A Behavioral Model for Rich Internet Applications

Sara Comai and Giovanni Toffetti Carughi

Dipartimento di Elettronica, Politecnico di Milano
P.zza L. da Vinci, 32 - I20133 Milano
comai@elet.polimi.it, toffetti@elet.polimi.it

Abstract. Rich Internet Applications (RIAs) are reshaping the way in which the Web works. They change not only the appearance of the Web interfaces, but also the behavior of applications, permitting novel operations, like data distribution, partial page computation, and disconnected work. In this paper we try to understand the differences between the behavior of traditional dynamic Web applications and RIAs, considering the WebML modeling language and its actual implementation.

1 Introduction

In a "traditional" dynamic HTML-based application the interface is an HTML document, computed by the server at each user's request. When the user interacts with the page, by following an hyperlink or submitting a form, the server is invoked and the destination page is computed from scratch and sent back to the client. The role of the client is to intercept the user's action, deliver the request to the server, and display the response.

In a Rich Internet Application (RIA), the client is assigned a fraction of the data and of the computation, so that the user can perform a complex interaction with the interface without invoking the server, unless data needs to be exchanged. Furthermore, if the user's interaction requires a server round-trip for refreshing some data, the client can selectively retrieve from the server only the information that need to be changed, update its internal status, and redisplay the modified content.

This paper investigates RIAs by analyzing their behavior in order to understand if they constitute only a technological improvement over the architecture of conventional dynamic Web applications or if they can alter significantly the way in which Web applications behave, and therefore pose novel challenges to their design, implementation, and evaluation. As a reference model we consider WebML [2].

2 Traditional Web Application Model

A traditional dynamic Web application is described by its structure and behavior, as defined by the following models.

L. Baresi, P. Fraternali, and G.-J. Houben (Eds.): ICWE 2007, LNCS 4607, pp. 364–369, 2007.
© Springer-Verlag Berlin Heidelberg 2007

The **structural model** comprises a *data model*, which specifies the content objects underlying the applications (e.g., represented by an Entity-Relationship or an UML class diagram), and an *interface model* (or hypertext model), which describes the front-end exposed to the user.

The key ingredients of the interface model are: 1) content components[1] , 2) interface containers, 3) parameter passing, and 4) interaction mechanisms.

1) A *content component* is used to extract content stored in the data layer and is characterized by: a (possibly empty) set of *input parameters*, a *query*, to retrieve the desired objects, exploiting the values of the input parameters; a *population*, storing the result of the query, and a (possibly empty) set of *output parameters*.
2) *Interface containers* are structured collections of content components. In the Web context, they typically represent Web *pages*, or sub-modules of pages and can be organized hierarchically.
3) *Parameter passing* is represented by *parameter passing rules*. A parameter passing rule expresses a dependency between a pair of components; it consists of a *source component*, a *destination component*, and a *mapping* between the output parameters of the source and the input parameters of the destination.

4) *Web interaction mechanisms* are specified by means of *links*, representing hyperlinks and input submit commands, typically rendered as anchors or form buttons. A link connects a source and a destination component and is associated with one parameter passing rule.

The **dynamic model** of a traditional Web application explains what happens when the user accesses a page or navigates its internal links. The behavior of a traditional Web application can be automatically inferred from the structural model presented above. In this section we present a sketch of the the page computation algorithm adopted by WebML and highlight the main characteristics of the behavior of traditional Web applications. A formal version of the WebML semantics defined by means of Statecharts can be found in [3].

In traditional Web applications page computation occurs entirely at the server: a page request causes the page to be computed from scratch as described by the following algorithm.

```
INPUT: page to be computed, initial assignment of input parameters IP
OUTPUT: computed page
CURRENTINPUT=IP; COMPONENTS=FindComputable(CURRENTINPUT);
WHILE (COMPONENTS is not empty) DO {
  C=Choose(COMPONENTS);
  I=ChooseInput(C);
  OutP=Execute(C, I);
  FORALL Rule in ParameterPassingRules(C)
      CURRENTINPUT += AssignInput(Rule.destination,OutP);
  COMPONENTS=FindComputable(CURRENTINPUT);
}
```

[1] For brevity we don't consider business components (i.e. WebML operations) here.

The page computation process starts from the requested page and from an initial assignment of values to the input parameters of the content components (e.g., the OID of the object selected from an index, the values input in a form). The maximal set of content components is then computed, starting from the received parameters and respecting a partial order imposed by the parameter passing rules: the set of computable units is determined (procedure *FindComputable*); one of such units is selected for execution (procedure *Choose*); the input parameters to be used are chosen (*ChooseInput*) and the query/method of the unit is executed and output parameters are computed (*Execute*). The output parameters are then passed through the parameter passing rules to the destination components (*AssignInput*).

As an example, consider the hypertext in Figure 1 showing the list of folders, a selected folder, the list of messages contained in this folder and a selected message. L1 is a navigation link recording the last selected value (it is an history link, denoted with h), L2 is a parameter passing rule not associated with a navigable link, L3 is a navigation link with no history. When the user selects a new message in the current folder by navigating link L3, the page must be recomputed. The initial assignment of input parameters contains the id of the newly selected message, while the history contains the id of the current folder. In the first iteration, it can evaluate the *Folders* component, which does not require input parameters, and the *Message* component, whose input parameter comes from the initial assignment. When the *Folders* component is evaluated, it provides the value of the input parameter of the *Folder* component (taken from the history). After the *Folder* component is evaluated, the input parameter of the *Messages* component becomes available and also this component can be evaluated, which terminates the page computation process.

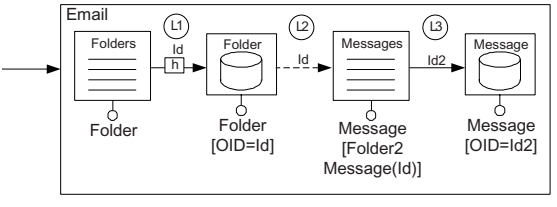

Fig. 1. An hypertext with four components, three parameter passing rules, and two navigational links

3 Rich Internet Application Dynamic Model

Rich Internet applications require the extension of the structural model with a distribution of the data and of the functionalities between the client and the server (see [1] for details) and a more flexible run-time behavior w.r.t. traditional applications. Upon the interaction of the user, the client can selectively request from the server only a portion of the data and maintain unchanged all the pieces of information that are not affected. Furthermore, the interaction of the user may

cause some pieces of content, which were previously displayed, to be invalidated because they are no longer consistent with the rest of the page.

A novel dynamic model where each link is associated with the notion of *computation sequence* (or *sequence*, for short) is needed; when a link is navigated, the dynamic model dictates explicitly the effects on all the components of the page. In a sequence, the following operators are used:

- *Evaluate*: causes the computation of the input parameters of a component and of its query. It is denoted by the term c_i++, where c_i is a component.
- *Refresh*: causes the computation of the query. It is denoted by the term c_i+.
- *Invalidate*: discards the input parameters of a component and its population. It is denoted by the term c_i--.
- *Empty*: discards the population of a component. It is denoted by c_i-.

Given a link, a sequence associated with it is *legal* if it satisfies the following constraints: 1) the operators affect components belonging to the page of the destination component; 2) each component appears at most once in the sequence associated with an *Evaluate* or a *Refresh* operator, thus avoiding cyclic computations; 3) the order of evaluation of the components is such that all the components that may provide input parameters to a given component are evaluated before the dependent component.

The proposed dynamic model subsumes the traditional one, and is able to express also more complex behaviors.

Examples. Consider the interface model of Figure 1.

The sequence on link L1: Folder++, Messages++, Message– specifies a behavior where only the components directly depending on the navigated link are updated, whereas the remaining components are no longer displayed. In other words, only the affected information is requested again to the server, as customary in AJAX and FLASH interfaces.

The sequence on link L1: Folder++, Messages++ specifies instead that when link L1 is navigated, the *Message* component is kept fixed and continues to show the same information irrespective of the (possibly repeated) change of folder. Such a behavior may be chosen to spare a data request to the server, and defer the re-evaluation of the displayed message to the time when the user selects a new message.

Extended page computation algorithm. The page computation process for the extended model is represented by the following algorithm:

```
INPUT: page, computation sequence Seq,
navigated link L, initial parameter assignment IP;
OUTPUT: computed page
AssignInput(L.destination,IP);
FOREACH (op in Seq) do {
  SWITCH (op.type){
    case Evaluate:
         (I,AffectedQuery)=ChooseInput(op.component);
         OutP=Execute(op.component, I, AffectedQuery); break;
```

```
    case Refresh:
         OutP=Execute(op.component); break;
    case Invalidate:
         op.component.Input=op.component.Population=null;
         op.component.Output=null; break;
    case Empty:
         op.component.Population=null;
         op.component.Output=null;
  }
  FORALL Rule in ParameterPassingRules(op.component)
    AssignInput(Rule.destination,null);
}
```

4 Discussion

The study and the implementation of the page computation algorithms of the traditional and of the Rich Internet Applications models allowed us to better evaluate their main differences.

The behavior of the whole application becomes more difficult to understand, since it is given by the combinations that can be obtained through the application of the computation sequences associated with the enabled navigational links. The specified behavior should always guarantee properties such as computation termination and determinism, triggerability of links and computability of components, and so on. The traditional model presented in Section 2 guaranteed most of such properties, while the new extended model introduces also new issues. Here we overview only some properties of the proposed models, summarized in Table 1.

Table 1. Traditional vs RIAs guaranteed properties

Property	Traditional	Rich Internet Application
Component computability	NO (cycles)	NO (due to cycles and/or partial computation)
Reachability	YES	NO (due to partial computation)
Freshness	YES	NO (due to partial computation)
Consistency	YES	NO (due to partial computation)

Non-*computability* of a component can arise in both applications. It occurs when a component may never receive the needed input parameters: this may happen in particular configurations with cycles among components. In RIAs, non-computability may also originate from computation sequences in which explicit invalidation actions prevent a component from receiving all its needed parameters. For example, in Figure 1 the computation sequence (L1: Folder++, Messages–; Message++) associated to the navigation of link (L1) cannot produce the content of the Message component.

The proposed RIA model is then affected by further issues due to possibility of specifying partial computations. First of all, partial computations may cause non-*reachability* of components and triggerability of links: the set of sequences

associated with the links of a page may not allow to compute components and therefore trigger their outgoing navigational links. The model for the traditional Web applications, computing the maximal set of content components and using the parameter passing rules at each user's interaction, can instead guarantee such properties.

Moreover, stale data may be present in the interface. In Figure 1, if the computation sequence associated with link L3 is (Message++), only the *Message* component is recomputed. The list of messages is not recomputed and, therefore, new messages available at the server are not shown. *Data freshness* problems do not arise in traditional dynamic Web applications, because pages are always entirely recomputed from the data currently stored in the data layer.

Finally, *consistency* problems may arise. By consistency we mean that if a component receives inputs from another component, then whenever the population of the first component is evaluated and new output parameters are produced, also the input parameters and the population of the dependent component are evaluated. This problem does not arise with the proposed algorithm for traditional Web applications, since the computation always starts from scratch by evaluating the maximal set of components and parameter passing rules apply; it may instead occur in RIAs when partial computations are specified. Consider Figure 1 and the computation sequence (L1: Folder++, Messages++) associated to the navigation of link L1; consistency is not guaranteed between the *Messages* component and the *Message* component that is not recomputed: indeed, if the user selects a new folder, a message belonging to a different folder is shown.

5 Conclusions

In this paper we have presented a dynamic model for Web applications, distinguishing between traditional Web applications and Rich Internet Applications. In case of traditional Web applications a default behavior may be associated with the structural model as it has been done for the WebML language. RIAs can exhibit richer and more flexible behaviors that cannot be specified through a standard behavioral model. In particular, the computation of RIAs typically involves partial fragments of the interface: this may introduce several new issues that need to be considered in the design, implementation, and evaluation of RIAs.

References

1. Bozzon, A., Comai, S., Fraternali, P., Toffetti Carughi, G.: Conceptual modeling and code generation for rich internet applications. In: Wolber, D., Calder, N., Brooks, C., Ginige, A. (eds.) ICWE, pp. 353–360. ACM Press, New York (2006)
2. Ceri, S., Fraternali, P., Brambilla, M., Bongio, A., Comai, S., Matera, M.: Designing Data-Intensive Web Applications. Morgan Kaufmann, Seattle, Washington (2002)
3. Comai, S., Fraternali, P.: A semantic model for specifying data-intensive web applications using webml. In: Semantic Web Workshop, Stanford, USA, July (2001)