VRIJE
UNIVERSITEIT
BRUSSEL

Graduation thesis submitted in partial fulfilment of the requirements for the degree of Master of Science in Applied Sciences and Engineering: Computer Science

INTERTEXT

The Everything App

OGUZ GELAL

Academic year 2020–2021

Promoter: Prof. Dr. Beat Signer
Advisor: Prof. Dr. Beat Signer
Faculty of Sciences and Bio-Engineering Sciences

Abstract

Today, we often see the same repeating patterns in front-end systems, from the components that make up the user interface to features and practices such as navigation, routing and state management. It is not uncommon for these patterns to be re-implemented time and again in various shapes or forms. Due to the lack of a systematic enforcement mechanism in the open market, these implementations' correctness and completeness rely primarily on the developer. A proper online representation nowadays requires accessible front-end applications with best practices optimised for various devices, browsers, screens and platforms, often resulting in high costs or inconsistent, insecure and low-quality byproducts. From a users perspective, this entails a poor user experience and bring potential security and privacy concerns.

To address these problems, we propose Intertext, a novel approach ~~to~~ how front-end systems are developed and consumed. It consists of Intertext User Interface Description Language (IUIDL): a device, design and style agnostic XML-based markup language; and a family of software clients for web, mobile, desktop and various other platforms that can render IUIDL into fully functional front-ends. IUIDL provides essential building blocks, such as a layout system, User Interface (UI) components and commands. It incorporates both input and output components, which can be used in conjunction with commands to make applications interactive. Commands enable sending, receiving and handling data, managing application state, routing and much more. It supports interpolations and views derived from the application state. IUIDL can be assembled and served from a generic backend, where the business logic of the application shall live. Once served from an endpoint, users can access it through any Intertext client on any device or platform in a similar fashion to an internet browser. Clients receive and render IUIDL most appropriately based on the host device and platform. It allows users to browse all their data sources through the same familiar, stable, robust, accessible screen/device optimised interface. IUIDL is agnostic of style, so users can customise the application's look and feel to their liking. Most importantly, clients do not accept executable code from external sources and all interactions with the device and external servers are controlled, which guarantees safety and privacy to the users. Intertext aims to stand as an alternative to traditional front-end systems that significantly benefits both the user and the data provider.

Acknowledgements

Contents

1 Introduction

In recent decades, the rise of smart interconnected devices not only changed how we interact with information, but also introduced all-new ways of engaging with the world. The rise of smartphones enabled users to consume content and perform tasks on-the-go, tablets made people rethink the ways they work and travel, wearable devices made new sets of data available to the users and so on. Ericsson Mobility Report of February 2021 reveals that the number of connected devices has been steadily increasing [2], and with new devices and form-factors underway, such as smart glasses, smart vehicles and folding displays, this trend is likely to continue in the foreseeable future.

The increasing demand in the diversity of devices and device families also increased the demand for front-end application development, as it became more challenging and costly to have a decent user-facing online presence. This lead to advancements in front-end development; emerging new tools, techniques, frameworks, libraries, and SDKs made it possible to build consumer-facing products in ways that were never possible before. Nevertheless, the ever-growing front-end ecosystem and high user demands forced developers to spread their focus to overwhelming levels; a recent survey *State of JavaScript* reveals that around TODO: This is wrong -> 60% of developers use more than one Front-end framework, and for cross-platform frameworks, it is more than TODO: This is wrong -> 80% [9] (Fig 1.1). This transformation did help create a large and dynamic market of consumer products,

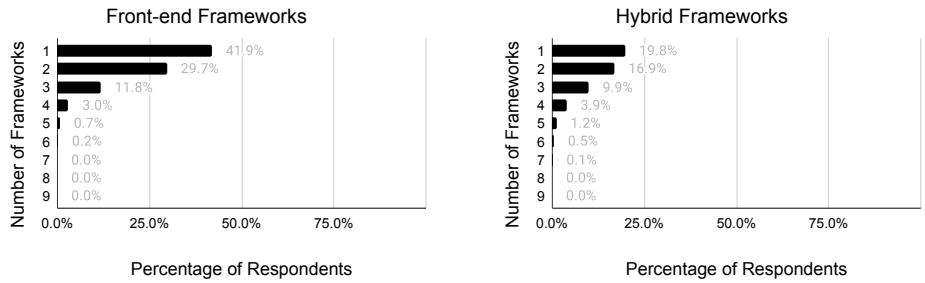

Figure 1.1: Number of frameworks used to Percentage of respondents

goods and services, though it did not come without some hurdles. In this thesis, we explore what these hurdles are and discuss how Intertext addresses them. subfigure env

1.1 Problem Statement

In this thesis, we group these hurdles mentioned above under three main categories; consuming data, privacy and security, and providing data. The consuming data section discusses end-users problems due to user interface and experience inconsistencies, and lack of device and accessibility support. The Privacy and Security section argues why there are no truly private and secure environments. Providing data section gives insights into the challenges faced by data and service providers in creating front-end applications.

Consuming Data

The simplicity of consuming data is lost within the complexity of modern-day applications; nowadays, something as simple as checking the weather, reading a news article, browsing an image gallery, buying a product or service and filling out a form can be frustrating and time-consuming. One common reason for this is the inconsistencies and errors in front-end implementations. In the open market, where everyone can create user-facing applications, there is little to no enforcement or quality control measures on how a user-interface should look or how it should behave. The presentation of the same functions can significantly differ based on the implementations. For instance, some websites' navigational menu component may be on the top, and others could have it on the left or right. Some could offer hamburger-menu style functionality where the user has to click/tap on the hamburger icon to toggle on and off, but some could work by swiping gestures. Swiping gestures might

toggle the navigation menu on and off, while some implementations could implement it as "go back", and so on.

In some cases, the developer can choose a poor selection of a colour palette, causing some components to blend into the background or make it hard for users to see. Users often have to take a second to adjust to every different experience from every different provider. While some providers with higher development budgets create flawless experiences for their users, this is not always the case, and there is no guarantee.

Another major hurdle in consuming data for users is device and screen size support. Providers typically need to spend significant extra effort to support various devices and screen sizes, should they want such support in the first place. Many providers choose not to have this support to reduce development cost, leaving some users with bad experiences or no experience at all. Different levels of support for different environments might cause confusion and frustration; for instance, some providers might offer a native mobile application with native mobile gestures, while some might have a Progressive Web App (PWA) that comes with native-like gestures, and some might offer a mobile-web experience that may or may not come with smooth gestures at all. Another significant experience difference between mobile applications and web-based applications is the style of navigation. Users accustomed to using native mobile applications could expect the navigational history to retain between tabs and get frustrated when they realize that this is not the case on a mobile web application. At times improperly handled navigation in a Single Page Application (SPA) might even cause the native "back" functionality to throw the user out of the application back to the previous one.

Creating accessible front-ends is another thing that providers have to spend significant efforts on, so much so that there are even developers who specialize only in this field. Creating accessible applications is not always the priority for many development projects due to its costs and efforts. The diversity in user interfaces directly affects the accessibility aspect, as it is another implementation detail that the developer needs to deal with. There are many different ways of creating accessible user interfaces, various implementations for different kinds of accessibility needs, and different ways of implementing each one of them. For instance, in a web application, adding "focus" states to DOM elements is the most basic form of accessibility implementation that allows users to tab into a specific element to interact with it. Designing the order in which they receive focus is a whole other dimension.

Privacy and Security

As mentioned before, there is a lack of a systematic enforcement mechanism for front-end applications. There are some protective measures in place; however, none of these measures can guarantee a completely secure and private environment [12], even the most ubiquitous browser or platform features could be used to relinquish the privacy of the users [10]. Most popular application stores typically have policies and guidelines on what the applications they distribute are allowed to do, but offending practices, especially the non-obvious ones, likely flies under the radar as most app stores do not require the source codes to be provided. Browsers and operating systems block suspicious activities to some extent, but they do not (or cannot) interfere with the legitimate (or legitimate-looking) ones. Governments implement various forms of cybersecurity laws to protect users, but laws are for the law-abiding, and as long as these laws cannot be enforced effectively, there is no safe environment for the users. As long as users are required to execute code on their devices, the bottom line is that they cannot be truly safe.

Providing Data

The word Providers refers to anyone who offers data and services to end-users, and it is safe to say that the providers also share the problems discussed under the Consuming Data section. The complexity of building a decent, well designed, accessible front-end experience, not even once but once per every client built for various platforms, forces providers to choose between supporting multiple devices, following best practices, creating a good experience. It is typically the case that providers cannot have it all initially, as it requires high costs and large teams.

Codesharing is one of the recent trends in front-end development gaining some traction, aiming to solve some of these problems mentioned above; using react-native, react-native-web or Flutter, one can have a single codebase that produces applications for multiple environments. Modern front-end libraries/frameworks typically abstract the view layer, making it possible to create applications for different environments by swapping the view layer and wiring it up to the logic layer, which seems to be the current cross-platform application development status quo. However, the issue that comes with this approach is the difficulty in reducing the codebases for different environments into one, as it is hard to address the various requirements and features of each platform. Even though it is possible to share some extent of the code, sometimes platforms are conceptually distinct, and the software for these platforms needs to be built differently.

Another pain point faced by providers is maintenance. The world of front-end development is an extremely fast-growing and evolving ecosystem. The changes are persistent, and at times breaking. There are thousands of tools, libraries, frameworks, SDKs available at the fingertips of developers at no cost. It is a common practice to make use of these libraries as they help with the development significantly. However, the diversity in the libraries used to build the software results in a diversity of maintenance problems. Even the most well-tested and maintained libraries could break after an update, causing a headache for the developers and hardship for the users.

1.2 Research Questions

We have listed the problems that we are interested to solve in the section above. In light of these problems, we aim to answer the following questions in our research:

How can we re-imagine front-end, and improve it in such a way that:

1. from the users perspective, for all applications it:
   (a) presents a consistent User Experience (UX) ?
   (b) works consistently on all supported devices and platforms ?
   (c) allow users to customize the look and feel ?
   (d) guarantees accessibility ?
   (e) guarantees privacy ?
   (f) guarantees security ?
2. from the data-providers perspective, it:
   (a) eliminates the need to create visually-appealing front-end applications ?
   (b) eliminates the need to create accessible front-end applications ?
   (c) eliminates the need to create front-end applications for every device / platform ?

How can we create an alternate front-end realm that:

1. brings product development costs to a minimum?
2. enables more variability of software products in the market?

3. unifies the experience, so individual developers and large companies compete on the same ground?

1.3 Contributions

As discussed in the Problem Statement (1.1) section, Intertext ambitiously attempts to solve many different problems from various domains. A solution at this scale requires novelty, rethinking the current state of art instead of an incremental update. The biggest contribution of Intertext is to borrow existing concepts that are mostly academic, combine them in new ways, and apply them to solve these real-world problems. With that said, here are some notable contributions:

- We reviewed existing research on User Interface Description Languages (UIDLs) and other similar topics to outline ways to utilise these concepts to solve the aforementioned problems.
- We designed IUIDL; while doing so, we decided:
  - What are the concepts that are crucial to modern front-end development that needs support out-of-the-box
  - What terminology should we use in order to create an optimal balance between the device-independent nature and developer familiarity
  - What the syntax should be like to maximise developer friendliness
  - How to overcome some limitation of XML in the most effective, clear and extendable way
- We created an engine using Javascript / Typescript to perform common tasks of Intertext clients, such as parsing IUIDL and managing the application state
- We created multiple software clients that can render IUIDL into functional front-ends optimised for the host device and are user customisable where possible

1.4 Methodology

To identify and effectively address all these problems, we utilized the Design Science Research Methodology (DSRM) [7] to carry out our research. DSRM

is a commonly adopted research framework that provides six executable steps to guide the researcher in being consistent with prior work and provides a theoretical process for doing the research, and guides through presenting and evaluating the outcome.

The first step of DSRM is **identifying the problem and the motivation**. It is not uncommon for research and inventions to stem from a problem or a necessity. Similarly, in our case, the problem was out there; we only needed to focus on it to better understand the issues we are tackling. In the Problem Statement (1.1) section, we discussed and justified what problems are we attempting to solve with Intertext. Then we proceeded to the next step, **defining the objectives for a solution**. As we narrowed down the problems that we attempt to solve in the Problem Statement (1.1) section, we drafted our objectives to approach this problem. We explained these objectives in detail in the Research Questions (1.2) section. Then, we were ready to take the next step of **design and development**. In the following sections TODO: add sections, we explained in detail our development process, all the challenges and design decisions that we have taken in order to address the problems in the best way possible.

Once we had a working prototype in our hands, we created a simple backend application that uses Intertext as its front-end. This sample application served two purposes; it demonstrated how IUIDL could be dynamically served from a backend, and it was used for the user evaluation. Consequently, we used this sample application to introduce Intertext to the users, communicate its goals and motivations, and then collect feedback. This last step combines the final three steps of **demonstration**, **evaluation** and **communication** into one, which is explained in details in TODO: add section section.

1.5 Structure

TODO: add structure

2
Related Work

This section explores and discusses the state-of-the-art research, tools, and technologies in this field or related to this field. First, we focus on existing UIDLs and how they compare to Intertext. We categorise them to demonstrate the relevancy as coherently as possible and investigate ones that fall into the same category with Intertext on a case-by-case basis. Then, we look into libraries and frameworks that aim to achieve a similar goal with Intertext. We will explain how Intertext is extending the existing solutions to do what it does. Moreover, we will explore commercially available tools and services and discuss why they are relevant to Intertext and what is to be learned from their efforts and success.

2.1 UIDLs

A UIDL can be defined as having two separate parts: a syntax that describes the user interface characteristics, and semantics that defines what these characteristics mean. They share the common goal of describing a UI without targeting any particular programming language or platform; nevertheless, the end-goal of UIDLs often varies [11]. UIDLs typically use XML or a similar markup language/notation that later gets transpiled into a programming language or is processed by a software to automatically or semi-automatically be translated into a UI, a visualisation, or any byproduct depending on the

goal of the project. A UIDL can be thought of as a tool designed to achieve a particular goal or to achieve a common goal in a particular way.

Souchon, Nathalie and Vanderdonckt et al., 2003 [11], and later Guerrero-Garcia, Josefina and Gonzalez-Calleros et al., 2009 [3] argued *"there is a plethora of UIDLs that are widely used, with different goals and different strengths"*. There are many ways of classifying and categorising existing UIDLs, as can be seen in *A review of XML-compliant user interface description languages, 2003* [11], and *A theoretical survey of user interface description languages: Preliminary results, 2009* [3]. However, in this section, to stay relevant to the comparison to Intertext, we group UIDLs under two main categories: compiled and interpreted UIDLs. Most of them belong to the first category, allowing us to focus on the general picture instead of on a case-by-case basis. This separation narrows down the comparison to a select few UIDLs, which we investigate in more detail.

Compiled UIDLs

Compiled refers to cases where UI descriptions are created at design time and are used to generate code or the final UI for different target platforms and environments. Most of the known UIDLs falls under this category. They often rely on a transformational approach; they utilise a reference framework for classifying UI elements with multiple levels of abstraction, and reify them into more concrete levels based on the target platform and context of use. UIML [1], XIML [8], TeresaXML [5], MariaXML [6] and UsiXML [4] are several examples of UIDLs that uses this model.

Interpreted UIDLs

TODO: BOSS? TODO: SeescoaXML TODO: OpenUIDL?

Comparison

We explored several of the existing UIDLs in this section. Some of them show similarities with Intertext in some respect, whereas others are radically different. The differences between many of them include but are not limited to:

1. Intertexts focuses on graphical user interfaces (GUIs), while many UIDLs try to cover different kinds of UIs. TODO: narrow down the mentions to focus on GUIs early on in the paper?

2. Intertext has a single level of abstraction, unlike many of the other UIDLs with multiple levels of abstraction, mostly due to their wider focus

3. Intertexts' IUIDL is interpreted. It relies on reifying the single level of its abstraction into the final UI on-the-fly, whereas the final UI for the UIDLs in the first category is generated during compile time.

4. The UI descriptions are created manually during design time for many UIDLs (or, in some cases, generated by several different means). However, for Intertext, it is meant to be generated and served on-the-fly from an endpoint based on the application logic and UI state.

5. Unlike Intertext, most UIDLs with a model-based approach incorporates heavy user interactions.

Nonetheless, if we were to take a step back to look at the big picture, Intertext has a fundamental difference that separates itself from the others; it is the purpose. The value added by the UIDLs is to improve the development process of user interfaces, reduce the cost and effort of creating UI descriptions that can target multiple platforms and environments with minimal to none additional effort. The byproduct of these UIDLs is a functioning user interface on each target platform. While we aim to take advantage of the nature of UIDLs to enable some of these advancements, we also have an equal focus on the user and improving the user experience. Unlike other UIDLs that doubles down on creating the best UI development experience and producing the most comprehensive outcome possible, we intentionally introduce calculated limitations to benefit users equally, if not more. Although we provide more details on Intertext clients and IUIDL in section TODO: add section, we can summerise our efforts that distinguish Intertext from other UIDLs as below:

1. Intertext is style-agnostic so that users can customise the look-and-feel of the UIs to their liking.

2. The final UI rendered for each platform and environment cannot be altered to ensure the quality and accessibility of the UI elements.

3. Intertext UIs are meant to be solely presentational with minimal user interaction for security reasons.

4. Intertext, in general, is a platform that comes with IUIDL and software clients that can interpret and render IUIDL. It has no intention of generating a UI that can run independently.

2.2 Frameworks and Libraries

2.3 Tools and Services

3

Intertext

Intertext is a platform based on a straightforward premise; it is a family of front-end applications that can interpret IUIDL and generate appropriate front-ends for the host platform. Simply put, a provider wanting to create an Intertext application creates a generic backend that generates IUIDL, and serves it from an endpoint, say at https://intertext.example.com. Users who wish to use this application pulls up an Intertext client on their preferred device or environment and visit this domain just like in a web browser. The Intertext client then requests this domain, fetch the IUIDL served by this endpoint, and generate the user interface as per the instructions received. Moreover, rather than rendering a simple static view, it performs some tasks such as accepting user input, navigating to different screens, making additional requests to fetch more data, keeping the UI updated and reading and writing some data to users local storage; all of which is again orchestrated based on the instructions received by the backend in IUIDL syntax. Intertext has multiple software clients built natively for various platforms that can interpret IUIDL most appropriately to the host device or platform. For instance, users browsing an Intertext app through a smartphone receives an experience optimized for touch screens, command-line interface client users receive an optimized experience for a text-based interface or a user browsing from a low-end device with limited capabilities use the version optimized for low-performance devices to get a comfortable viewing experience and so on.

Intertext gives providers a set of components to build and serve their UIs for their services. Given the involvement of a backend to handle all the logic, this service can be anything. In other words, users can enjoy their todo lists, habit tracker, notes, calendar, email client, social apps, news, weather and so on, all through one single app, using the Intertext client of their choice. IUIDL is agnostic of styles; for instance, a provider could create a "Button" component and specify what it says and what it does, but they cannot specify its looks, thus creating unified styles and user experience. This unity brings many advantages, most notably, consistency. Every single application on Intertext looks and feels the same, made out of components that users are familiar with. Customizability is another significant advantage; users can adjust the look and feel of these components, allowing them to personalize their browsing experience for all applications on the platform. Lastly, all components are accessible from the ground up. Thanks to this approach, the accessibility implementations are not the developers' responsibility; therefore, users with specific needs are guaranteed to have the accessibility features they need for every application Intertext platform.

Intertext is an open platform, and IUIDL is well documented for developers to build client applications, allowing the community to develop specific client applications that can interpret IUIDL in meaningful ways to respond to specific needs. Whether it is a VUI (Voice User Interface) client, a Tangible UI client, or even clients built for particular devices with specific requirements for targeting various communities or use cases, they can all support all existing Intertext applications served from backend services.

When it comes to privacy and security, the bottom line is that Intertext takes away the providers' ability to execute code on users devices. Applications running on Intertext clients are expected to implement all their application logic on the server-side, serve some instructions to the Intertext clients on what to do, how to function, what to show the user and so on. These instructions are a part of the IUIDL, and they are purely XML-based data; no executables are allowed from the providers. Nevertheless, Intertext allows applications to use a specific set of functions required to build a meaningful front-end application. For instance, an application can instruct an Intertext client to take actions like storing data, reading previously stored data, or making requests. Applications are also allowed to ask for permissions for things like accessing the camera (for devices that have one), notifications or location.

Intertext by default blocks cross-origin requests, meaning that a provider serving data from an origin can only ask the Intertext client to make a request to the same origin. Storage access is also bound to the origin; providers

serving data from an origin cannot access the data stored by another origin, preventing users from being tracked across the web for targeted advertisements. Furthermore, all requests that go back and forth are displayed to the end-users in a user-friendly way for transparency. This controlled approach guarantees the maximum level of privacy and security.

As convenient as Intertext is for end-users, this project aims to create advantages as attractive for data or service providers. It may not be the best option for all cases, such as front-end-heavy applications requiring client-side computations, custom styles or advanced graphics. However, for most cases, it has many advantages over building and maintaining front-end applications from scratch. Creating front-end business logic in the backend might be an overhead; however, the effort required is nothing compared to all the hurdles mentioned earlier. Intertext is agnostic of where the data is coming from and has no opinions on how it is generated. Therefore, the providers can easily use their existing backend services to add in the front-end logic. IUIDL is XML-based, and working with data is an essential part of every application; it is a light effort but greatly rewarding. Once a provider starts serving their front-end in IUIDL, that means they immediately get front-end applications for every platform that Intertext supports. Furthermore, as more Intertext clients get built, either officially or by the community, their applications immediately start working with those clients without requiring any change.

Intertext creates equal opportunities for everyone. The advancements in backend technologies these days made creating backend services possible in ways that were never possible before. Service providers such as AWS, Google Cloud and Azure made spinning up a backend infrastructure with necessary components as simple as a few clicks, all without requiring DevOps skills. Their generous free tiers, scalable infrastructures and the recently emerging serverless technology pulled the costs down so significantly that anyone familiar with backend technologies can create an application that could scale up to serve hundreds of thousands of users. However, one of the biggest blockers in launching applications that could gain popularity and evolve into a successful startup is arguably the branding and quality of the front-end applications. An application that suffers from user-facing problems mentioned earlier does not leave a good impression and creates a bad image, and it is relatively uncommon for such applications to be taken seriously, regardless of the quality of the data and services provided. Those who can achieve an exceptional user-facing presence while targeting multiple devices and platforms are commonly large companies with high development budgets. These costs hurt indie developers, particularly backend developers who do not specialize in front-end development but can create applications as

personal projects that could easily qualify as a good product. This project aims to remove this hegemony of "judging a book by the cover"; when all applications look the same and feel the same way, they are judged the same way. Not only indie developers have better chances of success, but there are also be more options for everyone.

A

Your Appendix

Bibliography

[1] Marc Abrams, Constantinos Phanouriou, Alan L Batongbacal, Stephen M Williams, and Jonathan E Shuster. Uiml: an appliance-independent xml user interface language. *Computer networks*, 31(11-16):1695–1708, 1999.

[2] Ericsson. Ericsson mobility report, 2021.

[3] Josefina Guerrero-Garcia, Juan Manuel Gonzalez-Calleros, Jean Vanderdonckt, and Jaime Munoz-Arteaga. A theoretical survey of user interface description languages: Preliminary results. In *2009 Latin American Web Congress*, pages 36–43. IEEE, 2009.

[4] Quentin Limbourg, Jean Vanderdonckt, Benjamin Michotte, Laurent Bouillon, and Víctor López-Jaquero. Usixml: A language supporting multi-path development of user interfaces. In *IFIP International Conference on Engineering for Human-Computer Interaction*, pages 200–220. Springer, 2004.

[5] Fabio Paternò and Carmen Santoro. A unified method for designing interactive systems adaptable to mobile and stationary platforms. *Interacting with Computers*, 15(3):349–366, 2003.

[6] Fabio Paterno', Carmen Santoro, and Lucio Davide Spano. Maria: A universal, declarative, multiple abstraction-level language for service-oriented applications in ubiquitous environments. *ACM Transactions on Computer-Human Interaction (TOCHI)*, 16(4):1–30, 2009.

[7] Ken Peffers, Tuure Tuunanen, Marcus A Rothenberger, and Samir Chatterjee. A design science research methodology for information systems research. *Journal of management information systems*, 24(3):45–77, 2007.

[8] Angel Puerta and Jacob Eisenstein. Ximl: a common representation for interaction data. In *Proceedings of the 7th international conference on Intelligent user interfaces*, pages 214–215, 2002.

[9] Sacha Greif, Raphaël Benitte. State of js, 2020.

[10] Konstantinos Solomos, John Kristoff, Chris Kanich, and Jason Polakis. Persistent tracking in modern browsers.

[11] Nathalie Souchon and Jean Vanderdonckt. A review of xml-compliant user interface description languages. In *International Workshop on Design, Specification, and Verification of Interactive Systems*, pages 377–391. Springer, 2003.

[12] Ting-Fang Yen, Yinglian Xie, Fang Yu, Roger Peng Yu, and Martin Abadi. Host fingerprinting and tracking on the web: Privacy and security implications. In *NDSS*, volume 62, page 66, 2012.