ACCESSIBILITY CHALLENGES IN E-GOVERNMENT: AN AUSTRIAN EXPERIENCE

Clemens Orthacker[1] and Thomas Zefferer[2]

Institute for Applied Information Processing and Communications,
Graz University of Technology, Austria
[1]clemens.orthacker@iaik.tugraz.at, [2]thomas.zefferer@iaik.tugraz.at

ABSTRACT

Secure user authentication and electronic signatures are key requirements for e-Government applications. Many EU Member States rely on smart-card technology for these purposes. In addition to security, accessibility is crucial for the integration of smart cards into web based e-Government processes. In Austria, the open-source software MOCCA provides access to smart-card functionality by means of Java Applet technology. Assuring MOCCA's compliance to established accessibility standards has turned out to be a challenging task. This paper discusses several accessibility issues that may arise in smart card based e-Government infrastructures. We focus on MOCCA and accessibility improvements that have been applied to this core component of Austria's e-Government infrastructure. We identify several accessibility issues related to Java Applets and show how faced problems have been overcome in order to achieve WCAG 2.0 compliance for MOCCA. The gained experiences and employed techniques presented in this paper can help to improve the accessibility of any Java Applet based application.

KEYWORDS

Accessibility, e-Government, smart cards, Java Applets, authentication, electronic signature

1. INTRODUCTION

For many decades, interaction with governmental administrations has been cumbersome for citizens due to complex procedures, limited office hours, and long queuing times. Positively influenced by customer-oriented services from the private sector and enabled by the emergence of information and communication technologies (ICT), governments have started to offer services to citizens also via web based technologies. The application of ICT to facilitate administrative procedures has become commonly known under the term “e-Government”. E-Government has rendered personal attendance of citizens in public offices unnecessary. Nowadays, citizens use the Internet to communicate with governmental administrations and to carry out administrative procedures. Due to intense effort that has been invested in e-Government initiatives during the past decade, Austria has been repeatedly awarded to be one of the leading European countries in terms of e-Government availability [4].

Security and accessibility are two crucial success factors for e-Government. Security requirements are usually met by applying established cryptographic mechanisms such as strong authentication and electronic signatures. In many European countries, smart cards act as enabling technology for electronic signatures and secure user authentication in e-Government processes. Among many other countries, also Belgium [5] and Austria [6] have introduced smart card based e-Government solutions. On the other hand, accessibility is usually considered by striving for compliance with the Web Content Accessibility Guidelines (WCAG) [7]. These guidelines have been proposed by the Web Accessibility Initiative (WAI) and define requirements and best practices to achieve accessibility in web-based environments.

Integration of smart cards into e-Government processes while preserving accessibility has turned out to be a challenging task. In this paper, we discuss accessibility in the context of smart card based e-Government services and address different issues that had to be overcome in Austria. We do so by reconsidering the open source software MOCCA (Modular Open Citizen Card Architecture) that has been introduced by the authors in [2]. By now, MOCCA represents a core component of Austria's e-Government infrastructure. MOCCA makes use of the Java Applet technology to integrate smart-card functionality into e-Government applications. Unfortunately, the Java Applet technology has turned out to be a potential source of various accessibility problems. In this paper we present our solutions to the faced problems and show how accessibility has been assured for one of the central building blocks of Austria's e-Government infrastructure.

The remainder of this paper is structured as follows. Section 2 discusses the two basic requirements of web based e-Government solutions: accessibility and integration of smart-card functionality. Subsequently, Section 3 describes different technical challenges that arise as a consequence of these two requirements. In this section we also present our approaches to overcome the faced challenges. Conclusions are finally drawn in Section 4.

2. Challenges in E-Government

Secure user authentication and electronic signature creation are crucial requirements of e-Government services. Similar to numerous other countries, Austria relies on smart-card technology to fulfil these requirements. Unfortunately, the use of smart cards raises various problems with regard to accessibility. This section discusses the relevance of smart card technology and describes how it is integrated into Austrian e-Government applications. We then emphasize the relevance of accessibility in the context of e-Government and identify various problems that have been faced during the development of MOCCA in Austria.

2.1. The Smart-Card Access Challenge

User authentication in common web based services is usually accomplished with username/password schemes. This method is vulnerable to identity theft[1] and does not provide strong authentication as required by legally binding transactions in e-Government. Contrary, smart cards provide two-factor authentication by relying on something the user possesses (the card) and something she knows (the PIN). Apart from strong authentication, smart cards also allow citizens to create electronic signatures that are legally equivalent to handwritten signatures [1]. Many EU Member States have adopted this technology as an integral component of their e-Government infrastructure and have rolled out smart cards to citizens.

In Austria, the majority of currently available e-Government services are web based. Hence, citizens mainly use web browsers to interact with public authorities. Unfortunately, no standardized approaches for accessing smart-card functionality from web applications exist and rollout of many different smart-card types and generations renders this task even more difficult. In order to hide the complexity of smart-card communication from web applications, an abstraction layer is in use in Austria. This abstraction layer for smart-card access is called *Security Layer* and has been introduced in [6] by Leitold et al. Figure 1 illustrates the basic idea of the Security Layer (SL) concept. Following a middleware approach, access to the functionality of different smart cards is provided through a common web based interface to higher-level services and applications.

[1] For instance, dictionary attacks pose a serious threat to the security of username/password based authentication schemes.

As the specifications of this abstraction layer are open and technology neutral, there are currently several different implementations in place in Austria[2]. However, MOCCA is the only open source Security Layer implementation and therefore plays a central role in Austria's e-Government infrastructure. According to the Security Layer specifications, MOCCA implements low level communications with smart cards and provides their functionality to e-Government applications through the Security Layer interface. MOCCA is therefore a crucial component and used by many e-Government applications that rely on smart card based user authentication or electronic signatures.

Figure 1 illustrates MOCCA's relevant interfaces. MOCCA accesses smart cards through the PC/SC interface. Applications may access smart-card functionality via the web based Security Layer (SL) interface. Required user input such as PIN entries is collected through a user interface.

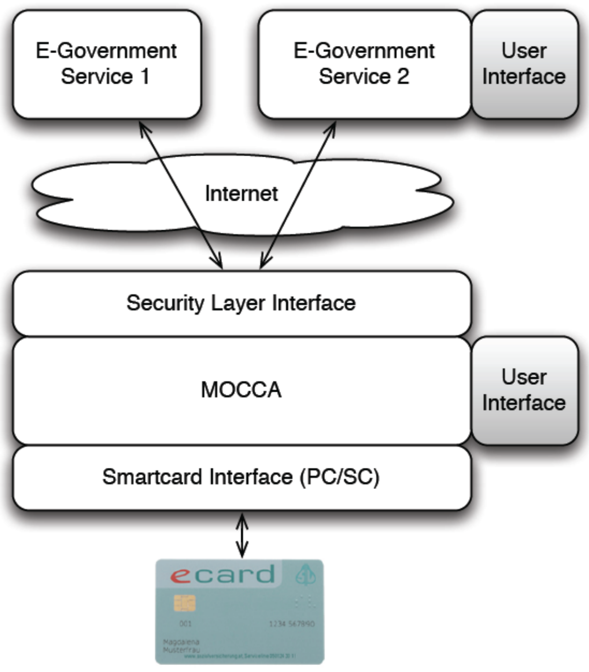

Figure 1. Layered architecture: MOCCA's user interface may be included in the e-Government service's web page. Smart-card PINs are entered via MOCCA's user interface (unless a pin-pad card reader is employed).

One of the basic ideas behind MOCCA was to follow a minimal-footprint approach. This should release citizens from the need to install software on their client PCs, which in the past has turned out to be troublesome for inexperienced users. By following a minimal-footprint approach, MOCCA significantly improves the usability of e-Government services.

Given the demand for a minimal-footprint solution and the requirement for system and browser independence, the available technological alternatives are actually limited. A survey of current rich Internet application frameworks and web-browser extensions finally yielded the Java Applet technology as most appropriate technology to access smart cards from web browsers.

[2] For a selection of such middleware implementations to be installed on the client PC see http://www.buergerkarte.at/download.en.php

MOCCA makes use of Java's smartcardIO[3] library to access the host system’s PC/SC[4] interface. Likewise, Java's Swing library is used to implement the user interface and to enable users to interact with MOCCA through a provided Java Applet. Due to Java's platform independence, MOCCA can basically be used on all common operating systems, which again improves usability and accessibility.

2.2. The Accessibility Challenge

Information and communication technologies have the potential to significantly improve governmental and administrative processes in terms of efficiency. It is therefore reasonable that an increasing number of governmental services are already offered electronically. However, this trend involves the risk that impaired citizens, who are unable to use these new technologies, are excluded from these services. This phenomenon is commonly known as *Digital Divide*.

To counteract the Digital Divide, especially governmental and administrative services need to be accessible to all citizens, regardless of existing disabilities or other factors that limit the use of ICT. Therefore, e-Government services have to be designed and implemented such that their degree of accessibility is maximized. Since most e-Government services are currently web based, compliance to the Web Content Accessibility Guidelines (WCAG) that have been introduced by the Web Accessibility Initiative (WAI) is a crucial requirement for accessible e-Government services.

Conformance with WCAG is actually an integral part of Austria's e-Government strategy. The need for accessibility in e-Government services is also regulated by Austria's e-Government Act [3]. Being a relevant component of Austria's e-Government infrastructure, the requirement for WCAG compliance applies also to MOCCA. Due to MOCCA's Java Applet based nature, the need for accessibility has brought up several challenges. In the following section, we elaborate on observed problems, show how several technical issues have threatened to compromise accessibility, and present solutions to the faced challenges.

3. PROBLEM ANALYSIS AND SOLUTIONS

It has been shown that the demand for minimal-footprint integration of smart-card functionality into web based e-Government services leads to the requirement for Java Applet technology. Furthermore, it has been discussed that accessibility is another important issue, especially in e-Government. Attempts to overcome both the smart-card access challenge and the accessibility challenge have brought up several problems during the development of MOCCA. In this section we discuss selected problems regarding accessibility that have been faced during implementation and present the corresponding solutions we have come up with.

The Java programming language features support for accessibility technologies, allowing programmers to build software that can also be used by people with disabilities. However, especially for Java Applets running in web browsers, several problems regarding accessibility arise. Additionally, the design of the Austrian Security Layer interface and the special architecture of MOCCA still complicate the situation. During the development of MOCCA, the accessibility challenges discussed in the following subsections had to be overcome.

3.1 Keyboard Navigation

According to WCAG, input elements such as text fields or links must be selectable via keyboard using the TAB key. Current web browsers assure this by default for input elements in web pages

[3] http://jcp.org/en/jsr/detail?id=268
[4] http://www.pcscworkgroup.com/specifications/overview.php

and the Java technology provides means to support keyboard navigation within Applets. However, web browsers in general do not provide means for switching the focus from the web page to the embedded Applet and vice versa. The user, cycling the focus with the TAB key, cannot reach the Applet's input elements if the focus is within the web page. Similarly, the web page's elements cannot be reached with the TAB key if the focus currently lies within the Java Applet.

The seamless integration of Java Applets into web sites in terms of keyboard navigation has turned out to be a challenging task. In fact, this problem seems to be a weakness of the web browser and requires a workaround. We have chosen the following JavaScript based approach to overcome this issue.

To allow users to move the keyboard focus into the Applet using the TAB key, a JavaScript function is called whenever the HTML input element that is located right before the Java Applet loses the focus, i.e. ceases to be active[5]. Within this JavaScript function, a public method of the embedded Java Applet is called via JavaScript. This Java method requests the focus for one of the Applet's currently displayed input elements.

To move the keyboard focus from the Applet to the embedding web site, a similar approach has been followed. Within a Java Applet, the Applet's context can be accessed through the Java class *java.applet.AppletContext*[6]. In our solution, this class is used to call a JavaScript function of the embedding web site. The called JavaScript function then assigns the keyboard focus to an arbitrary element outside the Applet.

To switch the keyboard focus between a Java Applet and its embedding web site, we employ the feature that public methods of a Java Applet can be called from its embedding web site using JavaScript and that Java Applets can call JavaScript functions of the embedding web site. This satisfies the accessibility requirement that all elements of a web site have to be selectable by keyboard.

3.2 Applet Scaling and Font Resizing

Web browsers usually allow users to easily change the font size of the displayed web site via keyboard shortcuts or menu items. However, this does usually not affect the font size of label texts within Java Applets. While all text in the embedding web site increases and decreases as supposed, neither the font size of text labels nor the size of several other user interface components in Java Applets do change.

We have addressed this issue by assuring that all Applet components are scaled according to the Applet’s current size, which is automatically adjusted with any change of the embedding web site’s font size. Whenever a resizing of the Applet is detected, all displayed Applet GUI components are re-rendered appropriately.

Additionally, two HTML buttons have been added to the embedding website, allowing the user to increase or decrease the size of the Applet and all of its GUI components along with the web site's font size. This resizing is again triggered by JavaScript functionality, which is used to assign a new size to the embedded Java Applet object.

Either by including HTML buttons or by relying on web browsers’ means for changing the font size, our approach guarantees that the Applet and all of its displayed GUI components can be resized dynamically. Both methods contribute to the satisfaction of the accessibility requirement for dynamically scalable contents.

[5] Alternatively, an invisible HTML input element could be inserted right before the Applet. Whenever this element receives the focus it could call the JavaScript method instead.
[6] For further information on this Java class refer to
http://download.oracle.com/javase/6/docs/api/java/applet/AppletContext.html

3.3 Screen Reader Support

By default, contents of Java Applets cannot be accessed by screen reader software. Java's Accessibility API[7] and the Java Access Bridge[8] provide means for the inclusion of interfaces required by assistive software such as screen readers.

While contents of the embedding web site can be read out, the Applet's contents are by default ignored by screen reading software. To solve this problem, our solution makes use of Java's Accessibility API. Through this API, each GUI element can be assigned with additional information that is then provided to assistive software such as screen readers through a well-defined interface.

Using the Java Accessibility API is in general good practice to achieve accessibility for Java programs. We have used this technique to make MOCCA compliant to screen reader software and similar assistive technologies.

3.4 Pop-Ups

A context sensitive help system is crucial for the usability of any complex software. In particular, this applies to software for creating electronic signatures with legal effects. Thus, the Austrian e-Government style guide convention [8] requires e-Government forms to provide context sensitive help to the user via a defined clickable help icon.

MOCCA provides a collection of web pages covering help topics for every single step and possible error situation in the signature creation or authentication process. Given the complexity of these processes, every help topic consists of an entire web page. In order not to interfere with the authentication process, a new browser window or tab should open if the user clicks the help icon. Since the help page to be displayed depends on the current state of the authentication or signature-creation process, it seems natural to open the new window via the Java Applet.

Modern web browsers feature pop-up blockers that prevent new pages from being opened automatically, i.e. if the user does not click on a link in the current page. Unfortunately, web browsers do not recognize the user's click on the help link inside the Applet and thus would block the help page from being displayed. In general, pop-up blockers display a warning, asking the user to either permit or disallow new pages being opened depending on their host URL. Once allowed, all help pages would open in a new help tab without the pop-up blocker interfering.

Apart from the unsettling warning about potentially insecure content, another, more severe problem persists: if the user confirms the pop-up, certain pop-up blockers reload the entire web page that caused the new page to open. Without going into details about the integration of the smart-card based authentication process in web applications, a reload of the entire page would restart this (multi-stage) process from the beginning[9], leaving the user with what appears as an aborted login.

A first solution yielded a new Java Frame opening (thus circumventing the browser's pop-up blocker) and displaying the help page using standard Java HTML components. However, the Java HTML rendering engine’s functionality is rather limited and does not support several features required for WAI conformance, such as ALT texts or TAB navigation.

Current web browsers feature very good accessibility support. It therefore seems natural to dispose of them to display the help pages. In order to circumvent the pop-up blockers, the help link must be placed in the embedding web page and not in the Applet. Again, JavaScript was

[7] http://www.oracle.com/technetwork/java/javase/tech/index-jsp-140174.html
[8] http://www.oracle.com/technetwork/java/javase/tech/index-jsp-136191.html
[9] See Section 3.5

employed to request the current help context via the Applet's public interface and construct the hypertext link to the respective help web page.

3.5 Application Integration

The Austrian smart card based authentication process consists of several stages and involves several modules (including MOCCA) that cooperate via HTTP and HTML forms. Application providers usually try to create a user experience comparable to simple username/password based authentication mechanisms. The entire authentication process is hidden from the user apart from the MOCCA Applet being displayed as part of the application's web site. Tight integration of the authentication process into the embedding web application is typically achieved by means of an HTML inline frame (IFRAME) in the application's web site. Unfortunately, with the use of the IFRAME element several accessibility issues arise.

Seamless integration requires to fix the inline frame's size to the space required by the Applet and to hide scrollbars. This however causes issues if any content different from the Applet shall be displayed. Error messages, which are for instance shown when the Applet cannot be loaded, might not fit into the IFRAME's limited area. Since the process requires secured communication with multiple online modules, server certificates of these modules have to be reviewed and accepted by the user. Upon first interaction with a secured site, browsers generally block the site's content and display a security warning instead. The user may add the server certificate to the trusted sites by clicking a button on the security-warning page. Any further communication with that server will be indicated as secured and no further warning will be displayed. Unfortunately, if the inline frame is neither resizable nor scrollable, the contained security warning cannot be displayed entirely and the user cannot accept the certificate by clicking the cropped or even completely hidden button.

In the common setting where online modules and the application share a common server certificate, this issue is not relevant at all. Otherwise, it can be avoided by ensuring that the certificates are trusted prior to the initiation of the authentication process. The application site may include referenced content from the secured site in order to force the user to accept the certificate prior to authentication. The fixed size of the inline frame causes another potential issue if MOCCA and the application are hosted by different operators. MOCCA updates might introduce changes in the Applet's size, requiring the application site to be adjusted. In our experience this is effectively a major problem.

As already mentioned in the previous section, a page reload causes the inline frame to reset to its initial state and the authentication process to abort. The authentication process requires an initial request to a remote authentication module. The embedding web page contains some input element to trigger this authentication request, causing the IFRAME to be inserted in the web page by means of JavaScript. The HTTP communication between the modules involved in the authentication process takes place within that IFRAME. Of course, a reload of the entire page causes the IFRAME to disappear. This might cause confusion with the user.

The issues arising from the integration via inline frames could be avoided by not integrating the authentication process and the MOCCA Applet as tight into the application site. However, we consider that the gain of usability caused by the tight inline frame integration outweighs the above-mentioned drawbacks of this solution. Of course, this trade-off leaves some room for improvements.

The discussed problems show that Java Applets can cause several severe problems regarding accessibility. At the same time, Java Applets are currently the only appropriate technology to implement an operating-system independent minimal-footprint middleware for smart-card access. Striving for WCAG 2.0 compliance of MOCCA[10], we have shown how Java Applet

[10] In 2010, MOCCA has been certified to be compliant to WCAG 2.0

related accessibility challenges can be overcome in practice. By assuring accessibility for MOCCA, we have contributed to the overall accessibility of Austria’s e-Government infrastructure.

4. Conclusions

Web content accessibility is a crucial requirement for providers of web based services. This especially applies to providers of e-Government applications, as these services need to be accessible to all people including those with physical disabilities. In order to counteract the Digital Divide, this challenge has to be faced. Assuring accessibility in e-Government has turned out to be challenging due to given constraints and the indispensable integration of complex technologies such as smart cards.

In this paper we have reported on our experiences with assuring accessibility for Austrian e-Government services. We have focused on the design and development of MOCCA, which is an integral part of Austria's e-Government infrastructure and mainly used to integrate smart-card functionality into web-application based e-Government services. We have addressed several accessibility issues that arise mainly due to MOCCA's Java Applet based nature. Furthermore, the identified problems have been analyzed and accessibility-preserving solutions to the found problems have been presented.

The implementation of MOCCA has shown that accessibility requirements can pose serious challenges to smart card based web-applications. Nevertheless, these challenges can be overcome. We have shown how smart-card functionality can be integrated into web-based services while preserving accessibility. This way, our proposed solution contributes to an accessible Austrian e-Government infrastructure and assures that e-Government services remain accessible to all Austrian citizens.

References

[1] Directive 1999/93/EC of the European Parliament and of the Council of 13 December 1999 on a Community framework for electronic signatures, http://eur-lex.europa.eu/LexUriServ/LexUriServ.do?uri=CELEX:31999L0093:EN:HTML.

[2] Centner, Martin & Orthacker, Clemens & Bauer, Wolfgang (2010) *Minimal-Footprint Middleware for the Creation of Qualified Signatures*, Proceedings of the 6th International Conference on Web Information Systems and Technologies.

[3] Austrian E-Government Act (Bundesgesetz über Regelungen zur Erleichterung des elektronischen Verkehrs mit öffentlichen Stellen) (2004), http://www.bka.gv.at/DocView.axd?CobId=31191(https://www.ris.bka.gv.at/GeltendeFassung.wxe?Abfrage=Bundesnormen&Gesetzesnummer=20003230).

[4] Capgemini: 9th Benchmark Measurement of European eGovernment services (2010), http://www.ch.capgemini.com/insights/publikationen/egovernment-benchmark-2011/.

[5] De Cock, D. & Wouters, K. & Preneel, B. (2004) *Introduction to the belgian eid card*, Katsikas, S.K., Gritzalis, S., Lopez, J. (eds.) Public Key Infrastructure, Lecture Notes in Computer Science, vol. 3093, pp. 621-622. Springer Berlin / Heidelberg.

[6] Leitold, H. & Hollosi, A. & Posch, R. (2002) *Security architecture of the austrian citizen card concept*. Proceedings of the 18th Annual Computer Security Applications Conference. pp. 391-. ACSAC '02, IEEE Computer Society, Washington, DC, USA.

[7] W3C: Web Content Accessibility Guidelines (WCAG) 2.0 (2008), http://www.w3.org/TR/WCAG20/

[8] Wagner-Leimbach, H., Kainz, G.: E-government styleguide (2010), http://reference.e-government.gv.at/AG-PS-Styleguide-sg-stg-2-1.2454.0.html