A new approach to distributed context-aware computing

Paul Swoboda John Plaice
School of Computer Science and Engineering
The University of New South Wales
UNSW SYDNEY NSW 2052, Australia
{pswoboda,plaice}@cse.unsw.edu.au

2004

Abstract

We present the æther, a tree-structured, pervasive and active distributed context that is directly useable as an efficient medium for intensional computing. An æther allows participants to register over a network at arbitrary nodes in the context and listen to operations which selectively modify the context. Using a distributed model similar to an IRC channel, context-sensitive software components can engage in contextual discussion, affecting their combined runtime semantics. Discussion is given on the nature of the context, as well as the means of transmission and potential applications.

1 Introduction

This paper presents a new infrastructure for distributed context-aware computing, using an active context called an *æther* [1]. The æther is a reactive machine which receives and transmits *context operations*, analagous to the deltas of software configuration. This model of distributed context allows the direct control of the runtime semantics of networked software components with an unprecedented level of structure and efficiency, by using the context directly as a medium for pervasive runtime configuration management.

The infrastructure that we have created follows the tradition of *intensional programming* (§2), in which programs are understood to adapt their behaviour according to a pervasive multidimensional context. We significantly extend this paradigm by transforming the context into a first class entity which can be transmitted or manipulated by any context aware program. For reasons both of fine-grained control and efficiency, most transmissions will consist of context operations — context modifiers — rather than full contexts (§3). To support distributed computing, we use the æther (§4), along with the æther protocol (§5), to ensure registered participants all receive timely updates of their local views of the global context.

We have written extensive software libraries to build and manipulate contexts and context operations, allowing software components to share and react to changes in localised context in a structured manner. We have also developed the necessary software for the æther protocol, allowing networked participants to share context efficiently across differing languages (Java, C++, Perl) and architectures. We conclude by discussing the use of this code in existing and future applications (§6–7).

2 Intensional Programming

Intensional programming [2] is an approach to computing that supposes that there is a multidimensional context, and that programs are capable of adapting themselves to this context. The semantics of an intensional program is an *intension*, in the logical sense, i.e., a mapping from contexts to *extensions* (ordinary programs). In practice, an intensional program can be built either

by direct access to and manipulation of the context or parts thereof, dimension by dimension, or through the definition of multiple *versions* of software components (objects, procedures, functions, types, etc.). As the context changes, a runtime system must ensure that the *best-fit* version of each of these components, with respect to the new context, will be selected as needed.

Such intensional programs can be implemented directly using intensional languages, or using adaptive libraries such as libintense, presented below, that provide access to the pervasive context through native constructs, as in C++ templates. The intense project was born of a need to aggregate and distribute the intensional context of otherwise separate context-aware entities and processes, so that they could change in cooperation with one another. The resulting system can be used either as an active (driving participant execution) or passive (polled) medium of contextual exchange, with optional use of intensionality.

3 Context and Context Operations

The context used in intense is a tree-structured dictionary of arbitrary depth and arity, where each node has a label and an optional *base value*, which in practice can be any software object, often just a scalar. The labels are termed *dimensions*, and a path to a given node is simply the list of dimensions from the context root to the node, or a *compound dimension*. The canonical string representation of a context is formed by enclosing a node in `< >` delimiters, optionally preceded by a dimension. For example, the context `<reactor:<temp:1400+state:"warn">>` has populated dimensions `reactor:temp` and `reactor:state`.

Similarly structured *context operations* are used to manipulate contexts, with modification and clearing of base values, as well as pruning of entire subcontexts at any level. An example context operation is `[user:[u1:[level:[14]]+u2:[---]]]`, which has the effect of removing all context for the dimension `user:u2` and of adding or modifying the base value of dimension `user:u1:level`. Context operations are associative, so that, for example, the application of thousands of successive operations in a single stream can be handled by efficiently collapsing multiple operations to a single effective operation, prior to its application to the target context. Figure 1 shows such an application.

If we view the context as a complete record of state for pervasive configuration management, then context operations correspond directly to the deltas of revision control systems. The clear difference here is that we have a state allowing the *runtime* version control of complex, distributed software systems.

Using intense, a version of an entity is simply a particular instance of that entity, along with a context tag, called an *extensional context*. The aggregation of multiple versions of the same entity is an intension. The pervasive runtime context is called an *intensional context*. The process of mapping an intensional context and an intensional object to a concrete extension is performed by using a partial order on contexts known as *refinement*, introduced by Plaice and Wadge [3].

4 Æthers: Active Contexts with Participants

An *æther* is an active machine containing a context. At any node in this context, processes may register *participants*, bodies of code to be executed should the associated subcontext in the æther be changed. Processes with registered participants send context operations to the æther, which applies these to its own context. When an operation is applied to an æther, participants at or above affected nodes are notified of the operation, relative to their respective nodes. For example, if the context operation above, `[user:[u1:[level:[14]]+u2:[---]]]`, were applied to an æther with a participant at the dimension `user:u1`, the participant would see only the operation `[level:[14]]`.

When using æthers for intensional progamming, participants serve the dual purpose of both modifying an entity's local intensional context, if any, and resolving best fits on or within the entity, as changes in the context occur.

Figure 1: Associative Application of Multiple Operations to a Context

5 AEP: A Protocol for Context Distribution

To allow remote collaboration through an æther, we have developed a service for both the remote registration of participants and the delivery of contexts and operations. This service is encapsulated in the *Æther Protocol* (AEP) and is implemented using a centralised and clusterable AEP server (aepd, for *æther protocol daemon*). Client libraries exist in several languages for use by remote participants, including C++, Java, and Perl.

The initial version of AEP/aepd allows clients to be in a number of states once connected to the server, including "listening" and "ignoring" states, and allows clients to move around between nodes in the context. The primary task of the æther server is to collect and serialize context operations from all connected participants into a single stream, while keeping track of the locations and states of participants; the aggregate operation stream is then applied to the server's æther. Unless an operation is specifically designated by a participant as an *associative fence*, the default behaviour of the æther server is to collapse multiple operations prior to applying the result to the internal æther and rebroadcasting the result to the appropriate listeners. This has required a highly threaded implementation with extensive queue management and internal synchronisation; the resulting server has nonetheless proven to be extremely efficient in handling large volumes of traffic with a significant number of clients.

The AEP protocol is itself abstract, specifying only states and abstract message types; the first implementation of AEP, AETP, is a textual protocol designed to be as lightweight and efficient as possible, while retaining the potential for human readability. The line-oriented nature of AETP is derived from similar protocols, such as SMTP and HTTP, and uses protocol-configurable serialisation

modes to perform transmission of contexts and context operations, similar to an HTTP `POST`, with serialised content. The first version of AETP allows pure binary (native), XDR (an architecture-independent big-endian format from Sun RPC), and textual (canonical-form ASCII) serialisation. It is envisioned that subsequent implementations of AEP could use existing middleware infrastructure such as CORBA for purposes of transport and context encapsulation. Figure 2 shows the overall distributed architecture of an æther server.

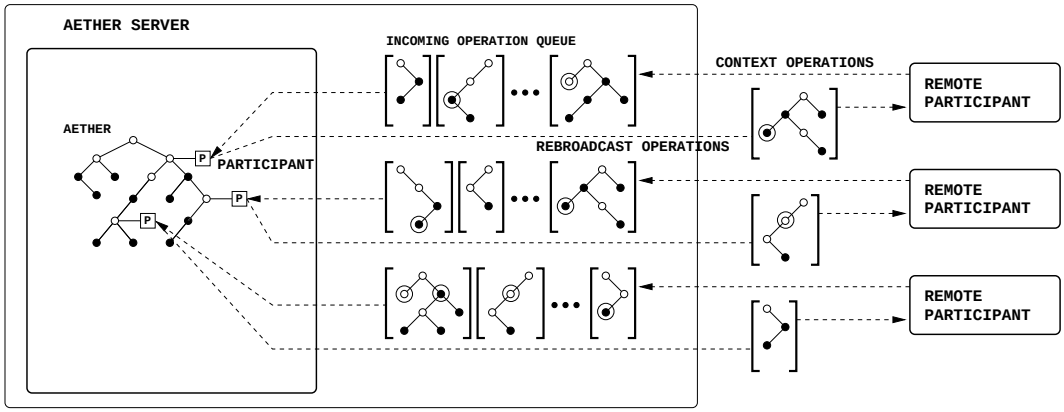

Figure 2: Æther Protocol Architecture

Two or more AEP servers can be bridged, and their respective æthers merged, with the use of special *bridge participants*. Effectively, this means that AEP servers can be clustered in a manner similar to Internet Relay Chat (IRC) servers, where a particular subcontext in a bridged æther is analagous to an IRC channel. This distribution model has the advantage that entire contextual communities can be linked, without any one community or community member being dependent on contact with another community. If a link between merged æthers is temporarily severed, they can function independently.

6 Applications

The infrastructure that we have described above allows the æther to effectively "reach into" the applications which have registered participants with it. To make it easier for each of the applications to adapt appropriately when its own local context changes, we have developed a number of mechanisms for context awareness [1]. These include versioned data structures and versioned flow control constructs. However, the real strength of our approach lies in the ability to quickly adapt the intense software to new situations, using standard OO techniques such as subclassing.

The potential applications for our system are at least as extensive as those for any other context distribution mechanism or directory service such as LDAP. With the added benefits of an active context, as well as the direct use of the context for purposes of intensional programming, we can now create distributed systems that react to shared context, from the level of fine-grained software components, to the level of global system properties.

6.1 Desktop Systems

The intense framework lends itself directly to fine-grained, active control of whole desktop environments, such as those provided by the GNOME and KDE projects. Distributed runtime versioning of aspects of UI, such as desktop themes, fonts and user access control, as well as active updates of desktop and application content, can all be performed through participation in a networked

æther. An example application is the simultaneous control of multiple desktop environments in one or more laboratories, a classroom, or a corporate-area network. While current technology allows the themes of some desktop environments to be altered on a per-machine basis, typically by causing desktop software to re-read configuration files, the intense infrastructure allows us to develop applications whose UI *and* internal behaviour both make use of global context.

6.2 Web Development

Web developers are already familiar with directory services such as LDAP, usually for purposes of authentication and limited storage of user session data. With an active, distributed context, we can use the intense framework directly with existing dynamic content technology, most notably JSP and modperl-based server pages, to create sites whose behaviour and content are versioned synchronously, either with other such sites, or with external entities such as databases, applications, or entire systems.

Intensional programming has been proven to be well suited to Web development, with tools such as IHTML [4]. Initial experiments using intensional JSP pages have demonstrated the feasibility of linked contextual browsing using participant applets, as well as æther-based session maintenance.

6.3 Wireless Systems

An obvious application of AEP is demonstrated in the emerging popularity of contextual communications between mobile devices, typically using Bluetooth, and fixed services such as Internet access in a cafe. Existing systems that provide "service discovery" layers based on *tags* (akin to dimensions in an æther) do not make direct use of the context itself to govern computation; rather, the client is left to analyze the tags provided and merely uses the content as a sort of dictionary.

With wireless æthers, provided as a service unto themselves, mobile clients can be coded to modify their behaviours synchronously with dynamic context available in their environments. For example, a mobile PDA user in a restaurant could use a wireless æther to view a menu, or procure Internet access, with individual components of the PDA client software being responsive to, and interactive with, the æther of the restaurant.

6.4 Remote Pervasive Configuration

Perhaps the most interesting application our framework is in the networked and collaborative configuration of remote systems. Networked æthers are directly useable as an infrastructure for control systems, both as a medium of contextual communication, and as a medium of runtime configuration; the reach of this technology is limited only by connectivity and security constraints.

Using æthers, both the systems control application and the remote service that it controls can be configured simultaneously. A control GUI for a remote system such as a power plant, for example, can make use of an æther to configure both the system and the GUI simultaneously. Further, multiple GUIs controlling the same system can interact with one another through the same æther.

7 Conclusions and Work in Progress

The central innovation of the intense project lies in its use of context operations to effect contextual communication. This innovation has allowed us to develop a complete infrastructure for the manipulation, distribution, and use of a pervasive intensional context. The infrastructure is provided in the form of a robust body of code, available in several popular systems development languages.

While the intense libraries are currently both stable and useable, there are a number of additions that will be made, primarily in the area of security. Similar to the first version of HTTP, the existing intense infrastructure is suitable for open, trusted communication. Subsequent versions of AEP

will incorporate technologies such as openssl to secure participant connectivity. Administrative features will also be provided, such as participant ownership of æther dimensions, as well as a permissions scheme like that of a file system.

We envisage that a distributed pervasive intensional context will change the way systems interact, and will continue to further these efforts.

References

[1] P. Swoboda. *A Formalisation and Implementation of Distributed Intensional Programming.* Ph.D. Thesis, The University of New South Wales, Sydney, Australia, 2003.

[2] A. A. Faustini and W. Wadge. Intensional Programming. In *The Role of Languages in Problem Solving 2*, Elsevier, 1987.

[3] J. Plaice and W. W. Wadge. A New Approach to Version Control. In *IEEE Transactions on Software Engineering* 19(3): 268–276, March 1993.

[4] W. W. Wadge, G. Brown, M. M. C. Schraefel, T. Yildirim. Intensional HTML. In *Principles of Digital Document Processing, LNCS* 1481: 128–139, 1998