Appearing in WebDB-99

Declarative web-site management with Tiramisu

Corin R. Anderson, Alon Y. Levy, Daniel S. Weld
Department of Computer Science and Engineering
University of Washington
Seattle, WA 98195
{corin, alon, weld}@cs.washington.edu

Abstract

Early research in declarative web-site management identified a key principle: the separation of data management, site structure, and page presentation. This separation was made through the introduction of a logical representation, the *site graph*, defined as a view over underlying data. While the separation of these three tasks provides many benefits, existing systems require that the user *implement* the web site with the same system they use to *design* it, and this restriction is problematic for three reasons. First, many users are familiar with and prefer another implementation tool. Second, other tools may provide useful, new, or specialized features. Third, a complex site may be controlled by multiple organizations whose standards require different tools. In this paper we present a new architecture for declarative web-site management that separates design and implementation. We describe the challenges we faced during the implementation of TIRAMISU, a system using this architecture.

1 Introduction

Recent research in the database community has investigated the use of declarative representations as a paradigm for building web-site management systems [5, 2, 4, 9, 1, 3]. Underlying these *declarative web-site management (DWSM)* systems is the observation that there are three main tasks in building a web site, and these tasks are intertwined in current tools. These tasks are (1) selecting, integrating and managing the data to be presented in the site, (2) defining the structure of the site (i.e., the set of pages, the data at each page, and the links between pages), and (3) designing the graphical presentation of the site.

Broadly speaking, DWSM systems seek to improve the speed and flexibility of web-site management by decoupling these tasks through the introduction of a *logical* representation of the structure and content of the web site as an intermediate step in the construction of the site. This logical model, which we refer to as the *site graph*, is defined as a view over some underlying data that is managed by a separate data management system. Differences between the DWSM systems are primarily in the precise data model and query language used to define the site graph as a view.

Previous declarative web-site management systems offer several advantages. First and foremost, these systems provide a flexible mechanism for modifying both the structure and the data in a site. This flexibility is important because the process of building web sites is inherently an iterative one. Furthermore, these systems enable building multiple versions of a web site from the same data, targeted for different classes of users (e.g., novice vs. expert users, public vs. internal sites). Finally, such systems enable the specification and verification of integrity constraints on web sites [6] and automatic development of efficient run-time management policies for sites [7].

Our experience building, using, and extending the STRUDEL DWSM system [5] has exposed a significant weakness in the aforementioned paradigm. While DWSM systems enable users to explicitly *design* the structure of a web site, they force users to *implement* the web site in the same system. For example, in STRUDEL, after users describe the structure of the web site using the STRUQL language, they have to implement the site (i.e., compute the site graph and the HTML pages) also using STRUDEL. There are several problems with this approach:

Familiarity: Many users are already using web site implementation tools with which they are comfortable or forced to use because of organizational constraints.

Functionality: There will always be tools that are specialized for certain tasks (*e.g.*, for specific graphical design, database connectivity, or electronic commerce) and we cannot expect STRUDEL or any other system to encompass the functionalities of all these tools.

Politics: Even if only a single tool is being used, the implementation of a web site may be fragmented across several sub-organizations of an enterprise.

This paper describes a new architecture for declarative web-site management systems, and discusses an embodiment of this architecture: the implemented TIRAMISU. Like previous DWSM systems, TIRAMISU provides the ability to distinguish the design phase of the web site using a variation on the entity/relationship model (called the *site schema*). However, after the designer completes the high-level design of the site, she chooses which implementation tools to use for the different parts of the site. TIRAMISU's *implementation manager* then breaks down the site specification appropriately and communicates with the different implementation tools to produce the web site.

Specifically, for each tool, TIRAMISU takes the appropriate fragment of the site specification and creates a partial implementation that is ready to be completed within the tool. For example, if the tool is a WYSIWYG HTML editor, the partial implementation includes a set of HTML files with appropriate links already in them. If the tool is a database middleware tool for creating web sites from relational databases, the partial implementation consists of a set of HTML templates that include embedded SQL queries, as the tool allows. After the implementation of each fragment of the site is completed using the tools, the implementation manager ensures that the partial web sites produced by the different implementation tools are fit seamlessly together.

In this paper, we illustrate the ideas and operation of TIRAMISU mostly through an example: creating the home page of one of the authors. We show how a site schema is created and annotated with the tools used. We demonstrate how TIRAMISU directs each tool to create each piece of the site. We discuss the requirements of the interface between the tools and TIRAMISU. We finally close with some discussion and directions of future work.

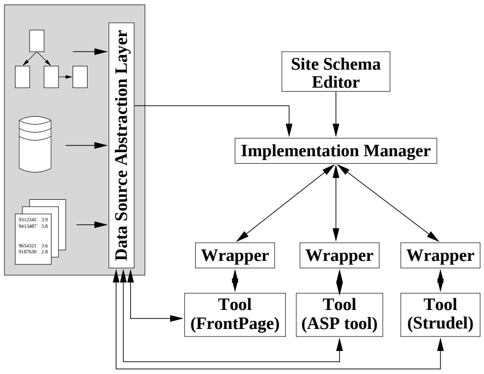

Figure 1: **Tiramisu architecture.** The data integration module (shaded box) provides a uniform view of the data to the web site. The web site designer interacts with the site schema editor to specify the site structure. Each piece of web content is built by external tools, shown with their wrappers, at the bottom. Coordinating the implementation of the entire site is the TIRAMISU Implementation Manager.

2 Tiramisu architecture

The TIRAMISU architecture, shown in figure 1, is broken into the following components:

Data management and integration: As in other DWSM systems, the content underlying the web site is stored in possibly multiple external repositories. TIRAMISU employs a data integration system to provide a uniform view of the data, thereby hiding the specific interface details of the particular data sources. Typically, the data is stored in either traditional databases, structured files, or legacy systems. Currently, TIRAMISU uses STRUDEL as the data integration system, but will soon be modified to use the TUKWILA [8] system.

Site schema editor: The site schema is a graphical representation of the structure of the web site. Intuitively, the site structure is defined by a set of queries over the underlying data sources. The web site designer can visually edit the site schema with TIRAMISU's site schema editor, including adding annotation about which implementation tool is to be used for each schema node. More discussion of the site schema is given in section 2.1.

Implementation manager: The key idea in TIRAMISU is that TIRAMISU itself does not create the web content, but rather coordinates a set of external tools that perform the task. The coordination is performed by the implementation manager component of the system. The actual communication with the tools is done via a set of wrappers. We discuss the implementation manager in more detail in Section 2.2. Section 3 discusses the issues that arise in designing an API for communicating with the implementation tools.

2.1 Definition of site structure

The structure of the web site is defined by a site schema [5], which is a visual representation of the query (or set of queries) defining the site graph from the underlying data sources. TIRAMISU's site schema is a variation of an entity/relationship diagram. The site schema for our example is shown in figure 2. Nodes represent groups of related web content, specified by a set of parameters and a query. For example, in figure 2, we see that the PubsByYear node has a single parameter y, whose values are the result of the query

q(y) :- pub(p), year(p,y).

Based on the data underlying our example site, the PubsByYear(y) node represents the set of three pieces of web content: PubsByYear_1996, PubsByYear_1998, and PubsByYear_1999.

Links between the nodes represent either hypertext links or links to embedded content. Embedded content refers to content that appears in another piece of content. For instance, the Schedule content will appear as part of the HomePage content. Like nodes, links also contain queries[1], which specify exactly which pairs of ground web content are linked together. For instance, the link from the PubsByYear node to the ExtendedPubEntry node specifies that only publications p that appear in year y should be embedded in the PubsByYear(y) ground instance.

Finally, each node in the site schema is tagged with the name of the tool that will be used to implement that node. In our example, FrontPage is used to create the Schedule and HomePage nodes, STRUDEL is used for the Resume and Publications nodes, and VB Script for Active Server Pages is used for the remaining nodes.[2]

2.2 Tiramisu Implementation Manager

The novel aspect of TIRAMISU compared to other declarative web-site management systems is the way it *coordinates* the implementation of the web site rather than producing the web site on its own. We demonstrate TIRAMISU's operation through our example. The tools we consider at the moment include FrontPage (a graphical editor for HTML); VB Script for Active Server Pages (ASP), a virtual scripting environment that enables embedding SQL queries in HTML; and the STRUDEL system. We note that the tools for creating web sites out of relational databases are similar in many ways to Active Server Pages for the purpose of our discussion.

The TIRAMISU Implementation Manager takes as input the site schema that the web site designer has created, including the annotations about which tools are to be used for each node. The TIRAMISU Implementation Manager begins by iterating through each of the tools mentioned in the input (FrontPage, ASP, and STRUDEL in our example). For each tool $\mathcal{I}$, the TIRAMISU Implementation Manager gathers the set of nodes, $\mathcal{N}_{\mathcal{I}}$, in the site schema that the web site designer has specified as being implemented by $\mathcal{I}$. In addition to these nodes (and their adjacent links), the TIRAMISU Implementation Manager also computes the set, $\mathcal{F}_{\mathcal{I}}$, of *frontier* nodes: the nodes in the site schema that are the destination of a link emanating from some node in $\mathcal{N}_{\mathcal{I}}$, but that are not in $\mathcal{N}_{\mathcal{I}}$. In the example,

$\mathcal{N}_{\text{FrontPage}} = \{$HomePage, Schedule$\}$

$\mathcal{F}_{\text{FrontPage}} = \{$Resume, Publications$\}$

$\mathcal{N}_{\text{Strudel}} = \{$Resume, Publications$\}$

$\mathcal{F}_{\text{Strudel}} = \{$BriefPubEntry, PubsByYear$\}$

$\mathcal{N}_{\text{ASP}} = \{$BriefPubEntry, PubsByYear, ExtendedPubEntry$\}$

$\mathcal{F}_{\text{ASP}} = \{\}$

Given the sets $\mathcal{N}_{\mathcal{I}}$ and $\mathcal{F}_{\mathcal{I}}$ for each tool, the TIRAMISU Implementation Manager can now invoke the tools to create each fragment of the web

[1] The query with no right hand side, :-, denotes the query TRUE. Links in the site schema whose queries have an empty right hand side produce links in the site graph between the cross product of all the source and destination pages. For example, the link L4 generates links between the Resume node and all the BriefPubEntry nodes in the site graph.

[2] We use ASP in this example for illustrative purposes. In practice, we would use STRUDEL for all parts of the site that are not created by FrontPage.

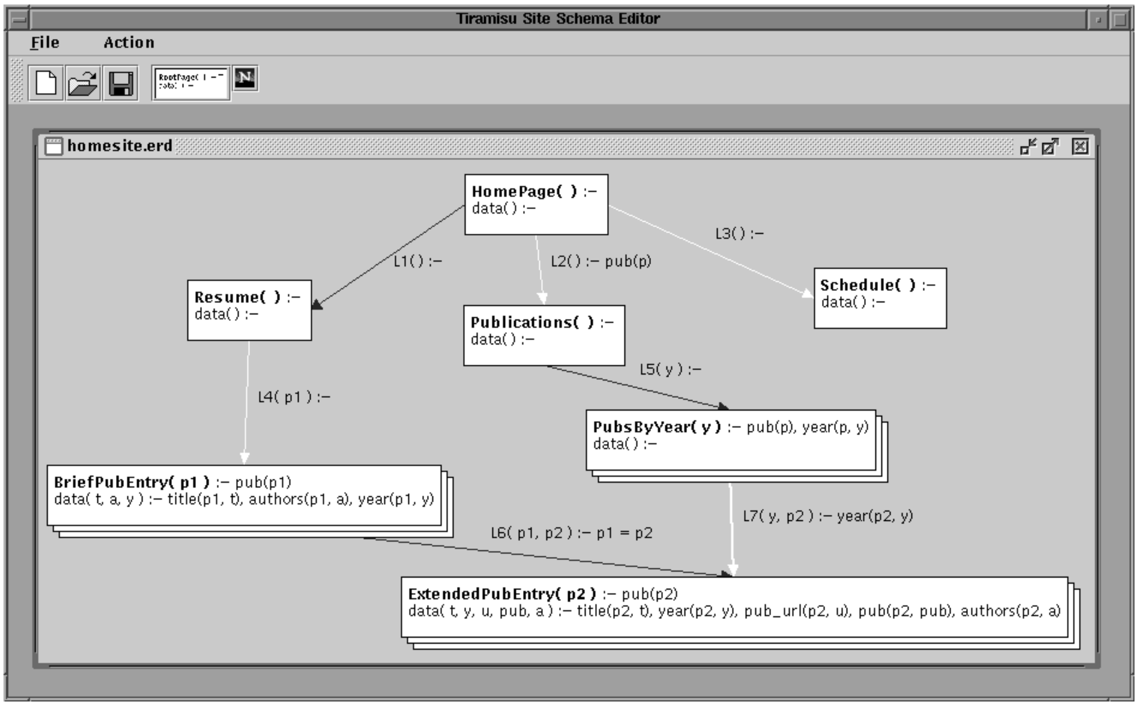

Figure 2: **An example site schema.** Rectangular nodes represent web content. Arrows represent either hypertext links (black) or embedded content (white). Nodes are parameterized and include queries for their parameters and displayed data. Links are also parameterized to specify which source and destination nodes are to be connected.

site. Specifically, for each tool, the Tiramisu Implementation Manager computes a partial implementation of the web content that that tool must create. In our example, the Tiramisu Implementation Manager creates partial implementations as follows:

FrontPage: Tiramisu creates HTML files for each node that FrontPage must implement. For the Schedule node, there is only one page (named `Schedule.html`). At the moment, this page has no data and no external links, and these will be filled in later as the web site designer creates the schedule with FrontPage. For the HomePage node, there again is no data, but now there are links. There is a link to the Resume node, for which Tiramisu creates a hypertext link in the `HomePage.html` file. There is also a link to the Publications node. For this link, however, Tiramisu first poses the query L2() :- pub(p). Only if the result is nonempty (*i.e.*, if there is *some* publication p in the database) will the link from `HomePage.html` to the Publications node exist.

Strudel: Tiramisu creates a StruQL query that includes the nodes Publications and Resume. Tiramisu also creates default HTML templates for these nodes that Strudel will use to create the resulting web pages. Given the StruQL query and the HTML templates, Strudel implements its part of the site graph.

ASP: Finally, Tiramisu creates an ASP for the PubsByYear node, the BriefPubEntry node, and the ExtendedPubEntry node. Each of these ASPs includes the SQL queries over the underlying data necessary to retrieve the data at that node and the HTML to display the data.

The final step in the implementation process is for the web site designer to refine the automatically created partial implementations (if desired). In this step, the web site designer performs the following tasks as needed: (1) adds content to the HomePage and Schedule pages with FrontPage, (2) edits the HTML templates that Strudel uses to build its content, and (3) edits the default VB script that the ASP will run.

There are a few important observations to be made at this point. The first observation is that content created by the different tools is integrated coherently into the web site. In particular, content that is created by one tool (the ASP tool, for instance) and referenced by another tool (STRUDEL) are matched up seamlessly. Such was the case with the link from the Publications node to the PubsByYear node. STRUDEL created the Publications content, but needed to reference the PubsByYear content created by the ASP tool. TIRAMISU ensures that these interactions between tools are coherent.

The second observation is that TIRAMISU makes no *a priori* commitment to the materialization policy – whether pages are created statically or dynamically. TIRAMISU cleanly supports both such methods, as can be seen by TIRAMISU's interaction with STRUDEL (which created its content statically) and the ASP tool (which will create its content dynamically).

3 API for a tool federation

The longer term goal of our project is to develop a federated architecture in which it is easy to incorporate new web-site development and management tools. A key element of this architecture is the definition of a standard API to web-site management tools (*e.g.*, as ODBC is for relational databases). The API defines the minimal set of functionalities that a tool must provide in order to participate in the federation. As a first step in this direction, we build on our experience in implementing TIRAMISU to highlight the main issues that need to be considered in such an API. As we describe, for the tools we considered in TIRAMISU, it is possible to simulate such an API by a set of wrappers *on top* of the existing tools.

Creation of content from an external specification: In order for TIRAMISU to be able to dispatch parts of the implementation of the web-site to a tool, the tool must be able to accept a partial specification of content from an external source. In its simplest form, a tool should be able to accept specifications for single web pages. Preferably, the tool should be able to accept specifications for sets of pages and links between them.

For the tools we considered, this functionality turned out to be relatively straightforward. For FrontPage, we simply created HTML pages that can be later refined using the tool itself. For ASP, we created scripts with embedded SQL queries. For STRUDEL, we created a STRUQL query and default HTML templates.

As a further step, it would be desirable to attach to the partial specification a set of integrity constraints that should be maintained by the tool as the content is fleshed out, and hints for run-time management of the site.

Access to tool's underlying data sources: Some implementation tools may have associated with them their own data sources[3]. In order to guarantee that there are no discrepancies between the fragments of the site that are created by different tools, the implementation manager must be able to access the underlying sources.

For example, suppose that one fragment of a university web site is created by the Computer Science Department and another by the registrar's office. The pages describing course listings are created by the registrar's office and derived from the registrar's data. But the faculty's homepages and schedules are created by the department and its data. To ensure a consistent web site, it must be the case that the Computer Science faculty in the registrar's database match those in the CS database. Hence, the implementation manager needs to be able to query both databases and warn of any discrepancy.[4]

This functionality is an easy one to support in practice, because web-site management tools tend to rely on ODBC compliant sources. The only question that arises is in contexts where a web site is built across multiple sub-organizations, and they are not willing to provide access to all of their data externally to the implementation manager. In such cases, the sub-organizations must expose at least the data that is relevant to the integration of the different fragments of the site (and, in fact, it is possible to derive which portions of the data are relevant by analyzing the site schema).

External naming of URLs: Because the fragments of the site created by different tools are linked in complex ways, special care needs to be given to the issue of naming URLs. Consider the case with two implementation tools, $\mathcal{I}$ and $\mathcal{J}$, where $\mathcal{N}_{\mathcal{I}} \cap \mathcal{F}_{\mathcal{J}} \neq \{\}$. $\mathcal{I}$ must create pages that can be referenced by the pages $\mathcal{J}$ creates, and $\mathcal{J}$ must create pages that have the correct external links to the pages of $\mathcal{I}$. To enable seamless linking,

[3] Recall that an "implementation tool" may also be an organization that is responsible for part of the larger web site.

[4] The issue of how to fix such a discrepancy is beyond the scope of our discussion.

these tools must be able to accept from the implementation manager names for the URLs that they create.

When the web site is created statically, this requirement is not strictly necessary. An alternative approach in this case would have been to allow each tool to name the URLs of the pages it creates, and report the names to the implementation manager. The implementation manager would then perform a second pass over all the content, in which it fixes the external references or introduces indirection to produce the correct effect.

Implementing this functionality for the tools we considered was mostly straightforward, except for STRUDEL, which insists on inventing its own URLs. In our implementation, we wrote a simple wrapper that iterated over the content produced by STRUDEL and renamed the URLs appropriately.

In conclusion, we clearly expect the set of features of the desired API to evolve over time. The observation that these features were easy to implement on top of existing tools suggests the such a federated architecture is indeed a viable one.

4 Conclusions

In this paper we have described the TIRAMISU system that provides an architecture for building web sites using a collection of web site management tools.

In the long run, this architecture has more promise than a single-tool system, because it will be able to leverage novel functionalities provided by emerging tools. Furthermore, one can replace an implementation tool used for building a web site without the effort of completely redesigning the site. We also argue that many of the advantages touted for previous DWSM systems can be conserved in our architecture as well. We mention two of these here.

The first is the ability to specify and verify integrity constraints [6]. As shown in [6], integrity constraints are verified at the intensional level of the site schema, and not the extensional level of the site graph or set of HTML pages. Hence, the same algorithms can be applied in TIRAMISU as well. Of course, when the web site is actually implemented using a specific tool, the web site designer can choose to ignore the constraints imposed by the design. However, it is possible to insert warnings in the partial implementations produced by the TIRAMISU to ensure that the web site designer is aware of possible violations.

The second point is the design of run-time management of web sites [7]. The problem is to decide automatically, based on the site schema and characteristics of the data, when to compute parts of the web site (either before browsing begins, on demand, or a combination of the two). Here too, the design of the run-time policies uses the site schema, and hence the algorithms described in [7] can be extended to TIRAMISU as well.

For more information about TIRAMISU, visit our web site at `http://data.cs.washington.edu/web/tiramisu/`.

References

[1] Gustavo Arocena and Alberto Mendelzon. WebOQL: Restructuring documents, databases and webs. In *Proc. of Int. Conf. on Data Engineering (ICDE)*, Orlando, Florida, 1998.

[2] Paolo Atzeni, Giansalvatore Mecca, and Paolo Merialdo. To weave the web. In *Proc. of the Int. Conf. on Very Large Data Bases (VLDB)*, 1997.

[3] Paolo Atzeni, Giansalvatore Mecca, and Paolo Merialdo. Design and maintenance of data-intensive web sites. In *Proc. of the Conf. on Extending Database Technology (EDBT)*, Valencia, Spain, 1998.

[4] Sophie Cluet, Claude Delobel, Jerome Simeon, and Katarzyna Smaga. Your mediators need data conversion. In *Proc. of ACM SIGMOD Conf. on Management of Data*, Seattle, WA, 1998.

[5] Mary Fernandez, Daniela Florescu, Jaewoo Kang, Alon Levy, and Dan Suciu. Catching the boat with Strudel: Experiences with a web-site management system. In *Proc. of ACM SIGMOD Conf. on Management of Data*, Seattle, WA, 1998.

[6] Mary Fernandez, Daniela Florescu, Alon Levy, and Dan Suciu. Verifying integrity constraints on websites. In *Proceedings of the 16th International Joint Conference on Artificial Intelligence*, 1999.

[7] Daniela Florescu, Alon Levy, Dan Suciu, and Khaled Yagoub. Optimization of run-time management of data intensive web sites. In *Proc. of the Int. Conf. on Very Large Data Bases (VLDB)*, 1999.

[8] Zachary Ives, Daniela Florescu, Marc Friedman, Alon Levy, and Dan Weld. An adaptive query execution engine for data integration. In *Proc. of ACM SIGMOD Conf. on Management of Data*, 1999.

[9] P. Paolini and P. Fraternali. A conceptual model and a tool environment for developing more scalable, dynamic, and customizable web applications. In *Proc. of the Conf. on Extending Database Technology (EDBT)*, 1998.