Content Request Markup Language (CRML): a Distributed Framework for XML-based Content Publishing

Chi-Huang Chiu, Kai-Chih Liang, Shyan-Ming Yuan
Dept. of Computer & Information Science, National Chiao Tung University
1001 Ta-Hsueh Rd,
HsinChu 300, Taiwan
+886-952-729654

chchiu/kcliang/smyuan@cis.nctu.edu.tw

ABSTRACT
Construct web applications to provide dynamic, personalized web contents with high scalability and performance is a challenge to the software industry in the new Internet era. In most available solutions, load balancing and caching mechanisms are introduced in front of web servers to reduce workload. In this paper we present Content Request Markup Language (CRML), an enabling techniques for distributed XML processing at the content level. CRML is a language based on emerging XML standards, XSLT and XPATH, to publish XML-based content over HTTP protocol. It provides hints to construct a distributed framework to support parallel XML-based content publishing. In addition, the content from databases or other sources could be cached before or after processing in block or page level. With the parallel content publishing and the caching mechanism, the CRML could provide a high performance platform for fully customized web service.

Keywords
Load balancing, Caching, Personalization, XML

1. INTRODUCTION
The volume of Internet content was getting higher and higher in the past few years. Meanwhile, dynamic multimedia contents are becoming the emerging demands from users. Content management, especially dynamic contents, is the issue that every service providers must face with.

The CRML is a content publishing framework that enforces distributed computation behind the Web server when managing Internet content. There are three major features of the CRML.

Pure XML. CRML is the markup language derived from XML. Contents are marked by CRML tags, which provides hints for distributed processing of the content.

Parameter-triggered caching mechanism. To increase the reusability of the cache data, CRML implementation utilize the query parameter to further reuse cached query result with different query parameters.

Parallel Processing. The implementation of CRML enforces the parallel process behinds the Web server.

2. The Language Definition
In CRML, the dynamic content should be embedded in the document prepared to return described by a CRB: Content Request Block. Each CRB should represent a block of data and contain the data source, inter-CRB communication, exception handling, parallel processing hint, and caching informtion.

2.1 Content Request Block
```
<crml:content id="auth"
 xmlns:crml="http://nctu/CRML">
<crml:concunrrency threadsafe="no"/>
 <crml:source type="sql">
  <crml:attr name="db" value="jdbc:db2:sample"/>
  [!CDATA[select name from users where
id='@{id}' and password='@{PASSWD}']]
 </crml:source>
 <crml:cache key=”id,password”/>
 <crml:variable name="name"
catch="RESULT/ROW[0]/id"/>
 <crml:exception code="100" type="page">
 <?xml-stylesheet href="error.xsl" type="xsl"?>
 <error type="autherror">
The User/Password you inputted is not correct !
</error>
 </crml:exception>
</crml:content>

<crml:content id="secdata"
xmlns:crml="http://nctu/CRML">
 <crml:concunrrency threadsafe="no"
wait="auth"/>
 <crml:source type="file">
  <crml:attr name="url"
value="file://c/doc/dept.xml"/>
 </crml:source>
</crml:content>
```

The above example shows two CRB in a document one of them is contain the data from the database and the other is from a local file. In the <crml:source> tag, the inner nodes will be passed to the processor of specified data source.

Current implementation includes JDBC, RMI, CORBA, local files, HTTP, and Java Source Code. No matter what type of data source is used, the result type of the data source must be XML. The final result could be process by a XSLT process if need.

2.2 Inter-CRB Communication
The inter-CRB communication is archived by the data exchange via page-level variable. Like the <crml:variable> tag in the previous code fragment, the XPATH string is used to extract the information from the data source to specified variable.

2.3 Exception Handling
If any exception is thrown by the data source, the <crml:exception> tag will catch it. If the type of the exception is page, the inner node of the tag will replace the whole result interrupt all other running or waiting CRB. Otherwise, it will replace the result of the CRB.

2.4 Parallel Processing Hint

The <crml:concurrency> tag provide the relationship between all CRB. The **threadsafe** attribute defines that if the CRB is thread-safe and the **wait** attribute defines the CRBs which should be executed before this CRB.

2.5 Page-Level Processing Instructions

In addtion to the CRBs embedded in the document, some page-level processing instructions are used to do the page-level caching, transformation, and error handling.

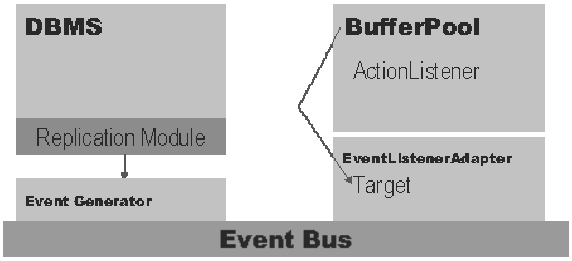
3. Parameter-triggered Caching Mechanism

The caching mechanism in this framework is easy. The concept is that the same parameter will get the same content. Each CRB include a cache tag to describe the parameter affect the result like the <crml:cache> tag in previous example. The CRB will not be processed if the same parameter could be found in the cache (Buffer Pool). Besides, to generate content with the dynamic information, the invalidation is done to remove expired content in the cache.

3.1 Self-Invalidation

Some content (such as HTTP and files) has the information about the modification or expiration information, which could be used to invalidate the cache.

3.2 Event-driven Invalidation



The above figure is an example of the event-driven invalidation scenario, which the event source is a DBMS. An event generator could be set via the replication module on the DBMS to publish the event. The buffer pool with an event listener adapter could listen the events on the bus to invalidate the out-of-date information.

3.3 Expression Variable

```
<crml:cache name="userquery" keys="~{#{min}/5}" />
```

The example shows a notation of Expression Variable, which could enclose an expression as a variable. In such example, the variable ~{#{min}/5} represent the value of the numbers of minutes from 1970 divide 5. The variable has the same value in the same block of 5 minutes and could reduce the effort of processing. Besides, the name of the cached element could let the CRBs in different CRML document share the same cache data. The scheme is suitable for the CRB, which is accessed frequently, but the real-time accuracy is not pretty important.

4. Implementation for Parallel Processing

The CRML doesn't include the detail information about how to process a CRML. It only contains some requirement of partial sequence and exclusion and is much depended on the implementation.

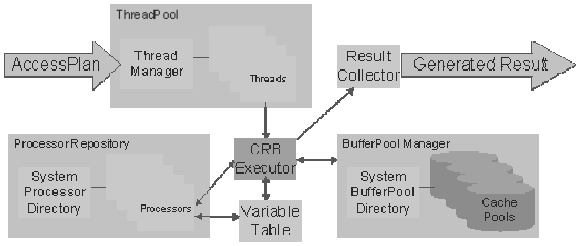
4.1 The CRML Engine

The CRML Engine is the component to process a whole request. First, the request will be associated with a CRML source file and be passed to the Engine with request parameters. Second, the compiler will be invoked to compile the CRML source file and generate the process plan of the request. Third, the process plan will be sent to the runtime and executed. Finally, if the result from the runtime contains another CRBs to process, the compiler will try to generate a process plan for next round and some page-level caching will be applied here.

4.2 The Process Plan

The process plan generated by the compiler will seperate the CRBs to several thread. The two CRB in the same thread represent that the dependence of them. In other words, it is useless if you execute the two CRBs concurrently because one CRB may wait the other one.

4.3 The CRML Runtime



Like the above, the access plan will be dispatched to several threads to execute. The CRB Executor is the core of each thread that executes the CRBs by the help of CRB Processors and Buffer Pools. The Result Collector could collect and manage the results and exceptions from each thread. The semaphores in the Result Collector also do the concurrency control. Such runtime could be implemented on the distributed environment easily if the centralized-controlled Result Collector is ready.

5. Conclusions and Future Works

In this paper, we have presented the core of CRML, a distributed framework for XML-based Content Publishing. Comparing with other publishing framework or tools, CRML: 1) stresses the distributed architecture which could be used either to do the load balancing for a heavy traffic website or to perform multithread processing in a single machine to improve performance; 2) provides the block-level and content-independent cache for the data before processing which is a great improvement for fully-customized web service; 3) uses the emerging XML standards to process the XML-based data without any sequential programming language.

CRML is a part of WebEngine, a developing project of Web Content Management System. The concept of WCMS is like the DBMS, which both manage the data and handle the request. As the role of SQL in DBMS, the CRML is designed to describe what user want to request not how to handle the request. Such spirit could let WebEngine improve the performance using a lot of techniques without change the original CRML source file. Since the CRML is used to request the content, the transaction management is not mentioned in the first version of design, which may be improved as future works.

6. Acknowledgment

This work was supported by both the National Science Council grants NSC88-2213-E-009-087, and NSC89-2213-E009-069.