

Handling Constantly Changing Metadata

Avigdor Gal

Department of Computer Science

University of Toronto

6 King's College Road

Toronto, ONT, Canada M5S 3H5

phone: (416) 978-7330

fax: (416) 978-1455

avigal@cs.toronto.edu

Abstract

One of the basic assumptions bestowed upon metadata in applications' design is its infrequent modifications with respect to the frequency of data changes, leading to a rigid selection of data structures and programs that are heavily dependent on specific metadata structure. However, this assumption is being challenged by the introduction of autonomous information systems in a distributed environment. This research presents an intuitive formulation of metadata that is particularly well suited for distributed heterogeneous systems, and develops methods for various metadata modifications. We present a lightweight tool to support frequent changes to metadata of related components by using a common active database model. This research is part of the CoopWARE project that takes place at the University of Toronto.

1 Introduction

Metadata is loosely defined as “data about data,” or “additional information that is necessary for data to be useful.” As such, it is tightly coupled

with data sources in general, and databases in particular. In the information age, in order to provide coherent and reliable information, data should be obtained from distributed, heterogeneous data sources on an as-needed basis. Thus, common assumptions bestowed upon metadata in applications' design should be revisited and adopted to contemporary data sources' architecture. A typical design assumption relates to the infrequent changes to metadata with respect to data modifications (e.g. [3]) and often leads to a rigid selection of data structures and programs that are heavily dependent on a specific metadata structure. This assumption is being challenged by the introduction of autonomous information systems in a distributed environment, e.g. federated databases [11] and the World Wide Web. In particular, the ever-changing environment of the World Wide Web poses serious challenges for repositories aimed at defining "global schema" of information over autonomous, distributed and heterogeneous information systems. Hence, applications can become legacy systems or completely useless even a short time after being designed.

This research presents an intuitive formulation of metadata for distributed heterogeneous systems, and develops methods for various metadata modifications, some of which are at the design level while others result from structural and functional changes to the underlying data sources. We present a lightweight tool to support frequent changes to metadata of related components by using an easy-to-use language where changes to metadata are easily maintained in order to stabilize a working application that faces frequent changes to its metadata. Metadata changes are handled either by using structural changes to the information stored at a repository or through the introduction of rules. We advocate the use of a dependency graph, an executable data structure that is generated by compiling a set of application rules into a graph, as a visual tool for design and analysis of metadata. An application's activities are translated into one or more paths in a dependency graph, where an initial node is a request for an activity and the final node is the conclusion of the activity. Using the dependency graph, the application can choose among alternative paths, retract whenever an element of an application becomes inaccessible, and dynamically choose the best path to achieve its goals.

This research is part of the CoopWARE project [9] that takes place at the University of Toronto. CoopWARE (Cooperation With Active Relationship Enforcement) is a generic integration architecture, based on active database

technology. CoopWARE uses a flexible and easy to set mechanism for supporting cooperative information systems and it has been successfully tested in a reverse engineering environment. The CoopWARE project is unique in taking an active approach towards data manipulation, as opposed to many other passive models (e.g. TSIMMIS [1] and DIOM [7]).

The paper examines a case study introducing an environment of heterogeneous distributed information services, based on a real-world example, as follows. Let an information service be an application program that takes data from certain addresses on the Web (information sources) and apply an information generation function to it. Changes to the relevant Web pages structures are a matter of routine and require the program to adopt itself to such changes. Also, some sites may be inaccessible, possibly interfering with the application program capabilities to provide the information service.

The rest of the paper is organised as follows. Section 2 introduces the metadata model, while Section 3 provides a variety of methods to handle specific metadata modifications in a WWW environment.¹ The paper is concluded in Section 4.

2 The metadata model

In this section we present a conceptual instrument for modelling and manipulating metadata information. We first provide a conceptual model of a metadata structure, followed by the definition of a dependency graph which captures inter-relationships among the various parts of an application. To demonstrate the conceptual model we use the following simplified WWW related case study. An information service probes an Internet version of the newspaper The Jerusalem Post. Triggered by (data) modifications to the newspaper's Web-site, an information service scans the front page of the Internet edition, that consists of seven top stories in search of given keywords (e.g. "Labor" and "elections"). The result of the search is embedded within an email message that is sent out to all users that are registered for this information service.

The metadata model is constructed from the information model, the communication model and the behaviour model, as follows. An elaborated de-

¹For space considerations, we refrain from providing formal definitions in this paper. Formal definitions of basic notions can be found in [2].

scription of the model is given in [5].

The information model: The information model consists of the following three primitives:

Component: an element of the application. A component can either be a specific machine, or a conceptual combinations of machines that are accessed in a unified way. Each component belongs to a component class, which defines its possible behaviour (see services description below). For example, we can define three components of the case study, namely a **WebAgent**, a **Mailer** and a **Calendar**: the **WebAgent** handles the access to the Web and the manipulation of data, the **Mailer** controls the correspondence with the users, and the **Calendar** is utilized by the coordinator to set events (see the communication model below).

Service: an operation a component can perform. Services define the functionality of a component in terms of specific operations. A service can have parameters that are instantiated in run-time. It also has a termination status, e.g. success, fail, unknown, etc. As an example, consider the components given above. The **WebAgent** has a single service with the following signature, **SearchFor:** `<WebSiteAddr, Keywords>`; it searches a Web site for specific keywords and returns statistics of the number of occurrences of each keyword. The **Mailer** has two services with the signatures **InformResults:** `<Users, WebSite, Keywords, Statistics>` and **InformFailure:** `<Users, WebSite>`; the mailer uses pre-defined forms to fill-in the statistics and send them to a distribution list or to inform of a failure to access the specific Web site. The **Calendar** has a single service, **Set:** `<Event, WebSite, Time>`, to set events. It is worth noting that the keyword **WebSite** refers to a logical description of a Web site address, while the keyword **WebSiteAddr** refers to an actual Web address.

Data source: A provider of data for the application. Strictly speaking, a data source is a component with well-defined services. Nonetheless, from a designer's standpoint it is a stand-alone element which represents a source of data. For example, the Web is accessed by

the use of HTTP calls from a client machine to a server machine. However, the process is transparent to the user and in most cases carries little importance to the user abstraction of the application. Therefore, an abstraction that relates to Web sites rather than HTTP calls defines better the user's view of the application domain.

The communication model: The communication model consists of one primitive, an event, a compact reliable occurrence that enables the flow of information regarding the state of an application. An event belongs to an event class, and is time stamped. Events can be either external or internal, where external events are events provided by external sensors, and internal events are provided by the components. An event can transfer parameters to be used by services. As an example of an external event, triggered by the system's clock, we can consider the event `DailyWebModification: <WebSite>` that notifies the modifications of Web sites that are known to be modified daily. Alternatively, it can be an internal event of `WebAgent`, a result of daily polling of Web sites in the domain of interest. In the latter case, an additional service (`Polling: <{WebSiteAddr}*>`) should be added to the metadata description. As another example of a set of internal events, we can consider the signals that are sent by `WebAgent`, in response to the instantiation of the `SearchFor` service. The events `SearchSuccess: <WebSite, Keywords, Statistics>`, `SearchFailure: <WebSite, Keywords>`, and `AccessFailure: <WebSite>` form the three possible responses of a `WebAgent` to a `SearchFor` service request. Finally, the `Calendar` has a single event, `SearchAgain: <WebSite>`, to trigger a repeated search.

The behaviour model: The behaviour model is defined using a set of rules. A rule is an event driven atomic element that defines the cooperation of the application's components. In accordance with active database conventions, each rule consists of three elements, namely an event, a condition and an action. An event component consists of a logic formula of events as defined above. A condition is a logic formula of data values, constants, etc. An action is either a set of services to be activated, where each service contains the appropriate parameters

or an update to the application state as stored in a repository. The following set of rules, based on the case study, serves as an example of a definition of a behaviour of an application.

Rule r_1

Event: DailyWebModification(WebSite) or SearchAgain(WebSite)

Condition: WebSite=jpost

Action: SearchFor(www.jpost.co.il, {Labour, election})

Triggered daily, the rule requests the **WebAgent** to search for the keywords at the Web site that is passed as a parameter by the event (jpost in this case). Alternatively, if a Web site is in accessible, a **SearchAgain** event can be set (in the sequel), and trigger a repeated search.

Rule r_2

Event: SearchSuccess(WebSite, Keywords, Statistics)

Condition: Keywords={Labour, election} and WebSite=jpost

Action: InformResults(John Doe, jpost, {Labour, election}, Statistics)
jpost.RepeatedTrials:= 0

If a search was successful, the results are sent, using the **Mailer**, to the interested users. The condition evaluates parameters given by the event. **RepeatedTrials** is a status flag that is associated with a Web site, to identify the maximal number of daily repeated search of the site.

Rule r_3

Event: SearchFailure(WebSite, Keywords)

Condition: Keywords={Labour, election} and WebSite=jpost
and JohnDoe.RequestFailure=True

Action: InformResults(John Doe, jpost, {Labour, election}, null)
jpost.RepeatedTrials:= 0

If a search was unsuccessful, null results are conditionally sent, using the **Mailer**, to the interested users. The **RequestFailure** is a status flag that is associated with a user, or a group of users, defining the interest of a user in receiving unsuccessful search notifications.

Rule r_4

Event: AccessFailure(WebSite)

Condition: WebSite=jpost

and jpost.RepeatedTrials \leq JohnDoe.RepeatedTrials

Action: Set (SearchAgain, jpost, 1hour)

jpost.RepeatedTrials:=jpost.RepeatedTrials+1

A failure to access the Web site results in a repeated trial within an hour, as many times as requested by the user (defined through the status flag `RepeatedTrials`).

Rule r_5

Event: AccessFailure(WebSite)

Condition: WebSite=jpost

and jpost.RepeatedTrials $>$ JohnDoe.RepeatedTrials

Action: InformFailure(John Doe, jpost)

jpost.RepeatedTrials:= 0

A failure to access the Web site after repeated trials results in a `Mailer` response to the user, regarding the failure to access the Web site.

2.1 The execution model

The execution model associated with the metadata model given above uses a dependency graph, $DG(MD) = (V, E)$, a data structure that is generated from the metadata to reason about the activities that should be taken at each step of the execution. A dependency graph can be informally described as a polygraph [10] where each node represents some activity of the application (the triggering of events, the evaluation of conditions or the execution of services) or a data source. An edge $\langle v_i, v_j \rangle$ represents a causal relationship, i.e. the successful termination of v_i is followed by the activation of v_j . Therefore, if an event ev is triggered, then any condition con such that $\langle ev, con \rangle \in E$ is evaluated and any service sr such that $\langle ev, sr \rangle \in E$ is executed. Also, if a condition con is evaluated to true (which is considered to be a successful termination of a condition), then any service sr such that $\langle con, sr \rangle \in E$ is executed. Finally, if a service sr is executed, than at least one of the events such that $\langle sr, ev \rangle \in E$ is triggered. An important property of a polygraph is that related edges are coupled together. This property will be discussed in

Section 3 in greater detail, but suffice is to say now that edges are coupled together if they serve as alternatives in achieving the same goal.

An automatic translation of a metadata model into a dependency graph is available for concrete models, e.g. [4] and [9]. In the general case, the connecting edges should be designed manually, using domain experts. Each of the edges, generated as a result of a rule causal relationship (i.e. event→condition, event→action, and condition→action) is associated with a label, to identify its rule origin.

Figure ?? provides a detailed graphic presentation of the case study's dependency graph. The graph consists of five events (depicted at triangles), five rules' conditions (depicted as diamonds), six services (depicted as circles) and two data sources (depicted as rectangles). Labels are associated with edges, to represent the rules for which an edge is valid. Edges that connect services with possible events do not have labels, as they are not part of the rule set. It is worth noting that an edge can be associated with more than one rule, stating that the same path can be taken by more than one rule.

3 Handling metadata modifications

In this section we provide several typical examples of metadata modifications in the framework of the World Wide Web, and show that these modifications are translated into a straight-forward model modifications. The first three examples (sections 3.1-3.3) relate to possibly frequent changes to the data sources while the fourth example (Section 3.4) is a typical design modification.

3.1 Page partitioning

A common situation in the evolution of sites on the Web is a page partitioning, where a page is partitioned into several pages, smaller in size, which can be reached from a single index page. In the context of the case study, a possible change is the partition of the `www.jpost.co.il` Web page into seven news pages, with the addresses `www.jpost.co.il/Article-*.html` ($* \in \{0..6\}$), where `www.jpost.co.il` serves as an index page that consists of the headlines and a short summary of each of the articles.

Based on the contents of the index page and the additional pages, the following two scenaria can occur:

- The relevant information is available in the index page, e.g. a search of keywords that is limited to the headlines. Therefore, no change to the metadata is necessary.
- The relevant information is no longer available in the index page. In this case, the search function should be repeated for each of the pages. For example, in the case study, such a change modifies rule r_1 to be

```

Rule  $r'_1$ 
Event:      DailyWebModification(WebSite) or SearchAgain(WebSite)
Condition:  WebSite=jpost
Action:     SearchFor(www.jpost.co.il/Article-0.html, {Labour, election})
...
           SearchFor(www.jpost.co.il/Article-6.html, {Labour, election})

```

3.2 Page relocation

Page relocation is a situation where an address of a page has been changed, due to the relocation of a site, a provider's replacement, or a structural modification to a site that involves the relocation of a specific page. As an example of the latter, assume that a front page for The Jerusalem Post is added, using the address `www.jpost.co.il`. The index page is relocated to `www.jpost.co.il/News` and the articles are relocated to `www.jpost.co.il/News/Article-*.html` ($* \in \{0..6\}$). The consequence of such a modification involves the modification of r'_1 to be

```

Rule  $r''_1$ 
Event:      DailyWebModification(WebSite) or SearchAgain(WebSite)
Condition:  WebSite=jpost
Action:     SearchFor(www.jpost.co.il/News/Article-0.html, {Labour, election})
...
           SearchFor(www.jpost.co.il/News/Article-6.html, {Labour, election})

```

3.3 Mirror sites

Mirror sites are common as a method to overcome accessibility problems. Roughly speaking, mirror sites consist of similar information and can serve

as alternatives for accessing data. The introduction of a mirror site results in multiplication of metadata rules that relate to one of the mirror sites and the coupling of relevant edges in the dependency graph. For example, assume that the site `www.jpost.co.il` is mirrored in the site `http://www.jpost.com`, conveniently located for the use of North American users. Consequently, rule r_1'' is cloned to provide the additional search path, as follows:

```

Rule  $r_1^*$ 
Event:      DailyWebModification(WebSite) or SearchAgain(WebSite)
Condition:  WebSite=jpost
Action:     SearchFor(www.jpost.com/News/Article-0.html, {Labour, election})
...
SearchFor(www.jpost.com/News/Article-6.html, {Labour, election})

```

It is worth noting that while all five rules, as given in Section 2, relate to The Jerusalem Post site, the only rule that relates to the physical location of the site is the first rule, while others use a virtual site name that can relate to any of the mirror sites.

Figure ?? provides the modified dependency graph, as a result of page partitioning, page relocation and site mirroring. the edges $\langle \text{SearchFor}, \text{www.jpost.co.il} \rangle$ and $\langle \text{SearchFor}, \text{www.jpost.com} \rangle$ are connected using a vertical line to suggest that these edges are coupled. Therefore, a derivation of the polygraph chooses one of the two as the accessed service. To assist in the decision process, each edge can be associated with a confidence metric that designates the reliability of the destination element. The confidence metric takes into account the number of successful hits using this edge and can be weighted to include the speed of access to the edge destination. During execution, whenever a coupled set of edges is accessed, the “best” edge is chosen, and the confidence metrics are updated. A failure to access an edge destination results in retracting and choosing another edge.

3.4 Design modifications

In the previous three sections, we have explored changes to the distributed system that are transparent to the application designer. A different type of modifications involve changes to the application itself. As an example, consider a situation where successful search results are stored in a local database rather than being sent to the user. This change requires an additional component `SearchDB` with one service `StoreSearchResults: <Users, WebSite,`

`Keywords, Statistics`>.² This service is translated into a parametric query in the language of the particular database. The behaviour model is modified such that rule r_2 is being affected as follows:

```
Rule  $r_2'$ 
Event:      SearchSuccess(WebSite, Keywords, Statistics)
Condition:  Keywords={Labour, election} and WebSite=jpost
Action:     StoreSearchResults(John Doe, jpost, {Labour, election}, Statistics)
           jpost.RepeatedTrials:= 0
```

4 Conclusion

The paper presents a metadata model that is particularly well suited for distributed heterogeneous systems. It is readily seen that the model can capture both design modifications and changes to the data source structure. Towards this end, the model provides a useful design tool for contemporary data-based applications.

The use of a dependency graph enables the designer to define alternative paths for achieving a task. A path can be chosen during execution, based on a confidence metric, associated with nodes in the graph. Unfortunately, identifying the best path from a given set of alternatives in a polygraph is \mathcal{NP} -complete.³ Therefore, hill climbing heuristics (e.g. “at each crossroad choose the node with the highest confidence metric”) can be applied at run-time.

Further research involves the applicability of the model to various applications, including Geosciences [8] and multimedia [6]. Another issue deserving attention is that of automatic and semi-automatic propagation of modifications to the data sources into model modifications. A further research is required into categorising such possible modifications and identifying possible propagations to the modifications.

²It is worth noting that the information retrieval is outside the scope of the application.

³This claim can be proven using a straight-forward reduction to the acyclicity of a polygraph, shown to be \mathcal{NP} -complete in [10].

References

- [1] S. Chawathe, H. Garcia-Molina, H. Hammer, K. Ireland, Y. Papakonstantinou, J.D. Ullman, and J. Widom. The TSIMMIS project: Integration of heterogeneous information sources. In *Proceedings of IPSJ*, Tokyo, Japan, 1994.
- [2] A. Gal. Modelling cooperation among information systems using control elements. In *the Proceedings of the Third International Workshop on Next Generation Information Technologies and Systems*, pages 125–132, Neve-Ilan, Israel, July 1997.
- [3] A. Gal and O. Etzion. A multiagent update process in a database with temporal data dependencies and schema versioning. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 10(1):21–37, 1998.
- [4] A. Gal, O. Etzion, and A. Segev. TALE — a Temporal Active Language and Execution model. In P. Constantopoulos, J. Mylopoulos, and Y. Vassiliou, editors, *Advanced Information Systems Engineering*, pages 60–81. Springer, May 1996.
- [5] A. Gal and J. Mylopoulos. Supporting distributed autonomous information services using coordination. *International Journal of Cooperative Information Systems*, 9(3):255–282, 2000.
- [6] J. Griffioen, R. Yavatkar, and R. Adams. Automatic and dynamic identification of metadata in multimedia. In *Proc. First IEEE Metadata Conference*, April 1996. available at <http://www.computer.org/conferen/meta96/adams/paper.html>.
- [7] L. Liu, C. Pu, and Y. Lee. An adaptive approach to query mediation across heterogeneous information sources. In *Proceedings of the International Conference on Cooperative Information Systems (CoopIS'96)*, pages 140–156, Brussels, Belgium, June 1996.
- [8] D.J. Medyckyj-Scott, I. Newman, C. Ruggles, and D. Walker, editors. *Metadata in the Geosciences*. Group D Publications, Loughborough UK, 1991. ISBN 1-874152-00-4.
- [9] J. Mylopoulos, A. Gal, K. Kontogiannis, and M. Stanley. A generic integration architecture for cooperative information systems. In *Proceedings of the First IFCIS International Conference on Cooperative Information Systems (CoopIS'96)*, pages 208–217, Brussels, Belgium, June 1996.

- [10] C.H. Papadimitriou, P.A. Bernstein, and J.B. Rothnie Jr. Computational problems related to database concurrency control. In *Proceedings of the Conference on Theoretical Computer Science*, 1977.
- [11] A. Sheth and J. Larson. Federated database systems for managing distributed, heterogeneous, and autonomous databases. *ACM Computing Surveys*, 22(3):183–236, 1990.