A Magic Lens for Revealing Device Interactions in Smart Environments

Simon Mayer
ETH Zurich
simon.mayer@inf.ethz.ch

Yassin N. Hassan
ETH Zurich
hassany@student.ethz.ch

Gábor Sörös
ETH Zurich
gabor.soros@inf.ethz.ch

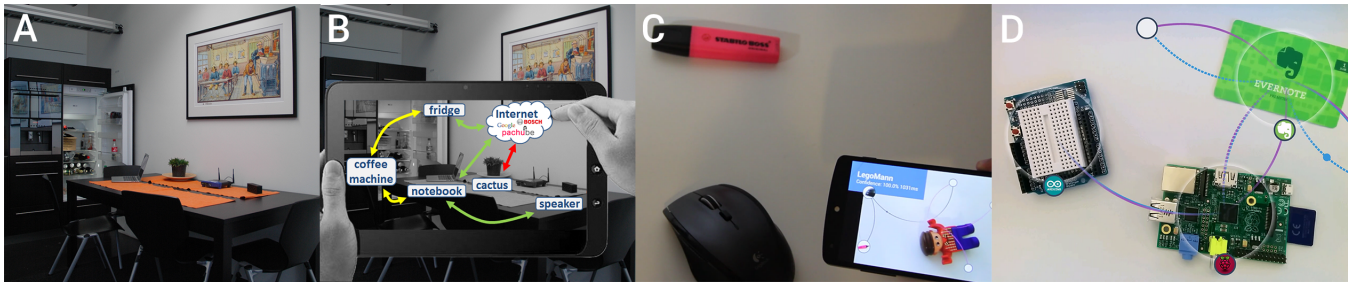

Figure 1: *A magic lens for visualizing communication links in smart environments. A) and B) Idea: Users can reveal communication links by observing devices through a tablet computer. C) and D) Screenshots of our tool showing messages being passed between connected objects.*

Abstract

Keeping track of device interactions in smart environments is a challenging task for everyday users. Given the expected high number of communicating devices in future smart homes, it will become increasingly important to put users more in control of their smart environments by providing tools to monitor and control the interactions between smart objects and remote services. We present a system for collecting and visualizing interactions of Web-enabled smart things and Web services in an intuitive and visually appealing way. Our tool displays device interactions both using a Web-based visualization application and in the form of a "magic lens" by augmenting the camera view of a tablet with relevant connections between recognized devices in the camera's field of view.

CR Categories: H.5.1 [Information Interfaces and Presentation]: Multimedia Information Systems – Artificial, augmented, and virtual realities C.2.3 [Computer-Communication Networks]: Network Operations – Network monitoring

Keywords: Augmented Reality, Network Management, Smart Environment, Visualization, Web of Things

1 Introduction

As ever more smart devices are connected to the Web of Things and start cooperating within smart environments, end users can easily lose track of interactions between those devices because the large amount of network traffic is difficult to monitor and manage. For example, LinkSys advertises routers with the slogan: "Do you have a home full of devices that seem to have their own agenda?" The feeling of not being in control of one's smart environment is not only inconvenient, but also induces privacy concerns and lowers user acceptance [Brush et al. 2011].

SA'14: Symposium on Mobile Graphics and Interactive Applications, Dec 03-06 2014, Shenzhen, China, ACM 978-1-4503-1891-4/14/12. http://dx.doi.org/10.1145/2669062.2669077

We believe that keeping users informed about what their connected smart devices are doing and why they are doing it will become increasingly important when more and more traditionally isolated devices such as refrigerators, stereo sets, and other domestic appliances get connected. Devices in future smart homes will not only interact with each other and with user interfaces, but also with services that are hosted remotely, for instance by utility companies to level peak loads in the energy grid, or by companies that analyze a household's data to provide advanced services to its inhabitants. This development gives rise to a number of challenges that, if not properly addressed, may hinder the widespread adoption of smart homes because people can lose trust in the smart things present in their homes and their remote services: domestic sensor data contains detailed information about inhabitants' lives that is potentially valuable to companies. For instance, by analyzing only the electricity load curve of a household, it is possible to determine whether the inhabitants have a job, and to estimate the household income [Beckel et al. 2013]. From the same data, an occupancy pattern can be obtained and it can be potentially predicted when – and for how long – the home will be unoccupied on a specific date in the future [Kleiminger et al. 2011]. Beyond the issue of sensitive data *leaving* the smart home, the fear of unauthorized remote *control* of devices constitutes the most severe anxiety of people regarding connected devices [Röcker et al. 2005].

End users must remain in control of which services their smart devices communicate with and what the transmitted data is used for by service providers. This is challenging because devices in smart environments form complex networked systems and communicate invisibly "behind the back" of inhabitants. Moreover, already today many users find it difficult to configure and maintain the rather simple home networks, mainly because of the "invisibility of settings and configuration information" as well as "poor strategies for diagnosis and troubleshooting" [Poole et al. 2008]. Managing a network requires inhabitants to use tools that are hard to use for them [Grinter et al. 2005] and some fear a loss of control of their environment as smart homes are growing increasingly complex [Randall 2003].

Our contribution is a tool for collecting and visualizing device interactions in smart environments. The tool consists of a logger that is simple to deploy, a back-end server that stores logged interactions between devices, and an augmented reality application that recognizes smart devices and visualizes their interactions on top of the camera view of a handheld device.

1.1 Related Work

Network management tools were originally created exclusively for industrial environments, but were adapted for private users as the complexity of networks in private homes increased. However, many such tools focus on static network properties such as parental control settings. This seems to be changing with modern products such as the LinkSys Smart Wi-Fi platform that enables users to monitor and control devices in the network in real time. On the other hand, applications such as Private Eye[1] let users monitor which remote services their computer is communicating with, a concept that was also demonstrated in the NetFlow project [Minarik and Dymacek 2008]. Recently, network monitoring tools have started to consider network nodes as embodied devices and applications such as CANVIS [Keith Mitchell 2005] were created that allow individuals to select networking devices using their smartphones to obtain information about how and how much they communicate with others.

We propose to use visual object recognition technologies and methods known from the domain of augmented reality to take these approaches to the next level, by visualizing data about connections between devices and services on mobile user interfaces as an overlay for the physical world. This approach could allow users to easily and intuitively monitor information flows between nodes inside their home networks and with remote services and thereby get more in control of their network installations. Similar ideas are applied for visualizing urban infrastructures, for instance in the Vidente project [Schall et al. 2013] where underground tubing is visualized using an augmented reality application on a smartphone. Our approach should enable users to better understand device associations in their smart home and to perceive unwanted interactions, where we aim to go beyond currently available network monitoring approaches also with respect to the inspection depth that our system allows: it is, using our methods, possible for users not only to see HTTP packets that are transmitted between devices, but also to inspect their contents in near real time. This has only rarely been done before, for instance in the *EtherPeg* tool[2] that can display a network graph and visualize image files as they are passed from one network node to another. Finally, our proposed network analysis software can not only be used by administrators to monitor information flows between devices, but also to control them using an approach that is known as *software-defined networking*: our system can be extended to give users the ability to control the flow of packets between devices in a smart home and remote services by configuring networking restrictions directly on their home networking hardware, by "cutting" visualized connections on the user's handheld device.

In the rest of this paper, we present our network monitoring and control system in greater detail, starting with a discussion of how it obtains the necessary metadata about traffic flows in the network. Next, we present two methods of how the collected data can be presented in an intuitive and visually appealing way: a Web view and an augmented reality view with device recognition and a live visualization overlay on a tablet device. Finally, we discuss potential use cases of our approach.

2 System Overview

Our system consists of a real-time logging component that records HTTP interactions between devices, a storage back end to persist recorded messages, and a front end to visualize device interactions for users (Fig. 2). A logging client is deployed on each observed device and forwards metadata about communication with others to the central logging back end. This component is responsible for storing the data about interactions and for forwarding them to connected visualization clients using HTML5 WebSockets.[3] Alternatively, visualization clients can also query the REST interface of the back end to obtain information about past interactions.

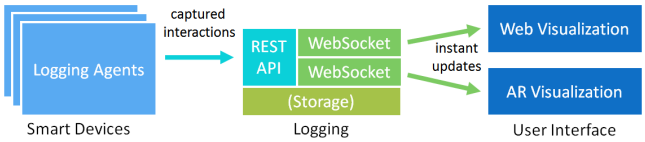

Figure 2: *System overview: Loggers are deployed on devices in a smart environment and record their interactions. This data is stored at a central back end and is pushed to visualization applications.*

We propose two different user interfaces to monitor device interactions: first, a HTML5 Web application that visualizes the captured interactions as a force-directed graph, and second, an augmented reality (AR) application on mobile devices. This AR application recognizes smart devices in the camera's field of view, associates the recognized devices with nodes in the interaction graph, and displays Web interactions between them as an overlay on the camera view (see Fig. 1).

3 Logging

In our system, metadata about device interactions is logged in a distributed way, where each Web server that is deployed on a smart device reports its interactions with remote endpoints to a central logging back end. It is therefore possible that the causal order of observed interactions at the back end is inconsistent with the global order of interactions between devices. However, since one main purpose of our system is to help users monitor the causes and effects of device interactions in their smart environment, it is important that the visualization of message flows between devices reflects the causal order of these interactions. To achieve a consistent ordering of messages, we record the causal relations between interactions using the vector clock algorithm [Mattern 1989]: The vector clocks are piggybacked on HTTP messages between devices and our logging software takes care of merging the local and incoming vector clocks upon receiving a request or response.

To also capture interactions with external services and unobserved devices (i.e., endpoints that are not under our control), our system uses a combination of server-side and client-side logging, where the concrete setup is chosen dynamically based on which of the two endpoints in an interaction is directly observed by our software: If *both* the server and the client are observed, we record the HTTP request on the server side and the HTTP response on the client side. If *only the client* is observed, it takes care of logging requests and responses from external services and unobserved devices that it interacts with. Conversely, if any observed node *receives* a request from an unobserved client, it logs that request/response pair.

3.1 Monitoring Interaction Chains

Since most Web interactions in a Web of Things context are part of a task that involves multiple HTTP requests and responses, our system does not only record isolated requests but aggregates them to *interaction chains*: all requests and responses within a chain are the consequence of a single initial HTTP request. By visualizing entire interaction chains, our tool thus enables users to inspect Web interactions *in their context*, for instance allowing them to better decide

[1] See `http://radiosilenceapp.com/private-eye`
[2] See `http://etherpeg.org/`
[3] See `http://www.w3.org/TR/websockets/`

whether a specific request to an external service was intended.

To enable our system to visualize series of interactions between devices, we assign each captured HTTP interaction to an *interaction chain*. Because it is not possible to understand causal relationships between messages by merely observing network communication,[4] our logging software adds an HTTP header entry that contains the identifier of the current interaction to each outgoing request. Additionally, whenever a logged Web server receives a request that contains an interaction chain identifier, it stores that ID and attaches it to each outgoing HTTP request that it generates while processing the incoming request. This technique represents a sound heuristic of assigning requests to interaction chains – it might, however, happen that a server generates a new request while processing another (and, thus, assigns the two requests to the same interaction chain) although the requests have no causal relationship. Likewise, new requests that are generated *after* the processing of another request has finished are assigned to different interaction chains although they might be related to the previously received request. Assigning requests correctly to interaction chains in these scenarios, however, would require our logging system to reconstruct the entire logic of applications running on the smart devices, which is not the focus of our project.

3.2 Intrusive and Non-intrusive Logging

In contrast to the non-intrusive sniffing of packets on the networking hardware, our approach requires that server applications attach the required HTTP headers to each outgoing request and merge vector clocks of incoming responses. To avoid burdening developers of smart devices with manually modifying their server implementations, our system uses *Java Agents* that modify the bytecode of Java classes using so-called *class transformers*. Using this approach, the logging client can be deployed to Web servers that are based on the Grizzly NIO framework[5] and use `HttpURLConnection` to make requests without the need of modifying a single line of code – instead, it is sufficient to attach our transformers when invoking the application (by using the *javaagent* JVM parameter).

We believe that this small burden of modifying the startup procedure of a deployed Web server is worthwhile for several reasons: by logging requests at the application layer, our system is able to trace the execution path of a request and thus can reconstruct interaction chains from HTTP messages. Furthermore, we gain access to the full URI of an endpoint and the message payload (in case it is unencrypted) which might be valuable to, for instance, automatically classify requests and warn users about interactions that seem suspicious. Finally, logging on the application layer is reasonable for several pragmatic reasons: A previous project where we non-intrusively logged device interactions by sniffing packets directly on network infrastructure devices [Mayer et al. 2012a] was in many cases unable to capture packets even on the network layer because routers attempt to handle communication already at the link layer for reducing processing overhead (i.e. we lose access to the source and destination IP addresses). With the mechanism presented here, we are not only able to differentiate between devices on the network level but even between different resources that are served by a single server which, in our opinion, by itself legitimates the approach of this paper.

4 Web Visualization Interface

To visualize logged interactions between devices, we implemented a HTML5 application that allows end users to inspect Web interactions between devices and remote services in real time and to replay recorded interactions. Our Web application visualizes device interactions as a force-directed graph where devices or services are represented as nodes and interactions between them are visualized as dots that travel on the edges of this graph (see Fig. 5). Our Web application makes use of several HTML5 Javascript APIs (HTML Canvas[6] and WebSockets). All the processing and filtering of the incoming data is executed on the client side. The server only stores and relays the captured interactions, the logging back end does not perform any pre-processing of the captured data.

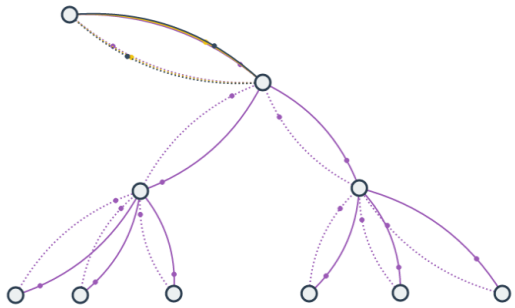

Figure 3: *The graph view of our application can visualize interactions of network endpoints in a force-directed graph. This figure shows a visualization of a search infrastructure (see [Mayer et al. 2012b]) whose nodes execute a distributed wave algorithm.*

Because the travel time of a message in a local network is only a few milliseconds and users would thus not be able to trace interaction chains, our application virtually increases the latency of each interaction to one second for the purpose of the visualization. Consequently, using the vector clocks that give a consistent causal ordering of interactions between devices, our application delays the visualization of some messages – still, the start of each new interaction chain corresponds to the actual beginning of that chain, in real time.

4.1 Web Visualization Modes

Our Web application provides two visualization modes, the Graph view and the Timeline view (see Fig. 5 A and B), as well as a "split view" that displays both next to each other.

Graph View The graph view of our Web application (see Fig. 3 and Fig. 5 A) renders and animates a dynamic force-directed graph where devices or services are represented as nodes and interactions between them are visualized as dots that travel on the graph's edges – the graph is aware of interaction chains: for instance, in Fig. 3, all interactions that are caused by the client's initial request to the top node of a distributed search infrastructure are assigned to the same chain. Interacting nodes exert an attractive force on each other and unrelated nodes are repelled. The graph topology is updated on each frame with the most recent interactions – nodes that do not participate in any interactions gradually disappear from the graph.

Timeline View The timeline view visualizes Web interactions between devices in chronological order by connecting send and receive events on different devices that are arranged along the y-axis of the timeline. This view can be used to inspect the exact order of

[4] This is due to the stateless nature of the HTTP protocol.

[5] See `https://grizzly.java.net/`

[6] See `http://www.w3.org/TR/2dcontext/`

the interactions. In contrast to the graph view, however, it does not provide any information about the topology of the communication.

4.2 Inspection and Filtering

Users can inspect details of each interaction (HTTP headers, status codes, and message payload) by clicking the corresponding edge in the graph and timeline views. They can also navigate to related interactions using previous/next buttons. Finally, the interface allows users to configure filters that change the visibility or color of interaction chains with specific properties that can be described using a simple domain-specific language. For instance, to filter all requests from the "TV" node to the `google.com` endpoint, the user would use this expression:

```
Request(from=TV)->Request(to="google.com")
```

Multiple such specifications can be combined using AND and OR operators to create custom interaction filters.

5 Magic Lens Visualization Interface

While the Web visualization gives a convenient overview of interactions between devices, we consider even more intuitive to use a personal mobile device that augments the user's environment with live connections. To enable this, we implemented a visual multi-object recognition technique that is able to detect several pre-defined objects in the camera view of a handheld device. The application then queries the logging back end for interaction metadata of the recognized devices and overlays the communication links on the camera frames (see Figs. 1 and 5), thereby transforming the handheld device into a "magic lens" [Bier et al. 1993].

5.1 Recognition pipeline

Out proof-of-concept visual recognition algorithm is able can recognize up to four distinct objects in the camera view simultaneously and we tested it with up to 12 object classes. The recognition was implemented using the OpenCV library.[7] We first resize the camera frames to a resolution of 320x240 pixels and then extract FAST9 keypoints [Rosten and Drummond 2006] with non-maximum suppression. To separate objects, we spatially cluster the keypoints with the DBSCAN algorithm [Ester et al. 1996] using a search radius of 30 pixels. Each keypoint cluster returned by DBSCAN is assumed to belong to a single object. For each keypoint within a cluster's bounding box (see Fig. 4) we extract SURF [Bay et al. 2008] descriptors. The object classes are then described in the Bag of Words model [Csurka et al. 2004]: the descriptors are quantized and a codebook of "visual words" is created. The description of each smart device is then a histogram computed over these words.

During the training phase we take 10-15 snapshots each smart device, process the images in the pipeline described above and train binary SVMs for each object. Later, at run time, camera frames are processed in the above pipeline and the SVMs determine the most probable object class in each bounding box of keypoint clusters. We assume that objects are sufficiently textured (i.e., contain at least 40 keypoints) and that there is little background clutter. We achieve a frame rate of about 1.8 FPS when having four devices recognized simultaneously on a Nexus 5 smartphone.

5.2 Limitations

Note that our approach does not track objects but re-detects them at subsequent frames therefore it cannot handle occlusions and cannot separate objects that are close to each other. Also, background clutter would cause the algorithm to fail because DBSCAN would return over-sized keypoint clusters that might contain more than one object. The algorithm cannot differentiate objects that look alike: for instance, recognizing cars represents an especially challenging task because of the *visual similarity* of distinct vehicles. Our algorithm can recognize specific cars only if unique features of that car are visible in the camera frame (and in the training images).

[7] `http://www.opencv.org`

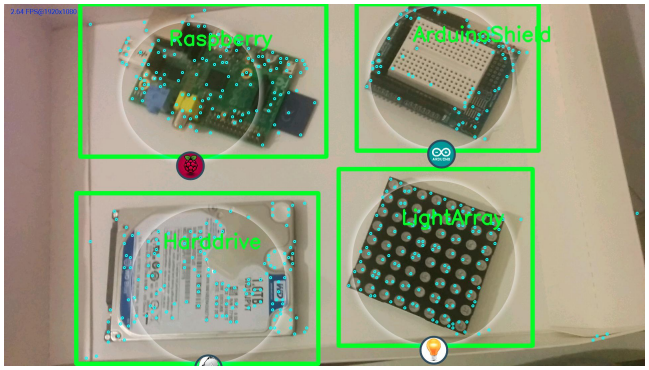

Figure 4: *Recognizing multiple objects in a plain background: extracted feature points are clustered spatially and each cluster's bounding box region is passed to the object classifiers.*

6 Applications

We envision the proposed system to be applicable in a number of use cases that range from educating students about distributed algorithms to enabling inhabitants of smart homes to track what kind of information is leaving the domestic environment, and where control commands to their devices come from. In this section, we describe five such scenarios to illustrate the merit of our proposed approach for visualizing Web interactions.

6.1 Smart Homes

Providing smart home inhabitants with a tool that enables them to stay aware of data that enters and leaves their home, as well as of interactions between smart devices within their domestic environment represents the original motivation for the work presented in this paper. Our system can clearly facilitate the monitoring of devices that handle privacy-sensitive data for end users, for instance with respect to smart electricity meters [Rial and Danezis 2011] – to find out which remote endpoints are accessing the domestic smart meter using our tool, users are merely required to point the camera of their tablet at the smart meter and observe the overlaid interactions (Fig. 5 C). Using the Web interface, users can additionally examine individual messages and play back earlier interactions between the domestic device and remote servers. Our system also enables users to monitor control commands that are sent to devices in their smart home, for instance to smart thermostats that are controlled by a remote service – this is especially relevant in scenarios where the control logic of appliances is provided by cloud services [Kovatsch et al. 2012]. Finally, our tool helps in monitoring interactions *between* devices in smart environments – while this is not demanded by users at the moment, we believe that the accelerating deployment of home automation solutions will make it increasingly relevant to track such "behind-the-back" communication. This is also important with respect to current research in the Web of Things domain that targets the ad-hoc creation of physical mashups based on user goals [Mayer et al. 2014]. In such scenarios,

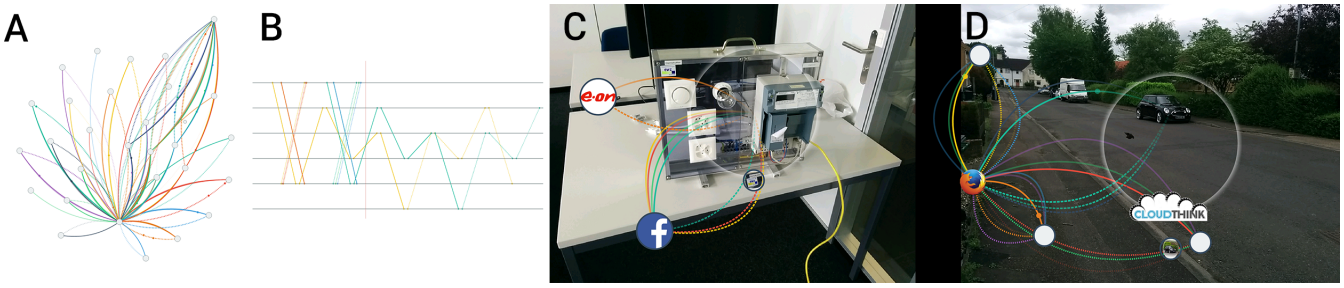

Figure 5: *A) The graph view of our Web visualization interface is useful for monitoring complex interactions between devices and services while its timeline view (B) emphasizes temporal relationships. C) Screenshot of the AR interface of our tool that visualizes interactions in smart homes in real time, for instance between a smart meter and utility companies or other parties that handle personal data. D) Screenshot that shows an overlay of interactions between clients and a smart car (as well as with three other endpoints).*

users should be provided with a tool that helps them to monitor and manage interactions between devices, if only to give them a feeling of being in control of their smart home.

6.2 Smart Factories

Production processes in factories increasingly involve dynamic interactions between individual manufacturing devices, to enable rapid reconfigurations which allow the process to evolve and adapt to the mass customization of products [Lastra and Delamer 2006]. The technologies proposed in this chapter could support operators within such environments to rapidly determine which devices talk to each other and what data is transmitted between them at any given moment. This is especially helpful when devices that are involved in a process start to be aware of their functionality, thus enabling the dynamic reconfiguration of the system at run time.

6.3 Smart Firewalls

With respect to both, smart homes and smart factories, we believe that it could be beneficial to combine the systems presented in this chapter with network infrastructure that can be remotely configured by users. In this case, if users discover an interaction between devices or with a remote service that they feel uncomfortable about, this connection could be instantly terminated and prevented in the future literally at the fling of a finger by configuring a firewall rule on a network router. This combination of software-defined networking with our augmented reality interfaces that display interactions holds great potential to facilitate not only network monitoring but also the *management* of device networks.

6.4 Smart Cars

Another scenario we find as an interesting use case for our system concerns smart cars where the ability to monitor interactions is relevant to protect drivers from attacks on their privacy such as location profiling or the misuse of private data by authorities [Dötzer 2006]. To demonstrate the applicability of our system in this domain, we combined the back-end server infrastructure of a vehicle-to-cloud communication platform called *CloudThink*[8] with our logging software. The combined system allows users to monitor interactions of clients with telemetry data from about 30 vehicles using the Web interface of our visualization tool as well as displaying these interactions as a live overlay on handheld devices (Fig. 5 D).

[8]The CloudThink project `http://cloudthink.mit.edu/` is a joint effort by Massachusetts Institute of Technology, ETH Zurich, the Singapore University of Technology and Design, and the Masdar Institute of Science and Technology to make vehicle telemetry data accessible to other applications on the Web, in near real time.

6.5 Smart Debugging and Education

We believe that our system could be applied in educational environments, similar to visualization tools for algorithms and data structures. Several studies in the domain of algorithm visualization have shown that students believe such tools support them in learning rather abstract programming methods such as sorting or line sweep algorithms, and examination results also reflect these benefits [Karavirta et al. 2010]. By visualizing interactions between distributed agents, our tool extends the scope of the broad domain of software visualization [Price et al. 1993] beyond individual programs. This is, for instance, relevant with respect to advanced topics in the domain of distributed algorithms (e.g., regarding leader election, distributed termination detection, and with respect to general wave algorithms, see Fig. 3). Our system can be used to illustrate these because it also takes care of preserving the causal relationships within interactions that stretch over multiple exchanged messages – at the same time, the tool itself represents a very convincing example to demonstrate the benefits of vector clocks in classrooms. We recommend using the Web visualization interface for these use cases – potentially, however, the magic lens interface of our visualization tool can be used in educational settings as well, for instance in robotics courses.

7 Conclusion

We presented a tool that gives end users immediate visual feedback about the Web interactions between devices within their smart environment. Our tool is easy to combine with current Web server technologies and allows to monitor HTTP messages that it aggregates to interaction chains to elicit the causes and effects of communications between smart devices and with remote services. We proposed a Web-based control interface and an AR visualization tool that recognizes devices and displays messages that are exchanged among them in real time as an overlay of the camera view of a handheld device. We believe our system will help to put users more in control of smart environments by enabling them to understand the communication of their devices, thus mitigating privacy concerns and increasing user acceptance of smart environments.

Future Work While our proof-of-concept prototype can recognize about a dozen smart objects, the applied rather simple object recognition is limited in terms of scalability and robustness. Our future work will focus on finding a recognition algorithm that is faster to compute and is more robust to background clutter. To overcome

the challenging task of differentiating similar-looking objects, we propose to make better use of context of the device that runs the recognition algorithm other than visual features in its camera feed. We particularly suggest to make use of the (indoor) location of the handheld device and also consider sensors such as its microphone, gyroscope, and accelerometer, broadening the scene understanding algorithm beyond the visual computing domain.

Acknowledgements The project has been partially funded by the Swiss National Science Foundation (grant number 134631).

References

BAY, H., ESS, A., TUYTELAARS, T., AND VAN GOOL, L. 2008. Speeded-Up Robust Features (SURF). *Computer Vision and Image Understanding 110*, 3, 346–359.

BECKEL, C., SADAMORI, L., AND SANTINI, S. 2013. Automatic Socio-Economic Classification of Households Using Electricity Consumption Data. In *Proceedings of the 4th International Conference on Future Energy Systems (ACM e-Energy 2013)*, ACM.

BIER, E. A., STONE, M. C., PIER, K., BUXTON, W., AND DEROSE, T. D. 1993. Toolglass and Magic Lenses: The See-through Interface. In *Proceedings of the 20th Annual Conference on Computer Graphics and Interactive Techniques*, ACM, SIGGRAPH '93, 73–80.

BRUSH, A. J., LEE, B., MAHAJAN, R., AGARWAL, S., SAROIU, S., AND DIXON, C. 2011. Home Automation in the Wild: Challenges and Opportunities. In *Proceedings of the ACM CHI Conference on Human Factors in Computing Systems (Vancouver, Canada)*, ACM, 2115–2124.

CSURKA, G., DANCE, C. R., FAN, L., WILLAMOWSKI, J., AND BRAY, C. 2004. Visual Categorization with Bags of Keypoints. In *Proceedings of the Workshop on Statistical Learning in Computer Vision (SLCV)*.

DÖTZER, F. 2006. Privacy Issues in Vehicular Ad Hoc Networks. In *Privacy Enhancing Technologies*, G. Danezis and D. Martin, Eds., vol. 3856 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 197–209.

ESTER, M., KRIEGEL, H.-P., SANDER, J., AND XU, X. 1996. A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. In *KDD*, 226–231.

GRINTER, R. E., EDWARDS, W. K., NEWMAN, M. W., AND DUCHENEAUT, N. 2005. The Work to Make a Home Network Work. In *Proceedings of the ninth conference on European Conference on Computer Supported Cooperative Work*, 469–488.

KARAVIRTA, V., KORHONEN, A., MALMI, L., AND NAPS, T. L. 2010. A Comprehensive Taxonomy of Algorithm Animation Languages. *J. Vis. Lang. Comput. 21*, 1, 1–22.

KEITH MITCHELL, NICHOLAS J. P. RACE, M. C. 2005. CANVIS: context-aware network visualization using smartphones. In *Proceedings of the 7th Conference on Human-Computer Interaction with Mobile Devices and Services (Mobile HCI 2005)*.

KLEIMINGER, W., BECKEL, C., AND SANTINI, S. 2011. Opportunistic Sensing for Efficient Energy Usage in Private Households. In *Proceedings of the Smart Energy Strategies Conference 2011*.

KOVATSCH, M., MAYER, S., AND OSTERMAIER, B. 2012. Moving Application Logic from the Firmware to the Cloud: Towards the Thin Server Architecture for the Internet of Things. In *Proceedings of the 6th International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS 2012)*.

LASTRA, J. L. M., AND DELAMER, I. M. 2006. Semantic Web Services in Factory Automation: Fundamental Insights and Research Roadmap. *IEEE Transactions on Industrial Informatics 2*, 1, 1–11.

MATTERN, F. 1989. Virtual Time and Global States of Distributed Systems. In *Proc. Workshop on Parallel and Distributed Algorithms*, North-Holland, C. M. et al., Ed., 215–226.

MAYER, S., BECKEL, C., SCHEIDEGGER, B., BARTHELS, C., AND SÖRÖS, G. 2012. Demo: Uncovering Device Whispers in Smart Homes. In *Proceedings of the 11th International Conference on Mobile and Ubiquitous Multimedia (MUM 2012)*.

MAYER, S., GUINARD, D., AND TRIFA, V. 2012. Searching in a Web-based Infrastructure for Smart Things. In *Proceedings of the 3rd International Conference on the Internet of Things (IoT)*, IEEE Computer Society, 119–126.

MAYER, S., INHELDER, N., VERBORGH, R., DE WALLE, R. V., AND MATTERN, F. 2014. Configuration of Smart Environments Made Simple - Combining Visual Modeling with Semantic Metadata and Reasoning. In *Proceedings of the 4th IEEE International Conference on the Internet of Things (IoT 2014)*.

MINARIK, P., AND DYMACEK, T. 2008. NetFlow Data Visualization Based on Graphs. In *Proceedings of the 5th International Workshop on Visualization for Computer Security*, Springer, 144–151.

POOLE, E. S., CHETTY, M., GRINTER, R. E., AND EDWARDS, W. K. 2008. More Than Meets the Eye: Transforming the User Experience of Home Network Management. In *Proceedings of the 7th ACM Conference on Designing Interactive Systems (DIS 2008)*, ACM, 455–464.

PRICE, B. A., BAECKER, R. M., AND SMALL, I. S. 1993. A Principled Taxonomy of Software Visualization. *J. Vis. Lang. Comput. 4*, 3, 211–266.

RANDALL, D. 2003. Living Inside a Smart Home: A Case Study. In *Inside the Smart Home*, R. Harper, Ed. Springer, 227–246.

RIAL, A., AND DANEZIS, G. 2011. Privacy-preserving smart metering. In *Proceedings of the 10th Annual ACM Workshop on Privacy in the Electronic Society (WPES 2011)*, ACM, 49–60.

RÖCKER, C., JANSE, M. D., PORTOLAN, N., AND STREITZ, N. 2005. User Requirements for Intelligent Home Environments: A Scenario-Driven Approach and Empirical Cross-Cultural Study. In *Proceedings of the 2005 Joint Conference on Smart Objects and Ambient Intelligence: Innovative Context-aware Services: Usages and Technologies*, ACM, 111–116.

ROSTEN, E., AND DRUMMOND, T. 2006. Machine Learning for High-Speed Corner Detection. In *Proceedings of the 9th European Conference on Computer Vision (ECCV 2006)*, Lecture Notes in Computer Science. Springer, May, 430–443.

SCHALL, G., ZOLLMANN, S., AND REITMAYR, G. 2013. Smart Vidente: advances in mobile augmented reality for interactive visualization of underground infrastructure. *Personal and Ubiquitous Computing 17*, 7, 1533–1549.