Efficient Matching in a Context-Aware Event Notification System for Mobile Users

George J. Lee
Computer Science and AI Laboratory
Massachusetts Institute of Technology
Cambridge, MA 02139 USA
gjl@mit.edu

Takefumi Naganuma Shoji Kurakake
Network Laboratories
NTT DoCoMo R&D Center
Yokosuka, Kanagawa 239-8536 JAPAN
{naganuma,kurakake}@netlab.nttdocomo.co.jp

Abstract

As the amount of information accessible to mobile users increases, the need for an effective system to deliver relevant information to users also increases. Event-based systems can help users deal with vast amounts of changing information, but existing approaches do not adequately address the needs of the growing number of mobile users. To address these needs, we built an automatic, personalized, context-aware event notification system that efficiently and flexibly matches users and events while automatically learning individual user preferences for the current context. We developed a concise and expressive multi–content graph event model for describing events and user interests, and a fast and efficient algorithm for matching events according to this model. In this paper we describe our event model and matching algorithm and present evaluation results illustrating the performance of our system.

1. Introduction

The amount of information available to Internet users today is increasing rapidly, but users lack effective tools for obtaining accurate, up-to-date, and relevant information. This problem is especially acute for mobile users, who could greatly benefit from context-specific event notifications but are constrained by the limited input, display, and processing capabilities of mobile devices.

Furthermore, existing event-based systems demand too much user interaction to specify accurate, personalized subscriptions and are not designed for extremely complex and descriptive event models. In addition, the set of events in which a user is interested may depend on their context, such as their location and task. These two problems are compounded for mobile users due to the input and display limitations of mobile handsets and the frequency with which mobile users may change their location and tasks.

To address this problem, we built an automatic, personalized, context-aware event notification system for mobile users. This system automatically delivers personalized information to users based on individual preferences for the current context. Figure 1 illustrates the architecture of this system. An event fetcher retrieves raw event data from event producers, converts them into event descriptions, and sends these event descriptions to a matching engine. A user agent learns user interests, stores user events, and transmits matching events to the client application running on a mobile handset. This user agent also makes context-aware subscriptions based on user interests and context information. The matching engine then matches event descriptions with user interests and delivers events to interested users. For this system to be useful, the event fetcher and user agent must be able to accurately describe a wide range of realistic events and user interests, respectively. Therefore, this system must support an expressive language for specifying event descriptions and user interests.

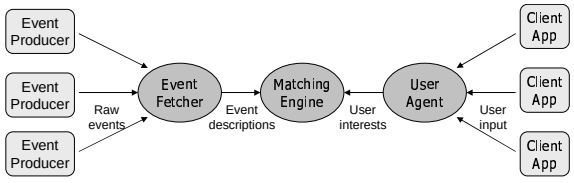

Figure 1. System Architecture

Two major challenges of developing such a system are 1) compactly representing a detailed, expressive event model, and 2) efficiently matching subscriptions and notifications according to such a model. To address these challenges, we developed a compact multi–content graph event model representation and a fast matching algorithm optimized for such event models. This matching algorithm is faster than regular graph search algorithms and allows more concise

subscriptions and notifications than content list approaches.

In section 2 we describe our method for compactly representing expressive graph-structured event models using multiple content graphs. Next we present an adaptive approach for multi–content graph matching in section 3. In section 4, we present an optimized content graph matching algorithm and compare it with other approaches. We describe our system implementation in section 5. Finally, in section 6 we conclude with evaluation results showing that our system can efficiently match new events to many users according to a multi–content graph event model.

2. Event Model Design

In our system, event descriptions (notifications) and user interests (subscriptions) are described according to an event model. This event model determines what information users can use to filter subscriptions and what attributes can be described in notifications. Our system must allow a wide range of realistic events and user interests, and so requires a concise and expressive event model. In order to support such event models, we developed a method for compactly and efficiently representing complex graph-structured event models.

Most event notification systems in use today take one of two approaches to matching: matching based on simple flat or hierarchical topic-based event models that is fast but does not allow expressive subscriptions, and sophisticated content-based matching based on detailed subscriptions but requiring more processing time. In our system we choose to use an intermediate approach based on a graph-structured event model that has more flexibility than simple topic-based matching and is faster than content-based approaches.

In a graph-structured event model, the relationships between different subscriptions are represented as a directed acyclic graph in which the set of nodes in the graph is the set of possible subscription and notification categories [8]. This graph is called a content graph. An edge (u, v) in this graph means that v is a subcategory of u. A subscription to a category s covers a notification in category t if and only if there is a path from s to t in this graph. Figure 2 shows an example of such a graph-structured event model.

Abstractly, a content graph can compactly represent the relationship between any two categories in the graph by joining them with an edge, but the size of this graph can be prohibitively large for event models that allow subscriptions composed of multiple parts, such as a subscription based on both a topic and a location. If there are x topics and y locations, then there are $x \times y$ combinations of topic and location to which users can subscribe. For such complex event models, this graph may contain billions of nodes, or more. This combinatorial explosion means that it is infeasible to represent the entire event model in a single data structure such as an adjacency list or adjacency matrix.

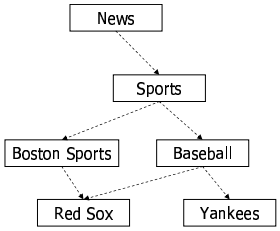

Figure 2. Graph-structured event model

In order to efficiently deal with such complex event models, we developed a more compact representation of this graph. In our system we represent the event model using a vector of content graphs $\mathbf{G} = (G_1, G_2, \ldots G_k)$, where k is the number of content graphs in the event model. We represent a subscription using a vector $\mathbf{s} = (S_1, S_2, \ldots, S_k)$ and a notification as a vector $\mathbf{t} = (T_1, T_2, \ldots, T_k)$, where each S_i and T_i is a set of categories in the content graph G_i. Then a subscription $\mathbf{s}$ matches a notification $\mathbf{t}$ if and only if for all i, there exists a path from a node $s \in S_i$ to a node $t \in T_i$ in G_i. This type of event model allows events and user interests to be efficiently described and matched along multiple dimensions. For example, if the event model consists of two content graphs describing topics and locations, respectively, then a subscription may be something like $(\{\texttt{Baseball}, \texttt{Weather}\}, \{\texttt{MIT}, \texttt{Boston}\})$. Then this user will receive all notifications that have a topic `Baseball` or `Weather` and a location `MIT` or `Boston`.

3. Multi–Content Graph Matching

Matching according to this multi–content graph event model requires more time to match than simpler flat or hierarchical models. In a flat or hierarchical model, the system can store subscriptions in a table indexed by category and efficiently look up all matching subscriptions, but using our event model the system must search each content graph separately to determine whether a notification matches a subscription.

3.1. Notification-oriented Content Graph Search

The simplest matching algorithm for this event model is to find the set of users that match a notification for each content graph and then compute the intersection of these sets. This is a notification-oriented approach because it starts from a notification and searches until it finds all matching users. This approach is constant in the number of subscriptions, but polynomial in the number of notifications.

3.2. Subscription-oriented Content Graph Search

For simple event models a notification-oriented method may work well, but in many situations, a subscription-oriented approach starting from the user may be much faster. For instance, if the notification category is the root of a dense tree and the user subscription category is a leaf node in this tree, searching starting from the subscription may be exponentially faster. This approach is constant in the number of notifications, but polynomial in the number of subscriptions. In a system that supports event history, a subscription-oriented approach may be more suitable for new users or users whose subscription changes due to changing context since only one user needs to be matched to many subscriptions [4].

Which approach is faster depends on the structure of the graph, the number of subscriptions, the number of notifications, and the distribution of subscriptions and notifications. Searching starting from a subscription works especially well for content graphs that resemble a tree when reversed, since searching a tree going from a leaf towards the root only requires $O(h)$ steps where h is the height of the tree; whereas searching from the root to a leaf requires $O(d^h)$ steps, where d is the degree of each node. Subscription-oriented searching requires one search per subscription category, however, so the overall time complexity is $O(h \times |S|)$, where $|S|$ is the number of distinct subscription categories.

Figure 3 illustrates the difference between the two approaches. In this example, there are four user subscriptions s_1, s_2, s_3, and s_4 to two distinct categories, and one notification t_1. The dots represent categories in the content graph, the dashed lines indicate edges in the content graph, and the solid lines indicate the edges traversed in the graph search. The notification-oriented search on the right must traverse six edges in this content graph, while the subscription-oriented search on the left only needs to traverse two edges twice. In more realistic and complex content graphs, the difference may be much greater.

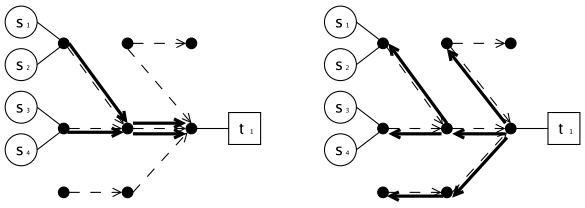

Figure 3. Subscription- vs. Notification-oriented Search

3.3. Adaptive Multi–Content Graph Search

To improve the performance of our system, we developed an adaptive approach that searches each content graph using either a notification-oriented approach or a subscription-oriented approach, depending on the structure of the content graph. The optimal matching plan depends on many factors including the number of subscriptions, the number of notifications, the size of the event model, the distribution of subscriptions and notifications in the system, and the structure of the event model. In our implementation we empirically determined whether to use a subscription- or notification-oriented search for each graph, but a more sophisticated system may choose a search method dynamically for each search depending the factors listed above.

For example, in a two content graph event model, this algorithm may choose to first perform a subscription-oriented search on one content graph and then a notification-oriented search on the other graph. In this case, the algorithm first finds all notifications that match a subscription based on the subscription-oriented content graph, and then eliminates non-matching notifications using the notification-oriented search. In general other matching plans are possible as well. For example, the system could start with notification-oriented searches first, or it could alternate between subscription- and notification-oriented approaches in some order. If the system performs more restrictive searches first, it can drastically reduce the number of searches required. For a complex event model, this method may require polynomial time in the number of users, but may be much faster than a constant time algorithm that searches the entire content graph.

4. Content Graph Matching

Adaptive content graph searching can greatly improve matching time over pure notification-oriented or subscription-oriented approaches, but if the content graphs are complex, searching these individual graphs can still be costly. There are several ways to perform these searches. We describe two existing approaches, unoptimized graph search and content lists, and an optimized matching approach we developed that is faster than an unoptimized graph search and allows more concise subscriptions and notifications than content lists.

4.1. Unoptimized Content Graph Search

A content graph search is a simple depth-first or breadth-first search of a content graph starting from a given node. If the content graph is represented using an adjacency list, this requires $O(|V|)$ time, where $|V|$ is the number of matching categories.

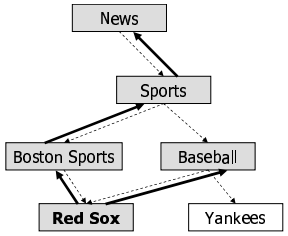

Figure 4. Unoptimized content graph search

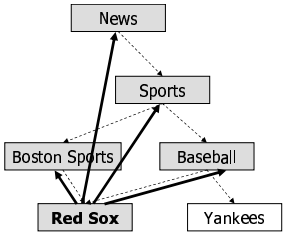

Figure 5. Optimized content graph matching

Figure 4 shows an example of matching a notification with topic *Red Sox* using unoptimized content graph search, where the shaded boxes are subscription topics that should match the notification topic *Red Sox*. Using this method, the system recursively searches the content graph to find matching topics. Searching the entire graph can be very costly if the event model is complex, however.

4.2. Content List Matching

The content list method described by Kulik greatly improves matching speed by including lists of matching categories in notifications and subscriptions [8]. Using content lists, a notification includes a list of all matching categories for notification-oriented content graphs, and a subscription includes a list of all matching categories for subscription-oriented content graphs. Matching is done by looking up subscriptions that match one of the notification topics and that have a location that matches the notification coverage. This approach is much faster than a content graph search, but in a complex event model it may be impractical to include all matching categories in a subscription or notification.

4.3. Optimized Content Graph Matching

We developed an optimized algorithm to perform matching more efficiently than an unoptimized content graph search but requiring less space to represent subscriptions and notifications than content list matching. This optimized matching approach precomputes the categories that match each possible notification. As mentioned in section 2, determining whether a subscription to category s matches a notification category t is equivalent to the question of whether a path exists from s to t in the content graph. In order to improve matching performance, the transitive closure G^* of a content graph G can be computed so that a path from s to t exists in G iff $(s,t) \in G^*$. This makes it possible to find all matches with just a single table lookup in $O(1)$ time instead of the $O(|V|)$ time required for the unoptimized approach. Computing this transitive closure can be costly, but if the content graph is static it only needs to be done once. Using this approach, even extremely complex content graphs, such as the ODP structure containing over 600,000 nodes and over 1,200,000 edges can be used for matching.

Figure 5 shows an example of matching a notification with topic *Red Sox* using optimized content graph matching. Using optimized matching, the system can retrieve all matching subscription topics with a single lookup.

5. System Implementation

We developed a prototype context-aware event notification system for mobile that uses the event model and matching algorithms described above. The event fetcher, user agent, and matching engine were implemented in Java on Intel Pentium 4 systems running the GNU/Linux operating system. The matching engine stores and retrieves events and subscriptions in a MySQL database. We developed a client application for NTT DoCoMo mobile handsets that displays events and transmits user input to the user agent. This system represents a user's context as their task and location and automatically changes their subscription when this context changes.

In our system our event model consists of two content graphs, a topic graph based on the The Open Directory Project (ODP), and a location graph based on the Getty Thesaurus of Geographic Names (TGN) [12, 15]. ODP classifies web sites by topic and the TGN provides an extensive database of relationships between geographic locations. The ODP structure is a directed acyclic graph, while the TGN structure is a tree. The detailed structure of these models allows complex subscriptions to be expressed very intuitively and compactly. In addition, the XML-based RSS, RDF, and DC standards support *topic* and *coverage* attributes in events, making it easy to adapt existing RSS feeds to use this event model.

In our event model, a notification matches a subscription if and only if the notification contains a subtopic of a subscribed topic and if the notification has a location that is a superset of the user's location. This allows users to subscribe to many subtopics by subscribing to a high level category, while receiving all events that pertain to their location or any superset of their location. To more efficiently search

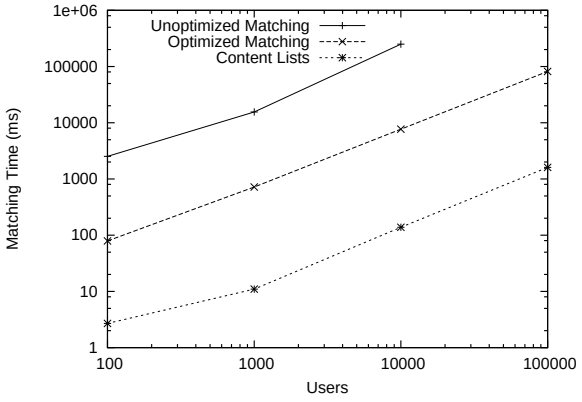

Figure 6. User scalability

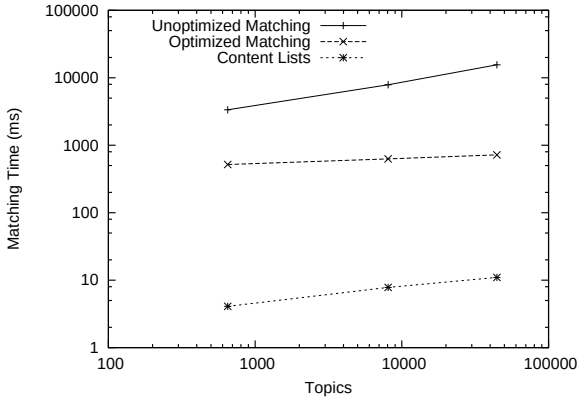

Figure 7. Event Model scalability

this event model, our system uses a notification-oriented approach for the topic graph and a subscription-oriented approach for the location graph.

6. Evaluation

To evaluate the ability of our system to support a complex multi–content graph event model, we consider several event models that differ in complexity. We evaluated our system using subgraphs of the ODP graph containing between 4,000 and 45,000 nodes. These contained all topics within 4, 5, and 6 links, respectively, from the ODP root topic, including symbolic links. For the location graph, we used a subset of the TGN database, consisting of 6,905 locations.

To evaluate the ability of this system to scale to a large number of users, we evaluated this system with 100 to 100,000 simulated users. Each user randomly subscribed to 5 topics and 3 locations. In our evaluation we measured the amount of time to match a random event to these subscriptions. We averaged the matching time of 10 random events, each with 3 topics and 2 locations.

Figure 6 shows the amount of time it takes to match a single random event given some number of users in the system and 44,506 topics. Optimized matching using transitive closure is 20 to 30 times faster than unoptimized matching. The simulation for the unoptimized content graph approach for 100,000 users requires too much time to perform and so is not plotted here. Unoptimized content graph matching takes about 1000 times longer than content list matching due to the complexity of searching the topic graph.

Content list matching is about 30 times faster than the optimized matching approach, but this speed comes at a cost in disk space and subscription and notification size. On average, the number of locations in a subscription increases by a factor of 4.0 and the number of notification topics increases 75 times in the medium size event model. This means that content list matching substantially increases the time required to parse and store new subscriptions and notifications, which was not measured in these simulations.

Figure 7 shows the relationship between matching time and the number of topics in the event model. As the number of topics increases, matching time also increases. Matching time for unoptimized content graph matching and content list matching increase fairly rapidly with the number of topics, while the time required for the optimized content graph approach remains relatively constant, showing that our optimized approach scales well to very complex event models.

These results show that optimized matching is much faster than an unoptimized approach while enabling much more compact representation of subscriptions and notifications than a content list approach. Each algorithm has its tradeoffs, however. Optimized matching is much faster than unoptimized matching but requires that the transitive closure be precomputed and stored. Unoptimized matching is much slower than other matching methods, but does not require computing transitive closure or storing content lists. The best approach to use depends on the complexity of the event model, the importance of matching speed, and the amount of memory and storage available.

Note that although all the algorithms we evaluated are polynomial time in the number of users, a constant time algorithm that searches the full event model to match users would require far more time than these polynomial matching algorithms. Also, though every effort was made to implement these matching algorithms efficiently, further tuning and optimization of this system may result in improved performance. We do not expect this to significantly change the relative performance or time complexity of the algorithms, however.

7. Related Work

There is a large amount of research in the design of event notification systems. Filho, *et al.* give a detailed description of the design of such systems [5]. There has been research in using event notification in context-aware systems and how to notify users in a context-aware manner [10, 3]. Huang provides a good overview of the network architecture design challenges of publish/subscribe in mobile environments [7]. Cilia designed a system for matching past events in a distributed event notification system for mobile users [4].

There has also been a great deal of research in efficient matching algorithms for distributed event notification [13]. The NaradaBrokering system efficiently implements several matching methods, including XML and SQL based matching, but does not consider event models with complex graph structures. Kulik describes some techniques for efficient matching for such complex event models, but does not focus on how to optimize content graph searches [8]. There has also been work on distributed matching [13, 2].

The techniques we use in our optimized matching algorithms are related to methods for database query plan optimization. Graefe provides a good overview of query optimization [6]. Avnur developed a method for dynamically reordering a query plan [1].

Our event notification system incorporates context awareness to provide more relevant information to users. Schmidt discusses various types of information that may be considered part of a user's context [14]. Lieberman gives an overview of context-aware systems [9]. Naganuma models a mobile user's context using a task ontology [11].

8. Conclusion and Future Work

In this paper, we presented an compact, expressive multi–content graph event model and a fast algorithm for efficient and scalable matching according to this event model. We showed how we used this event model and matching algorithm to develop a context-aware event notification system for mobile users. We presented an adaptive method for searching multiple content graphs and an optimized matching approach for individual content graphs. Our evaluation results suggest that our optimized content graph approach greatly improves matching performance over an unoptimized content graph approach for complex event models while allowing more compact subscriptions and notifications than content list approaches. Areas of future work include further matching optimizations, designing a distributed event notification system based on this framework, and more performance evaluations.

References

[1] R. Avnur and J. M. Hellerstein. Eddies: continuously adaptive query processing. In *SIGMOD '00: Proceedings of the 2000 ACM SIGMOD international conference on Management of data*, pages 261–272. ACM Press, 2000.

[2] A. Carzaniga, D. S. Rosenblum, and A. L. Wolf. Design and evaluation of a wide-area event notification service. *ACM Transactions on Computer Systems*, 19(3):332–383, 2001.

[3] P. Chou, M. Gruteser, J. Lai, A. Levas, S. McFaddin, C. Pinhanez, and M. Viveros. Bluespace: Creating a personalized and context-aware workspace. Technical Report RC22281, IBM Research Division, Thomas J. Watson Research Center, Dec. 2001.

[4] M. Cilia, L. Fiege, C. Haul, A. Zeidler, and A. P. Buchmann. Looking into the past: enhancing mobile publish/subscribe middleware. In *Proceedings of the 2nd international workshop on Distributed event-based systems*, pages 1–8. ACM Press, 2003.

[5] R. S. S. Filho, C. R. B. de Souza, and D. F. Redmiles. The design of a configurable, extensible and dynamic notification service. In *Proceedings of the 2nd international workshop on Distributed event-based systems*, pages 1–8. ACM Press, 2003.

[6] G. Graefe. Query evaluation techniques for large databases. *ACM Computing Surveys*, 25(2):73–170, June 1993.

[7] Y. Huang and H. Garcia-Molina. Publish/subscribe in a mobile enviroment. In *Proceedings of the 2nd ACM international workshop on Data engineering for wireless and mobile access*, pages 27–34. ACM Press, 2001.

[8] J. Kulik. Fast and flexible forwarding for internet subscription systems. In *Proceedings of the 2nd international workshop on Distributed event-based systems*, pages 1–8. ACM Press, 2003.

[9] H. Lieberman and T. Selker. Out of context: Computer systems that adapt to, and learn from, context. *IBM Systems Journal*, 39(3&4):617–631, 2000.

[10] S. Mitchell, M. D. Spiteri, J. Bates, and G. Coulouris. Context-aware multimedia computing in the intelligent hospital. In *Proceedings of the 9th workshop on ACM SIGOPS European workshop*, pages 13–18. ACM Press, 2000.

[11] T. Naganuma. A task oriented approach to service retrieval in mobile computing environment. In *Proceedings of IASTED International Conference on Artificial Intelligence and Applications (AIA 2005)*, 2005.

[12] Netscape Communications. About the Open Directory Project. `http://dmoz.org/about.html`.

[13] S. Pallickara and G. Fox. On the matching of events in distributed brokering systems. In *Proceedings of IEEE ITCC Conference on Information Technology*, volume II, pages 68–76, April 2004.

[14] A. Schmidt, M. Beigl, and H.-W. Gellersen. There is more to context than location: Environment sensing technologies for adaptive mobile user interfaces. In *Proceedings of the Intl. Workshop on Interactive Applications of Mobile Computing (IMC98)*, volume 23, pages 893–901, November 1998.

[15] The J. Paul Getty Trust. About the TGN. `http://www.getty.edu/research/conducting_research/vocabularies/tgn/about.html`.