key support process and strategy mainly at the organization's software project level, in order to assure high-quality products and services by providing the main project stakeholders with the appropriate visibility, control, and feedback on resources, processes, and associated products throughout the software and Web life cycle.

Quality assurance applies to evaluation not only of products, processes, and services but also of resources as development methods, development teams, among others. To be effective, the quality assurance strategy should be planned and integrated to the main processes in the early phases of a project: That is, plans, activities, and procedures that will add value to the project and satisfy functional and nonfunctional requirements should be established from the very beginning. Quality assurance as a support process deals ultimately with preventive, evaluative, and corrective actions.

To measure, evaluate, verify, and validate functional and nonfunctional requirements from the quality assurance standpoint, different classes of methods can be categorized, including, for example, testing, inspection, simulation, and surveys, among others. In turn, for each category, particular methods and techniques can be applied (e.g., feature analysis method, heuristic guidelines inspection, Web usage analyses, white box testing, and user testing, among many possibilities) regarding the specific evaluation objectives and information needs.

Functional requirements actually represent what the Web application must do and provide (i.e., the scope) in terms of functions, contents, and

services for the intended users. Nonfunctional requirements actually specify the capabilities, properties, and constraints that those functions, contents, and services should satisfy under certain conditions, for the intended users and contexts of use. The former are supported by different Web development methods by providing constructors and models at the conceptual, navigational, or presentational level, etc., as introduced in earlier sections here (and illustrated throughout this book). These models usually serve as input to many evaluation, verification, and validation activities, in addition to particular models for functional testing as test cases models, among others. The latter are currently supported to some extent by a couple of Web development methods; moreover, very often methods are not well integrated with quality assurance activities.

For instance, in order to measure and evaluate the external quality or quality in use of a Web application (or its components), models for quality, or quality in use, or subcharacteristics such as usability, security, reliability, etc. should be specified. These models, which represent nonfunctional requirements, can be performed by means of characteristics and attributes, or by means of heuristic guidelines, or by categories and questionnaire items.

Therefore, as the reader can surmise, conceptual frameworks for evaluation (as we will discuss in Chapter 13) that allow specifying nonfunctional requirements at different stages, in addition to the measurement (e.g., based on metrics) and the evaluation (e.g., based on indicators) components, might be necessary in order to support the analyses and recommended actions. In fact, some of the measurement, evaluation, and verification activities may be integrated and even automated in software and Web development methods in a sounder way.

4.8 THE PROBLEM STATEMENT

We deliberately kept the specification of the example for this book simple to make it more understandable for a broad audience. The requirements of the proposed application case are just described textually to leave place for each author to express it using the corresponding method's primitives.

The goal is to model an Internet site like www.imdb.com (the Internet Movies Database). The site allows users to explore movies and television programs, their actors, directors, and producers. Movies descriptions contain director, actors, genre(s), user comments, user ratings, country of origin, qualification, etc. Information about soundtracks can also be obtained. Relationships with related movies can be explored. External links (such as the official movie Web site) are also provided. Actors and directors are described by a short bio, a photograph, and his/her filmography (as actors,

producers, writers, directors, and other related roles). For example, for each actor one can explore all the movies he participated in. Photo galleries of the actor/director can also be explored. The filmography can be explored according to different criteria: awards, user's votes, genre, etc.

The site provides daily information on new movies, biographies of selected actors, and news on the world of movies. Regarding nonnavigational behaviors, it is possible to add comments to films (in the style of sites such as www.amazon.com). It is also possible to explore show times and to buy tickets in selected cinemas. In this regard it is possible to select a city and a movie (currently on exhibition) and get the list of cinemas in that city that are showing the film. It is possible to select one show time of a given cinema and buy a ticket to see the chosen movie. The site maintains a list of films currently playing, giving a short description, together with access to user comments.

In a similar sense, the site maintains a list of upcoming movies, information on festivals, awards (like Emmy, Oscar, etc.), and miscellaneous news. It is also possible to buy DVDs for some movies online, with the usual functionality for online stores. Search facilities are also provided for movies, actors, companies, etc.