C3S: a Content Sharing Middleware for Smart Spaces

Marcos Paulino Roriz Junior, Marco Aurélio Lino Massarani, Leandro Alexandre Freitas,
Ricardo Couto Antunes da Rocha, Fábio Moreira Costa
Institute of Informatics
Federal University of Goiás
Goiânia - Goiás - Brazil
{marcosjunior, lino, leandroaf, ricardo, fmc}@inf.ufg.br

***Abstract*—Current sharing-based applications combine new computing devices with smart spaces to provide content-level ubiquity, *i.e.*, the possibility to exchange and move content freely in a ubiquitous environment. However, due to the environment complexity and lack of infrastructure platforms, most of the work in the area is repeatedly built from scratch using raw techniques, such as socket and rpc, to express content sharing. Aiming to provide an infrastructure for the development of this kind of applications, we propose Content Sharing for Smart Spaces (C3S), a middleware that offers a high-level programming model using primitives that are based on a set of content sharing semantics. They express a set of behaviors, *move*, *clone*, and *mirror*, which serve as a building blocks for developers to implement sharing and content ubiquity features.**

***Keywords*—C3S; Content Sharing Middleware; Pervasive Middleware; Middleware; Ubiquitous Computing**

I. Introduction

Over the past years, there has been a significant increase in the number of compact and intuitive computing devices, such as tablets and smartphones with touchscreen [1], [2]. The integration of these devices into smart spaces, which are physical spaces permeated and coordinated by a responsive computational infrastructure that enhances users ability to interact with others seamlessy in this environment [3], [4], enables building application scenarios which were previously only glimpsed by the ubiquitous computing vision of Mark Weiser [5], [6]. Such advances enabled the materialization of these scenarios within a particular ubiquitous computing perspective, for instance, mobility, context or content ubiquity.

Content level ubiquity envisions the possibility of scenarios that transport content, relevant users' data abstractions, seamlessly trough the smart space. In this ubiquitous computing aspect, contents are first-class entities that are exchanged using the smart space infrastructure serving as a medium to integrate users in the environment. This concept is strongly linked to sharing and can be used to "extend" existing sharing-based applications contents, such as collaborative and interactive applications (public displays), into the smart spaces. For example, Figure 1 portraits a ubiquitous collaborative IDE that uses content sharing to enable collaborative programming using gestures. In this application, users can send and synchronize source code files with each other to build their software.

Several ubiquitous applications focused on content sharing were developed with the intention to explore content level ubiquity in smart spaces. The communication layer of these applications is built from scratch using raw techniques, such as socket and rpc, to express content sharing in the smart space. Because these low-level abstractions do not target this scenario [7], developers are exposed to several non-trivial problems, such as reference integrity, concurrency access, seamlessly mobility, when transporting contents in the smart space. For example, the application can fault or provide wrong answer due to inconsistent references left by a transferred content.

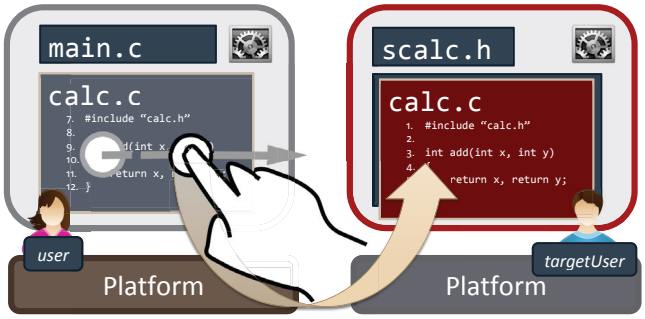

Fig. 1: Sharing a content (*calc.c*) using a move gesture.

Treatment of these problems by the programmer is costly and diverts his/her focus from the application domain. In fact, due to the complexity of these problems and lack of infrastructure platforms, most of the work in the area are repeatedly built from scratch using low-level abstractions and dealing only with parts of the problems mentioned [8]–[10]. Seeking to provide a more intuitive infrastructure for the development of this kind of application, we propose **C3S** (***C**ontent **S**haring for **S**mart **S**paces*), a middleware that offers a high-level programming primitives that are based on a set of content sharing manipulations in the smart space. These primitives express a set of behaviors, *move*, *clone*, and *mirror*, which serve as a building blocks for developers to implement sharing and content ubiquity features. The primitives also abstract completely the problems mentioned, reference integrity, concurrency access, and location transparency, that are involved in content sharing.

The paper is structured as follows. The next section explores the content sharing problem and expresses it in the form of feature that that platforms should provide to developers. Section III then presents our approach, which consists in the

978-1-4673-5077-8/13/$31.00 ©2013 IEEE

generalization of content sharing semantics, such as move, clone, and mirror, into primitives. Section IV describes a few important C3S structural concepts that describes how can applications and other services be partitioned in the smart space to provide support for the primitives. Section V shows how these concepts and primitives are implemented by the components that form the C3S architecture. Section VI then gives a brief description of some implementation issues, while Section VIII discusses related work. Section VII uses an application as an use case to evaluate the middleware constructions. Finally, Section IX concludes the paper by highlighting the results and contributions, and discussing future work.

II. Content Sharing

Low-level raw techniques mentioned before are used to build content sharing in smart spaces, *i.e.*, forms to exchange and synchronize content with other users in this environment. Each one of these forms represents a different sharing semantic, a specific way to express an interactive sharing operation. For example, a move semantic would migrate content from a user to another, while a clone semantic would copy and retain the content from the source user. Each specific sharing semantic that the application uses needs to be coded manually in those low-level raw techniques, making hard to use multiple sharing semantics since each one needs to be separately implemented. Also, not only these low-level techniques provide a poor abstraction level, they were not made with content sharing or ubiquitous computing in mind thus they do not address problems that appears in these domains. The rest of this section lists and explain briefly some problems that appear recurrently in this environment in these operations.

Content items often reference other contents in order to express a relationship or to construct complex structures, such as composition or aggregation, contents can reference others to build complex structures. Some sharing operations, such as moving contents between users, can leave the application with inconsistent references after concluding the operation. This problem can cause the application to fail by either crashing (null pointers) or returning wrong results (content is not deleted). Figure 2 shows this problem in the move operation described in the introduction.

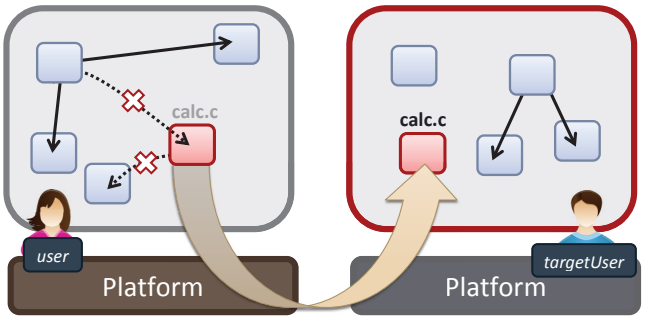

Fig. 2: Integrity issues that can happen when contents moves.

Collaborative applications typically use users' location abstraction to execute sharing operations. However, in smart space environments, there can be both user and device mobility, *i.e.*, not only users can enter and leave the space spontaneously but they can change devices too. This means that in order to abstract a user address we need to know his current location and active device. This is also a non-trivial to developers given the fact that developers will need to track and manage all the users that interact with the environment.

Middleware platforms that provide abstractions for building sharing-based ubiquitous applications mainly offer a programming for smart spaces. However, it is desired that this model can also abstract some of the environment problems mentioned, such as managing references, and abstracting user location and active device. C3S provides three high-level primitives, move, clone, and mirror, which not only offer an intuitive programming model using the sharing semantics approach but also address and turns these problems transparent to developers. These primitives express a set of functions sharing behaviors and serve as a building block for the developer to build sharing-based ubiquitous applications. By using and combining these primitives, developers can focus on their application domain rather than in sharing problems. For instance, in the introduction scenario the developer would execute the *move*(*calc.c, targetUser*) primitive to share the *calc.c* content with *targetUser*. C3S would automatically extend remotely the content references, as we can see Figure 3.

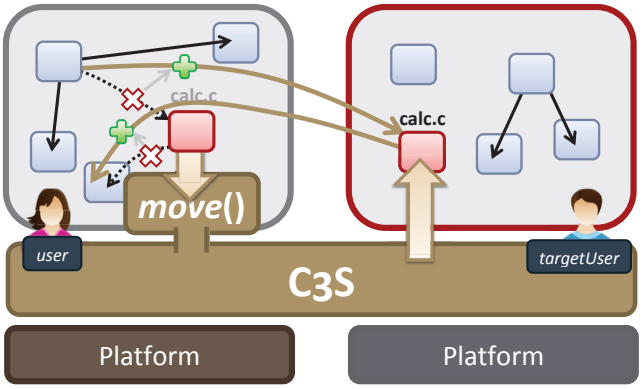

Fig. 3: Using the C3S move primitive to share a content.

III. Primitives

We analyzed content-level ubiquitous applications in smarts spaces, both in literature and with prototypes, to identify common sharing semantics. This exploration discovered an intersection with three content sharing semantics, *move*, *clone*, and *mirror*, which were the primary way to express content sharing in this environment. These semantics were mapped directly into C3S middleware primitives, see Figure 4. Developers can combine these primitives and create new sharing semantics, for instance, a trade semantic using move primitives, or use them as construct block for a high-level communication layer.

The move primitive migrate contents between application instances. When moving content from one user to another, it ceases to exist in the origin and is created on the target user instance. After the execution of this primitive, references that point to the moved content becomes inconsistent, since the content is no longer present. These references can either

be extend the reference to the new (remote) content location or delete completely depending on how the application or infrastructure address the issue. Both solutions are non-trivial [11] and requires local and remote reference management, including tractability and transparency aspects.

The clone primitive copies content between application instances. From developers and users point of view it is an interesting operation because, when copying an item of content, it is not eliminated at the source, thereby allowing individual development by another user (in his/her application instance). Since cloned contents are not removed in the origin, there is no need to treat invalid reference issues.

Primitives described so far do not have a continuous semantic, *i.e.*, after the operation, there is no connection between the source and target content. This semantic is very common on shared screen application such as Classroom Presenter [8], where digital ink is shared and synchronized across students and teacher. The mirror primitive materializes this behavior by coping the content and establishing synchronization between the original and its new copy. After the operation, the content exists in both instances, in a way, that changes made in any of them are reflected on copies. Synchronization and keeping these contents consistent is a hard task due to the volatility of users in smart space, programmers needs to design temporary transaction and locks scheme, both are not trivial.

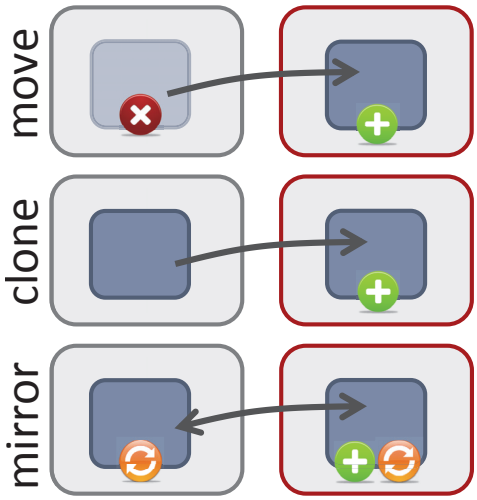

Fig. 4: The primitives semantic identified and provided by C3S.

IV. C3S Concepts

To express the C3S primitives, *move*, *clone*, and *mirror*, we defined some basic concepts about applications and services in smart spaces. These auxiliary concepts describe how the application can be partioned in this environment and the structure and behaviour of the sharing unit (*ubiquitous content*).

A. Application, Activation, and Ubiquitous Application

We consider an application as the combination of its executable and metadata. In the metadata, there are fields for application name, version, and platform support for each executable. The application is first installed on a server and then is made available to users in the smart space. We call each instance of the application as activation, and we define the ubiquitous application in C3S as a set of activations, *i.e.*, it is formed by the distributed instances of the application. The ubiquitous application activations can be restored or paused, allowing collaboration to be maintained into several sessions. Furthermore, it should be noted that the C3S can run several ubiquitous applications simultaneously.

B. Ubiquitous Content

These applications share content, so it is necessary to define a structure, generic and opaque, that the C3S middleware can handle. We explored pre-existing definitions, such as share object [12]–[15], and mobile object [16]–[18], however, they lacked features of sharing, such as migration, synchronization, and ubiquity, that these kind of application requires.

To fill this gap, we defined ubiquitous content as: the ubiquitous sharing unit in smart spaces. The term content is used to emphasize that this is an application abstraction and that it is implementation-independent, and thus can have different structures such as objects, modules or tuples. The ubiquitous part of the content is related to their ability to migrate between users and exists in multiple locations. Furthermore, contents may continue to exist after the activation ends, thereby enabling them to be reused by other activations or keep sharing when a user leave.

The structure of a ubiquitous content follows a standard format that allows their manipulation in smart spaces. Its creation is made at runtime in an activation, which binds the content to the running activation and a given external representation. The content internal structure can have multiple of these external representations, for instance, one for tablets and other for desktops. In C3S, the basic format of a ubiquitous content is defined by extending the abstract class *UbiquitousContent*. This class defines metadata, such as ID, activation, owner and synchronization status, which are used by the sharing primitives. When a developers instantiate an ubiquitous content, using the *createUbiContent()* function, it creates and returns an ubiquitous reference. This reference is a proxy that acts as a membrane that controls the content consistency and intercepts method calls to synchronized copies, both concepts will be explained further in Section V.

A ubiquitous content may also reference others to build complex structures or simple relationship, this is done using the link concept. A link is a connector that C3S uses to attach two pieces of content together. C3S provide default behavior for links when it is inconsistent, for instance when the content moves, they are break or extend the link remotely. However, developers can also define their own behavior specifying a *connect()* and *disconnect()* method which are called respectively when establishing or canceling links. When developers want to link two contents, for instance X to Y, they execute the *X.link(Y, behavior)* function, which calls the *X.connect()* operation (default behaviors) or the developer custom method. Links can be stored in the content and developers grab their endpoints using the *getTarget()* method, in the sample case it returns the ubiquitous reference of Y. Links relationship, *e.g.*, $X \rightarrow Y$, are registered in a C3S graph, which are used later to manage references, *e.g.*, when they become inconsistente.

V. Architecture

The above described primitives are implemented by a series of components that together collaborate to form the C3S architecture. It is composed mainly by users' peer-to-peer interactions in the sharing process, however, it also has a client-server style to use and explore smart space services. The smart space provides naming, storage, and concurrency control services, for example, user devices can query the list of current running ubiquitous application or discovers present users in the smart space. These services are provided by a C3S Server instance that runs in the smart space infrastructure which is exposed to users' applications using well-known identifiers, such as package-data type and port number.

C3S components are located on top of the user runtime platform and a communication middleware, as exposed in Figure 5. The communication middleware serve as a message bus between C3S instances (peers and server) which abstract well-known problems, such as user platform, different communication models, that serve as low-level building block of our primitives. Primitives are implemented separately through a set of interactions between C3S components thus allowing to use various communication infrastructure, including, tuple space, distributed objects, and message-oriented middleware. Overall managers perform the following tasks:

- **Content Management**: Provides an interface for ubiquitous content manipulation by activations, which includes methods to create, delete, share and link these contents. Uses a direct graph structure to manage content relationships and a local storage scheme to store and track activations contents.
- **User Management**: Manages and registers users' entrance and exit in the environment. Provides conventional methods, *i.e.*, register, remove, login, list users, to discover and explore user services in the space. This component is also responsible in abstracting the user name to his/her current location and active device.
- **Application Management**: Deals with management aspects of applications. It allows to create, list, join, and delete ubiquitous applications in the smart space.

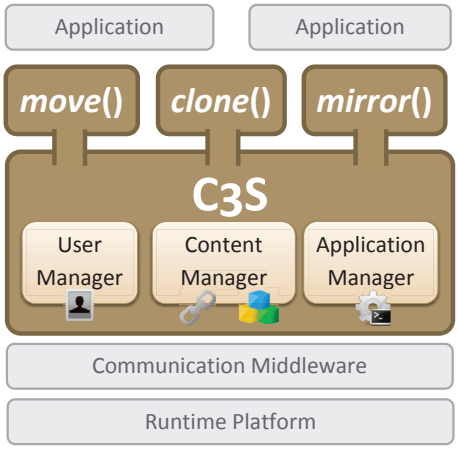

Fig. 5: An overview of C3S architecture.

A. Entering and Leaving the Smart Space

User enters the C3S smart space executing a well-known *login* service provide by the Server instance in this environment. The Server can uses different back-ends, *e.g.*, Google, OpenID, Facebook, or its own infrastructure to validate this operation. If the operation is successfully, the user is registered or updated with an abstract username pointing to his/her current location and active device.

Users might be carrying multiple devices and just spontaneously change between them or other devices presented in the environment, to keep the primitives location-independent we need to address this issue. Whenever users interact with an activation ubiquitous content, the proxy creates and sends signals to user manager. These signals serve as heartbeat messages and allow the manager to update the username abstraction in the smart space. Developers, can also manually specify the current device using a *setcurrentdevice()* function.

B. Move and Clone

The move and clone primitives are similar since the both do not establish a continuous sharing binding between the source content and its destination after the operation. Cloning can be seen as a "light" move operations, since it doesn't remove contents from the source location and don't need to deal with inconsistent references, thus we will mainly describe the move primitive. The steps involved in this process are:

1) Activation initiates the sharing process executing the *move(content, targetUser)* primitive;
2) User manager translate the *targetUser* name to its low-level address in the communication middleware;
3) Content manager takes the content structure from the storage, transform it on an external data representation with metadata and send to the address provided;
4) The target C3S instances receives the communication middleware message, which contain the serialized content and activation metadata. Using this external data representation the content is rebuilt and added to the activation content storage.
5) Using the message metadata, content manager creates an event and signalizes to the activation the receiving content and the user who sent.
6) After the content is delivered, the source content manager removes it from its activation storage. To deal with possible inconsistent references the manager checks for violations in the relationship graph for links behaviors that need to be addressed.
7) The current reference is linked to a *keep* behavior, the content proxy will redirect future calls to the remote copy using the subjacent communication middleware.

C. Mirror

Mirror primitives, in turn, provide a continuous effect in content sharing. The first part of the operation is identical to copying, however, after doing that it adds the new copy address to a sync queue in the Server. Interactions with synchronized ubiquitous content are intercepted by C3S, using the proxies, and are broadcasted to the other replicas. The broadcast message contains metadata about the intercepted function, such as arguments and function name, which are locally executed in

each replica. To control concurrency C3S uses the Mediated Updated pattern [19] which minimize update loads using a central instance (Server) that dispatches updates. Whenever a synchronized content changes, *e.g.*, through a function, the proxy sends the changeset to the central instance which locks the contents and propagate the update in the smart space.

VI. Implementation Issues

When ubiquitous contents are instantiated, proxies are dynamically created using reflection to manage the content structure as described in the Active Object pattern [20]. This pattern decouples the method execution from its invocation and allows the C3S middleware to intercept and control the content consistency. During the primitive execution, the content can receive further method calls from other threads. These methods are transformed into requests metadata and are stored in a buffer that is sent and executed on the destination location after transferring the content.

C3S combines the Active Object [20] with the Mediated Updates [19] pattern to manage the consistency throughout mirrored content in the smart space. While the first pattern can lock contents the second dispatches updates and control these locks. This provides full transaction support, however it slows down other replicas in high interactive applications given the fact they are blocked when propagating changes.

VII. Case Study

The Secret Words application was designed to validate the C3S architecture and primitives in a smart space. It is intended for children undergoing literacy and employs a simple methodology that combines images with words. Secret words are distributed among the students and they have to hit the correct letters to reveal the word. These words can be shared between students, for example, if a student knows one letter, you can move to him/her. Figures 6 and 7 shows the screen of the application and its execution in tablets respectively.

Hidden words represent the sharing unit of the application and were modeled as ubiquitous content. Among its attributes and operation it can list the string that contains the original word, inset letter, and get hints. The entire application communication layer to exchange words was mapped to C3S primitives. However, we identify primitives not supported by C3S, particularly, the merge semantic. For example, when a student clones a secret word to another, it is interesting that this student could apply modifications developed by colleague in the original copy.

Fig. 6: Secret Words screenshot.

VIII. Related Work

There are few specific middleware platforms for collaborative applications in smart spaces. However, there are some frameworks and middleware that meet some of the requirements of these applications. This section compares them with C3S using the abstractions that they provide to deal with content-level ubiquitous applications identified in Section II. Table I summarizes this comparison.

A. Sharing Semantics

A sharing semantic represents a way to share content, for instance, moving, cloning, and mirroring. This requirement expresses the content sharing semantics that platforms provide to developers. We are not aware of work that provides semantics other than the ones presented in C3S.

DOLCLAN [12], DreamObject [13], and Agilo [15] are middleware and frameworks developed by CSCW researches, which happens to all focus on syncing (mirroring) semantics in stable scenarios. Mica [14] provides a blackboard approach using tuple space that presents only a manual mirror semantic through reading and writing. AOM [16] and ProActive [18] use distributed objects middleware such as CORBA to provide a service that resembles move and clone. However, both requires that developers manually program parts of the operation. Finally, NetMorph [17] presents an interesting and different migration approach than the others. It divides the smart space in 2D coordinates and content are moved to a position (x, y) rather than an user. The related works usually provide only one semantic or lower-level abstraction compared to C3S.

B. Sharing Implementation

Content sharing can be realized through procedural primitives or content built-in types. In the first case, sharing is done with the use of functions expressing the different sharing semantics, while in the second case the semantic is embedded in the content type. Primitives are usally defined using functions, thus, they can be classified as dynamic, since it can choose at runtime the function (or sharing senatic) desired. With data typing, the semantis is static and integrated on the content type, which means, that any created conent can only be shared with the pre-defined type semantic.

CSCW work (DOLCLAN, DreamObject, Mica, and Agilo) use built-in types. The problem with built-in types is that it restricts the contents to a given semantic, and thus it can only be shared using that predefined form. The other related works and C3S uses the approach based on primitives, which

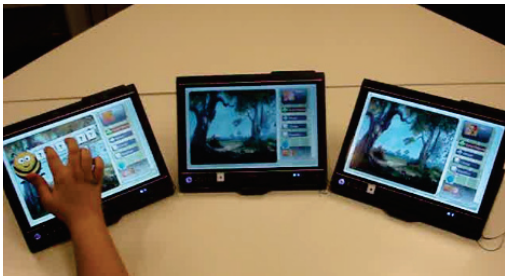

Fig. 7: Tablets executing the Secret Words application.

Middleware Platform	Sharing Semantic	Sharing Implementation	Referencial Integrity
DOLCLAN	Mirror	Built-in	None
DreamObject	Mirror	Built-in	None
Mica	Mirror	Built-in	None
Agilo	Mirror	Built-in	None
NetMorph	Move	Primitives	None
ProActive	Move, Clone	Primitives	Auto
AOM	Move, Clone	Primitives	Auto
C3S	Move, Clone, Mirror	Primitives	Manually, Auto

TABLE I: Comparison of abstractions that content sharing platforms provides.

allow higher abstraction to developers, given the fact that the semantics is defined in the primitive, which means, that the same content can be shared in different ways.

C. Referential Integrity

Referential integrity is an interesting problem that only some platforms try to solve. Since DOLCLAN, DreamObject, Mica, and Agilo do not have the move and clone semantic sit is quite expectable that they do not provide solutions to this problem. AOM and ProActive use a narrow copy during these operations and they fix these references using an lazy approach, *i.e.*, when the reference miss they discover and redirect to its new location. In C3S we provide the delete and keep behaviors or allow developers to specify custom functions to operate when the problem happens.

IX. Conclusion

C3S is a middleware platform that provides a high-level programming model base on the sharing semantics and primitives approach to build content-level ubiquitous application in smart spaces. These primitives abstract recurrent problems encountered in sharing applications, such as referential integrity and transaction support. The primitives include sharing semantics of move, clone, and mirror. Move transfer and delete the content from the origin to the target user. Clone copies the content and mirror provides a synchronous content replication.

The C3S architecture and primitives were built on top of a generic communication middleware. This allows the implementation to be flexible and agnostic to the subjacent low-level communication bus. It constructs the sharing semantics on top of this communication service. As future work we intend to explore the middleware in a larger scale than a closed smart space. Furthermore, we want to extend these primitives to explore and support other sharing semantic in the smart space, such as *trade*, *merge* and *peak*.

References

[1] G. Goetz, "Apple's inevitable path to a post=pc era," http://gigaom.com/apple/apples-inevitable-path-to-a-post-pc-era/, 2012.

[2] M. Lopez, "Four ways the post-pc era differs from today," http://www.forbes.com/sites/maribellopez/2012/05/01/four-ways-the-post-pc-era-differs-from-today, 2012.

[3] G. Coulouris, J. Dollimore, T. Kindberg, and G. Blair, *Distributed Systems: Concepts and Design*, 5th ed. USA: Addison-Wesley Publishing Company, 2011.

[4] M. Roman, C. Hess, R. Cerqueira, A. Ranganathan, R. Campbell, and K. Nahrstedt, "A middleware infrastructure for active spaces," *IEEE Pervasive Computing*, vol. 1, no. 4, pp. 74–83, Oct. 2002.

[5] A. Schmidt, "Ubiquitous Computing: Are We There Yet?" *Computer*, vol. 43, no. 2, pp. 95–97, Feb. 2010.

[6] M. Weiser, "The computer for the 21st century," *SIGMOBILE Mob. Comput. Commun. Rev.*, vol. 3, no. 3, pp. 3–11, 1999.

[7] P. Byrne, "MUSE - Platform For Mobile Computer Supported Collaborative Learning," Ph.D. dissertation, University of Dublin, 2011.

[8] R. R. Anderson, P. Davis, N. Linnell, C. Prince, V. Razmo, F. Videon, and V. Razmov, "Classroom Presenter: Enhancing Interactive Education with Digital Ink," *Computer*, vol. 40, no. 9, pp. 56–61, Sep. 2007.

[9] J. P. Hourcade, B. B. Bederson, and A. Druin, "Building KidPad: an application for children's collaborative storytelling," *Software: Practice and Experience*, vol. 34, no. 9, pp. 895–914, Jul. 2004.

[10] J. Steimle, O. Brdiczka, and M. Muhlhauser, "CoScribe: Integrating Paper and Digital Documents for Collaborative Knowledge Work," *IEEE Transactions on Learning Technologies*, vol. 2, no. 3, pp. 174–188, Jul. 2009.

[11] A. Fuggetta, G. Picco, and G. Vigna, "Understanding code mobility," *IEEE Transactions on Software Engineering*, vol. 24, no. 5, pp. 342–361, May 1998.

[12] J. Bardram and M. Mogensen, "DOLCLAN: middleware support for peer-to-peer distributed shared objects," in *Proceedings of the 7th IFIP WG 6.1 international conference on Distributed applications and interoperable systems*. Springer-Verlag, 2007, pp. 119–132.

[13] S. Lukosch, "Transparent and Flexible Data Sharing for Synchronous Groupware," Ph.D. dissertation, University of Hagen, 2003.

[14] M. W. Kadous and C. Sammut, "MICA: pervasive middleware for learning, sharing and talking," in *Pervasive Computing and Communications Workshops, 2004. Proceedings of the Second IEEE Annual Conference on*, 2004, pp. 176–180.

[15] A. Guicking, P. Tandler, and P. Avgeriou, "Agilo: a highly flexible groupware framework," in *Proceedings of the 11th international conference on Groupware: design, Implementation, and Use*, ser. CRIWG'05. Berlin, Heidelberg: Springer-Verlag, 2005, pp. 49–56.

[16] R. Kapitza, H. Schmidt, G. Söldner, and F. Hauck, "A Framework for Adaptive Mobile Objects in Heterogeneous Environments," in *On the Move to Meaningful Internet Systems 2006: CoopIS, DOA, GADA, and ODBASE*, ser. Lecture Notes in Computer Science, R. Meersman and Z. Tari, Eds. Springer Berlin, 2006, vol. 4276, pp. 1739–1756.

[17] M. Umezawa, K. Abe, S. Nishihara, and T. Kurihara, "NetMorph - an intuitive mobile object system," in *Creating, Connecting and Collaborating Through Computing, 2003. C5 2003. Proceedings. First Conference on*, 2003, pp. 32–39.

[18] B. Xu, W. Lian, and Q. Gao, "Migration of active objects in proactive," *Information and Software Technology*, vol. 45, no. 9, pp. 611–618, 2003.

[19] S. Lukosch and T. Schummer, "Patterns for Managing Shared Objects in Groupware Systems," in *Proceedings of the Ninth European Conference on Pattern Languages of Programs (EuroPLoP'04)*, K. Marquardt and D. Schutz, Eds. Konstanz: UVK Universitätsverlag, 2005, pp. 333–378.

[20] R. G. Lavender and D. C. Schmidt, "Pattern Languages of Program Design 2," in *Pattern Languages of Program Design 2*, J. M. Vlissides, J. O. Coplien, and N. L. Kerth, Eds. Boston, MA, USA: Addison-Wesley Publishing Co., Inc., 1996, ch. Active Obj, pp. 483–499.