Utilising the Tor Network for IoT Addressing and Connectivity

Felix W. Baumann[1,2], Ulrich Odefey[1], Sebastian Hudert[1],
Michael Falkenthal[3] and Uwe Breitenbücher[3]
[1]*TWT GmbH Science & Innovation, Ernsthaldenstr. 17, D-70565 Stuttgart, Germany*
[2]*Institute of Computer-aided Product Development Systems, University of Stuttgart, Universitätsstr. 38, D-70569 Stuttgart, Germany*
[3]*Institute of Architecture of Application Systems, University of Stuttgart, Universitätsstr. 38, D-70569 Stuttgart, Germany*

Keywords: Tor Network, IoT Connectivity, Internet of Things, Addressing.

Abstract: For Internet of Things (IoT) devices and cyber-physical systems (CPS), it is required to connect them securely and reliably to some form of cloud environment or computing entity for control, management and utilisation. The Internet is a suitable, standardized, and proven means for the connection of IoT devices in various scenarios. Connection over the Internet utilises existing protocols, standards, technologies and avoids investment in new, specialised concepts. Thereby, this connection requires a transparent addressing schema which is commonly TCP/IP, using domain names and IP addresses. However, in industrial, commercial and private networks, the addressability and connectability/connectivity is often limited by firewalls, proxies and router configurations utilising NAT. Thus, the present network configurations hinder the establishment of connections between IoT devices across different locations. Therefore, the method for connecting IoT devices in a client-server configuration proposed herein utilises the Tor (previously: The onion router/routing) network for addressing of and secured communication to IoT and CPS devices. It is an overlay protocol that was designed to allow for robust and anonymous communication. The benefit of this approach is to enable addressability and connectivity of IoT devices in firewalled and potentially unknown and changing network environments, thus allowing for IoT devices to be used reliably behind firewalls as long as outgoing communication is not blocked.

SCIENCE AND TECHNOLOGY PUBLICATIONS

1 INTRODUCTION

In scenarios where IoT (Internet of Things) systems are to be deployed, it is essential to address these devices reliably even if they are located behind corporate or other firewalls. The reliable addressing across various network scenarios for IoT devices is the key problem to solve. Furthermore, connectivity to and of such devices can be problematic due to restrictive network settings. This work is motivated by the *SePiA.Pro* project (Deutsches Forschungszentrum für Künstliche Intelligenz et al., ; Pfeil et al., 2016; Falkenthal et al., 2017), which provides such a scenario. In this project, industrial machinery is equipped with sensors to form cyber-physical systems (CPS) and enhanced with so called smart services, i. e., services operating on data generated by these CPS in a cloud-like manner (Falkenthal et al., 2016a). A special component in these smart services is a so called *DataHub*, which facilitates unified and secure access to and usage of various data sources and targets within smart service and industrial environments. These *DataHubs* have to be addressable and stackable within networks and over network boundaries for cross and inter-company access. Addressing and communication with these CPS devices is facilitated with the proposed mechanism, relying on a client-server architecture, operating on the *Tor* network (McCoy et al., 2008; Jansen et al., 2012). The solution provided here solves the issue of addressing and connecting CPS/IoT devices via a client-server system relying on the *Tor* network, it is motivated by a scenario of the *SePiA.Pro* project and described through its implementation in code. For an introduction to the nature and structure of the *Tor* network, section 3 is provided. The proposed mechanism is implemented utilising a *BASH* (Free Software Foundation, Inc.,) script for the client part, intended to be executed on a micro computing platform, in this case a *Raspberry Pi 2* (Raspberry Pi Foundation,). The server part is implemented as a RESTful (Represen-

Baumann, F., Odefey, U., Hudert, S., Falkenthal, M. and Breitenbücher, U.
Utilising the Tor Network for IoT Addressing and Connectivity.
DOI: 10.5220/0006591500270034
In *Proceedings of the 8th International Conference on Cloud Computing and Services Science (CLOSER 2018)*, pages 27-34
ISBN: 978-989-758-295-0
Copyright © 2019 by SCITEPRESS – Science and Technology Publications, Lda. All rights reserved

tational state transfer) API (Application programming interface) using the *LoopBack* library.

The remainder of this paper is structured as follows: This implementation is described in section 4, with the architecture discussed in Sec. 4.1 and the implementation details in Sec. 4.2. The implementation in and relationship with the *SePiA.Pro* project, in particular its

DataHub component is presented in Sec. 5. The *DataHub* component of the *SePiA.Pro* project is described in Baumann et al. (Baumann et al., 2017) and Falkenthal et al. (Falkenthal et al., 2017). This component controls, limits, manages, and facilitates the access and usage of data sources and targets in Industry 4.0 scenarios. This work concludes with a summary in Sec. 6.

2 RELATED WORK

Guth et al. (Guth et al., 2016) investigated several state-of-the-art IoT platforms and deduced a general and technology-agnostic IoT reference architecture. The presented approach in their work can be used to implement the abstractly described connections of their reference architecture between devices, gateways, IoT integration middlewares, and applications. While their analysis considers many present state-of-the-art protocols for connecting IoT devices to middleware systems and applications, the approach to overcome the ever-present difficulty of addressing and connecting IoT devices as presented in this work extends their list of investigated protocols.

Reinfurt et al. (Reinfurt et al., 2016; Reinfurt et al., 2017) capture the conceptual essence of different fields of IoT research into patterns. Thereby, they provide proven nuggets of advice about the design and characteristics of IoT systems. Our approach provides concrete solution implementations (Falkenthal et al., 2014) concerning different abstract solution principles which are described, e. g., by the patterns *Device Gateway*, *Remote Device Management* and their various different patterns for IoT devices.

Previous work by Baumann et al. (Baumann et al., 2016c; Baumann et al., 2016a; Baumann et al., 2016b) provides architectural compositions and implementations of distributed control systems for the use case of 3D printing in remote environments. These systems are based on unified control of 3D printing devices and associated adaptors via Internet based protocols. Prior works focus primarily on the server and communication components of these systems.

3 FUNDAMENTALS OF THE Tor NETWORK

The Tor network was developed around the mid 1990s and released in 2002, as an overlay protocol to allow safe, encrypted, and anonymous usage of the Internet and its services, such as web browsing, email, chat and file-sharing (Chaabane et al., 2010). Originally the name was used as an acronym, the onion router or routing, indicating the underlying design principle: onion routing (Haraty and Zantout, 2014). This routing principle is based on the nesting of data packages within each other that are partially unpacked by stations along the communication path to recover the encrypted package destined for the next hop or destination. The stations along the way are called *Tor* relays, more specifically *middle relays*, *bridges* (private relays not generally known to the public) and *exit* or *end relays*. Based on these nodes the Internet traffic is routed over several relays to its final destination in a way that the party on the other end of the connection can't trace the traffic back to the source. This principle ensures the anonymity of participants in the Tor network, which has reportedly been employed by journalists, law enforcement agencies but also for drug trafficking, the distribution of illegal content over the Internet and other illicit digital actions. Today the *Tor* network consists of several thousand relays (2017-07-20: 6914 relays with 12.3 $\frac{GByte}{s}$ bandwidth[1]), several hundreds of which are exit nodes (2017-07-20: 794 exit nodes), the relays connect a given request to its final destination in the Internet. The number of daily users was around 4 million (Dredge, 2013) in 2013. *Tor* uses a data structure that can potentially be identified and filtered by firewalls, but also provides means to obfuscate its traffic to avoid detection and disruption (The Tor Project,).

Furthermore, Tor allows the consumption of so called Hidden Services, which are only exposed and reachable within the *Tor* network and only by the users who connect their systems to the *Tor* network using a client software. These services are identified by a `.onion` address, a 16 alpha-semi-numeric character of 80 bit length. This domain was declared a special use domain in 2015 by the IETF (Willis, 2015).

The main drawback of the *Tor* network is performance. The principle of onion routing with all its relaying steps and the cryptographic operations involved slow down the experienced network speed. On the other hand, although *Tor* network ensures

[1]Tor Network Status http://torstatus.blutmagie.de

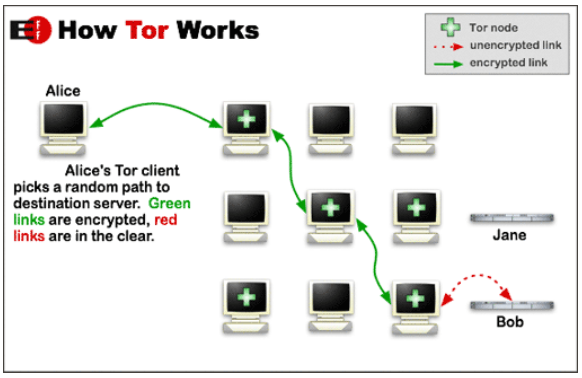

Figure 1: Working Principle of the Tor Network; Figure source: https://commons.wikimedia.org/wiki/File:Tor-onion-network.png; author: Electronic Frontier Foundation (EFF); licence: CC BY 3.0 US.

anonymity for Internet connections to a high degree, it is not a perfect system (Dingledine, 2011). There have been several successful attacks (mostly from law enforcement agencies while investigating for criminal activities) on the *Tor* browser, the most vulnerable part on the *Tor* system. The mere usage of an anonymity software can make participants move into the focus of law enforcement agencies.

So what value does *Tor* bring to our work? *Tor* provides the technological basis for static addressing within an overlay and some sense of firewall agnostic communication layer. Furthermore, it facilitates an additional layer of security, i. e., encryption, between the communicating parties, i. e., the server and the client. A requirement for the applicability of this method is that the client system, the CPS or IoT device, must be able to establish outgoing connections through the firewall to the Internet. The system is designed to be mostly uni-directional, polling or requesting data from the server, thus, the communication is coming from the client behind the firewall, mimicking ordinary user behaviour such as web surfing, which is likely to be allowed by the network configuration. The *Tor* network is actively researched (Ren and Wu, 2010) and developed, thus, making it more likely that drawbacks and flaws in the system getting fixed within reasonable time.

4 Tor-BASED ADDRESSING AND COMMUNICATION FOR IoT DEVICES

The system is composed of two parts, with the client component deployed to and operated on a micro-computing system, such as a *Raspberry Pi 2*, and the server component operating on another system, e. g., in the cloud. The server component is implemented as a RESTful API using *LoopBack* (StrongLoop,) for *NodeJS* (Node.js Foundation,). The server exposes and offers its functionality over HTTP (Hypertext transfer protocol) and is described in detail in the publications referred as (Baumann et al., 2016c; Baumann et al., 2016a; Baumann et al., 2016b). Only the necessary basic concepts of the server component are described herein. The conceptual and architectural composition of the system is described in the following section (4.1). A third component, the *DeploySupport* is implemented to facilitate the deployment and installation of the system and described in Sec. 4.2. The system has security as a key design principle with the utilisation of separated execution environments on the micro computing system, cryptographic signing and securing of data during transport and using secured data channels. The server component creates a *Tor* hidden service to which it binds its RESTful HTTP API for connection. A *Tor* hidden service, is a service within the *Tor* network, that provides a certain functionality within the network by a local service, ordinarily offered over the Internet, such as a HTTP server, a website or a chat service.

The micro computing platform on which this system (the client) is deployed, is a *Raspberry Pi 2*. In this figure, the system is connected to an Ethernet cable (red cable) and with USB (black cable) to an external device, a 3D printer. Furthermore, the platform is connected with a USB serial connection to a host system for debugging purposes. In operation, this USB serial connection cable is removed.

To further allow control of the platform, besides the client-server model described in the following, a SSH server is utilised on the client. On the first startup of the system, passwords for the user and administrator are generated randomly. These passwords are supplied to the server in encrypted form, along with other installation information.

4.1 System Architecture

Fig. 2 depicts an overview of the implemented system. In this figure, the CPS consists of the 3D printer, associated sensors, the micro computing system, i. e., the *Raspberry Pi 2*, and their digital representation and controlling capability. The system depicted here is communicating over the *Tor* network over the Internet. In the figure, the red text *¡ENC¿* is supposed to indicate an encrypted and secured channel through the Internet. The central server component is deployed to a system reachable over the Internet, e. g., a cloud-based hosting system. The *Tor* network provides addressability based on cryptographic hashes,

as indicated by the text *Tor Address* in the figure. These hashes are unique and identify components. The client is periodically polling the server for information and instructions to be executed on the micro computing system. All communication is initiated by the client and targeted outwards of the network. The server provides work instructions as work instruction packages in a queue, which is synchronised with an information storage component to ensure the consistency of instructions. Conceptually, the work instruction packages are self-contained compressed packages, that are cryptographically signed and encrypted. In these packages, the required instructions, software, and data can be included, thus, enabling function shipping.

Fig. 3 depicts the composition and distribution of the components of the system. The *DeploySupport* component consists of *BASH* scripts, that facilitate installation and configuration of the client system on the micro computing system. These components are described in the following at a high level.

The installer acquires an image of the latest *Arch-Linux OS* distribution and prepare a storage medium for use in the micro computing system with it. In our case, this storage medium is an SD-card. Alternative deployment strategies and mechanisms, such as *Tosca4IoT* (Ebner et al., 2017), can be used to deploy the system onto the micro computing system. For further iterations of this software, the extension and usage of alternative, automated and controlled, deployment strategies and mechanisms is intended.

4.2 Prototypical Implementation

Conceptually the client is created from one central, continually running loop, which checks for internal state, such as connectivity and utilisation, and performs subsequent actions.

The initial action of the client, prior to entering the control loop, hardens its execution environment to make it more robust against attacks.

The following action of the client checks for the required binaries and scripts in the respective paths of the operating system. In case the required binaries and scripts are unavailable, the execution of the client is terminated. If all required software is present, then the configuration information is processed. Configuration information is stored locally to enable consistent utilisation over reboots and system interruptions. The configuration consists of information such as a unique identifier, the remote endpoints and encryption components. The main loop of the client repeatedly checks the following information in the specific order:

- Status of the client daemon itself; checking if the client software is still running on the system and if the watchdog is present. The client watchdog is a component to prevent the client from hanging and restarts the system in case of aborts or errors.
- Connectivity; checking the connectivity of the system in general, i. e., checking if a network connection is established, if name resolution can be performed, if certain hosts on the Internet are reachable and if the central server can be connected with.
- Remaining disk space and space utilisation; Checking if sufficient disk space for the downloading of work instruction packages is available. In case the disk space is low and insufficient, further execution, i. e., fetching of instructions, is paused and this status is communicated with the central server.

In case the periodic checks are completed sufficiently, the client polls the central server for work. This polling for work queries a specific RESTful endpoint and the result of this poll is either the information that currently no work is expected to be performed or information on the number of work packages and logical location of the work packages. If the client has work to perform, the information on the existing work packages is used to query for the work packages on another specific RESTful endpoint. The work package is then transferred over an encrypted HTTP transport over *Tor* and stored locally. The work package is decrypted, deflated and checked for integrity and correctness. In case the correctness and integrity checks are sufficient, the work package is deployed into a jailed environment, for additional security, and executed. During the execution, the output of the work package is acquired and, alongside, the resulting information of the work package, stored for historical and analytical purposes. This information is submitted to the server with additional information. The client execution environment then cleans the temporary and execution folders to avoid space issues and data leakages.

Independent of this main loop there is a secondary loop executed periodically. This secondary loop acquires status information on the host operating system and the micro computing system in general and submits this information to the central server which can be considered health data. This data includes information on the system's uptime, its CPU and memory utilisation, the status of network adapters and memory devices. It is executed completely independent of the main loop.

As depicted in Fig. 3, the *Execution Engine* is part of the client package and executes the acquired work packages locally. For the execution of the

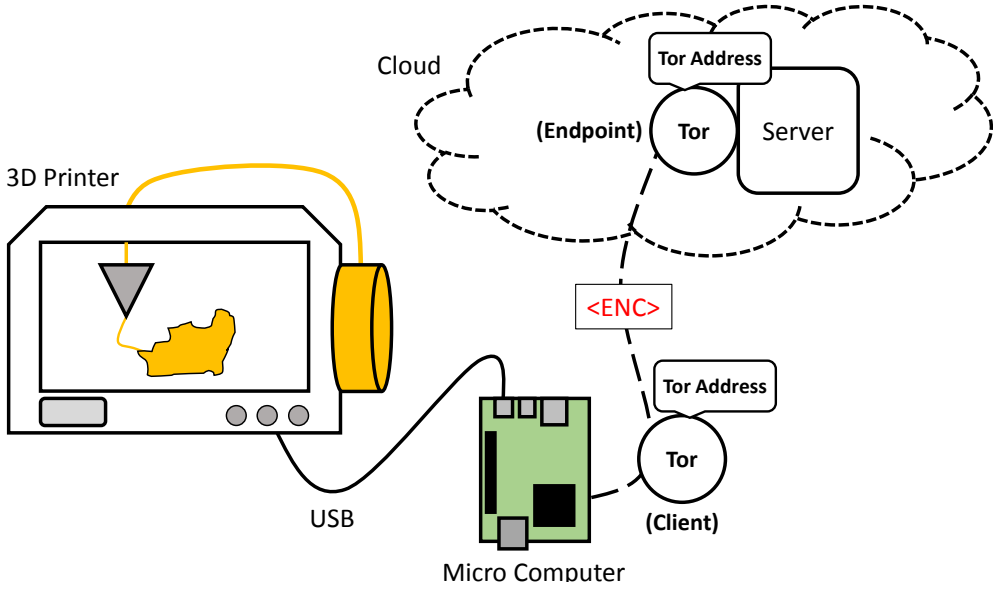

Figure 2: Overview Structure of Implemented System.

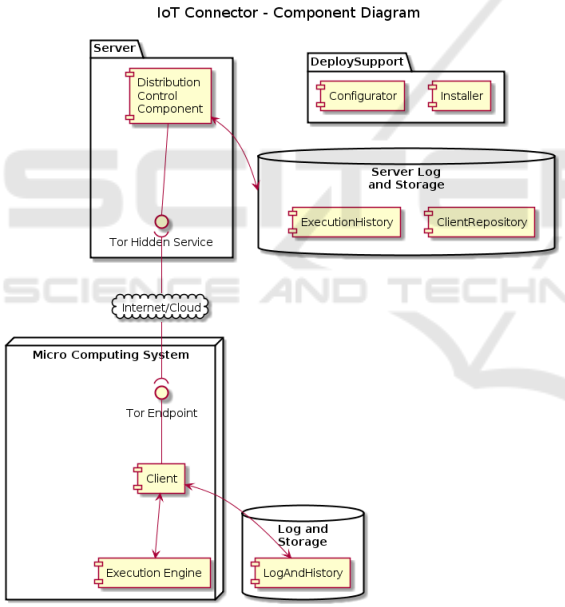

Figure 3: Component Structure of Implemented System.

work packages, a specific Linux jail environment using *chroot* is created and the execution is performed therein. These separate and distinct environments are utilised to avoid corruption of the system by malicious or malformed work package content. The initial creation of a prototypical jail environment is performed by the *Installer* and the *Configurator* as part of the *DeploySupport* component as depicted in Fig. 3. The *Log and Storage* components on the server and the client are implemented as file and directory based storage systems, to hold the historical execution output from the work packages and the operation of the client and server itself. The implementation of the client is a *BASH* script and consists of approximately 1900 lines of code. The code is available on *GitHub* and licensed as CC-BY. See https://github.com/baumanfx/RaspberryPi_Client for more details on the client software. In the following pseudo code, the structure of the client execution is depicted:

```
harden();
statusHealthLoop & // The ampersand
    indicating background/parallel
    operation
checkBinaries();
if ( binaryMissing ) { abort(); }
readConfig();
if ( noConfigFound ) { createNewConfig();
    }
while ( true ) { // main loop
  checkClient();
    if ( clientProblem || watchdogFailed )
        {
     solveProblem(); // e.g. restart
         watchdog or client
    }
    checkConnectivity();
    if ( noConnectivity ) {
     continue; // skip execution of loop
         and start anew
    }
    checkDiskSpace();
    queryServerForWorkpackage();
    if ( workpackageAvailable &&
        diskSpaceAvailable ) {
     downloadWorkpackage();
       unzipWorkpackage();
       checkIntegrityOfWorkpackage();
```

```
            deployWorkpackageToJail();
            executeWorkpackage();
            aquireLogAndStatus();
            submitWorkpackageInformationToServer();
            cleanUp();
}   }   }
```

The server component is adapted from another project, a cloud-based 3D printer control system (Baumann et al., 2016c; Baumann et al., 2016a; Baumann et al., 2016b), which was designed to distribute work packages to micro computing platforms that directly controlled attached 3D printers. The adaption for the *SePiA.Pro* project is that the work packages now contain execution instructions for the acquisition of sensor data from CPS and control and management instructions for the control of industrial machinery over standardized protocols, e. g., *OPC UA* (Open Platform Communications Unified Architecture (Leitner and Mahnke, 2006; TC 65/SC 65E, 2015)).

Furthermore, the components for the installation and configuration of the system are available on the same repository. The components for the central server component are published separately and have been described in previous publications.

Further supplementary material include service definition scripts for the *systemd* system on *ArchLinux OS*, exemplary udev rule files for the unified access of a 3D printer and webcams to the system.

5 INTEGRATION TO DataHub

The problem to be solved with this work is motivated by the research project *SePiA.Pro* (Deutsches Forschungszentrum für Künstliche Intelligenz et al.,) which aims to integrate industrial machinery to smart services. In this project, the addressability and connectivity of the machines or their CPS representation is a problem that is solvable via the proposed client-server system. The project aims to solve challenges in Industry 4.0. CPS in Industry 4.0 settings demand a seamless integration of a potentially huge set of data sources, potentially across geographic (such as production plants) or even organisational boundaries (for example when optimizing a whole supply chain) for large scale data-analysis to support concepts such as predictive maintenance. A data integration solution, explicitly addressing such scenarios is currently developed within the *SePiA.Pro* research project (Deutsches Forschungszentrum für Künstliche Intelligenz et al.,).

The *DataHub* in this project is a component that builds upon a modular architecture for integrating various data sources into one overarching hierarchical meta model. Each data source (e. g., sensor or even enterprise database) thus "exposes" only a projection of the overall meta model. By employing such a meta-model integration approach, we can define and enforce aggregation, access control, and data redundancy together with master/slave definition and synchronisation topics in a central manner, simply at the *DataHub* node. Designing the *DataHub* nodes self-referencable and equipping them with automated policy enforcement functionality (Baumann et al., 2017) allows us to create hierarchical systems of data integration nodes, covering intra as well as secure inter company data analytics.

System architectures as depicted in this paper integrate perfectly into such a system of integrated data analytics. Sensors and other CPS devices can be addressed in a stable manner, which becomes even more of a necessity and at the same time a technical problem in cross-company scenarios. Also the stable connectivity across company boundaries and thus firewalls or NAT (network address translation) configurations is also essential for the depicted setting.

The *DataHub* and its integration in the *SePiA.Pro* project and the connection to this proposed system is depicted in Fig. 4. In this figure, the different connection mechanisms from industrial machinery to the *DataHub* component are depicted by three different exemplary machines of which two are connected using the proposed client-server model over the *Tor* network. The figure depicts components (Trust Center, Smart DataHub, IoT/CPS, Models, Data, Machines) and information flows (Meta Data, Operating Data, etc.) between the components. The *Trust Center* component in this figure is an authoritative instance that controls, defines, and manages access and usage rights on all further components and acquires the relevant models and data from stakeholders. The bi-directional access to the industrial machines is indicated by the dual-arrows. Data that is acquired and processed includes machine and environmental meta data, operating data, and the associated instructions for the machines and sensors.

6 CONCLUSION & FUTURE WORK

In this publication, a method to address CPS or IoT devices based on the *Tor* network was proposed and described. The description was performed in a high-level manner based on the architecture and through an example. The code for this client-server system is published on a public repository and available for

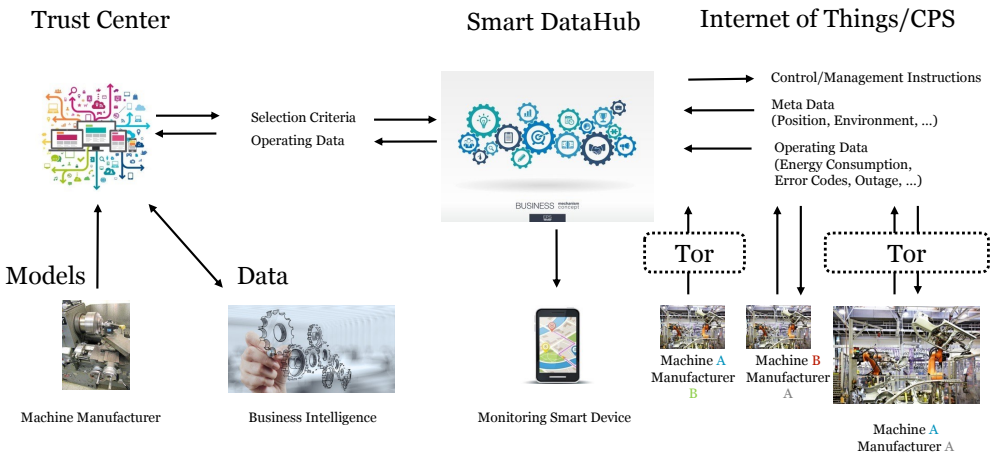

Figure 4: Smart DataHub Component and Relationship to Project and Proposed Client-Server Model.

study and application. The addressing of the client and communication with the server, as described, relies on the systems capability to make outbound connections through a firewall. The system presented here enables the usage of remote systems behind firewalls and in potentially unknown networking environments. The system is motivated by the *SePiA.Pro* project and its central component, the *DataHub*.

In future work we plan to automate the provisioning and management of whole IoT integration scenarios including IoT devices and middleware, such as presented in this work, but also data processing components, such as analytics platforms as demonstrated by Falkenthal et al. (Falkenthal et al., 2016b).

REFERENCES

Baumann, F. W., Eichhoff, J., and Roller, D. (2016a). *Collaborative Cloud Printing Service*, pages 77–85. Springer International Publishing.

Baumann, F. W., Falkenthal, M., Breitenbücher, U., Grünert, G., and Hudert, S. (2017). Industrial Data Sharing with Data Access Policy. In *Proceedings of the 14th International Conference on Cooperative Design, Visualization and Engineering*. In Press.

Baumann, F. W., Kopp, O., and Roller, D. (2016b). Abstract API for 3D Printing Hardware and Software Resources. *International Journal of Advanced Manufacturing Technology*, pages 1–17.

Baumann, F. W., Kopp, O., and Roller, D. (2016c). Universal API for 3D Printers. In Mayr, H. C. and Pinzger, M., editors, *INFORMATIK 2016. Lecture Notes in Informatics (LNI)*, volume P-259, pages 1611–1622. Gesellschaft für Informatik.

Chaabane, A., Manils, P., and Kaafar, M. A. (2010). Digging into Anonymous Traffic: A Deep Analysis of the Tor Anonymizing Network. In *2010 Fourth International Conference on Network and System Security*, pages 167–174.

Deutsches Forschungszentrum für Künstliche Intelligenz, Universität Stuttgart – Institut für Architektur von Anwendungssystemen, TWT GmbH Science & Innovation, Daimler AG, and TRUMPF Werkzeugmaschinen GmbH & Co. KG. SePiA.Pro – Service Plattform für die intelligente Anlagenoptimierung in der Produktion. Last accessed on 13th July 2017.

Dingledine, R. (2011). *Tor and Circumvention: Lessons Learned*, pages 485–486. Springer Berlin Heidelberg, Berlin, Heidelberg.

Dredge, S. (2013). What is Tor? A beginner's guide to the privacy tool. *The Guardian*.

Ebner, T., Meissler, M., and Steiler, F. (2017). TOSCA4IoT.

Falkenthal, M., Barzen, J., Breitenbücher, U., Fehling, C., and Leymann, F. (2014). From pattern languages to solution implementations. In *Proceedings of the 6th International Conferences on Pervasive Patterns and Applications*, pages 12–21. Xpert Publishing Services.

Falkenthal, M., Baumann, F. W., Grünert, G., Hudert, S., Leymann, F., and Zimmermann, M. (2017). Requirements and Enforcement Points for Policies in Industrial Data Sharing Scenarios. In *Proceedings of the 11th Symposium and Summer School On Service-Oriented Computing*. In Press.

Falkenthal, M., Breitenbücher, U., Christ, M., Endres, C., Kempa-Liehr, A. W., Leymann, F., and Zimmermann, M. (2016a). Towards Function and Data Shipping in Manufacturing Environments : How Cloud Technologies leverage the 4th Industrial Revolution. In *Proceedings of the 10th Advanced Summer School on Service Oriented Computing*, pages 16–25. IBM Research Division.

Falkenthal, M., Breitenbücher, U., Képes, K., Leymann, F., Zimmermann, M., Christ, M., Neuffer, J., Braun, N., and Kempa-Liehr, A. W. (2016b). Opentosca for the 4th industrial revolution: Automating the provisioning of analytics tools based on apache flink. In *Proceedings of the 6th International Conference on the Internet of Things*, IoT'16, pages 179–180. ACM.

Free Software Foundation, Inc. GNU Bash. Last accessed on 17th July 2017.

Guth, J., Breitenbücher, U., Falkenthal, M., Leymann, F., and Reinfurt, L. (2016). Comparison of iot platform architectures: A field study based on a reference architecture. In *2016 Cloudification of the Internet of Things*. Xpert Publishing Services.

Haraty, R. A. and Zantout, B. (2014). The TOR data communication system. *Journal of Communications and Networks*, 16(4):415–420.

Jansen, R., Bauer, K., Hopper, N., and Dingledine, R. (2012). Methodically Modeling the Tor Network. In *Proceedings of the 5th USENIX Conference on Cyber Security Experimentation and Test*, CSET'12, pages 1–9, Berkeley, CA, USA. USENIX Association.

Leitner, S.-H. and Mahnke, W. (2006). OPC UA - Service-oriented Architecture for Industrial Applications. *Softwaretechnik-Trends*, 26.

McCoy, D., Bauer, K., Grunwald, D., Kohno, T., and Sicker, D. (2008). *Shining Light in Dark Places: Understanding the Tor Network*, pages 63–76. Springer Berlin Heidelberg, Berlin, Heidelberg.

Node.js Foundation. Node.js. Last accessed on 13th July 2017.

Pfeil, M., Odefey, U., Ritter, Y., Schuermann, M., and Fäßler, V. (2016). Smart services - the smart implementation of Industry 4.0. In *NAFEMS Seminar Simulation von Composites – Bereit für Industrie 4.0?*

Raspberry Pi Foundation. Raspberry Pi - Teach, Learn, and Make with Raspberry Pi. Last accessed on 13th July 2017.

Reinfurt, L., Breitenbücher, U., Falkenthal, M., Leymann, F., and Riegg, A. (2016). Internet of things patterns. In *Proceedings of the 21st European Conference on Pattern Languages of Programs*. ACM.

Reinfurt, L., Breitenbücher, U., Falkenthal, M., Leymann, F., and Riegg, A. (2017). Internet of things patterns for devices. In *Ninth international Conferences on Pervasive Patterns and Applications*, pages 117–126. Xpert Publishing Services.

Ren, J. and Wu, J. (2010). Survey on anonymous communications in computer networks. *Computer Communications*, 33(4):420–431.

StrongLoop. The Node.js API Framework. Last accessed on 17th July 2017.

TC 65/SC 65E (2015). IEC 62541-100:2015 OPC Unified Architecture - Part 100: Device Interface. IEC 62541-100, International Electrotechnical Commission.

The Tor Project. Tor: Pluggable Transports. Last accessed on 17th July 2017.

Willis, N. (2015). Tor's .onion domain approved by IETF/IANA. Last accessed on 17th July 2017.

SCITEPRESS SCIENCE AND TECHNOLOGY PUBLICATIONS