OptimAX: Optimizing Distributed ActiveXML applications

Serge Abiteboul, Ioana Manolescu and Spyros Zoupanos

[1] GEMO group, INRIA Saclay – Île-de-France

July 18, 2008

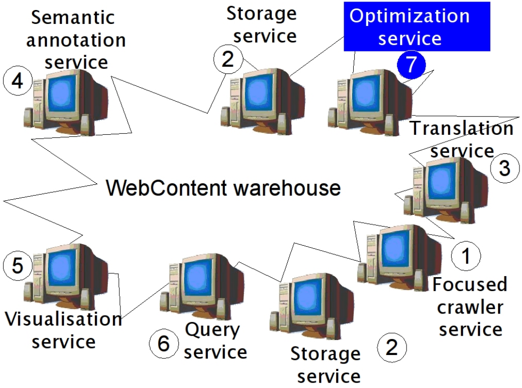
Outline

1. Overview
   - WebContent project
   - ActiveXML language
2. A framework for AXML optimization
   - Extended AXML
   - AXML rewriting
3. OptimAX
   - Design Principles
   - Performance
4. Related (sub) problems
5. Conclusion

Outline

Distributed web service based applications

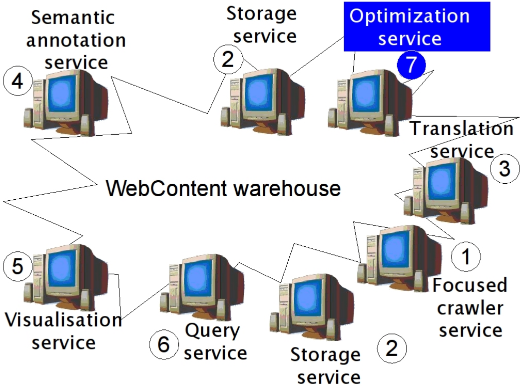
WebContent project (R&D project)



- Different services provided by different partners.
- Service call interaction described in (A)XML documents.
- Optimizer needed to optimize the execution plans.

Distributed web service based applications

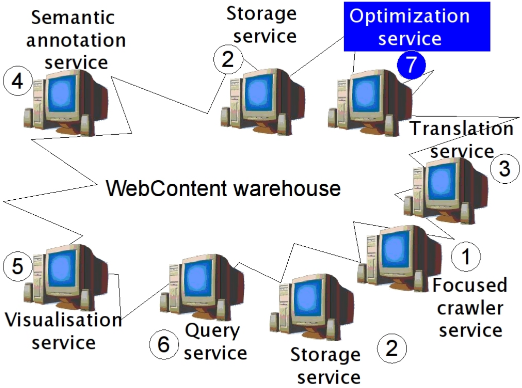
WebContent project (R&D project)



- Different services provided by different partners.
- Service call interaction described in (A)XML documents.
- Optimizer needed to optimize the execution plans.

Distributed web service based applications

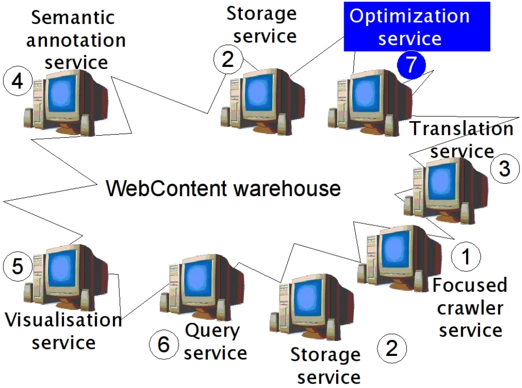
WebContent project (R&D project)



- Different services provided by different partners.
- Service call interaction described in (A)XML documents.
- Optimizer needed to optimize the execution plans.

Distributed web service based applications

WebContent project (R&D project)



- Different services provided by different partners.
- Service call interaction described in (A)XML documents.
- Optimizer needed to optimize the execution plans.

The ActiveXML language

Data-centric Web service composition

ActiveXML document = XML document including calls to (continuous) Web services

- A service call contains contact info for the Web service
- When the calls is activated, results are added to the document as siblings of the service call.

```
<myPage>
 <axml:sc service="getProgram" peer="tvchannel.com">
  <parameter>Movies</parameter>
 </axml:sc>
</myPage>
```

The ActiveXML language

Data-centric Web service composition
ActiveXML document = XML document including calls to (continuous) Web services

- A service call contains contact info for the Web service
- When the calls is activated, results are added to the document as siblings of the service call.

```
<myPage>
 <axml:sc service="getProgram" peer="tvchannel.com">
  <parameter>Movies</parameter>
 </axml:sc>
 <program day="today">
  <movie>Shrek 3</movie>
 </program>
</myPage>
```

The ActiveXML language

Data-centric Web service composition
ActiveXML document = XML document including calls to (continuous) Web services

- A service call contains contact info for the Web service
- When the calls is activated, results are added to the document as siblings of the service call.

```
<myPage>
 <axml:sc service="getProgram" peer="tvchannel.com">
  <parameter>Movies</parameter>
 </axml:sc>
 <program day="today">
  <movie>Shrek 3</movie>
 </program>
 <program day="tomorrow">
  <movie>Persepolis</movie>
 </program>
</myPage>
```

The ActiveXML language

Data-centric Web service composition

ActiveXML document = XML document including calls to (continuous) Web services

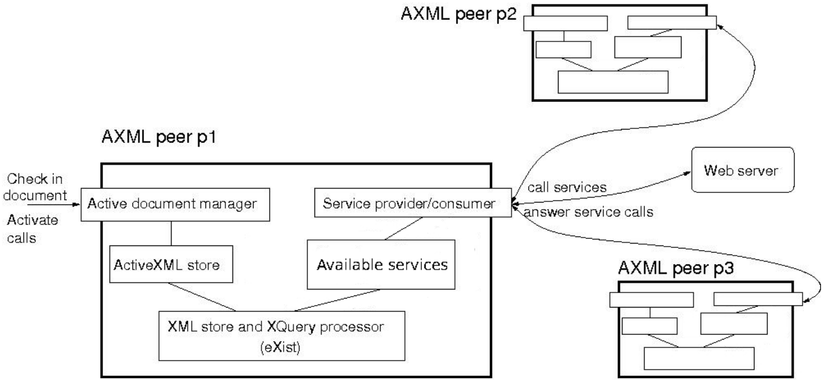
- A service call contains contact info for the Web service
- When the calls is activated, results are added to the document as siblings of the service call.

Continuous services may be continuous query services.

Example

Please inform me whenever there is a movie in the TV program (query over an RSS feed).

The ActiveXML peer v2: overview



AXML evaluation

AXML document $d@p_1$, $sc \in d@p_1$ is the call $s@p_2(\$in)$
Activating sc entails:

- Stream $\$in$ to p_2
- Evaluate $s@p_2$
- Stream the results of $s@p_2$ to p_1

Remark

$\$in$ may contain (continuous) service calls

Remark

$s@p_2$ results may contain (continuous) service calls

Call activation can be controlled:

- On a per-call basis
- Globally (per-subtree basis)

AXML evaluation

AXML document $d@p_1$, $sc \in d@p_1$ is the call $s@p_2(\$in)$
Activating sc entails:

- Stream $\$in$ to p_2
- Evaluate $s@p_2$
- Stream the results of $s@p_2$ to p_1

Remark

$\$in$ may contain (continuous) service calls

Remark

$s@p_2$ results may contain (continuous) service calls

Call activation can be controlled:

- On a per-call basis
- Globally (per-subtree basis)

AXML evaluation

AXML document $d@p_1$, $sc \in d@p_1$ is the call $s@p_2(\$in)$
Activating sc entails:

- Stream $\$in$ to p_2
- Evaluate $s@p_2$
- Stream the results of $s@p_2$ to p_1

Remark

$\$in$ may contain (continuous) service calls

Remark

$s@p_2$ results may contain (continuous) service calls

Call activation can be controlled:

- On a per-call basis
- Globally (per-subtree basis)

AXML evaluation

AXML document $d@p_1$, $sc \in d@p_1$ is the call $s@p_2(\$in)$
Activating *sc* entails:

- Stream $\$in$ to p_2
- Evaluate $s@p_2$
- Stream the results of $s@p_2$ to p_1

Remark

$\$in$ may contain (continuous) service calls

Remark

$s@p_2$ results may contain (continuous) service calls

Call activation can be controlled:

- On a per-call basis
- Globally (per-subtree basis)

Contributions

A framework for AXML optimization

- A small set of predefined services
- Precise evaluation semantics for AXML documents
- Equivalence-preserving AXML rewriting rules
- Classification of AXML optimization problems

OptimAX, an extensible AXML optimizer

- Search strategies
- Performance

Contributions

A framework for AXML optimization

- A small set of predefined services
- Precise evaluation semantics for AXML documents
- Equivalence-preserving AXML rewriting rules
- Classification of AXML optimization problems

OptimAX, an extensible AXML optimizer

- Search strategies
- Performance

Outline

Extending AXML

Purpose:

- Modest extensions turning AXML into a rich "executable" language
- Optimization = AXML-to-AXML equivalence-preserving rewriting
- Three new services:
  - `send`
  - `receive`
  - `newNode`

Extending AXML

Purpose:

- Modest extensions turning AXML into a rich "executable" language
- Optimization = AXML-to-AXML equivalence-preserving rewriting
- Three new services:
  - send
  - receive
  - newNode

Extending AXML

Purpose:

- Modest extensions turning AXML into a rich "executable" language
- Optimization = AXML-to-AXML equivalence-preserving rewriting
- Three new services:
  - `send`
  - `receive`
  - `newNode`

AXML as an executable language

How to order the activations of several calls?

- Default activation order: inside-out (activate parameter calls before the parent call)
- User-specified activation order:
  - sc_1 activated afterActivated | afterTerminated sc_2

The send service

Sends (a stream of) (A)XML trees as children of a given node

```
<axml:sc service="send" peer="p" id="#1">
  <where node="#2" doc="d2" peer="p2"/>
  <what> ... AXML expression ... </what>
</axml:sc>
```

Default activation order for send

A call to send is activated before activating the descendant calls.

The receive service

Receives (a stream of) (A)XML trees at a given node

```
<axml:sc service="receive" peer="p2" id="#2">
  <from node="#1" doc="d1" peer="p1"/>
  <what> ... AXML expression ... </what>
</axml:sc>
```

Default activation order for receive

A call to `receive` is activated when the first message from the corresponding `send` arrives.

The `what` child of `receive` only describes data being received. Its calls are not activated.

Global integrity constraint: `send` $\Leftrightarrow$ `receive`

The receive service

Receives (a stream of) (A)XML trees at a given node

```
<axml:sc service="receive" peer="p2" id="#2">
  <from node="#1" doc="d1" peer="p1"/>
  <what> ... AXML expression ... </what>
</axml:sc>
```

Default activation order for receive

A call to `receive` is activated when the first message from the corresponding `send` arrives.

The `what` child of `receive` only describes data being received. Its calls are not activated.

Global integrity constraint: `send` ⇔ `receive`

The receive service

Receives (a stream of) (A)XML trees at a given node

```
<axml:sc service="receive" peer="p2" id="#2">
  <from node="#1" doc="d1" peer="p1"/>
  <what> ... AXML expression ... </what>
</axml:sc>
```

Default activation order for receive

A call to `receive` is activated when the first message from the corresponding `send` arrives.

The `what` child of `receive` only describes data being received. Its calls are not activated.
Global integrity constraint: `send` $\Leftrightarrow$ `receive`

The newNode service

Installs an XML tree as a new document on a peer.

```
<axml:sc service="newNode" peer="p3" id="#3">
  <what> ... AXML expression ... </what>
</axml:sc>
```

Default activation order for newNode

A call to newNode is activated before activating the descendant calls.

AXML activation order

Given a document $d@p$, the AXML peer p computes a partial order $\mathscr{O}$ including:

- all explicit activation order constraints
- as many default order constraints as possible

There can be several legal schedules.

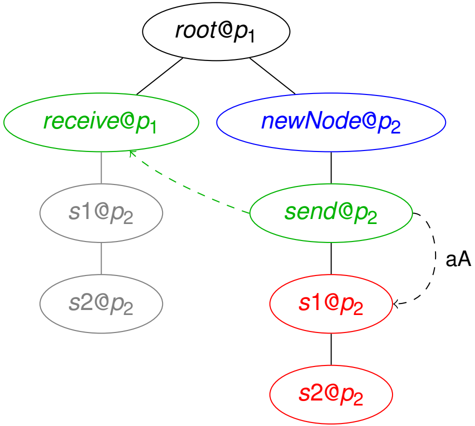
AXML activation order

Given a document $d@p$, the AXML peer p computes a partial order $\mathscr{O}$ including:

- all explicit activation order constraints
- as many default order constraints as possible

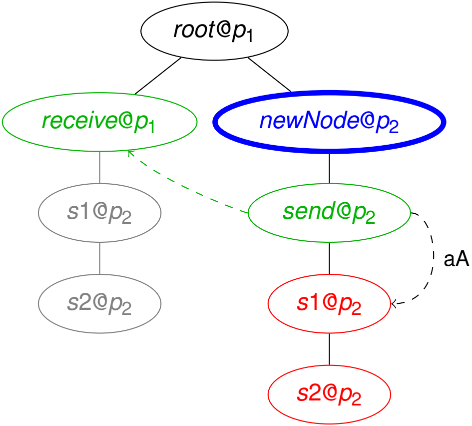
There can be several legal schedules.

Sample legal schedule



- Our document is installed at peer1.
- We want the s1@peer2 and the s2@peer2 to be called by peer2.
- We are interested in receiving the final answer at peer1.

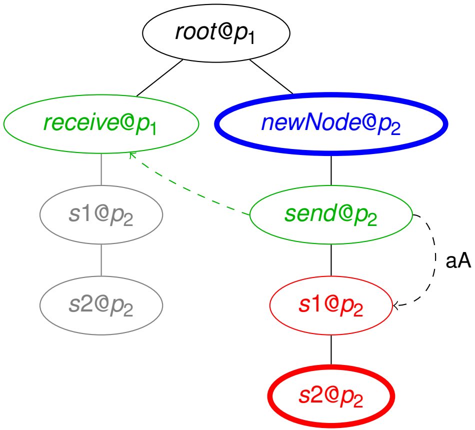
Sample legal schedule



Services activated:

- newNode@peer1

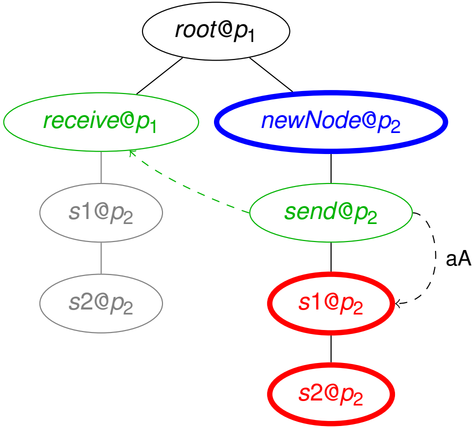
Sample legal schedule



Services activated:

- newNode@peer1
- s2@peer2

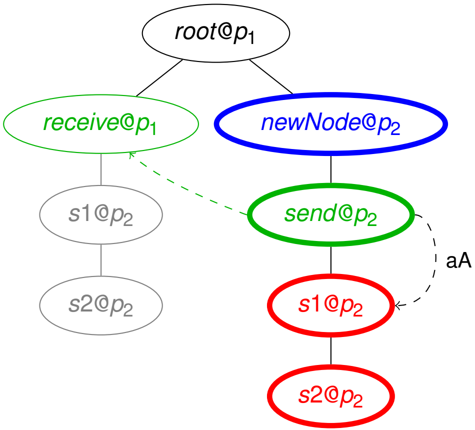
Sample legal schedule



Services activated:

- newNode@peer1
- s2@peer2
- s1@peer2

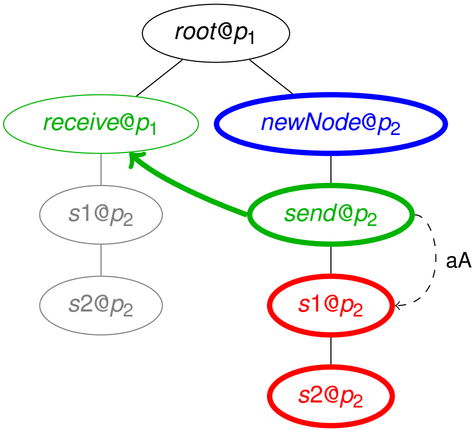
Sample legal schedule



Services activated:

- newNode@peer1
- s2@peer2
- s1@peer2
- send@peer2

Sample legal schedule



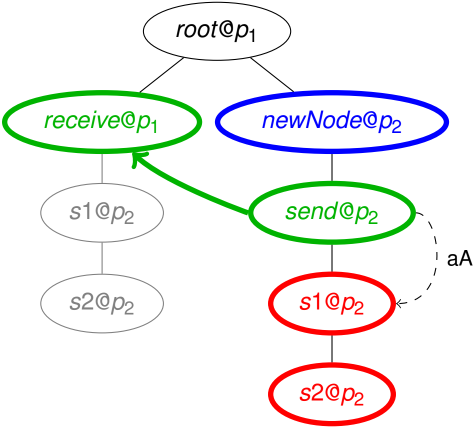
Services activated:

- newNode@peer1
- s2@peer2
- s1@peer2
- send@peer2

Data transmition

send@peer2 starts to send data to receive@peer1

Sample legal schedule



Services activated:

- newNode@peer1
- s2@peer2
- s1@peer2
- send@peer2
- receive@peer1

Receive activation

receive@peer1 is activated on first result received by send@peer2

AXML equivalence

AXML fixpoint: the final state of the document (assumed finite)

- Service calls may bring service calls
- Fixpoint is reached after full evaluation

Two documents are equivalent if their fixpoints are identical (modulo terminated service calls)

Two documents are one-stage equivalent if activating all their service calls leads to identical documents

- Do not activate calls included in the results

AXML equivalence

AXML fixpoint: the final state of the document (assumed finite)

- Service calls may bring service calls
- Fixpoint is reached after full evaluation

Two documents are equivalent if their fixpoints are identical (modulo terminated service calls)

Two documents are one-stage equivalent if activating all their service calls leads to identical documents

- Do not activate calls included in the results

AXML equivalence

AXML fixpoint: the final state of the document (assumed finite)

- Service calls may bring service calls
- Fixpoint is reached after full evaluation

Two documents are equivalent if their fixpoints are identical (modulo terminated service calls)

Two documents are one-stage equivalent if activating all their service calls leads to identical documents

- Do not activate calls included in the results

AXML equivalence

AXML fixpoint: the final state of the document (assumed finite)

- Service calls may bring service calls
- Fixpoint is reached after full evaluation

Two documents are equivalent if their fixpoints are identical (modulo terminated service calls)

Two documents are one-stage equivalent if activating all their service calls leads to identical documents

- Do not activate calls included in the results

AXML equivalence

AXML fixpoint: the final state of the document (assumed finite)

- Service calls may bring service calls
- Fixpoint is reached after full evaluation

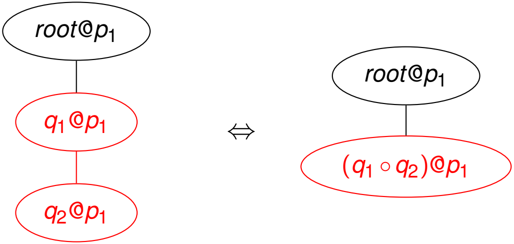
Two documents are equivalent if their fixpoints are identical (modulo terminated service calls)

Two documents are one-stage equivalent if activating all their service calls leads to identical documents

- Do not activate calls included in the results

Equivalence-preserving AXML rewriting rules

Rules specific to query services:

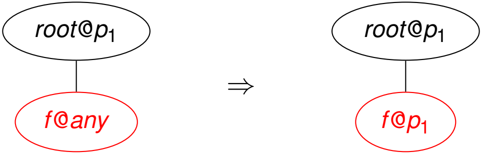


Query composition/decomposition

$q_1@p_1(q_2@p_1) \Leftrightarrow (q_1 \circ q_2)@p_1$

Equivalence-preserving AXML rewriting rules

Generic rules:

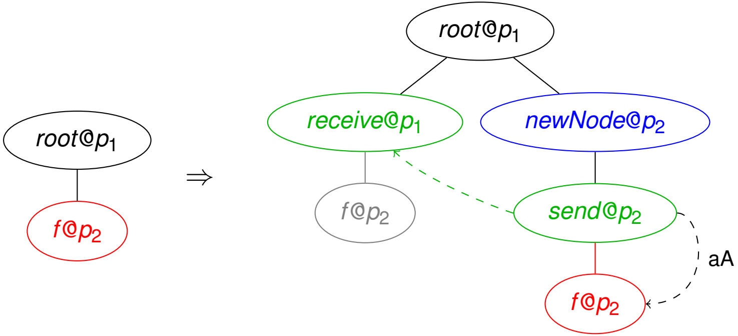


Instatiation

$f@any \Rightarrow f@p_1$ (the same f service)

Equivalence-preserving AXML rewriting rules

Generic rules:

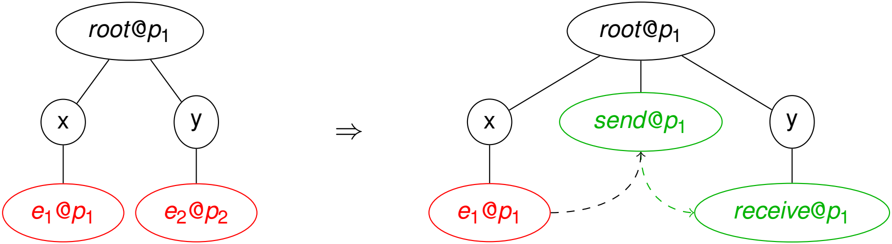


Delegation

$$f@p_1(e@p_2) \quad \Rightarrow \quad \begin{array}{l} \#1 : receive@p_1(e@p_2), \\ newNode@p_2(send@p_2(e@p_2, \#1@p_1)) \end{array}$$

Equivalence-preserving AXML rewriting rules

Generic rules:



Factorization

$$r(x(e_1),\ldots,y(e_2)) \quad\Rightarrow\quad r(x(\#1:e_1),$$
$$e_1 \equiv e_2 \qquad\qquad \#2:send@p_1(\#1@p_1,\#3@p_1),\ldots,$$
$$y(\#3:receive@p_1)$$
$$)$$

AXML optimization

Given:

- Rewriting rule set $\mathscr{R}$
- Cost function for *sc* evaluation

Full AXML optimization: repeat until fixpoint

1. choose one among
   1. pick an *sc* ready to be activated, activate it, add results to the document
   2. pick an AXML subtree t and a rule $r \in \mathscr{R}$, rewrite t with r
2. so that the total cost of evaluation (+optimization) is minimized

Undecidable if service calls may return other service calls.
In the decidable case, exhaustive optimization prior to any activation is optimal.

AXML optimization

Given:

- Rewriting rule set $\mathscr{R}$
- Cost function for *sc* evaluation

Full AXML optimization: repeat until fixpoint

1. choose one among
   1. pick an *sc* ready to be activated, activate it, add results to the document
   2. pick an AXML subtree t and a rule $r \in \mathscr{R}$, rewrite t with r
2. so that the total cost of evaluation (+optimization) is minimized

Undecidable if service calls may return other service calls.
In the decidable case, exhaustive optimization prior to any activation is optimal.

AXML optimization

Given:

- Rewriting rule set $\mathscr{R}$
- Cost function for *sc* evaluation

Full AXML optimization: repeat until fixpoint

1. choose one among
   1. pick an *sc* ready to be activated, activate it, add results to the document
   2. pick an AXML subtree t and a rule $r \in \mathscr{R}$, rewrite t with r
2. so that the total cost of evaluation (+optimization) is minimized

Undecidable if service calls may return other service calls.
In the decidable case, exhaustive optimization prior to any activation is optimal.

AXML optimization

Given:

- Rewriting rule set $\mathscr{R}$
- Cost function for *sc* evaluation

Full AXML optimization: repeat until fixpoint

1. choose one among
    1. pick an *sc* ready to be activated, activate it, add results to the document
    2. pick an AXML subtree t and a rule $r \in \mathscr{R}$, rewrite t with r
2. so that the total cost of evaluation (+optimization) is minimized

Undecidable if service calls may return other service calls.
In the decidable case, exhaustive optimization prior to any activation is optimal.

AXML optimization

Given:

- Rewriting rule set $\mathscr{R}$
- Cost function for *sc* evaluation

Full AXML optimization: repeat until fixpoint

1. choose one among
   1. pick an *sc* ready to be activated, activate it, add results to the document
   2. pick an AXML subtree t and a rule $r \in \mathscr{R}$, rewrite t with r
2. so that the total cost of evaluation (+optimization) is minimized

Undecidable if service calls may return other service calls.
In the decidable case, exhaustive optimization prior to any activation is optimal.

One-stage (static) optimization

Given a document $d@p$ and a set of rewriting rules $\mathscr{R}$

1. Let $S := \{d\}$
2. Repeat
   1. Pick a rule $r \in \mathscr{R}$, a document $d_1 \in S$ and a tree $t \in d_1$.
   2. Let $d_2 := r(d_1, t)$. If $d_2 \notin S$, add d_2 to S.
3. Until S stationary
4. Return cheapest plan from S

One stage optimization - Return cheapest document up to one stage equivalence to d.

One-stage (static) optimization

Given a document $d@p$ and a set of rewriting rules $\mathscr{R}$

1. Let $S := \{d\}$
2. Repeat
   1. Pick a rule $r \in \mathscr{R}$, a document $d_1 \in S$ and a tree $t \in d_1$.
   2. Let $d_2 := r(d_1, t)$. If $d_2 \notin S$, add d_2 to S.
3. Until S stationary
4. Return cheapest plan from S

One stage optimization - Return cheapest document up to one stage equivalence to d.

Outline

Generic ActiveXML optimizer: OptimAX

Available with the AXML peer v2 (www.activexml.net)

Extensible set of tree rewriting rules

Search algorithms: depth-first, breadth-first, cost-driven variants

Hint language:

- "Exhaust factorization, then 20 delegation steps"
- "Explore at most 50 rewritten plans"

Checks to preserve send-receive channel integrity

Generic ActiveXML optimizer: OptimAX

Available with the AXML peer v2 (www.activexml.net)
Extensible set of tree rewriting rules
Search algorithms: depth-first, breadth-first, cost-driven variants
Hint language:

- "Exhaust factorization, then 20 delegation steps"
- "Explore at most 50 rewritten plans"

Checks to preserve send-receive channel integrity

Generic ActiveXML optimizer: OptimAX

Available with the AXML peer v2 (www.activexml.net)
Extensible set of tree rewriting rules
Search algorithms: depth-first, breadth-first, cost-driven variants
Hint language:

- "Exhaust factorization, then 20 delegation steps"
- "Explore at most 50 rewritten plans"

Checks to preserve send-receive channel integrity

Generic ActiveXML optimizer: OptimAX

Available with the AXML peer v2 (www.activexml.net)
Extensible set of tree rewriting rules
Search algorithms: depth-first, breadth-first, cost-driven variants
Hint language:

- "Exhaust factorization, then 20 delegation steps"
- "Explore at most 50 rewritten plans"

Checks to preserve send-receive channel integrity

Generic ActiveXML optimizer: OptimAX

Available with the AXML peer v2 (www.activexml.net)
Extensible set of tree rewriting rules
Search algorithms: depth-first, breadth-first, cost-driven variants
Hint language:

- "Exhaust factorization, then 20 delegation steps"
- "Explore at most 50 rewritten plans"

Checks to preserve send-receive channel integrity

OptimAX performance

We measure: optimization time and reduction of estimated plan cost

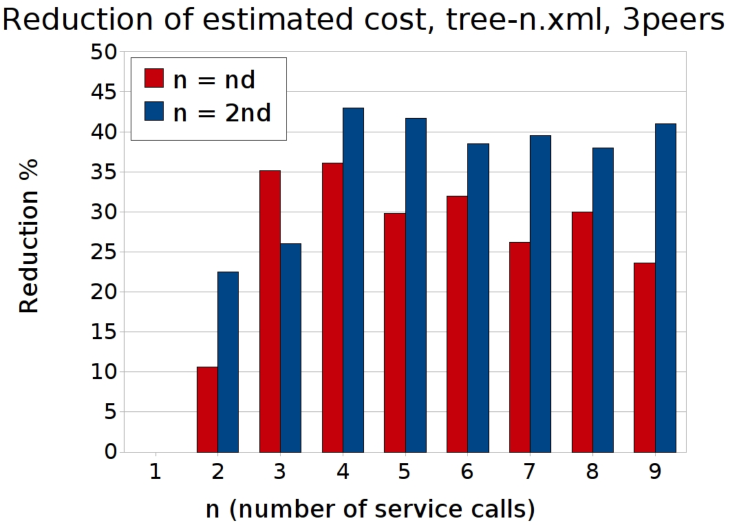
Synthetic documents:

- deep*n*.xml
- flat*n*.xml
- tree*n*.xml, max fan-out=6

Services assigned with uniform probability distribution over n_d services.

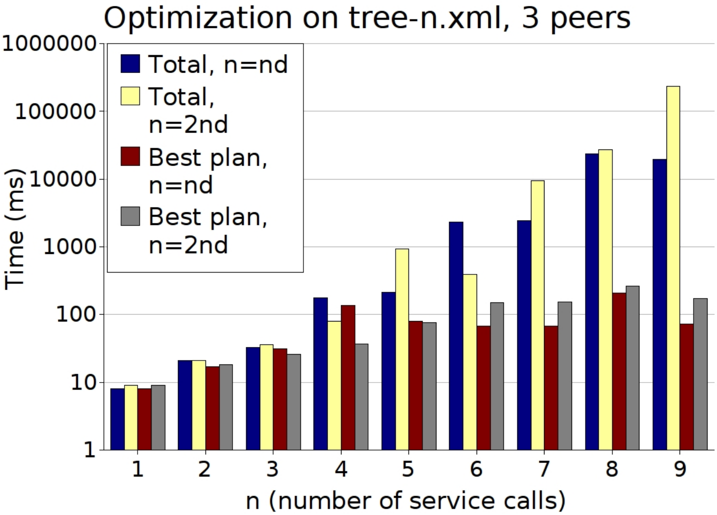
Optimization considers a network of p peers.

OptimAX performance



Search space size 3000, depth-first cost-driven strategy

OptimAX performance

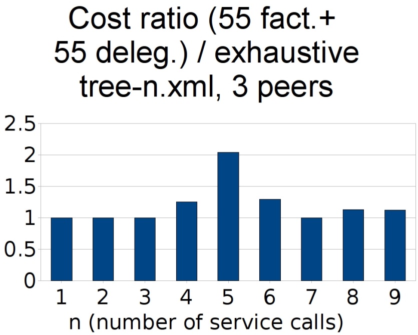


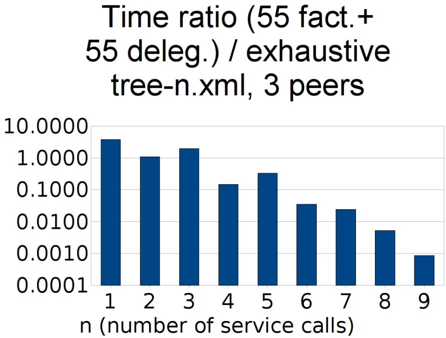
Depth-first cost-driven strategy

OptimAX performance

We compare an exhaustive search with a limited one:

- Cost ratio
- Time ratio





Outline

Related problems from previous works

Rewriting

Transforming an input or type τ_1 to an output of type τ_2 Extend XML Schema to include the types of services referred by each *sc* node.
Rewriting problem: find a sequence of activations which brings the document from type T_1 to type T_2 [MAA+03,AMB05].

Distribution

Assimilate service calls to remote tree references.
Query evaluation over an AXML document = local + remote evaluation.
Query shipping optimization rules [ABC+03].

Lazy evaluation

Decompose the query: $q@p_1(a(\alpha, f@p_2, \beta)) \Rightarrow q_1@p_1(a(\alpha, \beta)) \oplus q_2@p_1(f@p)$
Prune calls to f such that $q_2(f)$ is empty (irrelevant calls)
Full optimization algorithm (decomposition, call elimination) [ABC+04]

Related problems from previous works

Rewriting

Transforming an input or type τ_1 to an output of type τ_2 Extend XML Schema to include the types of services referred by each *sc* node.
Rewriting problem: find a sequence of activations which brings the document from type T_1 to type T_2 [MAA+03,AMB05].

Distribution

Assimilate service calls to remote tree references.
Query evaluation over an AXML document = local + remote evaluation.
Query shipping optimization rules [ABC+03].

Lazy evaluation

Decompose the query: $q@p_1(a(\alpha, f@p_2, \beta)) \Rightarrow q_1@p_1(a(\alpha, \beta)) \oplus q_2@p_1(f@p)$
Prune calls to f such that $q_2(f)$ is empty (irrelevant calls)
Full optimization algorithm (decomposition, call elimination) [ABC+04]

Related problems from previous works

Rewriting

Transforming an input or type τ_1 to an output of type τ_2 Extend XML Schema to include the types of services referred by each *sc* node.
Rewriting problem: find a sequence of activations which brings the document from type T_1 to type T_2 [MAA+03,AMB05].

Distribution

Assimilate service calls to remote tree references.
Query evaluation over an AXML document = local + remote evaluation.
Query shipping optimization rules [ABC+03].

Lazy evaluation

Decompose the query: $q@p_1(a(\alpha, f@p_2, \beta)) \Rightarrow q_1@p_1(a(\alpha, \beta)) \oplus q_2@p_1(f@p)$
Prune calls to f such that $q_2(f)$ is empty (irrelevant calls)
Full optimization algorithm (decomposition, call elimination) [ABC+04]

Outline

Conclusion

ActiveXML: very expressive language for data-driven web service integration

We take a database-oriented perspective: efficient, declarative evaluation of data-intensive computations

Many interesting database problems

Ongoing work: incremental query evaluation, integration with monitoring system

OptimAX demo in ICDE 2008 [AMZb08] and WebContent demo in VLDB 2008 [AAC+08].

Thank you!

References

AAC+08 S. Abiteboul, T. Allard, P. Chatalic, G. Gardarin, A. Ghitescu, F. Goasdoué, I. Manolescu, B. Nguyen, M. Ouazara, A. Somani, N. Travers, G. Vasile, S. Zoupanos, to appear at VLDB 2008

AMZa08 S. Abiteboul, I. Manolescu, S. Zoupanos: OptimAX: Optimizing distributed AXML applications, ICWE 2008.

AMZb08 S. Abiteboul, I. Manolescu, S. Zoupanos: OptimAX: efficient support for data-intensive mash-ups, ICDE 2008

AMT06 S. Abiteboul, I. Manolescu, E. Taropa: A Framework for Distributed XML Data Management, EDBT 2006

References

AMB05 S. Abiteboul, T. Milo, O. Benjelloun: Regular rewriting of Active XML and unambiguity, PODS 2005

ABC+04 S. Abiteboul, O. Benjelloun, B. Cautis, I. Manolescu, T. Milo, N. Preda: Lazy Query Evaluation for Active XML, SIGMOD 2004

MAA+03 T. Milo, S. Abiteboul, B. Amann, O. Benjelloun, F. Dang Ngoc: Exchanging Intensional XML Data, SIGMOD 2003

ABC+03 S. Abiteboul, A. Bonifati, G. Cobena, I. Manolescu, T. Milo: Dynamic XML documents with distribution and replication, SIGMOD 2003