Tearing down the walls: towards an activity-centered applicationless desktop interface

Jeroen Jillissen
Media Technology MSc Programme, Leiden University, The Netherlands
Graduation Thesis, supervised by Maarten H Lamers
August 2010

Abstract

Although computer applications have given us possibilities beyond our imagination, they are the same applications which can cause much confusion when interacting with computers. Would it be possible to create an applicationless desktop interface, which provides the user with the possibilities to reach their desired computing goals? We propose a conceptual framework containing the principles of an activity-centered applicationless desktop interface. This framework was used to create a prototype to see, experience and understand the possible issues and complications of using and implementing an activity-centered applicationless interface. Conclusions are drawn and presented in this paper.

1. Introduction

With increasing power and capabilities, the computer has become a prominent tool in our daily lives. Many activities nowadays involve a computer, ranging from everyday activities like enjoying entertainment and communicating with friends to work related activities like writing reports or doing calculations. Computers are part of our daily lives and it is hard to imagine our lives without them anymore.
Although we have come a long way many people seem to struggle with the use of computers. Almost everybody has the experience of getting strangled in the web of uncountable options, functions and windows that applications have to offer, or knows at least some other people 'just not getting it'.

Looking at the way computers work it could be said that the current personal computer technology is based on an 'application-centric' model. This model puts the emphasis on applications which have their specific functionalities and support the creation and manipulation of data, like an image editor to manipulate images.
Although the application-centric model has been introduced before the creation of the graphical user interface (GUI) more than 30 years ago, it was the GUI which popularized the application-centric model we know today. In those 30 years there has been relatively little change in the way the GUI works and thereby little change in the application-centric model. However the way the computer is used has changed tremendously in the same period. Therefore one could ask if the current application-centric model is a model that is still usable if users seem to struggle with the use of it after all these years. Shouldn't we look in other directions?

If one would want to change the application-centric model into something different one should first consider the main component of this model: the application. Looking at applications themselves, one could distinguish three major problems with applications in how they exist today.
Firstly, the goals of the user and the functions and tools offered by the applications do not always match. For example: a user might want to remove red eyes from a photo, however most photo manipulation applications do not offer a function in the line of 'remove red eyes'. What they do offer are functions and tools like a 'selection tool' and a 'fill tool', that might enable the user to fulfill the desired goal and thus these functions and tools only provide a possible partial solution. As a result the user must split a goal into several subgoals (which can be fulfilled by completing one or more *subtasks*), which should be accomplished separately using the provided functions and tools.
Secondly, applications can be seen as 'walled-gardens' or 'silos'. Applications are closed environments which contain a specific set of functions and tools that (in general) cannot be used in other applications, even while it might be logical for the user to do so. For example, it is not possible to use the spellchecker from the word processor in the image editor, which allows the user to add text to an image but does not offer a spellchecker itself. Furthermore, each application comes with its own standards resulting in inflexibility. An example of such standards are file standards: a Photoshop PSD file cannot be opened in a different image editor and a Word 2007 Docx-file cannot be opened in Word 2004.
Thirdly, every application is different and works differently. While this might sound logical, it is a serious problem. This problem basically forces the user to adapt

to every application he uses, since every application will have (slightly) different interfaces and might have tools and functions positioned at different locaitons. Even applications which have the same purpose, like for example email clients, might work differently making activities such as 'sending an email' unnecessary difficult for a user when performing this activity using a different application then he is used to.

So if there are problems with the applications in their current form and the current 'application-centric' model is not working for everybody, one could ask the following question: '*Would it be possible to create an interface that supports the desired activities and reach the desired goals of the user without the need of applications?'*
This paper explores the possibility of offering the user an interface that matches and has a strong focus on their activities and overall goals without the need of applications as we know them today. First, however, we discuss related studies and projects. Then a framework is proposed for an activity-centered applicationless desktop interface, which is then explored in the form of a prototype. The final section reflects on the goals set out.

2. Related work

Activity-centered thinking is not a new phenomenon. This way of thinking, based on Leon'tev's activity theory [1], is something that has been applied for many years in the field of human-computer interaction. It provides a framework where a task or activity can be broken down in actions, which can be divided into operations. This helps designers and developers to understand what steps are necessary for a user to successfully perform a task or activity. Most of the time this results into regular applications and thus the application-centric computer landscape we know today, where applications contain tools that help users perform operations which make up actions that make up the final activity.

However, research was done into systems which support an activity-centered design approach and look into how to approach human-computer interaction on high-level activities instead of breaking it down to low-level operations. This research can be divided in two main categories.
The first category represents research which looks into the use of overarching systems, which strive to support users on an activity level by looking for solutions in terms of task management. Bardram *et al* present their ABC (Activity Based Computing) system [2][3] which replaces the Microsoft Windows Taskbar with an activity bar. This lets the user manually create activities with corresponding applications and documents, in order to let the user quickly switch between activities gathering the applications and documents needed to complete the chosen activity. The system is an extension to the operating system which integrates support for activities in existing applications. In similar vein research has been done in this direction amongst which are Giornata [4], offering an activity-based computing system focused on knowledge workers; the Scalable Fabric System [5], grouping relevant windows together to support the user with task execution; Task Gallery [6], employing window management in 3D space; the GroupBar [7], which allows users to group windows in higher level tasks; and Elastic Windows system [8], allowing users to hierarchically organize windows in order to support tasks.
Additionally research has been done into systems which monitor the user's behaviour and activity data in order to automatically create task-context which helps the user to (re)perform the activities in the future. Examples of research in this direction are TaskTracer [9] and the UMEA (User-Monitoring Environment for Activities) system [10].

Although work in this category has a lot of resemblance with our work, namely the focus on activity-centered design for desktop interfaces instead of application-centered design, our work is distinct in the following ways.
Firstly, our research focuses on rethinking and totally replacing applications, and thereby the current application-centric way of working, by an activity-centered way of working. The works mentioned above offers extensions which can be incorporated into or onto an existing operating system. This keeps the 'traditional' computing paradigm in tact and merely adds an extra layer to manage the user's activities.
Secondly, research in this category does not specifically focus on improving the use of the computer by using activity-centered computing as a goal. Instead it has its main focus on different research goals like handling multiple parallel and mobile work activities [2] or supporting knowledge workers with multitasking and collaboration [4], where activity-based computing is a method to reach these goals. In our research we focus on activity-centered computing as the ultimate goal, where improvements in other areas are nice to have.

The second category represents research into creating systems that gather and group sets of information, objects and/or tools from other applications, placing them in a centralized location or application, and allowing the user to use these sets to complete an activity. An example is Activities [11], a centralized web-based service that allows the user to create an activity and gather all related objects and information (e.g. files, emails and contact information) in order to complete the activity. The system

itself does not support technology to perform the desired activity, but rather reflects what has to be done and groups the information to do so. Similar research includes Activity Explorer [12], a system that integrates objects and tools around activities and brings them together in a dedicated application, and Task Master [13], a system that transforms the email client as a task management tool and embeds relevant objects directly into the client.

Compared to this work our study is distinct in the fact that the work in this group focuses on creating new tools that are applications or services in itself which should help to manage tasks and activities performed with already existing applications. Like the research done in the first category, this approach keeps the 'traditional' paradigm and application-centered way of working intact.

Apart from the mentioned research there are several commercial products, like Enso (Humanized inc.), Quicksilver (Blacktree software) and Automator (Apple inc.), which enable the user to perform some form of 'activities'. However none of these products focus on an activity-centered way of working and are merely an application/solution on top of already existing applications.

Finally there is the Lightful.org [14] project, a project that looks into "*the next generation of personal computer interfaces"* (quoted from [17]). A small part of the initial ideas for Lightful.org are based around looking differently towards how applications should be used and how users interact with the computer. Since Lightful.org is in a very preliminary stage (and at this point appears to be nothing more than several lines in a sketchbook) nothing can be said about whether this project truly implements an activity-based way of working.

3. Proposed Framework

The proposed framework is based on the idea to help the user reach the desired goal as easily as possible by the means of an activity, however without the need of today's applications. The framework consists of eight principles which are derived from the following rule: the *user* has a certain *goal* which can be reached by changing (attributes of) an *object* by means of a *valid activity*, which is based on the *object* and/or *context* (schematically shown in Figure 1).

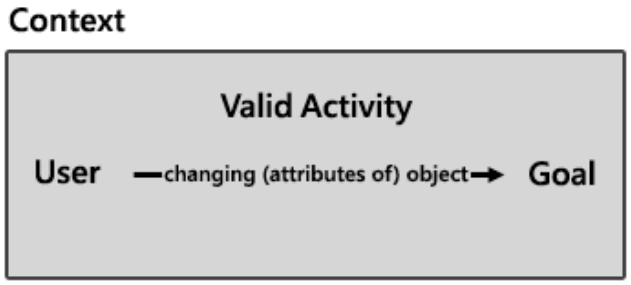

Figure 1: schematic representation of our basic premise.

We will illustrate this rule with an example. Let's assume the user wants to '*remove red eyes from a photo'*. The first and second aspect which should be taken into account are the user and his goal. These are straightforward. The user is the one who is in control and who determines the goal to be reached, in this case a photo without red eyes.
The third aspect is the object which must be altered in order to reach the goal, in our example this is the photo. Please note that in this example the photo itself will be altered, but in other situations it could be the case that the attributes of the photo will be changed, such as the position of the photo.
The fourth aspect is the activity. An activity is the process needed to change an object in order to reach the goal formulated by the user. In this example the desired activity would be *'removing the red eyes'*. However if we think in terms of a system, it is impossible to know what the goal of the user is without them explicitly telling it to the system and therefore it is impossible to present the user with exactly this activity. That is why a system has to present the user with activities that are 'valid' for the object and the user has to pick the activity that enables him to reach his goal. In this example valid activities could be *'send photo to friend'*, *'remove red eyes from photo'* or *'change brightness of photo'*, while invalid activities are activities that have nothing to do with the subject/object, for example: *'play song'* or *'spellcheck document'*.
The fifth and final aspect that should be taken into account is the context in which the activity takes place. It would be logical to present the user with the activity *'remove red eyes from photo'* if there is actually a person with red eyes in the photo. However, if the photo contains a beautiful landscape without any persons, it would be unnecessary to present this activity to the user thus making it an invalid activity for the landscape photo.

The rule explained above is translated in the following eight principles which make up the framework:

Open environment / unified workspace
All activities take place in a single unified workspace. There will be no applications which provide the user with tools, functions and windows to present data. Instead this will be covered by the next principles.

Make use of objects
In the framework we introduce the use of objects. Objects replace the files we know today. They are representations of the actual content the user works with and are placed in the unified workspace.
Examples of objects could be a song, a text or a photo. In the current situation a photo is a file somewhere on your computer, for example a TIF or a JPG file. The file must be opened in order to see the content, in this case the actual photo. In the new situation the photo is just a photo and it is sitting on the desktop ready to be viewed or used.
The use of objects allows for more natural activities than a file does, since a lot of activities we do happen to refer to the content itself instead of the file containing the content. If you want to send this photo to a friend it is more logical to say *'Hey, I'm sending you this great photo'* instead of *'Hey, I'm sending you this image file (which happens to contain a great photo)'*.

Next to the content, an object should contain information about itself. Although this might not be directly visible, this information offers the user and the system information which is necessary to properly interact with the object. Examples of this information could be information about the type of object which is important for the system so the system knows what kind of object it is dealing with, plus information about the photo itself like the date on which the photo has been taken and possible custom added information like the location, the subject or the persons in the photo.
The use of this extra information will be explained in the next principles.

Functions based on activities
Based on the rule mentioned earlier, the user has to perform a valid activity to reach the desired goal. In order to do so a system based on the framework should offer the user 'activity-based functions'.

Activity-based functions are functions which become available once a user selects an object and which are specifically designed to perform an activity on the selected object in order to reach the desired goal.
For example, when a user selects a photo, the system should display all functions which allow the user to do 'something' with the photo. This could be anything along the lines of *'email this photo'*, *'remove red eyes from photo'* or *'change the contrast of the photo'*.
By doing so, the activity-based functions replace functionalities which are normally offered by applications.

Ideally, the functions themselves should deal with everything that is needed in order to perform the activity for the user, with as little user input as possible. When a user selects a function, the function should do most of the work. For example if the user selects *'remove red eyes from photo'*, the function should automatically detect the red eyes in the photo and remove them for the user. It is an illusion to think that every function can automatically perform the activity without any user intervention. Therefore functions can optionally be accompanied with a window containing GUI elements in order to guide the function, specify certain variables or give the user feedback about the performed activity. For example if the user selects a photo and clicks on a function named *'email this photo'*, the system may present the user with a window where the user can select a contact to whom the photo must be sent.

Context sensitive
A system based on the framework has to be context sensitive. As described earlier, activity-based functions can be executed on objects. However the user should only be able to execute activity-based functions on objects for which they are intended. For example when a photo is selected the system should only present photo related functions, like *'remove red eyes from photo'*, while for a text object the system should present text related functions, like *'spellcheck this text'*. By doing so the user is only presented with relevant options and not bothered with options which could not be executed.

Consistency
A system based on the framework has to be consistent. If the user is able to execute a function on a certain type of object, the user should always be able to execute the same function on the same type of object independent of their state or location. For example if the user is able to perform a certain function on a photo that is located on the desktop, the same function should be available to the user wether the photo is part of an email or presentation. By doing so the system presents the user with a consistent set of functions per object type, so the user always knows what to expect.

Persistency
A system based on the framework must be persistent. Without applications and files there is no possibility (and no need) to open objects. If you cannot open an object, you cannot close it either, nor would it make sense to save it. Based on earlier principles of the framework objects reside on the desktop and are 'just there'. When a user alters an object by using an activity-based function, the

object takes on its altered form and remains this way. The objects are persistent much like objects in the real world; when you add a piece of text on a sheet of paper it will stay this way.
With a persistent system and without the possibility to open, close and save (copies of) objects the user needs an alternative way to manage errors or undo certain actions. Therefore the system should be able to undo every activity the user performed on an object at all times.

Unified event system
Applications not only provide functions and tools which allow the user to fulfill their goals, they also provide information to the user in the like of events, like an email client which informs the user when a new email arrived. Without applications there can be no events, therefore there must be a unified event system which allows the functions to communicate with the user in a standard uniform way.
Moreover, there need to be functions which are not controlled directly by the user, but let the user know that certain objects might need their attention. An example is checking email. This is a function that cannot be activated from an object on the workspace. Rather it automatically checks for new email and notifies the user when a new email arrived and automatically has been placed in the workspace.

Unified view system
Last but not least, applications provide views. An application collects a set of information which helps the user to easily access the information in a central location. For example by starting the email client, the user will get an overview of all his emails, while an address book presents the user with a view of all his contacts. On top of this an application may let the user sort the information in particular views, for example alphabetically or chronologically.
Without applications there will be nothing more than a collection of objects on the workspace. Therefore a unified view system is needed to manage all the objects. The unified view system allows the user to filter objects based on the extra information they contain as specified at the objects principle, allowing the user to create custom views like we know them from nowadays applications. For example if the user filters for 'email' the unified view system will only display the email objects on the workspace. On top of this the system allows custom views such as 'email John' which shows all emails received from John or 'email last week' which shows all emails received last week, allowing the user to create even more flexible views than is possible with current applications.

4. Prototype

To verify our framework, a prototype was built in order to see, understand and experience how the different principles of the framework could be implemented and to see where certain problems may arise. This prototype was built to a level where it could accommodate this goal, resulting in a product that contains the complete framework. To emphasize, the prototype was built with focus on this goal only and not on building the 'ultimate' system which is ready to be used by actual users. Therefore the level of technical-, graphical- and interaction design applied are sufficient for the prototype, however would lack the level needed for an actual usable product. Furthermore, we stress that the prototype is just one of the many possible implementations and interpretations of the proposed framework.

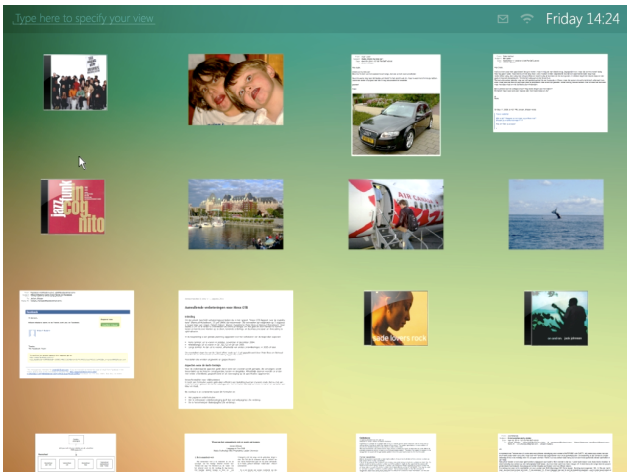

Figure 2: a screenshot of the prototype system.

Our prototype system, as shown in figure 2, contains a desktop on which everything takes place. The desktop remains the same and will never change, covering the *unified workspace* principle. The desktop contains 20 objects in three different categories (music, photo and text objects). The objects represent files as we know from current personal computers, such as a song, an email and a holiday photo. Every object can be selected and dragged around the desktop. This covers the first principle of the framework: *Make use of objects*.
When an object is selected a menu appears displaying the corresponding functions. At this moment the prototype contains twenty functions, five for each type of object and five for text selection. Of these functions four are actually functional in order to illustrate what functions and accompanying dialogs might look like and function. When a certain type of object is selected, for example a text, the menu shows the corresponding functions for the object type text. If a photo is selected, the menu will show functions that correspond to the photo object. The

functions based on activities and the *context sensitive* principles are hereby covered. Selecting a different photo will display the same functions. Even if the selected photo is part of a different object, for example a photo in a received email, the menu will display the functions corresponding to the photo object. However, if the text object that contains the photo is selected the menu will show the functions corresponding to the text object. This covers the *consistency* principle.
When an object has been altered, the changes will be saved automatically. In the prototype system the text of text objects, the brightness of photo objects and the position of the object on the desktop can be altered and are saved automatically, in accordance to the *persistency* principle.
The prototype contains an event module which allows functions or possible events to notify the user if necessary. At this point the event of checking email has been implemented in the prototype. For convenience of testing the prototype the event can be triggered with a single key press, showing a notification in the upper right corner containing a variable text and a reference to the related object. Clicking on the notification will take the user to the related object, in this example the newly received email, by bringing the object into focus so the user can interact with this object. This covers the *unified event system* principle.
Furthermore, the prototype contains a view system which lets the user navigate to, filter and select the desired objects. The view system allows the user to navigate the desktop as a whole, by dragging the desktop in order to reveal objects that might fall outside the screen. The system contains a zoom function to allow the user to get an overview of all objects, but also to allow the user to zoom in and focus on one object if desired. By double clicking on an object it will be brought into focus, bringing it to the center of the screen which allows the user to quickly select and use the desired object.
Finally, the view system contains a filter function which allows the user to filter objects by entering keywords into the filter which are compared to the information contained within objects. When a filter is applied, the filtered objects are moved to and displayed at the top of the screen while the other objects are dimmed so the desired objects can easily be selected. For example: if the user filters on the word 'photo' the system will display all the photos at the top of the screen while all other objects are dimmed. If the user would filter on 'photo holiday 2009' the system will again display photos, however this time only holiday photos from the year 2009 are shown. This covers the *unified view system* principle.

5. Results

As mentioned before the goal of building the prototype system was to gain insights into possible issues, flaws and problems that may arise when implementing the proposed framework. These insights are based on personal observations. No user tests were performed as the focus of this project is on a conceptual level resulting in a prototype which is not sufficient to perform user tests which lead to meaningful results.

We note that these issues are mentioned to learn from. This does not imply that these issues cannot be resolved using a proper or different implementation.

Focus on user experience
This might be very logical, but it is important to realize that the implementation of the framework not necessarily leads to a usable system. Although all the principles of the framework may be present within the prototype, it is the technical, functional and graphical design on a much more detailed level that probably makes the difference. For example the system might have the unified view system principle implemented, but if this implementation is clumsy and cumbersome the user might have a hard time reaching the objects he want to interact with, making the system suboptimal to use.

Scalability
The current prototype only contains twenty objects and already requires a proper functioning unified view system with its zoom tool and filter system in order to quickly and conveniently select the desired objects. Thinking about real life situations where a user does not have twenty objects but hundreds or even thousands of objects, it is apparent that this might lead to potential problems.

Apart from the issue of scaling up the number of objects, the same could be said about the number of functions or activities which are shown in the menu when a certain type of object is selected. While the prototype contains a couple of functions that illustrate how these would work together with the available object types, it is clear that if there would be functions for all desirable activities this quickly grows to an enormous amount of functions which cannot be easily displayed in a standard menu like we do in the prototype.

Translation of files into objects
In the current prototype files are fairly easily translated into objects since the objects used are fairly straightforward themselves. However this might be an area where problems could arise. One could question how

to properly translate files and content into objects thinking about issues like:

The template of an object type; how should an object be represented on the desktop so it is clear for the user what type of object it is and what to expect when it is clicked on. For example there could be two objects on the desktop which both appear as text objects, where one object is an email received from a friend and the other contains the contact information of a business contact. At this point both objects look more or less the same, but if the email looks like a text object and the contact information like a (custom) contact object, which might have its own look and feel, both objects might be better recognizable and distinguishable for the user.

The representation of the actual objects; how should objects be represented in the workspace so they correctly reflect attributes like the contents or size of the object. For example does it help the user to easily pick the lengthy email from his friend if the user knows he is looking for a lengthy email and the object is actually a tall object? But if we choose to represent objects in their correct scale how do we represent a scientific paper with a length of 50 pages?

The affordance of an object; since objects represented on the desktop are 'just there' it might be logical to directly interact with the object instead of using the activity-based functions. For example, it is logical to alter a text by placing the cursor in the text and start typing. However this kind of behavior is not directly clear for each type of object, for example should a song or movie start playing when it is being clicked?

Collections
Collections are everywhere. A book contains a collection of pages, while a page contains a collection of words. Your inbox is a collection of emails, your photo album is a collection of photos and a music album contains a collection of songs. The current prototype lacks a system for representing collections. Since every file is a separate object, your mailbox will be represented as hundreds of separate objects, your collection of photos from your most recent holiday as tens or hundreds of separate photos and your favorite music album as twelve separate songs. While this might be logical and correct from a framework perspective, it could be questioned if we should not introduce collections in order to preserve the already existing collections out there. Although collections can be accessed on the fly using the filter of the unified view system, by filtering on things like 'email' or 'photo holiday 2010', the introduction of collections could introduce more natural activities like 'play music album' on a music album or 'play slideshow' on a photo album.

The external world
Currently the framework focuses on what you want to do with the objects that are present on the computer. While this seems to be able to replace a fair share of nowadays applications, it is hard to replace any functionality based on activities you want to do on objects which are not physically present on your computer. For example, based on the framework a website might be a logical object. You can look at it, interact with it, and there would be corresponding activities like *'bookmark this website'* and *'send to a friend'*. However how would one view the website without having the website physically present on the computer? It seems impossible to present all websites on the internet as objects on the desktop.

On top of this there are applications that focus on getting information from the 'external world' to the user, which are very hard to translate into logical understandable objects. For example, a weather application that fetches information from different external sources and presents the weather to the user. How would this information be translated into a logical object that fits into the framework for the user to interact with?

6. Discussion

The goal of this research was to see if it would be possible to create an interface based on activities without applications as we know today. To do so, a framework was proposed which in turn was used to create a prototype in order to experience if an interface like this would be possible.

The current prototype implementation shows that it is possible to successfully create an activity-centered applicationless interface, offering the user activities which could be performed on objects, which are representations of the content, in order to reach the desired goal.
From the implementation questions and possible problems arose, ranging from questions on how to represent objects to how to deal with the 'external world'. Although these questions and problems are no deal breakers for the future of applicationless desktop interfaces, they have to be taken into account when creating a future implementation of a system based on the proposed framework.

Although an activity-centered applicationless interface has been successfully implemented, the current implementation is only a rough implementation developed to test the framework. This tells us that it could be done, but not about how usable it actually would be. Therefore it would be an interesting experiment to research what

content is used and what activities are performed by certain target groups on a daily basis and see if these objects and activities can be translated into objects and activity-based functions to find out to what degree a system like this could actually support these target groups in performing their daily activities.

Another interesting follow up study would be to look into how to solve the issues as mentioned in chapter 5 which surfaced by building the prototype. It is expected that most issues can be resolved fairly easy by the means of making some design and implementation decisions, however 'the external world', is something that might need more research on its own. This issue is something that stands out from the rest since it is fundamentally different from the other issues. The framework has a focus on activities based on content and in order to represent the content, it needs to be available on the computer. Since content from the external world, like websites, is not available on the computer, research must be done on how to make this content available within the principles of the framework.

A final suggestion would be to do some actual testing. The level of technical, graphical and interaction design of the current prototype is not sufficient for a regular user to use the current implementation. Therefore it would be an interesting experiment to implement a system which contains this level of detail in order to test the system with actual users. Doing so would give insight in issues such as the way people work with an applicationless interface, how people perceive an applicationsless interface and/or how an applicationless interface holds up compared to the interfaces we know today.

7. References

[1] Nardi, I. and Bonnie, A., Context and Consciousness: Activity Theory and Human-computer Interaction, MIT Press, 1996, Cambridge, USA

[2] Bardram, J. E., Bunde-Pedersen, J. and Soegaard M., Support for Activity-Based Computing in a Personal Computing Operating System, *Proceedings of ACM CHI 2006 Conference on Human Factors in Computing Systems*, April 2006, Montréal, Canada, pp 210 - 220

[3] Bardram, J. E., Activity-based computing: support for mobility and collaboration in ubiquitous computing, *Personal and Ubiquitous Computing Volume 9*, Issue 5, September 2005, pp 312 - 322

[4] Voida, S., Mynatt E. D. and Edwards, W. K., Re-framing the Desktop Interface Around the Activities of Knowledge Work, *Proceedings of the 21st Annual ACM Symposium on User Interface Software and Technology*, October 2008, Monterey, USA, pp 221 - 220

[5] Robertson, G., Horvitz, E., Czerwinski, M., Baudisch, P., Hutchings, D., Meyers, B., Robbins, D. and Smith, G. Scalable Fabric: Flexible Task Management, *Proceedings of the working conference on Advanced visual interfaces*, May 2004, Gallipoli, Italy, pp 85 - 89

[6] Robertson, G., van Dantzich, M., Robbins, D., Czerwinski, M., Hinckly, K., Risden, K., Thiel, D. and Gorokhovsky, V., The Task Gallery: A 3D Window Manager, *Proceedings of the ACM CHI 2000 Human Factors in Computing Systems Conference,* April 2000, pp 494 - 501

[7] Smith, G., Baudisch, P., Robertson, G., Czerwinski, M., Mayers, B., Robbins, D., and Andrews, D., Groupbar: The taskbar evolved, *Proceedings of Australasian Computer Human Interaction Conference, OZCHI 2003*, Brisbane, Australia

[8] Kandogan, E. and Shneiderman, B., Elastic Windows: Evaluation of Multi-Window Operations, *Proceedings of the SIGCHI conference on Human factors in computing systems*, 1997, Atlanta, USA, pp 250 - 257

[9] Dragunov, A. N., Dietterich, T. G., Johnsrude, K., McLaughlin, M., Li, L. and Herlocker, J. L., TaskTracer: A Desktop Environment to Support Multi-tasking Knowledge Workers, *Proceedings of the 10th international conference on Intelligent user interface*, January 2005, San Diego, USA, pp 75 - 82

[10] Kaptelinin, V., UMEA: Translating Interaction Histories into Project Contexts, *Proceedings of the SIGCHI conference on Human factors in computing systems,* April 2003, Ft. Lauderdale, USA, pp 353 - 361

[11] Moore, M., Estrada, M., Finley, T., Muller, M. and Geyer, W., Next Generation Activity-Centric Computing, *Presented as a demo at CSCW 2006,* November 2006, Alberta, Canada

[12] Geyer, W., Muller, M. J., Moore, M. T., Wilcox, E., Cheng, L. T., Brownholtz, B., Hill, C., and Millen, D. R., Activity Explorer: Activity-centric collaboration from research to product*, IBM Systems Journal Vol. 45*, No. 4, 2006, pp 713 - 738

[13] Bellotti, V., Duchenaut, N., Howard, M. and Smith, I., Taking email to task: The Design and Evaluation of a

Task Management Centered Email Tool, *Proceedings of the SIGCHI conference on Human factors in computing systems*, April 2003, Ft. Lauderdale, USA, pp 345 - 352

[14] Project *lightful.org*, online resource (accessed August 2010), www.lightful.org