

Templates and Queries in Contextual Hypermedia

Kenneth M. Anderson^{*}
Dept. of Computer Science
University of Aarhus, Denmark
kena@daimi.au.dk

Frank Allan Hansen
Dept. of Computer Science
University of Aarhus, Denmark
fah@daimi.au.dk

Niels Olof Bouvin
Dept. of Computer Science
University of Aarhus, Denmark
bouvin@daimi.au.dk

ABSTRACT

This paper presents a new definition of context for context-aware computing based on a model that relies on dynamic queries over structured objects. This new model enables developers to flexibly specify the relationship between context and context data for their context-aware applications. We discuss a framework, HYCONSC, that implements this model and describe how it can be used to build new contextual hypermedia systems. Our framework aids the developer in the iterative development of contextual queries (via a dynamic query browser) and offers support for context matching, a key feature of contextual hypermedia. We have tested the framework with data and sensors taken from the HyCon contextual hypermedia system and are now migrating HyCon to this new framework.

Categories and Subject Descriptors

H.5.4 [Information Systems]: INFORMATION INTERFACES AND PRESENTATION—*Hypertext/Hypermedia*

General Terms

Design, Experimentation, Standardization

Keywords

Structural computing, context-aware systems, search, templates, user interfaces

1. INTRODUCTION

Structural computing [24] has established itself as a generic approach to structuring information and services for a diverse range of application domains, including various hypermedia domains (e.g., navigational, spatial, and taxonomic). As an approach to software design and development, *structural computing* provides developers with techniques and

^{*}Professor Anderson performed this work while on sabbatical from the University of Colorado, Boulder.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

HT'06, August 22–25, 2006, Odense, Denmark.
Copyright 2006 ACM 1-59593-417-0/06/0008 ...\$5.00.

technology to manage heterogeneous sets of relationships between many diverse entities. Another area that handles sometimes ephemeral relationships between many entities is *context-aware computing*. Context-aware applications are those which can adapt to a user's current circumstances, detecting input such as time, location, available resources, social situation, etc. [36]. With a plethora of small sensors available, acquiring data is not a challenge, but establishing relationships between sensor input is, as is determining what at a given time best represents a user's context.

We present herein a structural computing approach to contextual hypermedia exemplified in the HYCONSC framework. Our contributions include a general model for context-aware hypermedia, a new definition of context based on dynamic queries, a general approach to comparing contexts, and an implementation of a dynamic user interface for exploring and discovering information via context. Our model of context as “dynamic queries over structured objects” offers a lot of power to developers looking to create contextual hypermedia systems or context-aware applications.

In particular, this model defines a user's context as the set of queries being used by a context-aware application to retrieve relevant information. This approach enables both developers and end-users to dynamically update what constitutes “context” for a given situation by simply updating the set of active queries. With respect to contextual hypermedia, this allows users to fine tune at run-time the types of objects that are being retrieved as “related” to their current physical and digital contexts and provides them with more power to find the information they need when they need it.

The paper is structured as follows: Section 2 describes related work within structural and context-aware computing. We then discuss three different types of context that are relevant to context-aware hypermedia applications in Section 3.1. In Section 3.2 we describe our new framework for contextual hypermedia systems, HYCONSC, and present its definition of context. Section 3.3 discusses our user interface for creating contexts and how it can be used to find information, sometimes serendipitously. In Section 3.4, we describe the framework's support for context matching as well as its ability to support “fuzzy” contextual search. We then discuss the architectural configurations that our framework supports in Section 4, describe systems related to HYCONSC in Section 5, and conclude the paper in Section 6.

2. RELATED WORK

We now provide background on the topics of structural and context-aware computing and discuss relevant related

work. A discussion of specific systems related to the HYCONSC framework is deferred to Section 5.

2.1 Structural Computing

The field of structural computing evolved from work on open hypermedia [25] in response to a 1997 paper by Nürnberg *et al.* entitled “As We Should Have Thought” [24]. Open hypermedia was, at that time, an area of research that devised techniques and tools to supply navigational hypermedia services to an open set of applications. These techniques allowed hypermedia concepts such as anchors and links to appear in third party applications that did not provide hypermedia support natively, and allowed users to traverse links between these hypermedia-enabled applications. Nürnberg *et al.* noted that there were many different types of hypermedia, and that open hypermedia systems supported only one of these domains—navigational hypermedia—and could not be easily extended to the other domains. They challenged the open hypermedia community to generalize open hypermedia infrastructure to support the structures and services of other application domains. This call led to work on component-based open hypermedia systems and structural computing environments, producing systems such as Callimachus [33], Construct [37], FOHM [22], IUHM [23] and Themis [2, 3, 18].

Relevant to our research here is the concept of *structural templates*. Structural templates have appeared in various forms in Callimachus, IUHM, FOHM, and Themis. The basic idea is to allow developers to assign names to particular structures of information and then to instantiate and manipulate instances of that information via those names. Here, structure refers to the attributes of the information being defined along with its internal and external relationships. In Themis, structural templates could be created by combining previously defined templates into a larger structure, often creating hierarchies of structured information. Each Themis template defined labels that could be used by clients of a structure to store and retrieve information. Thus, a template for a postal address could define the label “city” that allowed that part of the address to be manipulated by client code without needing to know exactly how or where that information was stored in the template. This feature enables *structural abstraction* (similar to object-oriented encapsulation) and allows developers to evolve the structure of a template without impacting existing client code.

As we discuss in Section 3, we make use of structural templates in HYCONSC as a foundational concept that allows the creation of many types of information objects that can participate in a variety of contexts. Indeed, templates underly our very definition of context and provide the basis for an efficient query system for finding objects contextually.

2.2 Context-aware Computing

Context-aware computing refers to software systems that can adapt their behavior, interface and structures according to a user’s context. Information about the context can be made available to the system either explicitly by the user (e.g., by entering a login name) or implicitly through associated sensory systems. Adding context-aware capabilities to systems is desirable for several reasons: firstly, interaction with applications can be greatly improved if an application adapts to a user’s situation and only provides information relevant to that situation. For example, filtering informa-

tion about a city based on a user’s location can make it much easier for users to gain an overview, especially given the limitations of browser user interfaces on small, mobile devices [17]. Secondly, knowledge about the physical environment surrounding a user can be used to create new powerful applications. Hypermedia, for instance, has traditionally been concerned with linking digital resources stored on computers, but the ability to identify physical resources through tagging or sensor input enables hypermedia systems to support linking to locations and physical objects.

2.2.1 Previous Definitions of Context

The idea of utilizing location in computing systems was advocated by Mark Weiser in his seminal 1991 paper on ubiquitous computing [36]. Some of the early research on location- and context-aware computing was performed on the Olivetti Active Badge systems in 1992 [35]. The term “context-aware computing” was however not defined until 1994, where Schilit *et al.* [30, 31] listed three important aspects of context: *where you are, who you are with, and what resources are nearby*. Schilit *et al.* defined context as a location and its dynamic collection of nearby people, hosts, and devices. A number of similar definitions exist that all try to define context by enumerating examples of context elements [6, 12, 29]. However, as discussed by Dey [13], these types of definitions may be too specific and hard to apply when trying to define general support for context-awareness: e.g., how is it decided whether a type of information can be regarded as context if it is not listed in the definition, and how does a system handle new types of contexts if its design is based on a fixed set of types?

Later, definitions of context became more general. Schmidt *et al.* [32] use the following definition: “[Context is defined as] *knowledge about the user’s and IT device’s state, including surroundings, situation, and to a less extent location.*” Chen and Kotz [9] define context as “*the set of environmental states and settings that either determines an application’s behavior or in which an application event occurs and is interesting to the user.*” Similarly, Dey [13] provides the following definition: “*Context is any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and application themselves.*”

These latter definitions are easier to apply than the former since information can be categorized as context on a per application basis, and they can be used as guidelines for designing general support for context-awareness.

2.2.2 Previous Models of Context

The different notions of context have led to a number of varied approaches to modeling and handling context in computer systems. Context defined as a fixed set of attributes can be represented by a hard-coded, optimized context model, whereas a more general definition will lead to more generic and flexible models. Typically, data structures include key/value pairs, tagged encodings, object-oriented models, and logic-based models [9].

In hypermedia, however, the prevalent way to model context has typically been as either composites serving as partitioning mechanisms on the global network or as key/value pairs—associated with links, nodes and other hypermedia objects—that describe parameters of the context the hyper-

media objects belong to. Context modeled as composites is often a purely structural partitioning concept constraining browsing and linking to some kind of context given by the user explicitly or implicitly. Key/value pairs associated with objects are typically used to describe in which context objects are visible. Neptune’s *Contexts* [11], Intermedia’s *Webs* [38], and the Webvise *Context* composites [15] belong to the first category while FOHM’s *context objects* [21] and Storyspace’s *guard fields* [4] belong to the second. Key/value pairs have also been used to model context in other context-aware systems e.g., Schilit’s and Theimer’s *located objects* [31] and the context-aware Web browser Mobisaic [34].

In our previous work with the context-aware HyCon framework [5], we experimented with several ways to represent context. HyCon provided an object-oriented data model (shown in Figure 1) with context data that could be described by subclassing an abstract data object. These specialized objects could then be associated with other data objects via a dedicated context composite. In addition to this approach, we also allowed key/value pairs to be associated with data objects to support the simple context tagging mechanism described above. During our evaluation of the HyCon framework [16] we found that while the composite mechanism was useful for modeling certain kinds of physical contexts, especially containment—such as locations where one location (a room) may be part of another location (a building)—this mechanism was seldom used in practice. The object-oriented approach, on the other hand, often resulted in minor changes to the existing framework service, so programmers preferred the simpler key/value pair model when new applications were created or old classes changed. This observation led us to evolve the HyCon data model (Figure 1) into the HYCONSC framework discussed below in Section 3.2 (and shown in Figure 2).

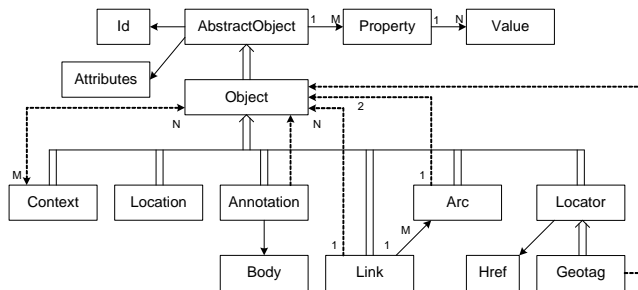


Figure 1: The original HyCon Data Model

3. CONTEXTS

In this section, we present our own notion of *context* and describe how this definition can be used to model and utilize context in applications.

3.1 Defining Context in Hypermedia

We view the definitions of context above (Section 2.2.1) as a starting point for a common context definition for hypermedia. When Chen and Kotz [9] describe context as “[a] set of environmental states and settings” and Dey [13] uses the term “any relevant information” to define context, it sets some requirements for the design of a hypermedia

framework. With context defined in general terms rather than specific entities, our framework must be able to handle a wide range of context data. Context must necessarily be modeled and defined as specific entities at some point, but this decision should be deferred to the design of concrete applications or perhaps even to run-time, where users should be free to specify the nature and format of context objects.

In order to ease the categorization of context information and to facilitate designing for context, we adopt the terminology developed by Brynskov *et al.* [7] to help distinguish the use of “context” as a concept at different levels of system development. Context is classified into three domains: *physical*, *digital*, and *conceptual*. These domains ease the transition from traditional digital-only hypermedia models to models that also encompass a notion of physical objects and contexts:

Physical context includes the physical surroundings of an entity. This includes physical location, physical objects, physical interaction, absolute time and space, and other physical measurements. Computer systems may be aware of the physical context by using sensors.

Digital context includes computer models, infrastructure, protocols, devices, resources and services, logs, and relative time and space. Specifically, traditional hypermedia structures such as links, collections, guided tours, etc. correspond to objects that can be used to represent relationships in the digital context.

Conceptual context describes user activity, intention, focus, and understanding of surroundings.

Most entities may have both a physical and a digital representation, i.e., the physical phenomenon is modeled in the computer system. Ideally, the relationship between the physical entity and the digital model should correspond to the user’s understanding of his or her current context (represented as the conceptual model or user model). The representation of physical entities in the digital model should thus reflect the conceptual context. As an example, consider a user using a smart phone with a context-aware hypermedia system. Picture the user located in front of a building, and the system being aware of the user’s position through the smart phone’s integrated GPS receiver. Based on this sensed data from the physical context, the hypermedia system can search its database (i.e., the digital context) for information pertaining to this particular physical context. In our example, the system finds a resource describing the building and presents it to the user. Furthermore, since additional resources annotate the displayed document, links to these resources are presented in the interface allowing the user to further explore the document’s digital context. In this example, the user will have no problem coupling the physical building with the document describing it in the hypermedia system, and the conceptual context therefore corresponds to the relationship between the physical and digital contexts.

The main motive for adopting these three domains is not to enumerate a definitive list of parameters essential to the understanding of context, but rather to partition the design space. This allows developers to focus on the aspects they wish to support in their context-driven applications.

Next, we discuss how structural templates in HYCONSC can handle the general requirements for modeling arbitrary

context information, and we demonstrate how the division of context into three domains can be used to *specify* and *explore* contexts with dynamic queries.

3.2 Modeling Context in HyConSC

We have created a framework, called HYCONSC that allows us to explore the ways in which our context domains can be realized in the design of context-aware applications. This framework is the result of evolving the data model of the HyCon [5] contextual hypermedia system with the SmallSC [1] lightweight structural computing framework. SmallSC has been significantly enhanced while working on the research presented in this paper, especially with respect to its ability to provide search functionality over the objects stored in its repository. These features are discussed below within the context of the HYCONSC framework. HyCon is a contextual hypermedia system that specializes in the creation of mobile hypermedia applications running on PDAs and smart phones. It also integrates these applications with a set of sensors deployed throughout a user's environment. HyCon's original data model (see Figure 1) hard coded the notion of context by having it as an explicit class with fixed connections to other elements of the data model.

We have transformed this data model by removing virtually all of its "leaf" nodes and recreated them as templates in the new HYCONSC framework (see Figure 2). In addition, we have made concepts such as repositories, sensors, and queries explicit members of the framework. HyCon had analogs of these concepts, but they lived in separate services that each shared the classes that were present in the original data model. However, when a new concept had to be added to the data model, HyCon developers experienced increasing development costs, as it impacted the code of existing services in sometimes unanticipated ways. The reformulation of the original data model's leaf classes as structural templates has mitigated this problem because all objects are now instances of the same class (HYCONOBJECT from Figure 2) that can be treated uniformly by HYCONSC's repository. Furthermore, services now interact only with objects of known templates and do not care if new templates are defined, as these new types cannot alter the existing templates and objects that they know about.

3.2.1 Templates in HyConSC

We now discuss the entire HYCONSC framework, starting with how it makes use of the concept of structural templates and eventually demonstrating how it models contexts as "dynamic queries over structured objects."

The first step in defining a system's context is to establish a mechanism for specifying the types of objects that will appear in them. As discussed in Section 2.1, the HYCONSC framework makes use of *structural templates* to define the different types of objects that will be accessible in contexts.

A HYCONSC *template* is a *named set of typed properties*. In the template, each property is assigned a name, a type, and a default value. The types are drawn from an extensible set of type definitions that all derive from the abstract type HYCONVALUE. The current set of supported types include integers, floating point numbers, strings, dates, ids, and the following collection types: lists, maps, and sets. The only restriction on the use of collection classes is that a collection class can only be populated with instances of the supported types. This range of types provides HYCONSC with the

ability to model a wide variety of information objects.

All templates have three properties defined by default: a name, a type and an id. The name property ensures that all HYCONSC objects have a human readable name, while the type property ensures that objects can be retrieved via their template's name. Ids are implemented as UUIDs (universally unique identifiers) and form the basis for modeling relationships between templates¹. In particular, any property with type HYCONID has the ability to point at other HYCONSC objects at run-time.

Objects in HYCONSC are created by instantiating a template. Instantiation creates a new object and populates it with the set of properties defined by its template. Additional properties may be arbitrarily added to and removed from an object at run-time; however, properties defined by an object's template may not be removed from the object. This decision guarantees that an object of a particular template always provides values for the properties defined in that template while allowing for individual instances to be tailored with additional properties as needed by their clients.

HYCONSC templates and the objects instantiated from them are stored in a *repository*. The repository maintains a number of indexes to facilitate efficient retrieval. In particular, the repository indexes objects by their names, their types, and by the properties they contain. The mechanism that defines these indexes is part of a larger, generic query system that makes it easy to add additional indexes to the repository. For instance, it is possible to add indexes to the repository that track the values of properties, such as a full text search index on string-based properties.

In order to support the creation of context-aware applications, HYCONSC also allows objects to be created from the information delivered by a *sensor* attached to the HYCONSC framework (the framework's API contains a sensor manager that provides the hooks needed to attach sensors to a HYCONSC repository). Indeed, the mechanism for creating such objects is the same as described above: object instantiation. We are able to accomplish this by simply adding a template for each sensor that is attached to the repository. The template for a sensor defines the properties described in a sensor's events. This allows the information contained in a sensor event to be copied directly into the corresponding HYCONSC object instantiated from that sensor's template.

Currently the HYCONSENSOR objects (see Figure 2) registered with the sensor manager determine when a sensor event is stored in the repository. Some sensors generate so many events (such as accelerometers or GPS receivers) that it does not necessarily make sense to store all of them. In the future, we may switch to a constraint-based mechanism for determining whether a sensor event should be stored in the repository; such a change would simplify HYCONSENSOR objects as they would only need to deliver events to the sensor manager and no longer worry about storage issues.

3.2.2 Context as Dynamic Queries over Structured Objects

Having all objects in HYCONSC associated with a template provides a solid structural basis for defining HYCONSC contexts. In particular, we model contexts as *queries* that reference the properties defined by the templates of a HYCONSC repository. Thus, we can create a context that contains all objects with a particular set of property names

¹UUID: <http://www.ietf.org/rfc/rfc4122.txt>

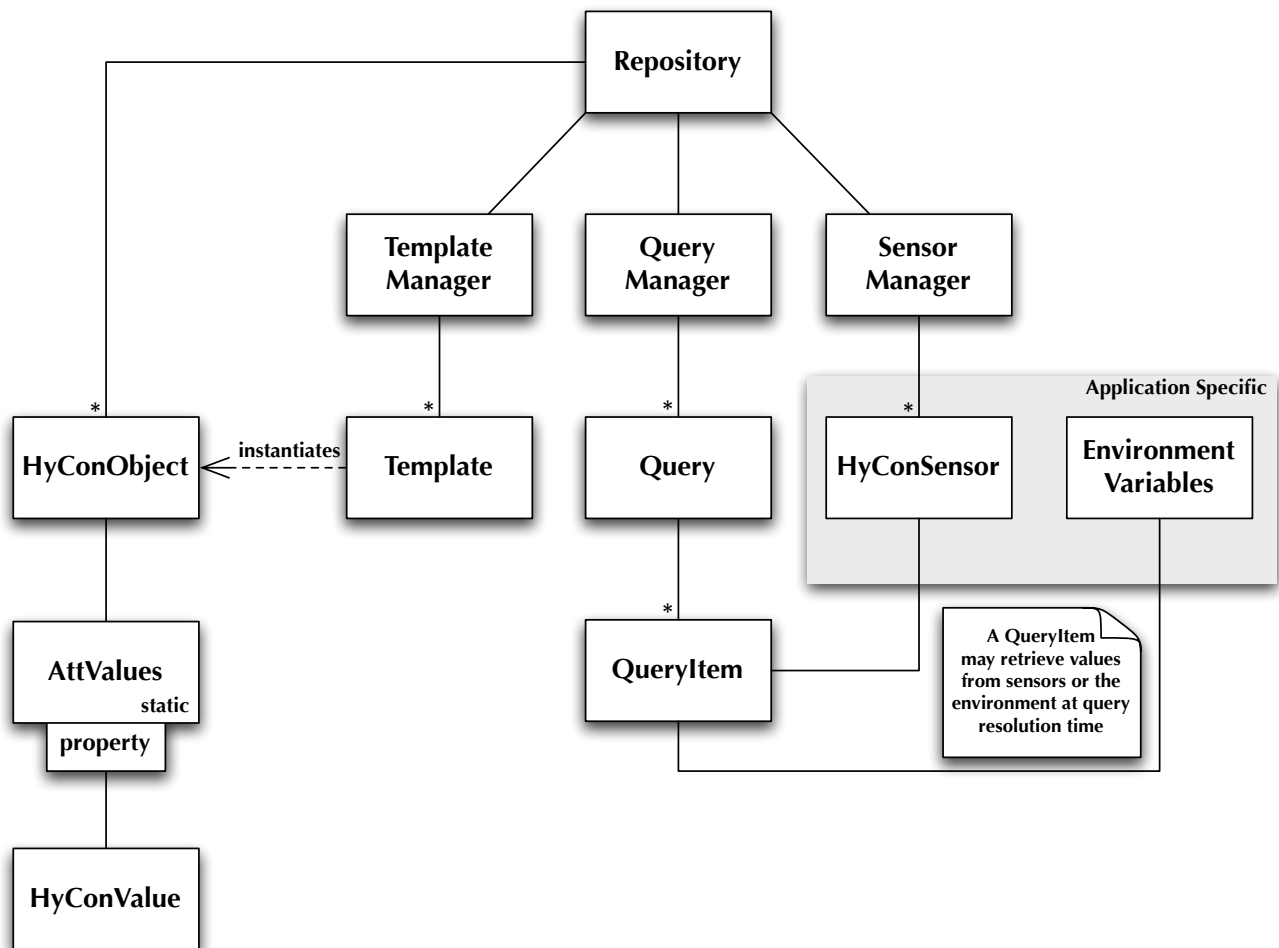


Figure 2: The HyConSC Framework

regardless of which templates defined those names. We can also easily create contexts that track objects of a particular type or that reference a specific physical location or time. More formally, a *context* is a *named set of query items* that is managed by the query mechanism of a HYCONSC repository. The query mechanism ensures that the result set of a query stays current as HYCONSC objects are created, updated, or removed (including when this occurs in response to the delivery of a new event by an attached sensor). The primary benefit of this definition of context is its inherent flexibility. A query can be created to act as the “context of annotation objects”, another query can act as the “context of documents about cathedrals”, and then a new query can easily be created that combines the first two and adds additional query items to represent “the context of annotations on documents about Notre Dame in Paris.” A user’s “current context” then becomes the set of active queries stored in the HYCONSC query manager (as seen in Figure 2) for the context-aware application that he/she is using.

Query items define three pieces of information: a *property name*, a *filter*, and a *target value*. As a search is performed, a query item is applied to a set of candidate HYCONSC objects. For a given object, the query item’s property name is used to retrieve a property value from the object. This

value is compared to the item’s target value using the filter which indicates whether a “hit” has been achieved. If so, the object is added to the query item’s result set and the next candidate object is evaluated. (Note: candidate objects for a search may potentially involve all of the objects in a repository. However, the indexes maintained by the query manager are applied to the query to attempt to narrow the list down to a relevant subset. Thus, if a query contains a query item that looks for objects with a particular name, then the query manager’s name index is used to ensure that the list of candidate objects contains only those objects with that name.) A filter is an operation that can be applied to properties of a particular type. For instance, string-based properties have filters such as: STRINGCONTAINS, STRINGIS, STRINGISNOT, and STRINGMATCHES. The filter returns true to indicate that a “hit” has occurred when the semantics of its operation indicate that the target value matches an object’s property value.

A query can either be an ANDQUERY or an ORQUERY. After a candidate set of objects has been applied to a query’s items, the individual result sets of each item are combined either by computing the intersection of all the sets in the case of an ANDQUERY or by computing the union of all the sets in the case of an ORQUERY.

3.2.3 Dynamic Queries and Sensor Input

The final result set of a query is only updated when the repository indicates to the query manager that a change has occurred that may impact a query's results. For instance, if a query references the property names of "city" and "state," then there is no reason to update that query when some object's "country" property is updated. Having the query manager update queries in response to changes in the repository is one aspect of HYCONSC's dynamic queries.

The other dynamic aspect of HYCONSC's query mechanism occurs when a query item's target value references the value of a sensor or environment variable. The values that sensors generate and the application-specific environment variables are available to query items via a special syntax that causes the referenced value to be used as the target value during query resolution time. The syntax for referencing a sensor value is $\$sensor.<sensor-name>.<sensor-value>\$$, e.g., $\$sensor.gps.x\$$ and the syntax for referencing an environment variable is $\$<namespace>.<variable>\$$, e.g., $\$firefox.current-url\$$. When a query is being resolved, a target value of this form is resolved by asking the sensor or environment variable for its current value; this value is then used when the query item's filter is evaluating candidate objects. Furthermore, queries which have query items that reference sensors and/or environment variables attach listeners to the relevant event source to ensure that they update whenever the sensor or variable is changed. This mechanism is critical for specifying contexts that update as a user moves through or manipulates his/her environment. As the referenced sensors update their values, the query is notified and it updates its result set accordingly.

We now present the user interface that we have developed on top of this framework that allows developers to specify contexts as queries and interact with their results.

3.3 Using Context to Explore Information

Figure 3 presents a query browser developed using the HYCONSC framework. This browser can be attached to any HYCONSC repository to interactively explore its contents via the use of contextual queries. Indeed, this browser can be used by developers to iteratively develop the queries that they need to represent context in their context-aware applications. Queries can be created, deleted, and selected in the top left pane of the query browser. Creating or deleting queries cause them to be added or removed from HYCONSC's query manager. The details of the query are displayed in the query editor, located in the top right pane of the query browser. This editor shows the name of the query, its type (AND/OR), whether its enabled and/or dynamic, and its query items. All aspects of the query can be edited in this window and its result set (displayed in the lower left pane of the query browser) updates dynamically in response to each edit.

Query items can be added and removed using the appropriate buttons, and pop-up menus are attached to each of the three fields of a query item to make it easy to enter legal values². The pull-down menu of the name field contains all of the different property names that have been defined by the repository's templates. The pull-down menu of the filter

field contains all filters valid for the selected property name. If a property name appears in more than one template with different types, then the filter pull-down menu is populated with all of the relevant filters, sorted by type. The target value pull-down menu is populated with the available sensor properties and environment variables as a convenience (as shown in Figure 3), however the user is not restricted to these values and may enter any desired value in this field.

If an object is selected in the results list, its associated information is displayed in the object inspector (see Figure 3), located in the lower right pane of the query browser. The object inspector displays all of an object's properties and their associated values, and allows a user to double click on any property that displays another object's id to traverse to the referenced object. The object inspector provides a "bread crumb" trail across the top of its display to allow a user to keep track of how the currently displayed object was reached. For example, in Figure 3, a user started by viewing an object called "Curriculum Vitae (May 2004)." From here, the user then double clicked on an id leading to the object named "balasuthasundararajah." Clicking on the arrow buttons allows a user to easily traverse the current set of bread crumbs.

Just below the object inspector pane is the digital context pane. This pane shows all objects *that point at* the object currently displayed in the object inspector. Double clicking any of the objects shown in the digital context pane causes the object inspector to traverse to the selected object. The digital context pane allows a user to serendipitously discover objects related to the currently displayed object, because as the number of templates and objects contained in a repository grows it will become increasingly difficult for a human to keep track of all the possible ways in which objects in the repository can interrelate. The rapid nature of the traversals provided by the object inspector and the digital context pane allows users to explore the information space around an object in an engaging and direct manner.

In order to further support a user who unexpectedly discovers an interesting object via traversal, we have implemented a feature that can auto-generate a query based on the currently selected object in the object inspector. This query can then be edited to make it more general (typically by removing or altering query items) and this provides the user with a set of similar objects displayed in the search results pane. The user can then select one of these objects and explore its digital context for additional interesting objects. These features form an efficient loop that enables users to explore large information spaces rapidly.

3.4 Context Matching

A key issue for contextual hypermedia systems is that of *context matching*. A basic feature of these systems is that a link is created with respect to a particular context. If a user returns to that context (or a similar context) at a later time, all links associated with that context should be made available. The problem then becomes to determine when a user's current context matches, or is sufficiently similar to, a context that has been stored in the contextual hypermedia system. A simple approach to this problem is to require that the current context match the stored context exactly, i.e., they each have the same set of properties with the same values. This approach however breaks down in the presence of sensor data which often cannot be made to match *exactly*.

²Our query mechanism supports sub-queries, in which query **A** can be added to query **B** to influence the result set of query **B**. However, the query browser currently does not provide controls to manipulate a query's sub-queries.

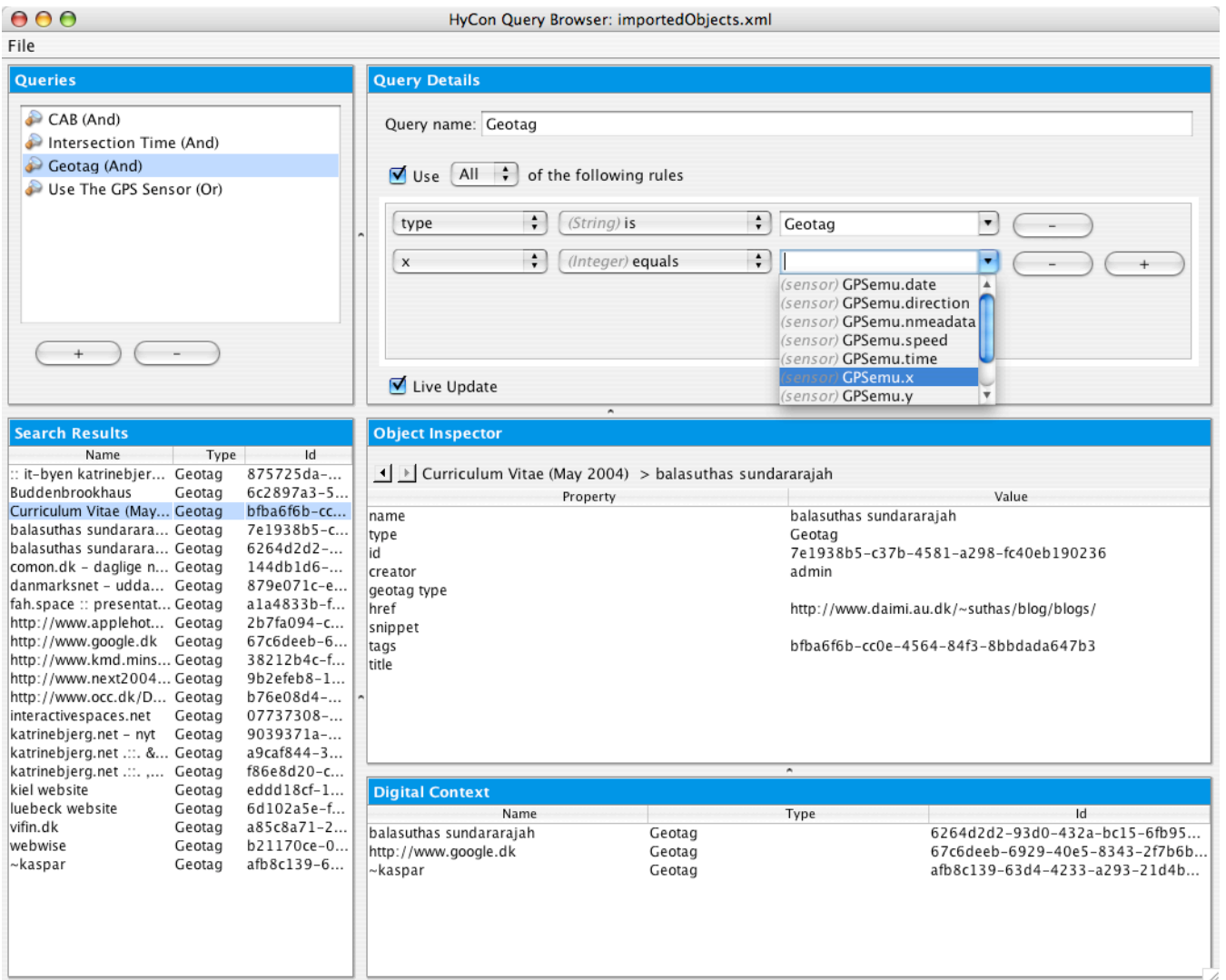


Figure 3: The HyConSC Query Browser.

For instance, a GPS sensor will produce slightly different x and y coordinates each time a user visits a location that they perceive as being “the same place,” such as “in front of the computer science department.” In such cases, it is better to support “fuzzy” matches where a range is associated with a particular value and where the value “matches” any value that falls within the range.

Our framework supports several approaches to context matching. The simplest method is to compare two queries to see how much of an overlap exists between the properties, filters, and target values contained in their query items. An exact match occurs when the two queries have identical sets of query items producing a similarity value of 1.0. On the opposite end of the scale, two queries with disjoint sets of query items have a similarity value of 0.0. Otherwise, a similarity value is computed using a simple algorithm with the following steps:

1. Given queries A and B, where $A.numberOfItems() \geq B.numberOfItems()$
2. Set $total = 0$

3. Set $items = A.numberOfItems()$
4. For item A_i in A, find all items in B that have the same property name, $B_0 \dots B_j$
5. If $j > 0$ then compute similarity measures between A_i and all items in $B_0 \dots B_j$ as $s_0 \dots s_j$.
6. Associate the largest similarity value s_k with A_i and remove B_k from B. Add s_k to $total$.
7. Once iteration is complete, return $\frac{total}{items}$

The similarity measure for query items is obtained with the following simple algorithm:

1. Given query items A and B
2. If $A.getName() \neq B.getName()$ then return 0
3. If $A.getFilter() \neq B.getFilter()$ then return 0.33
4. If $A.getValue() \neq B.getValue()$ then return 0.66
5. return 1.0 // name, filter & value are all equal

Given these algorithms, it is possible to retrieve all contexts that are similar to a given context. A contextual hypermedia system built on top of this framework can then decide the threshold it will use to include objects of one context

in another based on their similarity. Given the somewhat coarse nature of the algorithms above, we intend to provide applications with the ability to supply HYCONSC with alternative similarity measures³. However, these default algorithms provide contextual hypermedia developers with a useful starting point for experimentation.

Another approach to context matching enabled by the HYCONSC framework is the ability to create a hierarchy of contexts in which a child context of depth n differs from its ancestor context n levels up by n query items. Thus, a given context's children all share the same query items with it save one; while a given context's grandchildren all share the same query items with it save two, etc. If all contexts in such a hierarchy are of type ANDQUERY, then child contexts are more specific than their parents. A contextual hypermedia system built on top of the framework can then experiment with different algorithms for including objects in a context by looking for new objects in that context's parents (thus expanding the number of objects related to the context) or that context's siblings (thus looking for objects that are similar but prioritize different properties).

Finally, the HYCONSC framework can expand the matches made by a particular context by employing operators that match values based on a given threshold or range, i.e., what is commonly known as a "fuzzy" search. For instance, an *approximates filter* could be developed for integers that takes a target value and a range and computes a "hit" whenever a candidate property value falls within the range surrounding the target value. This would be useful, for example, when comparing the x and y values of GPS sensor readings where a developer might indicate that any point within five meters of a given reading should be considered the "same" location. In the current implementation of the framework, "fuzzy" filters are treated the same as filters that perform exact matches and are available to users from the operator pull-down menu of query items in the query editor. In a future version of the query browser, we intend to expose the range parameter of "fuzzy" filters to allow users to dynamically adjust the "fuzziness" of the item at run-time. This will allow a user to e.g., manipulate a slider specifying the fuzziness of a query item and watch as the query's search results dynamically update.

These mechanisms offer a powerful set of context matching options when creating new contextual hypermedia systems with the HYCONSC framework.

4. ARCHITECTURE AND DEPLOYMENT

The HYCONSC framework is intended to be used by other developers to build context-aware applications, including contextual hypermedia systems. The framework's architecture currently supports two deployment configurations. The first configuration involves linking the framework directly into a context-aware application as shown in Figure 4(a). In this configuration, HYCONSC acts as the application's data model. The application's services interact with the framework to register templates, instantiate objects, track queries, manage sensors, and access repositories. The application's user interface allows users to interact with the

³Indeed, one obvious point of improvement for the second algorithm is to make use of a distance measure for comparing the similarity of the target values of items A and B and using that value to influence whether a value greater than 0.66 is returned in the case where the two values are not equal.

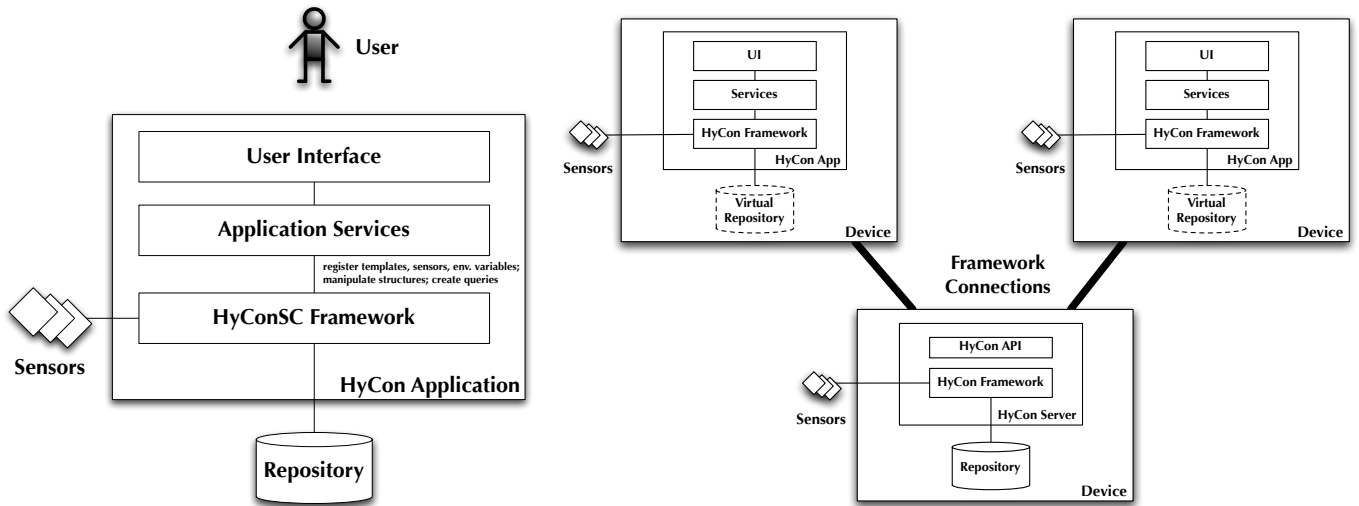
application and adapts the display of the information stored in HYCONSC objects to best utilize the capabilities of the device hosting the application.

The second configuration (shown in Figure 4(b)) reveals the framework's support for distribution. An instance of the framework is wrapped by a server that exports an API that allows templates, objects, repositories, queries, and sensors to be manipulated remotely. This server then manages connections to other instantiations of the framework embedded in context-aware applications running on remote devices. When the framework is employed in this fashion, the context-aware application indicates what templates it needs to do its job; its instance of the framework contacts the server and downloads objects of the indicated templates forming a virtual repository on the local device⁴. At application-specified intervals, changes to the virtual repository are synced with the repository being managed by the server. We make use of an optimistic concurrency control policy to manage updates to shared objects. In particular, if the same object is modified by two different context-aware applications, the last one to attempt to store that object in the server's repository will receive a notification that the object was changed from under it. The notification includes a copy of the changed object. It is then the responsibility of the application to manage the conflict by editing (in some fashion) that copy and submitting it back to the server. This approach greatly simplifies the HYCONSC server application as it never has to broadcast changes about the objects in the repository to all connected clients.

The server does however broadcast sensor events for sensors that are connected to it, but only for events for which a client has registered interest. In the distributed configuration, sensors can be attached at multiple points in the architecture; if a client's performance depends on receiving timely events from a particular type of sensor, that sensor can and should be attached directly to the client. Otherwise, sensors may be shared among multiple clients through associating the sensors with the server. The server's sensor manager coordinates with the sensor managers running on the connected clients to ensure that the clients are aware of the server's sensors. This allows queries on the clients to reference sensor values held by the server, as the client's sensor manager will make it appear that the remote sensor is in fact directly connected to the client.

The advantage of HYCONSC's approach to distribution is that a full instance of the framework exists at each of the nodes of a distributed application. This is useful when a client application becomes temporarily disconnected from the network as the user will experience almost no degradation of service from the context-aware application. Sensor information from remote sensors will be unavailable but data stored in the virtual repository will not be lost. Indeed, to further safeguard that information, the repository's API provides operations that allow it to be severed from its host repository and become a "real" repository by saving a persistent copy of its information on the client device. Later, when a connection to the network has been established, the real repository can be converted back to a virtual repository by having it connect to the server's repository, sending information about changed objects back to the server, han-

⁴The client does not have to download all the objects associated with its selected templates; it can use context definitions to limit the number of objects retrieved.



(a) Use of the HyConSC Framework by a Single Application.

(b) HyConSC's Support for Distributed Architectures.

Figure 4: Architectural Configurations of the HyConSC Framework.

dling any conflicts that arise, and finally flushing its local persistent copy of the repository.

We have constructed a proof-of-concept prototype that uses the framework in a standalone application. This prototype implements the query browser discussed in Section 3.3 (and shown in Figure 3) and we have applied it to a data set that was imported from the original HyCon system that contained over 1200 objects. Three templates were defined to specify the structures of the imported objects. We have used these objects to test the query mechanism of the framework. With respect to sensors, we have developed two emulators—a GPS sensor and an RFID scanner—that were used to generate events that could be used to test the ability to reference sensor values in HYCONSC queries. We are currently in the process of migrating the HyCon contextual hypermedia system to the distributed form of the HYCONSC framework and will be evaluating how well the framework supports HyCon's existing range of services as well as support the addition of new services with minimal impact on existing code.

5. RELATED SYSTEMS

While Section 2 positioned our work with respect to the fields of structural computing and context-aware computing, this section describes systems that make use of query mechanisms similar to the one implemented in HYCONSC.

The FOHM hypermedia model [20, 21] is primarily based on the traditional navigational model of open hypermedia (OHP-Nav), but with a number of extensions that generalizes it to other hypermedia domains such as spatial and taxonomic hypermedia. FOHM generalizes the standard OHP-Nav objects (Link, Endpoint, Anchor, and Node) into a new set of elements (Association, Binding, Reference, and Data objects) and allows each of these elements to be augmented with a context object. As described in Section 2.2.2, FOHM context objects are sets of key/value pairs that describe the context of their associated element. Furthermore, both contexts and context-attributes may have associated constraints that can be evaluated at run-time. The refer-

ence implementation of FOHM is the Auld Linky structure server [19] which implements a pattern matching mechanism on context objects. When a user queries the server for a set of links, a context may be supplied along with the query and the server will match that context against the contexts of the elements. If the contexts do not match, the corresponding substructure is pruned away. If no context is supplied, the structure is assumed to match, which effectively means that the systems will act as a normal (non-contextual) structure server. While this makes the FOHM context mechanism very powerful, it has some weaknesses, just as the original HyCon key/value approach. First of all, the context attributes are type-less, and the context objects are defined in an ad-hoc fashion without the use of a schema. Furthermore, the constraints are expressed as Perl scripts that have to be evaluated at run-time. These loosely defined structures will make it difficult to design something like the HYCONSC query editor that relies on the presence of structured objects (via the use of structural templates) for context data, and introspection of data types to aid developers and users in creating correct and meaningful queries.

The Context Information Service (CIS) [27] based on the Stick-e note data model [6, 26] is a system that also employs a template mechanism to specify objects and their relationships. CIS operates on a *world* that is composed of *artifacts*. An artifact has a name, a type, and a set of contextual states, known as trigger actions. Artifacts are named instances of templates from an artifact catalog. The *contextual states* which are specified in the templates are similarly chosen from a state catalog. Each state specifies the operations that are available for it (e.g., a location state may have distance operations and so forth). To create (generic) links between artifacts, a *relationship* can be chosen from a relationship catalog and anchored to an existing artifact or artifact state (e.g., a CLOSETO relationship called "nearby printers" anchored to a person's location state, with a range of "ten meters" and "printer" template instances as the candidate artifacts would result in nearby printers be-

ing presented to the user). When events are sent from CIS's associated sensors, the sensor data is matched against the trigger conditions of all artifacts in the system's world, and triggered artifacts are presented to the user.

Our approach to templates is less operational than the templates found in CIS. HYCONSC templates are used only to specify the typed attributes of a context object. In addition, the HYCONSC framework defers the specification of what represents context and context data as late as possible in the development of context-aware applications. CIS, on the other hand, offers developers already implemented templates that they can adapt to their needs before having to resort to defining their own. We believe that both approaches are valid and offer different benefits to developers: HYCONSC offers flexibility while CIS offers immediate functionality through software reuse.

The Gaia meta-operating system [28] and the Context Toolkit [14] employ mechanisms similar to HYCONSC's query items but use them to allow applications to subscribe to events about a user's environment. Unlike the 3-ary query items of HYCONSC, Gaia specifies context using 4-ary predicates like CONTEXT (location, chris, entering, room 3231) and CONTEXT (social relationship, venus, sister, serena). This approach adds a *subject* dimension to the context specification which specifies the object the query should be matched against. While this mechanism is similar to the HYCONSC approach, the focus is more on acquiring context information, than on defining an application's context.

6. CONCLUSION

The HYCONSC framework makes use of dynamic queries over structured objects as a model for providing significant power and flexibility to the developers of context-aware applications. Structural templates provide a flexible and extensible foundation for specifying a wide range of information objects. The use of dynamic queries to specify context over those objects, as well as the ability to integrate sensor data into those queries, again provides the flexibility needed to apply context-aware techniques to a wide range of application domains. In addition, HYCONSC's built-in support for context matching and "fuzzy" queries makes it an ideal platform for creating new contextual hypermedia systems. Furthermore, the framework can be configured for use in stand-alone applications as well as in distributed settings. We have tested the framework and its user interface by applying it to a set of over 1200 objects taken from the existing HyCon contextual hypermedia system, and we are continuing our evaluations of the framework by migrating HyCon to a HYCONSC based server. We believe our work so far has demonstrated the potential of our new model of "context as query" and we will continue to evaluate this potential via a new generation of powerful, context-aware hypermedia services hosted on top of the HYCONSC framework.

Acknowledgments

Dr. Anderson wishes to thank Professor Kaj Grønbaek for making it possible to visit the University of Aarhus during his sabbatical from the University of Colorado. In particular, this work has been supported by the Center for Interactive Spaces under ISIS Katrinebjerg, Aarhus, Denmark.

7. REFERENCES

- [1] K. M. Anderson. Towards lightweight structural computing techniques with SmallSC. In *Proceedings of the Metainformatics Symposium*, Esbjerg, Denmark, Nov. 2005.
- [2] K. M. Anderson, S. A. Sherba, and W. V. Lepthien. Structural templates and transformations: The Themis structural computing environment. *Journal of Network and Computer Applications*, 26(1):47–71, Jan. 2003.
- [3] K. M. Anderson, S. A. Sherba, and W. V. Lepthien. Structure and behavior awareness in Themis. In Carr and Hardman [8], pages 138–147.
- [4] M. Bernstein. Storyspace 1. In K. M. Anderson, S. Moulthrop, and J. Blustein, editors, *Proceedings of the 13th ACM Hypertext Conference*, pages 172–181, College Park, Maryland, USA, June 2002. ACM Press.
- [5] N. O. Bouvin, B. G. Christensen, K. Grønbaek, and F. A. Hansen. HyCon: A framework for context-aware mobile hypermedia. *The New Review of Hypermedia and Multimedia*, 9:59–88, 2003.
- [6] P. Brown, J. Bovey, and X. Chen. Context-aware applications: from the laboratory to the marketplace. *IEEE Personal Communications*, 4(5):58–64, Oct. 1997.
- [7] M. Brynskov, J. F. Kristensen, B. Thomsen, and L. L. Thomsen. What is context? Technical report, Department of Computer Science, University of Aarhus, 2003.
<http://www.daimi.au.dk/~brynskov/publications/what-is-context-brynskov-et-al-2003.pdf>.
- [8] L. Carr and L. Hardman, editors. *Proceedings of the 14th ACM Hypertext Conference*, Nottingham, UK, Aug. 2003. ACM Press.
- [9] G. Chen and D. Kotz. A survey of context-aware mobile computing research. Technical Report TR2000-381, Department of Computer Science, Dartmouth College, Nov. 2000.
- [10] D. de Roure and H. Ashman, editors. *Proceedings of the 15th ACM Hypertext Conference*, Santa Cruz, CA, USA, Aug. 2004. ACM Press.
- [11] N. M. Delisle and M. D. Schwartz. Contexts-a partitioning concept for hypertext. *ACM Transactions on Information Systems*, 5(2):168–186, 1987.
- [12] A. K. Dey. Context-aware computing: The CyberDesk Project. In *Proceedings of the AAAI 1998 Spring Symposium on Intelligent Environments (AAAI Tech. Report SS-98-02)*, pages 51–54, Stanford, CA, 1998.
- [13] A. K. Dey. Understanding and using context. *Personal and Ubiquitous Computing*, 5(1):4–7, 2001.
- [14] A. K. Dey, D. Salber, and G. D. Abowd. A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications. *Human-Computer Interaction (HCI) Journal*, 16(2-4):97–166, 2001.
- [15] K. Grønbaek, L. Sloth, and N. O. Bouvin. Open hypermedia as user controlled meta data for the Web. In *Proceedings of the 9th International World Wide Web Conference*, pages 553–566, Amsterdam, Holland, May 2000.
- [16] F. A. Hansen. Representing Context in Hypermedia Data Models. In *Proceedings of the International*

- Workshop on Context in Mobile HCI*, Salzburg, Austria, Sep. 2005. <http://mobilehci.icts.sbg.ac.at/context/papers.htm>.
- [17] F. A. Hansen, N. O. Bouvin, B. G. Christensen, K. Grønbæk, T. B. Pedersen, and J. Gagach. Integrating the Web and the World: Contextual trails on the move. In de Roure and Ashman [10].
- [18] W. Lepthien and K. Anderson. Unifying structure, behavior, and data with Themis types and templates. In de Roure and Ashman [10], pages 256–265.
- [19] D. Michaelides, D. Millard, M. Weal, and D. De Roure. Auld Leaky: A contextual open hypermedia link server. Position paper, OHS7, Aarhus, Denmark, 2001.
- [20] D. Millard, D. De Roure, D. Michaelides, M. Thompson, and M. Weal. Navigational hypertext models for physical hypermedia environments. In *Proceedings of the 15th ACM Hypertext Conference*, pages 110–111, Santa Cruz, CA, USA, 2004.
- [21] D. Millard, D. Michaelides, D. De Roure, and M. Weal. Beyond the traditional domains of hypermedia. In *the Workshop on Open Hypermedia Systems*, pages 26–32, College Park, MD, 2002.
- [22] D. Millard, L. Moreau, H. Davis, and S. Reich. FOHM: a fundamental open hypertext model for investigating interoperability between hypertext domains. In *Proceedings of the ACM Hypertext Conference*, pages 93–102, San Antonio, TX, USA, 2000.
- [23] M. Nanard, J. Nanard, and P. King. IUHM: A hypermedia-based model for integrating open services, data and metadata. In Carr and Hardman [8], pages 128–137.
- [24] P. Nürnberg, J. Leggett, and E. Schneider. As we should have thought. In *Proceedings of the ACM Hypertext Conference*, pages 96–101, Southampton, UK, April 1997.
- [25] K. Østerbye and U. Wiil. The flag taxonomy of open hypermedia systems. In *Proceedings of the ACM Hypertext Conference*, pages 129–139, Wash. D.C., USA, Mar. 1996.
- [26] J. Pascoe. The stick-e note architecture: extending the interface beyond the user. In *the 2nd International Conference on Intelligent User Interfaces*, pages 261–264, 1997.
- [27] J. Pascoe. Adding generic contextual capabilities to wearable computers. In *Proceedings of the 2nd IEEE International Symposium on Wearable Computers*, page 92, Washington, DC, USA, 1998.
- [28] M. Romn, C. Hess, R. Cerqueira, A. Ranganathan, R. Campbell, and K. Nahrstedt. Gaia: A Middleware Infrastructure to Enable Active Spaces. *IEEE Pervasive Computing*, pages 74–83, Oct–Dec 2002.
- [29] N. S. Ryan, J. Pascoe, and D. R. Morse. Enhanced reality fieldwork: the context-aware archaeological assistant. In V. Gaffney, M. van Leusen, and S. Exxon, editors, *Computer Applications in Archaeology 1997*, British Archaeological Reports. Tempus Reparatum, Oxford, Oct. 1998.
- [30] B. Schilit, N. Adams, and R. Want. Context-aware computing applications. In *Proceedings of the Workshop on Mobile Computing Systems and Applications*, pages 85–90, Santa Cruz, CA, USA, Dec. 1994.
- [31] B. N. Schilit and M. Theimer. Disseminating active map information to mobile hosts. *IEEE Network*, 8:22–32, 1994.
- [32] A. Schmidt, K. Aidoo, A. Takaluoma, U. Tuomela, K. V. Laerhoven, and W. V. de Velde. Advanced interaction in context. In *First International Symposium on Handheld and Ubiquitous Computing (HUC99)*, volume 1707 of *Lecture Notes in Computer Science*. Springer, Karlsruhe, Germany, Sept. 1999.
- [33] M. Tzagarakis, D. Avramidis, M. Kyriakopoulou, M. C. Schraefel, M. Vaitis, and D. Christodoulakis. Structuring primitives in the Callimachus component-based OHS. *Journal of Network and Computer Applications*, 26(1):139–162, Jan. 2003.
- [34] G. M. Voelker and B. N. Bershad. Mobisaic, an information system for a mobile wireless computing environment & engineering. In *Proceedings of the Workshop on Mobile Computing Systems and Applications*, pages 85–90, Santa Cruz, CA, USA, Dec. 1994.
- [35] R. Want, A. Hopper, V. Falcao, and J. Gibbons. The active badge location system. Technical Report 92.1, Olivetti Research Ltd, Cambridge, UK, 1992.
- [36] M. Weiser. The computer for the 21st century. *Scientific American*, 265(3):66–75, Feb. 1991.
- [37] U. K. Wiil, S. Tata, and D. L. Hicks. Cooperation services in the Construct structural computing environment. *Journal of Network and Computer Applications*, 26(1):115–137, January 2003.
- [38] N. Yankelovich, B. J. Haan, N. K. Meyrowitz, and S. M. Drucker. Intermedia: the concept and the construction of a seamless information environment. *Computer*, pages 81–96, Jan. 1988.