Lokale Benutzeroberflächen - überall verfügbar

Ein performanter Ansatz, um unabhängig vom Endgerät zu werden

Frank Borchard, XiSys Software GmbH

Die Ansprüche an die Benutzeroberflächen von embedded Systemen steigen mit der Einführung von IoT und Industrie 4.0. Existierende lokale Bedienoberflächen sollen möglichst ohne Probleme und Aufwand auf nahezu jedem denkbaren Endgerät, vom Smartphone bis zum Laptop, dargestellt werden können. Dazu wird üblicherweise die Bedienoberfläche als Webanwendung gestaltet und damit die Visualisierung der Oberflächenlogik auf den Webbrowser verlagert.

Embedded Systeme im industriellen Umfeld müssen in vielen Fällen direkt an der Maschine bedient werden und verwenden daher eine stationäre Bedienoberfläche. Zurzeit sind solche Anforderungen durch eine klassische Grafiklösung, basierend auf einem lokalen Grafikserver und lokaler Grafikhardware, am effektivsten abbildbar. Im Zuge von IoT und Industrie 4.0 tritt jedoch immer häufiger die Forderung nach Weboberflächen oder Remotefähigkeiten via browserbasierten Lösungen auf. Mit der Einführung von HTML 5 und den immer leistungsfähigeren Grafikmöglichkeiten kann nun dieser Forderung genüge getan werden. Man muss sich jedoch bewusst sein, dass webbasierte Lösungen von der technischen Seite einen erheblichen Mehraufwand bedeuten und auch schwieriger gegen Angriffe von außen abzusichern sind. Soll eine lokale Browserlösung implementiert werden, so steigen die Anforderungen an die benötigte Hardware erheblich und sind im industriellen Umfeld kommerziell oft nicht mehr darstellbar.

Remote-Desktop mit VNC, RDP, ...

Die gängigste Methode eine lokale Grafik weiterzuleiten ist die Verwendung einer Remote-Desktop-Software. Für unterschiedliche Betriebssysteme stehen Lösungen wie VNC (Virtual Network Computing), RDP (Remote Desktop Protokoll) usw. zur Verfügung. Auf dem embedded System muss ein entsprechender Service laufen, der die Inhalte des Videospeichers oder eines Framebuffers auf Änderungen untersucht und diese mittels eines geeigneten Protokolls an das Visualisierungsgerät weiterleitet. Auf dem Anzeigegerät muss immer ein entsprechender Empfänger installiert sein, der die Daten entweder in einem eigenen Fenster oder in einem Browser visualisiert. Da weder der Ort noch der Zeitpunkt von Änderungen genau vorhergesagt werden können, ist ein relativ hoher Aufwand nötig, um die Änderungen zu erfassen. Kleinere Systeme kommen hier relativ schnell an ihr Limit.

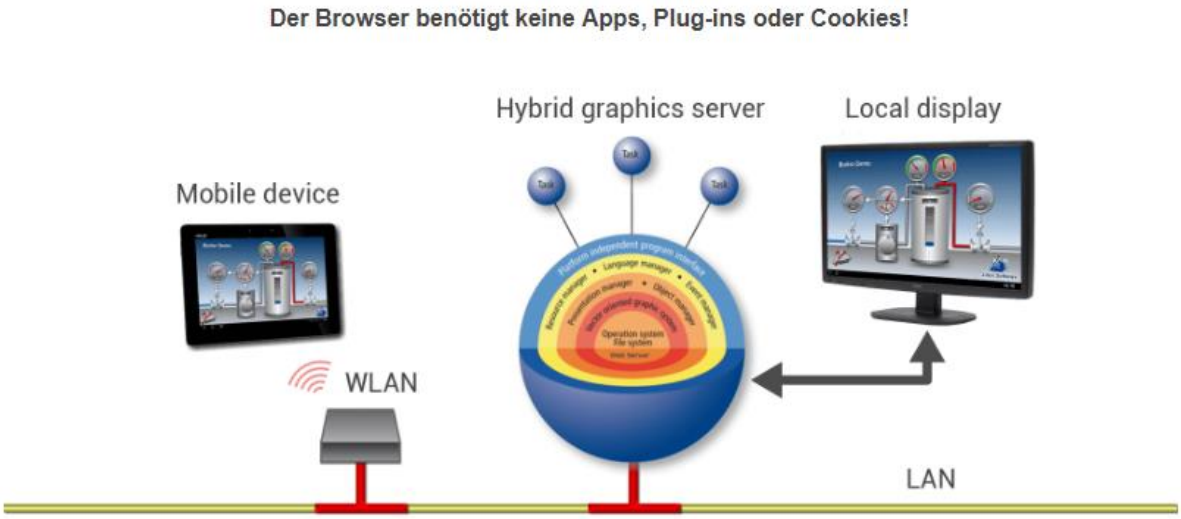
Remote via HTML5

Mit den Möglichkeiten die HTML5 bietet, kann auch eine Web-basierte Remote-Visualisierung realisiert werden. Diese übernimmt die Aufgaben einer Remote-Desktop-Software und kann auch als reine Webvisualisierung funktionieren. Die Applikationslogik verbleibt immer auf dem embedded System. Die Aufwendungen für das Zeichnen verschieben sich dann je nach Einsatz vom embedded System zum Anzeigesystem. Nach erfolgreicher Verbindung erhält der Browser ein Regelwerk in Javascript, welches Befehle, die der Webserver versendet, zur Laufzeit interpretiert. Der Browser erhält so die Grundfunktionalität einer externen Grafikkarte. Dieses Konzept erfordert auf der Anzeigeseite keine Apps, Plug-Ins oder Cookies.

Websockets vs. http oder warum HTML4 nicht als Plattform geeignet ist
Mit HTML4 war nur http-basierte Kommunikation möglich. Das heißt bekanntermaßen, dass eine zustandslose Verbindung, mit diversen „Hacks“ wie AJAX simuliert werden musste. Mit Websockets wird eine permanente Full-Duplex Verbindung mit einem im Vergleich zu http nicht existenten Overhead ermöglicht. Diese Eigenschaften des Protokolls ergeben zudem eine extrem geringe Latenz im Millisekunden-Bereich und öffnen damit die Tür zu Echtzeit-ähnlichem Verhalten der Benutzeroberfläche. Durch die Websockets-basierte Kommunikation ist die Animation der grafischen Objekte nahezu in Echtzeit möglich.

Konzept
Mit Augenmerk auf eine möglichst performante Visualisierung muss unnötiger Kommunikations-Overhead zwischen Grafikserver und Webserver vermieden werden. Der Webserver muss in möglichst kurzer Zeit und mit möglichst geringem Aufwand Zugang zu den Änderungsinformationen erhalten. Der effektivste Weg dorthin, besteht darin, den Webserver als integralen Bestandteil des Grafikservers zu realisieren. In diesem Fall können Änderungen ohne nennenswerten Zeitverlust vom Grafikserver an das Anzeigegerät weitergeleitet werden.

Eine weitere Effektivitätssteigerung wird durch Strukturieren der Webseite im Browser erzielt. Solange nur gerasterte Bildinformationen aus dem Videomemory an das Anzeigegerät weitergeleitet werden, genügt ein Canvas-Element für die komplette Darstellung. Scrollen, Verschieben oder Löschen von Objekten wird in diesem Fall mit sehr hohem Datenverkehr erkauft. Die Effektivität steigt deutlich, sobald für jedes Objekt ein eigenes Element angelegt wird. Der Grafikserver sendet dann - je nach Anforderung - gerasterte Bilder oder skalierbare Vektoren in Form von SVGs, die das jeweilige Objekt beschreiben. Die Aufwendungen für das Zeichnen verschieben sich damit mehr auf die Browserseite. Als angenehmer Nebeneffekt verringert sich die zu übertragende Datenmenge. Der Browser setzt aus den überlagerten Elementen das Gesamtbild des Displays oder einer Applikation zusammen. Um z.B. ein Objekt zu verschieben, muss nur die entsprechende Koordinate des jeweiligen Elements im DOM-Baum angepasst werden.



Hybrid-Grafik-Server
Die Firma XiSys Software hat einen Grafikserver entwickelt, der nicht nur über einen integrierten Window- und Objektmanager verfügt. In der neuesten Version ist auch die Funktionalität eines Webservers implementiert. Mit dem Objektmanager ist der Server in der

Lage, einen hierarchisch strukturierten Objektbaum aufzubauen und zu verwalten. Sobald sich ein Teilnehmer über eine Websocket-Verbindung anmeldet, wird die komplette Objekthierarchie aller auf einem Display dargestellten Objekte als DOM-Baum im Webbrowser abgebildet. Im DOM-Baum stehen in Abhängigkeit der Objekteigenschaften für jedes Objekt ein oder mehrere Canvas-Elemente oder SVG-Elemente zur Verfügung. Jedem Element werden automatisch individuelle Eventhandler zugeordnet, welche die Eventeigenschaften des Objektes am besten unterstützen. Z.B.: Beim Berühren eines Eingabeelementes wird automatisch ein Softkeyboard eingeblendet, das Element wird, falls nötig, in den sichtbaren Bildausschnitt geschoben und die Tastatureingaben an den Webserver weitergeleitet.

Ist im lokalen System keine Grafik-Hardware vorhanden oder soll nur eine Applikation im Browser visualisiert werden, kann der Browser auch eine asynchrone virtuelle Visualisierung anfordern. In diesem Fall wird ein Klon des Hybrid-Grafik-Servers erzeugt. Anstelle ein Objekt lokal zeichnen zu müssen, wird aus der Verktorliste ein SVG-Element abgeleitet und an den Browser verschickt. Der Browser übernimmt nun alle Renderaufgaben. Das embedded System kann sich hierbei auf reine Verwaltungsaufgaben beschränken - was natürlich dem Ressourcenverbrauch sehr entgegenkommt.

Performance Aspekte

Um eine hohe Akzeptanz der Bedienoberfläche zu erreichen, sollte diese ohne Verzögerungen auf Eingaben reagieren. Bei statischen Bildaufbauten ist dies kein Problem. Komplexe animierte Grafiken sind dagegen eine Herausforderung. Um flüssige Animationen und gleichzeitig schnelle Reaktionen auf Benutzereingaben zu gewährleisten, muss das Hauptaugenmerk auf einen möglichst hohen verzögerungsfreien Datendurchsatz gelegt werden. Latenzzeiten, verursacht durch lange Signallaufzeiten, schlechte Verbindungen oder die Überlastung eines Anzeigesystems müssen unbedingt vermieden werden. Daher darf weder der Server noch der Browser auf eine Quittierung eines abgesetzten Befehls warten müssen. Solange die Netzverbindung schnell genug ist und das Anzeigesystem die eingehenden Kommandos schneller bearbeiten kann als neue ankommen, läuft alles rund. Die Praxis zeigt, dass vorwiegend bei mobilen Geräten bisweilen Performance-Probleme auftreten können. Diese äußeren sich in erster Linie durch langsame Reaktionszeiten. Um dem entgegenzuwirken, ist es notwendig, im Webserver einen Algorithmus zu implementieren, der erkennt, welche Gesamtverzögerungszeit aufgrund von aufgestauten Befehlen zu erwarten ist. In Abhängigkeit dieses Ergebnisses muss dann die Abarbeitungsgeschwindigkeit des Grafikservers angepasst werden. Dieser Algorithmus wird auch zur Synchronisation mehrerer paralleler Anzeigegeräte verwendet.

Implementierungsaufwand

Ein bestechender Vorteil des hybriden Grafikservers ist der geringe Aufwand bei der Entwicklung mobiler Anwendungen. Die Benutzeroberfläche muss nur ein einziges Mal entwickelt werden, die Adaption an unterschiedliche Endgeräte wird vom Grafikserver automatisch geleistet. Es ist nicht erforderlich, eine parallele Infrastruktur mit den dabei anfallenden Entwicklungsaufgaben zu erstellen.

Alle Vorteile im Überblick

- Für die Visualisierung in einem Browser werden weder eine App, Plug-Ins oder Cookies benötigt. Die Visualisierung hinterlässt keine Spuren.
- Visualisierung mehrerer Systeme in iFrames möglich.
- Durch die Vorhersagbarkeit und Eingrenzbarkeit von Änderungen wird das zu steuernde System nur minimal belastet.

- Der integrierte Webserver kann quasi ohne Zeitverlust auf bereits gerenderte Daten zugreifen und diese sofort verteilen.
- Die Organisation eines DOM-Baumes für alle Objekte ermöglicht es, Bildschirmmasken vom Browser zusammensetzen zu lassen. Entfernen, Verschieben und Scrollen von Objekten wird vom Browser übernommen.
- Das Verwenden von SVGs erspart das Rendern von Objekten seitens des embedded Systems.
- Es können asynchrone virtuelle Sessions gestartet werden, die z.B. nur den Zugriff auf eine Applikation erlauben.
- Der Programmierer muss sich nicht um die Inkompatibilitäten verschiedener Browser kümmern. Er entwirft mit einem GUI-Builder nur einmal seine Bildschirmmasken. Diese werden dann lokal und im Browser identisch abgebildet.
-

Fazit

Das vorgestellte Konzept besticht durch seine Flexibilität. Damit können sowohl die Aufgaben eines lokalen GUI und einer Remote-Desktop-Software als auch Visualisierungen, welche nur eine Web-basierte Anzeige unterstützen, realisiert werden. Das GUI ist für alle Szenarien identisch. Der Programmierer muss sich nie mit den Techniken unterschiedlicher Anzeigesysteme auseinandersetzen.

Autor

Frank Borchard studierte Informatik an der Fachhochschule Würzburg. Nach dem Abschluss war er bei Alcatel-Lucent in der Telekommunikationsbranche tätig. Seit 2010 arbeitet er als selbständiger IT-Trainer- und -Berater im Netzwerk- und Telekommunikationsumfeld, als Berater für Softwaredesign mit dem Schwerpunkt IT-Sicherheit und als freier Mitarbeiter bei XiSys Software GmbH.

Kontakt:
Internet: www.xisys.de
Email: info@frankborchard.de
Email: info@xisys.de