Resolving Queries in a Heterogeneous Context Rich Environment

Ruaidhrí Power, Declan O'Sullivan, Owen Conlan,
David Lewis, Vincent Wade
E-mail: Firstname.Surname@cs.tcd.ie
Knowledge and Data Engineering Group,
Department of Computer Science,
Trinity College Dublin,
Ireland.

1 Introduction

The vision of ubiquitous computing[1] is that of many computing devices interacting in a natural way with humans in the real world. The software running on these devices is currently limited in how it can provide for users' needs by the amount and quality of the information it can retrieve about the environment in which it is operating. This context information must be provided in a form each software component can understand, which is a difficult problem given the wide array of heterogeneous information sources involved in any ubiquitous computing scenario such as computers, embedded sensors and information appliances.

One of the driving forces behind the design for a context system proposed in this paper is to minimise the effort required to make a piece of software context aware. These software components will come in many forms, from the e-mail clients and office tools that are prevalent today, to tiny embedded operating systems with minimal processing power, to massive mainframe or cluster computers running large databases. For the purpose of this paper, any of these software components are referred to as applications. While this term may bring to mind today's software which is not context aware, throughout this paper it refers to any software component which wishes to make use of context information. These applications need to have easy access to more information about the environment in which they are operating, rather than being limited to the explicit input or information from hardwired data sources provided to current applications.

In this architecture the 'context service' is the service provided to applications to make context information available to them. One role of a context service is to take queries from a context-aware client and to resolve those queries by acting as a mediator between the client and other information sources that the service has access to. As well as acting as consumers of context information (by executing queries), applications can also act as producers of context information by providing their context service with a description of the information they have available. If an application produces context information, a context service can advertise that information available to it to other context services.

This design uses an ontology-driven approach to bridge the heterogeneity of context information sources in ubiquitous computing systems. Ontologies are a technique for formally representing domain knowledge in an application independent way. Ontologies feature heavily in the Semantic Web initiative[2], which aims to provide ways of defining information so that it can be understood and processed by computers

more easily. Examples of ontology languages are W3C's OWL[1], the Web ontology language and DARPA's DAML[2].

In summary, this paper proposes an ontology-driven context system for heterogeneous context-rich environments that aims to minimise the effort required to make an application context-aware.

The paper is laid out as follows: Section 2 describes the state of the art with regard to integration of heterogeneous context information. Section 3 describes the example context scenario covered in this paper, and section 4 describes the process of designing a context-aware application. Section 5 describes how a context-aware device would operate, describing the interactions between applications and the context service. Section 6 describes the internal structure of a context service, which consists of query interface and analysis (section 6.1), query decomposition (section 6.2) and query routing (section 6.3). Section 7 describes our experimental work to date, and section 8 concludes the paper.

An initial experiment to verify the approach presented in this paper is underway, and is due for completion in September 2004. The full version of this paper will provide the results of this experiment.

2 Comparison with Existing Context Approaches

Existing context-aware computing research projects take a number of approaches to resolving queries in heterogeneous environments. Early context-aware applications such as CyberGuide[3] were individually tailored at design time to the scenario they were to be deployed into, with hard-coded data sources and formats for their context information. This is quite an inflexible approach in the long term, as applications must be redesigned when formats or locations of context information change.

Other approaches such as SIMS[4] use ontological models to integrate heterogeneous information, but frequently require that applications commit to communicating solely in terms of a global ontology. This ontology must be agreed by all the participants, or else they must define at design time a set of mappings from their local ontology to that global ontology. This causes difficulty when one wants to reuse the applications in a different scenario, as they must be redesigned to account for different sources of information.

Ontology-based information integration projects such as KRAFT[5] and OBSERVER[6] aim to integrate heterogeneous information through the use of multiple ontologies. We aim to extend work in this area to the field of ubiquitous computing, and particularly context information.

3 Example Scenario

In this paper the example taken is that of software controlling a college lecture theatre. The theatre administrators want to use the software to obtain a copy of the notes for each lecture and automatically display them on an attached projector without

1 http://www.w3.org/TR/owl-guide/

2 http://www.daml.org/

the lecturer having to intervene to select the notes she wants displayed.

However, the software was actually developed independently of the college and was originally designed to handle business meetings of companies. The mismatch between the two domains must be resolved for the software to be able to operate successfully and avail of context information available in the college environment. A simple example of a mismatch detailed in this scenario is the mismatch between the theatre software's 'meeting' and 'document' concepts and the college context server's 'lecture' and 'notes' concepts.

In this situation, the integration of the software into the college environment is being performed by people other than those who wrote the software. This will be a frequent occurrence in a ubiquitous computing scenario where the large number of users, devices and applications will prohibit any solution that relies on a developer being involved at every deployment of a context-aware application. Instead, the approach described in this paper allows applications to be integrated into a context system by another party. In our simple mismatch example this person can perform the integration by nominating terms that are functionally equivalent for the purposes of this application. In this example, the terms 'meeting' and 'lecture' are equivalent, as are 'document' and 'notes'.

Once this integration has been completed, the lecture theatre software can query the college context service for the lecture notes, and display them on the projector.

4 Design and Integration of Context-aware Applications

In a ubiquitous computing environment, a controlling factor over the deployment of context-aware applications will be the effort required on the part of the application developer to make use of this context information. Consequently making applications context-aware should be made as simple as possible, making use of good software engineering practices such as code reuse and automating repetitive tasks wherever possible.

In our current architecture, detailed in Section 5, an application can be designed as context-aware by defining an ontology that describes the domain of context information that the application is interested in querying, and also that it wants to make available to other applications. This ontology may be written from scratch, or it may be possible to reuse an existing ontology such as CoBrA-ONT[7]. This ontology is registered with the context service as belonging to the application, and is stored in the ontology repository.

The application developer has the option of providing mappings between concepts in the application's ontology and equivalent concepts in other ontologies used within the system. This is however not a requirement, as this step may be done at a later stage. If mappings are provided, they will be stored in the ontology mapping repository.

The greater the number of equivalence mappings provided here the more context information the context service will be able to query. If the application's ontology has been used by other applications, there may already exist mappings that can be accessed by the context service and reused. Otherwise, these mappings will have to be provided by a human, either by hand or through the use of automated tools such as

PROMPT[8] that will help to simplify the process. In the example scenario, the person integrating the lecture theatre software would record the mapping between notes and documents.

As well as providing mappings to concepts within a single domain, mappings can be provided to multiple domains, allowing any applications using those concepts to work across domain boundaries. Sharing and reuse of mappings is the ideal way for interoperability to be achieved across these rich sources of context information. If the application developer chooses not to provide mappings of their own, these can be provided when the software is integrated into the context system.

A key benefit of this approach is that it allows us to make existing applications context-aware by simply modelling the domain of context information that they are interested in as an ontology, reformulating queries to external data sources in terms of that ontology, and directing those queries to their context service instead of those data sources. This will give these applications access to their existing data sources, but also potentially new information that their context service can avail of through communication with other context services in the environment. Treating all data sources as context sources has advantages because the applications don't need to know the precise details needed to access a heterogeneous set of sources, but rather they are insulated from specific terms, formats and protocols by the abstraction of a context service.

5 Architecture of a Context-aware device

The internal architecture of a context-aware device is shown in Figure 1. Each box within the device represents an autonomous piece of software, with their interactions described by arrows. Starting from the bottom of the diagram, applications present queries to the context service. Each of these query messages contains the content of the query **Q**, a reference to the query language used **L**, and a reference to the ontology **O** that the query refers to. This query is taken by the context service which examines the query and ontology used. Combining these with mappings from the ontology mapping repository, the context service can then compose a new query that can be routed to other context services, which will attempt to return a corresponding result.

Any results that are returned are translated back from the ontology of the remote context service into the application's ontology before they are returned as a query response, **R**.

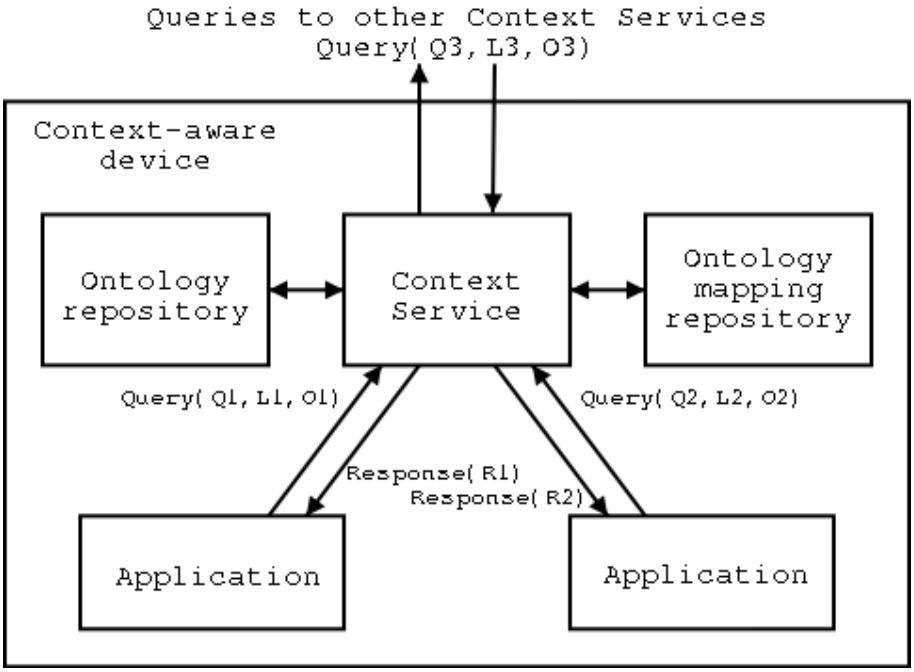

Figure 1: Context-Aware Device Architecture

6 Anatomy of a Context service

The internal structure of a context service is shown in Figure 2. The first major functional section of a context service are its query interface/query analysis modules. The query analysis module also handles query decomposition. The decomposed queries are then passed to the query routing module. We discuss these modules in the following subsections.

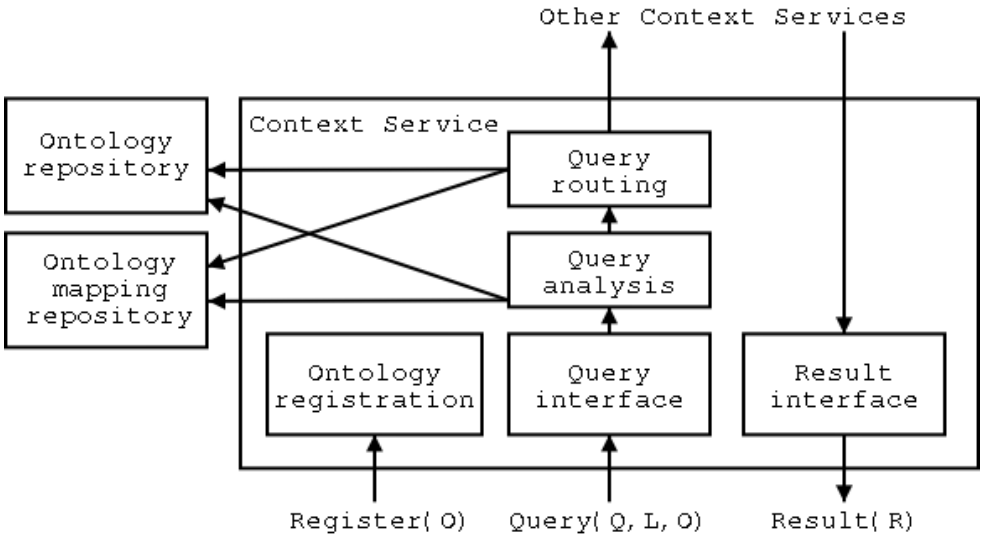

Figure 2: Context Service Internal Design

6.1 Query Interface and Query Analysis

A context service offers a query interface, which will accept a query in a supported query language, such as SQL, XQuery[3], RDQL[4]. The terms involved in the query must be a subset of those in the ontologies understood by the context service, for the service to be able to understand the query. Because each context-aware application has provided its own ontology to the context service, the local context service is guaranteed at a minimum to be able to understand queries posed by its applications.

Associated with each context service there is a repository of ontologies that describe the domains of knowledge that this service has. These ontologies are provided to the service by the applications for which it acts as a source of context information, and can also be discovered through communication with other context services. The context service must take each of these queries and communicate with other context services to resolve them. It does this by sending the query to some number of context services, after translating the query into terms that they will understand.

Considering the example scenario, the lecture theatre software has been developed for use in any meeting scenario and provides in its ontology the terms 'meeting' and 'document'. When the software was installed, the college provided mappings to the context service to say that these terms were equivalent to the terms 'lecture' and 'notes' (stored on a college context server) for their purposes. The application's query to its local context service is phrased in terms of 'meeting' and 'document', allowing the software to remain unchanged when deployed in this scenario, but still communicate with a college context service about which nothing was known when the application was designed. The lecture theatre context service transforms the query for the lecture notes and passes it on to the college context service which then returns the notes in question.

These context services will be potentially quite large in number and can be distributed around the network in whatever way seems most appropriate, most likely based on how devices are managed. For instance, if all applications on a PDA are provided by the same vendor, they may be happy to use a single context service on the PDA. They are not limited to this configuration though, as for example some devices may not have the processing power to host a context service and will rely on a remote service that will provide context to them over a network connection.

6.2 Query decomposition

The field of distributed databases provides insight into how queries across distributed information sources can be decomposed and optimized. This research has addressed the process of taking a query that references data across multiple databases, and breaking down that query into a set of sub-operations that can be distributed to the individual databases for processing. The order and location of these sub-operations form a plan of execution[9]. Plans can be formulated before queries take place, or can be composed on the fly and therefore tailored for the query at hand at the expense of some computation.

When evaluating potential plans and breaking them down into sub-plans for the

3 http://www.w3.org/XML/Query

4 http://www.w3.org/Submission/RDQL/

resolution of a query, a cost factor must be associated with each (sub-) element of the plan. The classic approach is as described in [10]. These cost factors will include computational cost, communication cost, storage cost, etc. Ideally, inefficient plans will be eliminated as early as possible to avoid potential overall plans being investigated even though they are based on inefficient sub-plans.

One approach to query optimization is to break down a stored query into its sub-parts and precompute the join order, join methods and access paths before the query is executed. This approach can save time, but also produces sub-optimal plans for particular queries, for example when the locality of different pieces of data isn't taken into account. Two-step query optimization is described in [11].

Distributed databases do however make use of a global ontology that describes the structure of the database, which means that these approaches need to be modified to route context queries in a heterogeneous environment.

6.3 Query routing

Once the client application's query has been received and understood, the context service must decide where these queries should be routed to. In the scenario presented above, the context service knew of one location where it could find information to answer its query. However, there will be many situations in which the set of context services which will be able to answer the query is not known.

The context service can decide where to send the queries based on a routing algorithm. In the simplest case, this is just to route the query to one other service that will understand the query (has equivalent terms in its local ontologies) and is therefore able to return a meaningful reply. This query can be understood by any other context services that have equivalent terms in their own local ontologies. Because the lecture theatre's context service knows that the college context service has these equivalent terms, it translates the original query into the new terms and forwards it on. When it receives a reply to this query, it can then translate it back into the original terminology and pass it on to the application.

Alternatively, this query could be multicast or sent over a content-based network to a number of services that will be able to answer the query. These approaches have different advantages: queries that are only sent to a small number of services are efficient in terms of bandwidth and processing costs in routing of queries, but queries sent to a large number of services will have the greatest chance of being answered satisfactorily.

A problem arises however when a large number of context services can understand a query. Obviously they should not all be sent the query (as a broadcast), as this would be wasteful of bandwidth. The context service must decide how to route these queries to achieve the optimal effect. The most simple methods for query routing (unicast and broadcast querying) will not scale with the explosion in the numbers of devices heralded by ubiquitous computing. Queries posed by applications should be routed to those services that can answer them, but should not be spread any further than is necessary.

Research in content-based networking has produced an enhancement to traditional

publish/subscribe mechanisms that may prove useful in the design of a context system. In a content-based network, rather than messages being given an explicit destination address, they are routed to a set of hosts based on predicates applied to their content. Hosts are assigned `addresses' in the content-based network based on a function which describes the type of messages they are interested in. Content-based routers can then decide which of their neighbouring routers to send the message to, based on a routing table composed from these predicates. For the purposes of a context system, the messages are the context queries and the message content is the set of terms involved in each query.

A good introduction to content-based networking can be found in [12], which describes a content-based network operating as an overlay network above existing network technologies such as IP. Comparisons are made between content-based networking and current (though limited) approaches to this information routing such as IP multicast in [13]. This document also evaluates approaches that minimize the time taken to make a routing decision and produces concrete performance evaluations of these forwarding algorithms, which show promising results.

Chan et. al. describe in [14] a method of content-based routing that is specifically tailored to XML documents. Consideration is again given to the space and efficiency characteristics of such a solution. Here, XPath is used as the XML pattern specification language. XML seems an appropriate choice as an information interchange markup, and XPath an effective way of querying those XML documents and specifying patterns of interest.

7 Experimental work

Experimental work in progress focuses on implementing the architecture outlined in this paper, and applying it to a student project meeting scenario. Particular technologies are under evaluation for their suitability for use in this architecture.

- OWL is under evaluation as the ontology language.
- XQuery is being used as a query language.
- Content-based networking is being investigated as a routing mechanism for context queries.

While these technologies are under evaluation, it is not foreseen that one set of technologies will be prescribed for use in this architecture. However, this work will characterise desirable features in each, in order that they can be more easily be identified as new technologies arise.

Another strand of work is addressing the fact that responses to context queries may produce a definite answer, or they may return that the answer to the question is unknown. This will require application programmers to be aware that context-aware applications may operate in an environment where not enough information is available to answer their queries, and adjust their behaviour accordingly.

An initial experiment to verify the approach presented in this paper is underway, and is due for completion in September 2004. The full version of this paper will provide the results of this experiment.

8 Conclusion

We see a number of advantages to the approach outlined in this paper. These advantages are as follows:

- Applications can be designed independently of the environment in which they are finally run, because of the encapsulation of their domain knowledge in an ontology. This is the minimum amount of work that an application programmer will have to do to enable his application to be context-aware.
- Integration into a context system is done through mappings between equivalent terms. These mappings can be set up and also changed at a later stage, all independently of the original application programmer. These mappings will either have to be provided by the application programmer, or (we believe more likely) they will be provided by the people integrating the software in the environment.
- Mappings between equivalent terms can be stored and reused in future. In the simple example above, the equivalence between the lecture theatre's term 'meeting' and the college's term 'lecture' can be stored and deployed automatically in future, saving effort.
- A system that uses content-based networking for the routing of queries is much more powerful and efficient than one that uses unicast queries or a more simple publish/subscribe architecture.

The use of this architecture will allow context-aware applications to make the most use of the rich sources of context information available to them in future ubiquitous computing environments.

References

[1] M. Weiser, "Some computer science problems in ubiquitous computing," Communications of the ACM, July 1993.

[2] T. Berners-Lee, J. Hendler, and O. Lassila, "The semantic web." Scientific American, May 2001.

[3] S. Long, R. Kooper, G. D. Abowd, and C. G. Atkeson, "Rapid prototyping of mobile context-aware applications: the cyberguide case study," in Proceedings of the Second Annual International Conference on Mobile Computing and Networking, (White Plains, NY), pp. 97--107, ACM Press, November 1996.

[4] Y. Arens, C. A. Knoblock, and C.-N. Hsu, "Query processing in the SIMS information mediator," Readings in Agents, 1998.

[5] P. R. S. Visser, M. D. Beer, T. J. M. Bench-Capon, B. M. Diaz, and M. J. R. Shave, "Resolving ontological heterogeneity in the KRAFT project," in Database and Expert Systems Applications, pp. 668--677, 1999.

[6] E. Mena, V. Kashyap, A. Sheth, and A. Illarramendi, "OBSERVER: An approach for query processing in global information systems based on interoperation across pre-existing ontologies," in Proc. of the First IFCIS International Conference on Cooperative Information Systems (CoopIS'96), pp. 14--25, IEEE Computer Society Press, June 1996.

[7] H. Chen, T. Finin, and A. Joshi, "An ontology for context-aware pervasive computing environments," Special Issue on Ontologies for Distributed Systems, Knowledge Engineering Review, 2003.

[8] N. F. Noy and M. A. Musen, "The PROMPT suite: Interactive tools for ontology merging and mapping," International Journal of Human-Computer Studies, vol. 59, no. 6, pp. 983--1024, 2003.

[9] A. Ip, W. Rahayu, and S. Singh, "Query optimization in a non-uniform bandwidth distributed database system," in Proceedings of the Fourth International Conference on High-Performance Computing in the Asia-Pacific Region, vol. 2, (Beijing, China), pp. 818--823, May 2000.

[10] L. Mackert and G. Lohman, "R* optimizer validation and performance evaluation for distributed queries," in Proceedings of the Conference on Very Large Databases (VLDB), Kyoto, Japan, 1988.

[11] Z. Nie and S. Kambhampati, "Joint optimization of cost and coverage of query plans in data integration," in Proceedings of the tenth international conference on Information and knowledge management, (Atlanta, Georgia, USA), pp. 223--230, ACM Press, 2001.

[12] A. Carzaniga and A. L. Wolf, "Content-based networking: A new communication infrastructure." NSF Workshop on an Infrastructure for Mobile and Wireless Systems, 2001.

[13] A. Carzaniga and A. L. Wolf, "Forwarding in a content-based network," in Proceedings of ACM SIGCOMM 2003, (Karlsruhe, Germany), pp. 163--174, Aug 2003.

[14] C.-Y. Chan, W. Fan, P. Felber, M. Garofalakis, and R. Rastogi, "Tree Pattern Aggregation for Scalable XML Data Dissemination." Bell Labs Technical Memorandum, February 2002.