OVERCAST SKIES
What Cloud Computing Should Be?

Mark Wallis, Frans Henskens and Michael Hannaford
Distributed Computing Research Group, School of Electrical Engineering and Computer Science
University of Newcastle, Callaghan, NSW, Australia

Keywords: Cloud computing, Enterprise service bus, Web engineering, Client computing.

Abstract: From an consumer perspective the Cloud is opaque. Online storage and the rise of web applications are changing the way users work. There continues though to be no distinction from a user experience point of view between accessing a Cloud-based application and accessing a web application deployed on a classic server. We propose a new paradigm for online application development which takes the best from web applications, thick client applications and the new "application store" market. This approach expands the cloud to encompass all resources that belong to a user; be it local client resources or server-farm resources procured using a traditional cloud model. By implementing these concepts we can bring the benefits of cloud computing directly to the end user while breaking developers out of the confines of the web browser.

1 INTRODUCTION

From a consumer perspective, the Cloud (Boss et al., 2007) is opaque. The web browser provides the solitary interface between the user and the application. Application developers are moving to web application models to take full advantage of the benefits that cloud computing provides. This has forced the end user experience into the web browser where security (WebDevout, 2009) and usability continue to be a concern. While technologies such as HTML5 (Coursey, 2009) attempt to address the growing pains of the web browser, they appear to be destined to encounter early adoption issues (Krill, 2010).

Thick client applications have a history of providing stable and secure end-user experiences, but lack such benefits as rapid deployment and single support platform. A new trend towards "application store" platforms does attempt to address some of the traditional issues with thick client deployment. In review, it is obvious that there are benefits from each of these three deployment styles.

We propose a new paradigm for component-based software engineering. The paradigm takes advantage of each of the three historical application deployment methodologies. A truly component-based system allows us to expand the concept of the cloud to encompass all resources to which a user has access. In turn, the user experience can break outside of the web browser into process-based user-interface components while maintaining the benefits of dynamic patching and ease-of-deployment.

This new global component-based design addresses some of the published concerns with cloud computing, which are delaying wide-spread adoption. We show that issues such as compliance and security in the cloud can be addressed by designing our applications in a component fashion based around a global-scale component service bus.

2 PROBLEM DESCRIPTION

The competition between web application and traditional thick client applications has largely been decided. The web application is now seen as a viable deployment methodology that provides many benefits over the traditional thick client approach. While web applications are thriving, especially in the cloud, a number of key issues remain;

- Security concerns about user data in the cloud continue to stifle uptake. Concerns that revolve around end-user privacy, transparency and compliance are key (Buyya et al., 2009; Newman, 2009).
- The user experience is limited by the virtual environment provided within the web browser. Web

Wallis M., Henskens F. and Hannaford M..
OVERCAST SKIES - What Cloud Computing Should Be?.
DOI: 10.5220/0003390900730078
In *Proceedings of the 1st International Conference on Cloud Computing and Services Science* (CLOSER-2011), pages 73-78
ISBN: 978-989-8425-52-2

Browsers are not able to provide full feature-rich user interfaces whereas traditional thick client applications can be implemented using a wide range of interface features.

- Application development in a cloud environment is still prone to vendor lock-in issues due to a lack of common application programming interfaces (Brandel, 2009)

Issues such as these have led to continued use of thick client applications. In addition, we are seeing adoption of a new model revolving around "Application Stores". These application stores provide a way for thick clients to be easily deployed and updated on end-user devices. Unfortunately, each application store to-date has been limited to a single platform. No common abstraction has been developed that allows applications to execute across multiple platforms, distributed by multiple stores. Application stores present a move away from large integrated software packages to smaller, function-specific applications. These small fat client applications provide service-style, single purpose, functionality. Another defining point is that application stores are generally controlled by the hardware vendor for the device to which they relate. It has been proposed that this tight relationship provides a level of quality control, but often it has been critized as a way of limiting innovation and controlling the user (Kaneshige, 2010).

The three main application development and deployment approaches - web application, thick application and application store all have well documented issues (Kaneshige, 2010; Gilbertson, 2007). These represent a hindrance to adoption of a single consolidated approach. We propose a new global component-based paradigm, which takes the best from all three models, while addressing key concerns found in the areas of Web 2.0, user experience and cloud computing.

3 EXISTING FRAMEWORKS

There are many distributed component-based frameworks that aim to resolve key points raised in the above problem description.

The .NET framework and corresponding Azure (Microsoft, 2009) cloud computing platform present a component-based environment for code execution. A key failing with the .NET approach is the lack of global scale and open protocol. Non .NET components must use bridges to access the .NET environment and transparent migration of components across WANs is not supported.

Component migration and location transparency is a feature of the OMG's CORBA (Object Management Group, 2004) component technology. Again, CORBA fails to scale to a global environment in which objects housed in multiple virtual clouds discover/locate each other and exchange messages. Many technologies and concepts from within CORBA have been leveraged in our new proposed framework.

Web Services and the traditional Service-Orientated Architectures (Bell, 2008) approaches also address issues relevant to this paper. Web Services rely on DNS names for transparency and component locality functions are restricted to discovery services and ill-defined service specifications.

The web browser itself can be viewed as a distributed execution platform, especially due to recent developments in web browser 'plugin' and 'extension' technology. Enhancements to the web browser model to support a lower-level of component execution have been suggested (Henskens and Ashton, 2007) and may provide a stepping stone for a complete component-based solution.

In Section 4 we present a truly global component-based architecture, which builds on lessons learnt from the above technologies.

4 COMPONENT-BASED ARCHITECTURE

Component-based software engineering is not a new concept. Component-based systems from a design perspective are commonly compared to Object-Oriented approaches (Szyperski, 1997). The key difference is that each component retains its own execution thread within the system. By taking a component-based approach to application development we can clearly define each application as a set of components. These components do not need to all execute within the one run time environment. The application developer may define one group of components as server-components and have them execute within a traditional remote cloud environment. Another group may be client-components that execute on end-user devices, outside of a sandbox such as the web browser. Some components are defined as data storage components and are in charge of persisting application information. There is no need for all application data to be stored by a single component using this approach. For example, the design allows a user to instantiate different storage components for private and public data.

Such a component-based system requires technol-

ogy that allows components to locate and communicate with each other. While technologies such as CORBA and SOA address some of these requirements, no existing system can scale to a global model. The quest for a global model leads us to Cloud Computing.

4.1 Cloud Computing

Currently, the Cloud encompasses processing and storage resources within one or more data centres. The Cloud is opaque to the end-user with the internal resources being hidden. Taking this traditional cloud approach excludes incorporation of end-user resources. Despite this, there are many benefits that the Cloud can provide. Dynamic resource scaling, rapid deployment and abstraction of resources are all key to a global deployment model. Bringing together a component-based design with the inclusion of client based resources within a cloud environment provides a powerful, global approach to application development.

A key aspect of this expanded cloud concept relates to the end-user's perspective of the system. Currently, any processing or data storage that occurs on the local PC is seen as being 'my data'. When moving to a cloud computing based model this distinction is blurred as the end-user is seen as giving their data and processing away to a 3rd party system. This represents a loss of control, transparency and trust in the overall system. Applications in the cloud, and in Web 2.0 in general, are also tailored to suit the general user base. Traditional thick client applications, in contrast, can be configured for that specific end-user.

By expanding the boundaries of the cloud (Wallis et al., 2010b) to encompass all resources available to the user we can extend deployment of the application components to include client devices and workstations. This creates a 'virtual cloud', for each user, which surrounds all resources that the user can access - be they local computing resources or remote server resources. Users essentially become part of the cloud with full control over where data is stored and processing takes place. Specific components of applications can be tailored and even replaced to suit end-user specific needs. Each component has its own execution thread, which ensures that the end-user sees the executing application as 'my application', as compared to a multi-tenant shared instance.

To bring this concept to the end-user requires a new component-based system that can extend the boundaries of the cloud and give processing and control back to the end-user.

4.2 The New Paradigm

Figure 1 shows the current model of cloud computing in which the user and their workstation/device is external to the cloud. Notice that the role of the workstation/device in this model is as an input/output device only. Figure 2 depicts the enhanced model in which the users' resources are brought into the cloud, allowing components of applications to execute across all available resources.

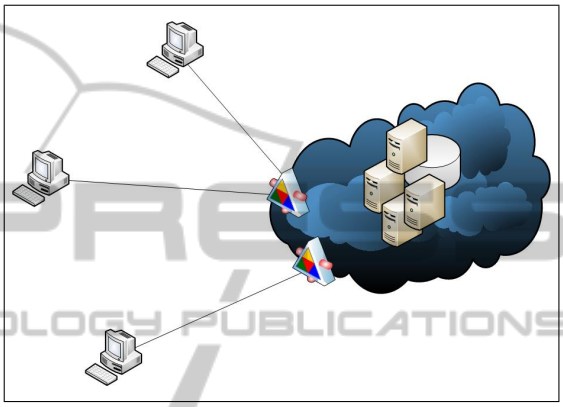

Figure 1: Existing Cloud Computing Model.

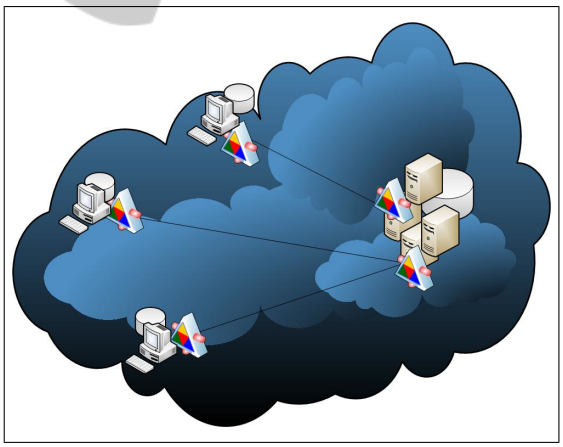

Figure 2: Expanded Cloud Concept.

Application developers can release updates at a component level, and share versioned components between applications. Components can be listed as mandatory and pre-loaded onto client devices when an application is purchased. These techniques allow for an application store approach to deployment while retaining the openness of a traditional thick client platform.

At a high-level, the following key advances can be expected for a global component-based model:

- Local data storage of sensitive data while retaining web application access to the data for processing.
- Component level application patching which will allow application developers to push out updated code in a distributed manner, retaining many of the deployment benefits found with existing web applications.
- Distributed execution will allow unused resources in a user's local environment to be consumed by processor-intensive and storage-intensive components.
- Component mobility will allow users to move components from system to system as required.
- User interaction to the application is abstracted as a graphical user interface (GUI) component and is deployed into the system in the same manner as other application components.
- Existing technology such as web browsers can be wrapped in component shells and executed within the new web platform environment providing a large amount of reuse of existing technology.

5 INTER-COMPONENT COMMUNICATION

To provide a global communication backbone for these components an extension to the classic enterprise service bus is required. A hierarchical deployment of multiple buses will allow components in LANs to communicate with a single bus, while buses can communicate in a hierarchical fashion to pass inter-component messages across WANs and internets. This model allows inter-cloud communication and treats each cloud as a logical LAN in a global internet. A service bus also facilitates inter-language and inter-platform communication. This allows components within an application to be developed using distinct languages. For instance, the developer may decide to implement a user interface component in native C++/OpenGL while implementing data storage using J2EE. The service bus will handle the communication between those components and provide any required data transformations.

A service bus architecture also allows for component mobility. Combined with abstraction on the run time environment, this allows for components to migrate from one processing resource to another in a transparent fashion without losing connectivity to other components within the application. It is foreseeable that user-interface components could migrate from device to device without losing state or connectivity to the system.

Security is implemented at the service bus layer, inspecting all messages between components. Inter-cloud communication also occurs through publish service bus endpoints and as such allows the bus to validate all external requests. Public-key cryptography and existing trust-relationships have be leveraged to provide request signing and validation functions.

Client-side bi-directional connectivity will be required and hence provisions have to be made to address such end-user technology blockers as NAPT. Previous research (Wallis et al., 2007) provides a dynamic technique for addressing such issues in a IPv4 setting. The introduction of IPv6 (Deering and Hinden, 1998) will also assist in addressing this issue due to the removal of NAT and translation layers.

6 DATA STORAGE

A key focus of this research is data storage within the web/cloud environment. Previous research (Wallis et al., 2010c; Wallis et al., 2010a) has presented a solution to the issues surrounding data duplication, data freshness and data ownership online. The more data stored by web applications the greater these concerns become. By making data storage a resource within each user's virtual cloud we can move the onus of data storage away from the application and back to the data owner.

Our previous research presents a model for data storage which has the end-user being responsible for the storage of their own personal information. Web applications can broker access to this data and present it to 3rd parties in a transparent fashion using authentication tokens based on protocols such as SAML (OASIS, 2007). Figure 3 depicts this approach.

Taking this approach to a component-based system allows these data storage services to become components addressed within each user's virtual cloud. High-risk information such as date of birth and credit card numbers can be stored on private data storage services managed directly by the end-user, while public information such as photo libraries and blog entries can be stored on less-secure infrastructure. By connecting these data storage services to a common communication bus it allows each service to be accessed transparently by any services requiring access to this information. Requests from 3rd party components outside of the user's domain (such as a merchant requesting access to a credit card number) can be tightly controlled and validated by the service bus before making it to the data storage service for further

validation.

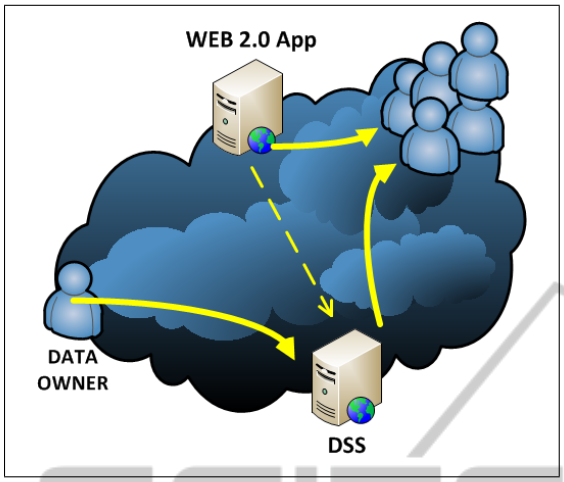

Figure 3: Data Storage Service model.

7 IMPLEMENTATION

To prove the model presented by this new paradigm, an implementation that focuses on the following key issues has been created:

- Application design and definition.
- Component deployment, installation and updating.
- Component locality and mobility.
- Inter-component communication.

A focus has been placed on providing an open API for the component-based system, which allows cloud vendors and operating system designers to adopt the model at a core level. This provides a common infrastructure across both local and remote resources and addresses many of the vendor and technology lockin issues that continue to stifle cloud adoption.

This proof of concept implementation will be used to prove the model is providing a step forward beyond what is capable with existing technologies. A generic set of applications will be developed across all four application approaches - web application, thick application, application-store and the new global component-based model. Metrics will then be collected around data transfer, user interface response and general performance before a feature-comparison is completed to review non-function requirements such as ease of development and deployment.

The current implementation is focused on executing components developed in the Java programming language. The actual execution environment for the components is generic as long as the components can obtain access to the service bus communication backbone. The API being developed as part of this research does not limit the application developer to any specific language or framework as long as the API is adhered to and communication with the service bus is possible.

8 CONCLUSIONS AND FUTURE WORK

This new paradigm continues to be an open project with a key emphasis being placed on providing an open API for discussion within the cloud and software engineering community. The concept of defining an application as a series of components where those components can be deployed on a global scale with high levels of transparency is novel and non-trivial. This research will provide a platform that addresses this issue while also providing for future application development, which does not limit the developer to the confines of a single runtime environment.

For cloud computing this new approach opens up the resources of the client for use by the cloud. Clouds will no longer encompass data centres, but instead, surround all resources belonging to a single user or corporation. These virtual clouds can communicate through well-defined service bus entry points, which logically firewall one cloud from another.

Open issues such as how to allow the application developer to define where each component should execute need to be addressed. While maximum flexibility is ensured, it is unreasonable to expect the end-user to be able to direct components to resources in a micro-managed fashion. Generic application profiles can address these issues in a fashion that still allows power-users full control of their resources.

Adoption of the API being developed as part of this work can be used to help facilitate inter-cloud communication and break the user away from a web-browser based experience. Operating Systems have been providing rich user experiences for years at a native level without an abstraction such as the web browser. The presented technology allows for native execution of user interfaces while retaining all of the benefits of a dynamic web application. By moving out of the web browser to a component-based model we can help address some of the security concerns raised in the web browser space which have been caused by the rapid evolution of the browser far beyond its original intention.

REFERENCES

Bell, M. (2008). *Introduction to Service-Oriented Modeling, Service-Oriented Modeling: Service Analysis, Design, and Architecture*. Wiley and Sons.

Boss, G., Malladi, P., Quan, D., Legregni, L., and Hall, H. (2007). Cloud computing. http://download.boulder.ibm.com/ibmdl/pub/software/dw/wes/hipods/Cloud_computing_wp_final_8Oct.pdf.

Brandel, M. (2009). The trouble with cloud: Vendor lock-in. *CIO.com*.

Buyya, R., Yeo, C. S., Venugopal, S., Broberg, J., and Brandic, I. (2009). Cloud computing and emerging it platforms: Vision, hype, and reality for delivering computing as the 5th utility. *Future Generation Computer Systems*, 25(6):599 – 616.

Coursey, D. (2009). Html5 could be the os killer. *PCWorld Business Centre*.

Deering, S. and Hinden, R. (1998). Rfc 2460 - internet protocol, version 6. Technical report, Network Working Group.

Gilbertson, S. (2007). Jakob nielsen on web 2.0: "glossy, but useless". *Wired Magazine*.

Henskens, F. A. and Ashton, M. G. (2007). Graph-based optimistic transaction management. *Journal of Object Technology*, 6(6):131–148.

Kaneshige, T. (2010). Apple iphone kill switch: Can cios trust apple ? *cio.com*.

Krill, P. (2010). W3c: Hold off on deployment html5 in websites. *InfoWorld*.

Microsoft (2009). Microsoft azure. http://www.microsoft.com/windowsazure/.

Newman, H. (2009). Why cloud storage use could be limited in enterprises. *Enterprise Storage Forum*.

OASIS (2007). Security assertion markup language (saml) v2.0 technical overview. Technical report, Working Group.

Object Management Group (2004). *Common Object Request Broker Architecture: Core Specification*.

Szyperski, C. (1997). *Component Software: Beyond Object-Oriented Programming*. Addison-Wesley Professional.

Wallis, M., Henskens, F. A., and Hannaford, M. R. (2007). A system for robust peer-to-peer communication with dynamic protocol selection. *The 8th International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT)*.

Wallis, M., Henskens, F. A., and Hannaford, M. R. (2010a). A distributed content storage model for web applications. In *The Second International Conference on Evolving Internet (INTERNET-2010)*.

Wallis, M., Henskens, F. A., and Hannaford, M. R. (2010b). Expanding the cloud: A component-based architecture to application deployment on the internet. In *IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*.

Wallis, M., Henskens, F. A., and Hannaford, M. R. (2010c). Publish/subscribe model for personal data on the internet. In *6th International Conference on Web Information Systems and Technologies (WEBIST-2010)*. INSTICC.

WebDevout (2009). Web browser security statistics, http://www.webdevout.net/browser-security. *WebDevout*.