Computers in Human Behavior 89 (2018) 385–394

Contents lists available at ScienceDirect

Computers in Human Behavior

journal homepage: www.elsevier.com/locate/comphumbeh

Full length article

Enabling visual community learning analytics with Internet of Things devices

István Koren*, Ralf Klamma

RWTH Aachen University, Advanced Community Information Systems (ACIS), Informatik 5, Ahornstr. 55, 52056 Aachen, Germany

ARTICLE INFO

Keywords:
Visual analytics
Internet of Things
Community of practice

ABSTRACT

Industry 4.0 is currently transforming industrial workplaces into sensor-assisted high-tech environments. While it is often feared that jobs will be lost through increasing automation, we are convinced that there is an enormous potential for versatile and competent workers at innovative workplaces. Training at the workplace may contribute to this. In this context, both body-near wearables and industrial devices produce an enormous amount of data. However, the question is how to deal with this amount of data. To this end, visual analytics is a combination of computational techniques from data mining and machine learning with human perceptional methods from human-computer interaction. In this article we present a method and tool support to create rich and interactive visual analytics charts to analyze innovative training solutions in high-tech workplace settings. This brings together the computer's capacities to handle large data and calculation resources with the human ability to quickly grasp relationships. Our technical evaluation shows, that the approach is feasible from a computational perspective; usability tests revealed that the developed pipeline metaphor reaches its goal. Our results may help in designing future systems that fulfill the needs of both trainers and learners in Industry 4.0 settings.

1. Introduction

The emerging paradigm shift to digital Industry 4.0 workplace settings has several profound effects like the need to restructure entire manufacturing and training processes. The term Industry 4.0, originally introduced by a German government initiative, describes the interconnectedness of manufacturing devices on the shop floor, and resulting applications such as asset management and predictive maintenance (Lasi, Fettke, Kemper, Feld, & Hoffmann, 2014). The process of creating digital twins of physical artifacts, human interactions and even complex workflows is thereby known as digitization. It may not only speed up innovation cycles, but also open new opportunities for innovative businesses. On the downside, digitization is often seen as a boogey man and threat for industrial workforces, saying that on the long run, technologies like robots will replace human workers. Undeniably, the constant stream of innovation puts pressure on workers to acquire new skills to deal with new workflows and new machines. Hereby, continuous training plays a special role. Constant evaluation and self-assessment of the training helps to render it more effective and useful. Therefore, it is advantageous that in these high-tech Industry 4.0 settings, both industrial devices and wearable devices of workers collect huge amounts of information. Combined, the gathered data is a valuable asset for further analysis in order to provide insights into human-machine interactions; additionally it may be used for replaying training data to future workforces. A challenge of these workplace settings is that the educational context is not necessarily tied to a particular location and time, but instead may happen at different places over various periods of times.

On the technical level, there are many hurdles on the way to make sensor data a valuable asset in analyzing Industry 4.0 learning settings. The obstacles are manifold and include ethical and privacy issues, as well as more technically, synchronization of different data sources and limitations of the devices involved. On a profound level, the large number of network- and application-layer protocols make it hard to access smart devices. Regarding the technical interplay of devices and information systems, Web standards have already helped bridging the gap between various mobile platforms in the last years: any mobile device, smartphone, tablet computer or smart watch is able to display HTML5 Web pages with dynamic JavaScript content. Beyond the user interaction aspect, Web standards are tremendously useful for connecting to arbitrary devices. A number of protocols for the Internet of Things (IoT) provide bindings for making communication available to browsers; examples are Websockets endpoints of XMPP and MQTT systems. While technical challenges on the protocol and device level are one side of the coin, the other side conceals further obstacles, namely the large amount of data. Besides the acquisition level, further

* Corresponding author.
E-mail addresses: koren@dbis.rwth-aachen.de (I. Koren), klamma@dbis.rwth-aachen.de (R. Klamma).

https://doi.org/10.1016/j.chb.2018.07.036
Received 16 February 2018; Received in revised form 11 July 2018; Accepted 23 July 2018
Available online 02 August 2018

challenges include privacy, the format, semantics and the interpretation of data (Labrinidis & Jagadish, 2012), rendering it difficult to grasp trends and understand correlations. To this end, *visual analytics* combines the capability of machines to collect and provide immense amounts of data with the ability of the human brain to visually structure and classify data. The goal of visual analytics is to gain knowledge through model building, visualize it dynamically and then feed back the acquired evidence into the data gathering and filtering process.

Finally, a critical view needs to be taken on the scaling aspect, not only on the analytical level, but also in terms of application development. The more users a software has, the greater the variety of practices, making it hard to serve all kinds of combinations of sensors, scenarios and learning needs. Especially in the *long tail* of customers, many niche use cases may be found (Anderson, 2006). These special needs may be met by employing more developers trained to implement the needs of the users. However, this approach does not scale as the number of users and possible use cases exceeds the number of developers in orders of magnitude. Similarly, analyzing the learning settings of the high number of workers is a challenge. To serve the increasing number of usage scenarios for learning analytics involving Internet of Things devices, we need to rethink current workflows. To this end, we present a method and tool support to place the use of the tools in the hands of the adopters and creates societal development processes embedded within professional communities. The open character of our suggested workflow starts with acquiring data from arbitrary sensors that are capable of exchanging data using Web-based protocols. SWEVA (Social Web Environment for Visual Analytics) is a platform on which learners and teachers alike can build their own data visualization pipelines. Our conceptual findings of visual analytics pipelines can be applied in various disciplines while our technical results can be embedded into any Web-based platform leading to ubiquitous visual wearable-enhanced learning analytics.

The remainder is structured as follows. In Section 2 we first present related work in the areas of visual learning analytics and end user development of Internet of Things systems. Section 3 then introduces a number of key terms, standards and technologies for making physical aspects of the environment available on the Web through interconnected sensors. Section 4 shows how the sensor data serves as input for dynamic processing pipelines whose instantiations result in state-of-the-art interactive visual analytics. Our standards-based cross-device software framework conceptualized in Section 5 is able to connect with industrial Internet of Things machines and wearable devices based on open interface documentation formats. The easy-to-use Web frontend presented in Section 6 allows non-technical end users to collaboratively design data pipelines that ultimately get executed to output dynamic visual analytics charts. Section 7 shows the evaluation of the system. Section 8 discusses the conclusions and points to future work.

2. Related work

In this section, we analyze related work in the areas of Internet of Things (IoT) and visual analytics. We start with an overview of the societal impact of the IoT, then we present work on visual analytics, before we conclude with related research that combines these areas.

Internet of Things (IoT) (Carretero & García, 2014; Gubbi, Buyya, Marusic, & Palaniswami, 2013; McEwen & Cassimally, 2013) frameworks are focusing either on the consumer or the business market. Main differences are for example the acceptable level of security aspects of solutions. While in consumer markets the majority of vendors and customers are quite tolerant against security issues like data protection breaches and possible exploitation of gathered data, the situation in business-oriented markets is very different with respect to emerging Industry 4.0 and security threats through criminals and foreign agencies up to the national security level. While the latter is targeted at by new regulations, government institutions and new defensive agencies in the military, the intelligence sector and the security authorities, the former is expected to be self-regulated at the moment. Therefore, in the area of the Internet of Things, governmental investments are mainly targeted at defense and attack prevention capabilities in the cyberspace, leaving a lot of room for consumer-driven innovations, but also problems for the adoption of technology. Several initiatives have already discovered the enormous potential of the IoT for the Web (Web of Things - WoT) (Guinard, Trifa, Mattern, & Wilde, 2011; Krawiec et al., 2017) or social applications (Social Internet of Things - SIoT) (Atzori et al., 2011, 2012, 2014; Lin & Dong, 2017; Perera, Zaslavsky, Christen, & Georgakopoulos, 2014). But only a few development platforms are available (Eisenhauer, Rosengren, & Antolin, 2009). It will be a major problem for users to organize themselves in the face of the speed of technology, protocol and tool development (infrastructure), while keeping their own agency. For technology and infrastructure, the frequency of adoption waves following the diffusion of innovations (Rogers, 2003) is increasing, as the next technological wave is already rolling while the previous one has only just spread.

Visual analytics (Keim, Kohlhammer, Ellis, & Mansmann, 2010) aims at integrating the human capabilities into the data analysis process by using visual representations and interaction techniques. The user gets involved in the analysis process and data interpretation and reasoning are supported by visualizing the important aspects of the data. When this process is distributed and enlarged to a bigger group of people, additional social processes become active. Thus, visual analytics is even more important if many stakeholders view possibly different views and interests on the data are involved. Therefore, the model needs to be extended by a social dimension. It leads to multi-faceted visual analytics with possibly conflicting debates about interpretation of results and far leading analytic activities. The context of a community using an information system gives a common social structure for the stakeholder in case of missing institutional context. Hereby, information system use and development are inseparably interlocked with each other by balancing agency and structure (Orlikowski & Robey, 1991). With the advent of social software, the development process has become much more complex and engineering methods have to consider informal community structures much more than before. Understanding and exploiting the differences between organizational information systems as planned and community information systems as in use contribute to better utilization of organizational and societal resources. Within the information system, Community Learning Analytics (CLA) are the processes of identification, analysis, visualization and support of informal community-regulated learning processes (Klamma et al., 2013). Professional communities of practice (CoP) are learning informally. CoP are groups of people who share a concern or a passion for something they do and who interact regularly to learn how to do it better (Wenger, 1998).

There are various examples of applications allowing communities to interact with their environment through the use of IoT technologies. Snap-To-It (Freitas et al., 2016) allows users to opportunistically interact with any appliance simply by taking a picture of it. The authors of (Mikusz et al., 2015) repurpose existing Web analytics technologies for IoT applications. Crowd-based analytics on top of a video archive is realized in (Satyanarayanan et al., 2015). These approaches demonstrate the uptake of visual analytics by communities of practice and motivate us to steer our research in this direction. However, the complex interplay between social learning, technology appropriation and community learning analytics is coming up with a set of heterogeneous requirements that are not easy to fulfill in the moment.

3. Background

In this section, we explain the underlying technologies necessary for any solution tackling visual analytics with IoT data. First, we introduce common Internet of Things protocols, with a particular focus on those that are accessible on the Web. We then discuss API documentation specification formats and show existing prototypes.

3.1. Internet of Things protocols

A protocol describes a set of rules that is applied when two communication partners exchange information. Standardized protocols allow implementers to create software that is able to interact with other components following the same set of rules. An example is HTTP, which regulates data transfers on the Web. Without the HTTP protocol, browsers would not now know how to contact a server to retrieve HTML pages or images from it. In the Internet of Things world, protocols are limited by the resource constraints of devices. Because of this, messages that are exchanged between nodes must not exceed a certain length depending on the devices and their hardware limitations. Also, verbose messages like the ones typically exchanged within SOAP (see Section 3.2) are not feasible on low-end sensor devices with weak processors and low memories.

For this reason, standardization organizations in collaboration with academia and industry have come up with a number of solutions on multiple levels of the OSI reference model, a conceptual model categorizing network protols into abstraction layers. For the IoT, on the lower levels, there is *ZigBee*, *Bluetooth* and pure *TCP/IP*. ZigBee is a specification for wireless networks with low data traffic. Every module has a unique 64 Bit identifier; when entering a network, an additional 16 Bit identifier is assigned that is used for the communication in that specific network. The most notable commercial adopters of the ZigBee protocol are Philips with its *Hue* lighting system and the *TRÅDFRI* device family by IKEA. While the *ZigBee IP* extension allows directly connecting ZigBee networks to the Internet, it lacks adoption. Another prominent example of IoT protocols is Bluetooth. Especially with the introduction of *Bluetooth Low Energy* with version 4, the IEEE standard was adopted by wearable device manufacturers for fitness trackers and smart watches; it is also known from Bluetooth speakers and car entertainment systems. The low energy standard allows to connect devices in under 5 ms and keep the connection at a maximum distance of at most 100 m. The biggest selling point of Bluetooth IoT devices is that Bluetooth is already built into all state-of-the-art smartphones and it is available in state-of-the-art Web browsers via the Web Bluetooth specification.[1] While the protocol is popular for occasionally connected devices, it has not yet get real traction in smart home scenarios. In contrast, the highly used TCP/IP is a family of protocols best known for the HTTP protocol that is built upon it. Every participant in the network is identified with an IP address. Besides PCs, laptops, also routers, print servers, IP phones and IP radios are connected to the Internet via TCP/IP.

Application-layer protocols are built on top of lower-level protocols like the ones presented above. They make the data exchange between software applications sitting on different devices possible. Examples are *XMPP*, *MQTT* and *CoAP* (Waher, 2015). XMPP is an XML-based protocol consisting of federated servers. Every user is addressable via a unique Jabber ID (JID) and a resource suffix identifying the concrete device the user is using, e.g. alice@provider.com/phone. The format resembles an e-mail address. MQTT and CoAP are representatives of protocols that are reusing concepts of HTTP. HTTP works as request/reply protocol where stateless resources are retrieved from a server. Every message consists of a header and a body. The header starts with a method, e.g. GET. Then, further metadata like the data type of the body or the requested data type is described. The Constrained Application Protocol (CoAP) is standardized by the IETF. CoAP knows two message types, request and response. A message contains headers and the body. Finally, MQTT is a client/server protocol with the server called 'broker'. The broker arranges messages in hierarchical topics separated by forward slashes. Other clients subscribe to these topics and are notified by the broker once new messages arrive. Clients may also define a "last will" for a topic which is published by the broker once the client goes offline. This is useful for gracefully notifying subscribers that the device went offline.

Although the aforementioned event-driven protocols are not directly compatible to the request/reply-oriented HTTP, there exist gateways to translate the exchange pattern to protocols which are understood by Web browsers. For this reason, the WebSockets Web standard (W3C, 2018) can be used. WebSockets allow to setup two-way communication channels between browsers and servers. A further mechanisms in the HTTP standard for creating asynchronous APIs are *Server Push* for sending information from the server to a client. It enables servers to send out asynchronous notifications to interested clients.

3.2. APIs

An application program interface (API) is a clearly defined access point to reuse processing capability of a distinct piece of software. APIs can be found in operating systems, libraries and frameworks, and on the Web. For instance, the Portable Operating System Interface (POSIX) is a standard that in its first version dates back to 1988 (Walli, 1995). It provides a specific communication standard to maintain compatibility between different operating systems. Included are definitions for process control and creation, signals and input/output to hardware interfaces. On the Internet, the term service-oriented architecture (SOA), first mentioned in the year 1996 by Gartner, refers to a model, where functionalities are spread across computing nodes in a distributed network. The respective standard is SOAP (originally Simple Object Access Protocol), an XML-based message exchange format. SOAP APIs are described using Web Services Description Language (WSDL), a platform, programming language and protocol-independent description language for services on the Web. The seminal dissertation by Roy Fielding *Architectural Styles and the Design of Network-based Software Architectures* (Fielding, 2000) describes a programming paradigm for distributed services that uses the HTTP protocol. The architecture is today known as REST standing for Representational State Transfer. RESTful APIs use the methods *GET* for retrieving information from a server, *POST* for creating new resources, *PUT* for updating and *DELETE* for removing resources from a server. REST is an architectural style primarily suited for synchronous request/response based interactions between software artifacts.

3.2.1. API documentation formats

To describe the functionalities of an API, API documentation formats can be used. As explained above, a WSDL document is an XML-based documentation format for SOAP APIs. It contains definitions for *data types*, *messages*, *ports*, *bindings* and *porttypes*. A client accessing a webservice can read in WSDL files to determine which functions are available on the server. The format allows to automatically generate source code in various programming languages. Many integrated development environments (IDEs) provide life cycle functionalities to trace changes in remote interfaces, to adapt the generated code accordingly. WSDL only defines the syntax of the interface and omits semantics like quality of service or guaranteed response times. To this end, extensions like WSDL-S or OWL-S were published that describe the semantics of a SOAP webservice. The Web Application Description Language (WADL) targetting RESTful webservices was submitted to the W3C (Hadley, 2009), yet it is still not formally standardized. Similarly to WSDL, it is a machine-readable service description format. Following the RESTful architectural style, it models resources and relationships between them. WADL is an XML-based format, but can be used to describe JSON APIs. Alternative formats are RAML by MuleSoft (MuleSoft, 2017), API Blueprint by Apiary (API, 2017) or OpenAPI by the OpenAPI Initiative (Swagger, 2018). MuleSoft has lately joined the OpenAPI initiative, leading to a market consolidation towards OpenAPI. In the next section, we present the OpenAPI specification in detail and show how the concepts were adopted by AsyncAPI, a derivate documentation

[1] https://webbluetoothcg.github.io/web-bluetooth/.

format for asynchronous APIs on the Web.

3.2.2. OpenAPI specification

The OpenAPI Specification is a description format for RESTful APIs on the Web. It is both human- and machine understandable. The specification document can be written in the text-based formats JSON or YAML, both formats are convertible to each other. It starts with an info header that defines the basic information of the API it is describing, such as the version, the name, the license and the authors. The server block contains pointers to server addresses the API is running on. Then, the paths of the REST API are listed. Each path item object describes the method that is possible, together with possible request and response parameters, as well as response codes. For describing data types, the items refer to the components section of the OpenAPI documentation. Amongst other definitions, it contains JSON schema descriptions of parameters used to validate input or output data. Examples that are compatible with the respective schema can be provided.

While the specification was renamed to OpenAPI, the name Swagger now stands for a whole framework that revolves around automated tools built around the specification. Swagger-UI is an online tool for creating demo pages with try-out functionality of an API, where requests can be created with the help of simple text boxes and buttons. Furthermore, there exist code generators to generate code libraries for a wide variety of programing languages. They create methods, parameter objects and validators.

The OpenAPI specification is language-agnostic and allows to be extended by vendor-specific extensions. Additionally, in OpenAPI 3.0.0, callbacks can be defined to register out-of-band event listeners, to notify an external service in response to certain events. These event listeners are also known as *webhooks*. However, no real asynchronous behavior can be specified that is native to asynchronous event-based protocols like MQTT and CoAP. To this end, the AsyncAPI was created that is based on OpenAPI.

3.2.3. AsyncAPI

AsyncAPI is an API documentation specification which is based on the recent OpenAPI 3.0.0 standard described in the section above. IT allows a human- and machine-readable documentation for APIs that are not based on request/response, but publish/subscribe. This is typically the case for sensors employed in the Internet of Things. Examples are temperature sensors who regularly update the temperature on a server. In this case, the server publishes the AsyncAPI description. The clients can then parse out the needed endpoints, either based on a decision by the developer, or dynamically via schema-matching.

Similarly to OpenAPI, the specification starts with an info part listing the version and license of the API. Then, server endpoints are listed. The asynchronous equivalent to *paths* are *topics*. AsyncAPI defines a best practice for topic descriptions in the format company.service.1.event.user.signedup. Here, an event is emitted once a user signs up to a service by the company. The numeric literal stands for the version number of the API. Instead of event the specification can also list a command, which describes an operation that needs to be performed. Based on the concrete protocol the specification is used for, the topics are translated to equivalent descriptions in the development process. In MQTT, the dots in the above topic description are translated to forward slashes. Topic names may also include variable parts encompassed in curly braces.

The components part at the end of the specification lists the data types and schemas of the input or output parameters. They are based on JSON schema (Galiegue Zypet al., 2013, p. 32). Similar to the Swagger ecosystem, there are code generators for AsyncAPI available.

3.3. Existing prototypes

In the following two sections, we present two prototypes we developed in earlier work. Both target specific aspects needed for our overall goal to create a visual analytics Web application for workplace learning scenarios.

3.3.1. Direwolf

Direwolf is a framework for cross-device user interfaces, accounting for the fact that users today employ multiple device types in parallel (Kovachev, Renzel, Nicolaescu, Koren, & Klamma, 2014). Thereby, the framework manages the synchronization of the state (video running or not) across devices. In the most recent extension of Direwolf, we presented how to integrate Web services and IoT devices in a typical smart home setting into the Direwolf cross-device application framework (Koren et al., 2018). We did this by equipping each device with a QR code or NFC tag; both point to a URL to an OpenAPI or AsyncAPI description. The framework then reads out the URL, parses the documentation and generates user interface components into the existing application space. This way, the framework remains arbitrarily expandable to support new device types in the future.

3.3.2. SWEVA

The Social Web Environment for Visual Analytics (SWEVA) is a collaborative Web application that enables communities to create their own visual analytics tools. It was developed to enable open source development communities to visualize their development efforts. For this reason, various data sources like GitHub code repository statistics and issue tracker data can be accessed and modeled together in a pipeline editor, by multiple participants simultaneously. Updates are propagated to the collaborators in near real-time. At the end of the pipeline, a visualization charting type can be defined. SWEVA offers various options like line charts, bar charts, or graph-based visualization. The visualization part shows the visualization next to some input elements that are responsible for the interactivity of the visualization. End users can change parameters of the visualization, like the data range, dynamically, and the visualization is updated respectively. More advanced programming is possible via an integrated scripting language.

SWEVA also supports asynchronous events coming from Internet of Things sensors. The events are processed by an event-driven message queue based on XMPP or MQTT. Using the same data pipeline principle, the visual analytics modeling engine allows to include modules that retrieve their data from these asynchronous data sources. The data is then combined and transformed towards the visualization charting engine.

4. Societal software development methodology

Our research is driven by the social learning theory of Communities of Practice (CoP) (Wenger, 1998). In a CoP, community members collaborate to mutually improve their practices. Our communities consist of end users who work in a digitized environment, developers who create software, and researchers analyzing the community-specific interactions. The roles are overlapping, as we enable end users to participate in the development process. *Infrastructuring* is the provision of infrastructure that adapts to the needs of a community using it (Pipek & Wulf, 2009). The term is closely related to participatory design (Schuler & Namioka, 2009) and end-user development (Lieberman, Paternò, & Wulf, 2006), which happen on different stages of the development process. We link these levels in our overarching methodology that we present in this section. It starts from requirements and ends with means for self-assessment with the means of visual analytics.

First, social requirements engineering (Renzel & Klamma, 2014) opens up the requirements gathering process from low-scaling focus groups to the crowd of the Web 2.0. On Requirements Bazaar,[2] new ideas, feature requests or bug reports can be entered, liked, shared and commented. This enables developers to get in touch with end users at a

[2] https://requirements-bazaar.org.

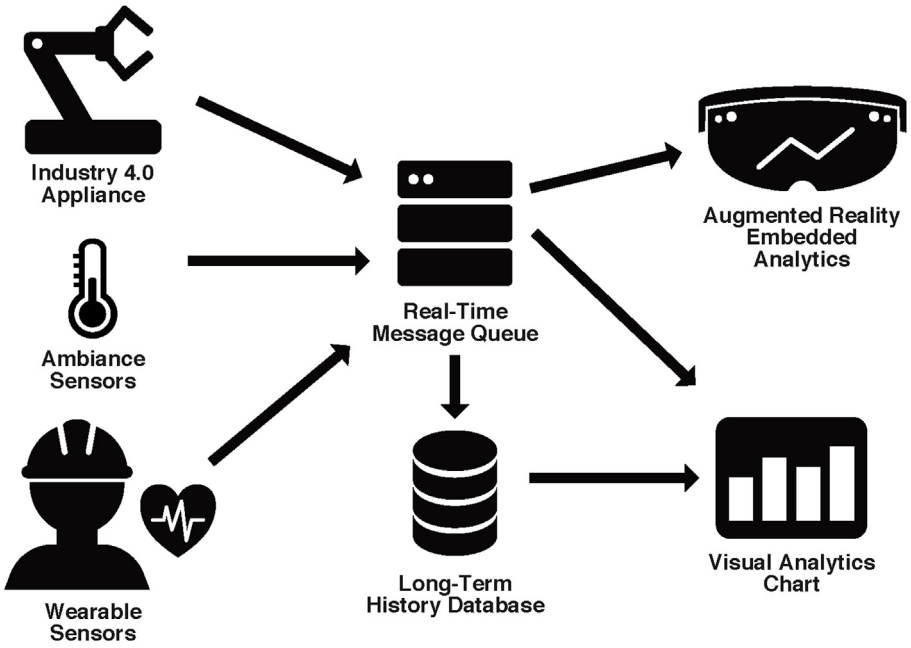

Fig. 1. Conceptual view on the visual analytics system.

very early stage, to enable a feedback loop. From here, requirements can be exported to more developer-related issue trackers. At any time, the current state of the implementation of the requirements is communicated clearly, leading to a transparent workflow. If enabled, contributors get automatically notified by emails upon any change of the state of a requirement. The states are *open*, *assigned* and *realized*. When entering, a requirement is first marked as open. Developers can list requirements and sort them in various ways. After a developer decided to develop a certain requirement, it gets marked as assigned. After the developer is finished, the requirement gets a realized flag.

The next step after requirements are turned into code is putting the developed software into practice. In the following we are describing a general software architecture based on Web technologies. In this architecture, services and frontends are deployed and provided by backends on a server. Services process input and serve their output to be retrieved by clients. This is done in a *request/reply* manner; for every request to the service, a reply is generated and served. They may be executed on workstations, dedicated servers or fully virtualized servers in a cloud environment. Frontends are made available as HTML markup files, JavaScript files containing execution code, and CSS stylesheets including style definitions. The software running on low-powered sensor nodes is much more comprised. For instance, a temperature sensor often only consists of the physical hardware measuring physical values, and a tiny piece of software that transmits the measured sensor data to a server, either directly or via a gateway. In contrast to the aforementioned request/reply pattern, such ‘dumb’ sensor hardware follows a *publish/subscribe* pattern. In a publish/subscribe system, events are sent to a central entity, which then notifies all subscribed actors in the system about the event. To stay in the terms of our architecture, services deployed on the backend would typically be clients of such a publish/subscribe system, turning them into hybrid services. On the one hand they are available for request/reply style interactions, on the other hand they are notified of events happening in their network. A client willing to show the historical data in its frontend can then easily request the historical records from the service. This example system already shows the simplified underlying principle of the system presented in this article.

After the software is put into practice, we are interested in the runtime behavior of the system including its usage. For this reason, we perform formative evaluation during the use of the deployed information system. As mentioned before, the learning context in Industry 4.0 settings is particularly challenging because of the various interdisciplinary data sources. Interacting with industrial and wearable devices naturally produces a lot of data. Some of the sensor data is of environmental origin, like temperature, humidity or brightness. Other data originates from the operating process, such as pushing a button, turning a rotary switch or moving a slide switch. Yet other input is coming from the human machine operator itself and is captured by wearable devices. Such data include the heart rate, body temperature or general body motion sequences. Finally, there are context-dependent data like the time of operation, the age of the machine operator or the years of work experience. It is easy to follow that the amount of data becomes more complex with the number of inputs. Moreover, it makes it hard to grasp interrelations, such as connecting an increased heart rate with a particular machine operation. To this end, we need an overarching analytics strategy that is serving both software usage as well as learning analytics. In our community of practice, the analytics can serve the goal to strengthen the learning goals, and to make the information system more successful in various aspects like failure behavior and usability. Visual analytics is set to combine the power of computer-generated analytics and human interpretation.

Computers are capable of evaluating enormous amounts of data in a very short time. The human brain, conversely, is optimized to quickly identify correlations. Visual analytics is an approach to combine the best of both worlds (Keim et al., 2008). It involves human judgment in the analytics process, utilizing characteristics such as flexibility, creativity and background knowledge. It is possible to directly interact with the represented data, gain new knowledge and therefore make better decisions in the future. The goal of visual analytics is to gain knowledge through model building, visualize it dynamically and then feed back the acquired evidence into the data gathering and filtering process. Visual analytics is a multidisciplinary field with many focus areas (Thomas & Cook, 2006). Examples include weather forecasts, disaster and emergency management, software analytics and physical simulation in engineering.

We identify visual analytics as a useful tool in creating learning analytics for wearable learning solutions in an Industry 4.0 context. However, we acknowledge that while we can reuse various analytics services available, it remains hard to integrate various data sources. To this end, in the next section we present our concept that links the both

worlds APIs and visual analytics in an end-user-oriented way.

5. Conceptual design

Fig. 1 shows a conceptual overview of the system. On the left side, multiple Internet of Things sensors deployed in Industry 4.0 machines and wearable devices provide input to an asynchronous message queue. In the center, we see the real-time message queue, which routes messages to both a database for long-term memory of historical data, and to devices interested in showing the data in visual analytics charts. On the right, the outputted visual analytics chart is shown that can be modified by user input through input text boxes, sliders and checkboxes.

In this section, we start with the requirements for a Industry 4.0-based system for visual analytics. Then, we present conceptual considerations on previous systems we have built, namely Direwolf and SWEVA. We then show how the two concepts are interwoven.

5.1. Requirements

Our requirements were collected within an innovative joint European partnership project in the area of wearable-enhanced learning. The WEKIT project[3] is a consortium of twelve partners from six countries and runs under the Horizon 2020 funding of the European Commission. WEKIT stands for *Wearable Experience for Knowledge Intensive Training*. The objectives are to develop an open technology platform for augmented reality experiences, to augment training in situ with live expert guidance and to create a roadmap for augmented reality learning together in a community of stakeholders. The application cases are in aeronautics, medical engineering and space. Requirements for the platform are sourced by the *WEKIT Community*.[4] A dedicated webpage for the community, together with outposts at Twitter, Facebook, Google Plus and LinkedIn, collects interested stakeholders and gives a voice to them. During the first months of the project, the community was asked to submit ideas at the WEKIT community website. The idea collection was technically realized via a WordPress plugin showing content from Requirements Bazaar. For the WEKIT idea collection, people could enter their idea and comment and vote on existing ones. The development partners in the WEKIT project then developed the most promising scenarios for the project and employed the result in big trials organized together with the application partners. After the trials, new requirements were entered into Requirements Bazaar and prioritized there. Besides requirements for the existing software, also new idea categories were posted; one of them is *analytics*, providing us with real-world requirements in the following. The analytics category mainly contains requirements in terms of post-processing sensor data for learning purposes. In the following, we describe these requirements in a descending order based on the priorities given through the voting process. The actual list of requirements can be retrieved in Requirements Bazaar.[5]

The most important requirement is capturing the execution time of learning activities with the framework. This includes the full duration of the training as well as the single durations during each action step. An action in the WEKIT terminology defines a well-defined act, such as pushing a certain button in the mockup of the International Space Station. The second requirement, *Attention Tracking*, is also related to the course of actions. It proposes timeline visualizations with levels for each action step. Such a timeline should optionally visualize different body and machine data, e.g. the heart rate in combination with an action. This requirement shows the interdisciplinary background of the analytics component: While the heart rate is captured by a wearable heart rate sensor, the action comes from the equipment operated on. The next requirement asks for context-based performance metrics, name analyzing the number of repetitions. This includes, how often the learner practiced a certain training; how many times a learner did a certain step, to understand how difficult it was; how many times the learner replayed instructions, to understand how to do the work step; and the learning curve, to understand how much quicker the learner is when doing repetitions. To see the progression of a single student, analytics should compare the performance of the same person. The same students could be asked to repeat the procedure to evaluate if the whole process is improving. Gaze tracking could be done to analyze if a student is looking at the correct place when performing tasks. Similarly, a gaze trail could be analyzed as frequently changing gaze direction may indicate uncertainty. Analytics could also be helpful for trainers to receive data about their performances, i.e. total procedure duration, duration of each task etc. The performance of different students should be compared to see if the majority is stuck in a particular action; this can be also used to detect if a specific instruction of the expert is unclear. In the same way, the performances of students and experts should be compared. The next requirements are about real-time feedback during the training scenario; an engaging graphical attention indicator should change according to the attention level. In the same way, the number of times a mistake is repeated could be highlighted. The movement of tools and objects should be tracked to see if they are handled too roughly.

The requirements presented above show the main need for an interactive analytics strategy well aligned with the goals of visual analytics. Further requirements were omitted here, but can be found in the Requirements Bazaar. From the depicted requirements here, we mainly derive the need of using different data sources. In addition, they require the expertise of experts from possibly various disciplines. For this reason, we aspire a collaborative solution. To be inclusive towards heterogeneous data sources without needing a developer for adjusting the software to various protocols, the inclusion needs to be automatically handled. Regarding the non-functional requirements, multiple ideas expressed in the Requirements Bazaar refer to immediate feedback within the device. For this reason, the solution should be available cross-device, embeddable and running on different device types.

5.2. Combining IoT and services within visual analytics

In Section 3.3 we introduced the prototypes Direwolf and SWEVA. Direwolf is a cross-device framework for integrating heterogeneous devices and Web services into a collaborative app. On one hand its features allow to create applications whose state is synchronized across various devices of its users. On the other hand, it provides functionality to dynamically integrate Internet of Things devices and Web services. SWEVA is an online tool for collaboratively creating visual analytics pipelines. However, the integration of data coming from Industry 4.0 appliances and wearable devices was not achieved yet because of the complexity of dealing with different vendor-specific APIs.

To this end, we combined the strengths of both prototypes and integrated their approaches in the combined Direwolf & SWEVA prototype. For simplicity, we refer to it as *SWEVA* in the following. As a result, we got support for creating cross-device, collaborative Web applications where the application state is synchronized. We now can integrate various device types in the interactions modeled with SWEVA. On top of that, we added a SWEVA module generator based on API documentation specifications for both synchronous and asynchronous APIs. This is useful in particular when considering the challenges of the large amount of sensor data as described in Section 1. For instance, services can be provided and integrated that are responsible for filtering or cleaning data, or others for combining different data sources. As shown in Fig. 2, historical data may play a role in analytical charts. To integrate historical records, a SWEVA module can be provided that accesses a Web service providing the history.

[3] https://wekit.eu.
[4] https://wekit-community.org.
[5] https://requirements-bazaar.org/projects/155/categories/640.

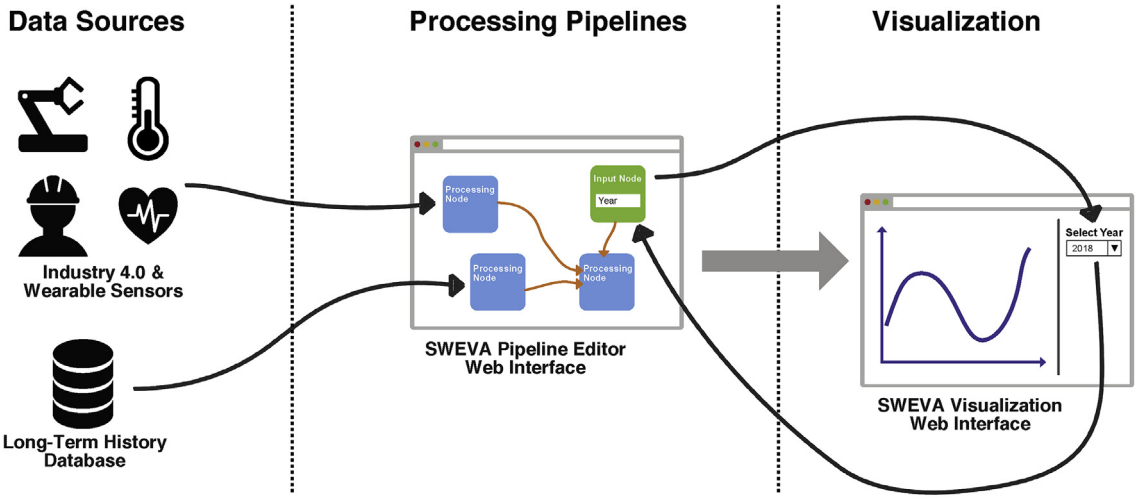

Fig. 2. Illustration of the SWEVA processing pipeline.

Once developed, SWEVA modules are reusable in other contexts as well. For instance, filtering or enriching modules targeting specific wearable IoT sensors can be integrated in other learning scenarios, creating a market of data-handling operations. The details of our implementation are detailed in the following section.

6. Social Web-based environment for visual analytics

This section presents the implementation of the extended SWEVA system. We show the components of the system and the technologies, libraries and frameworks used. Fig. 2 depicts the processing of real-time and historical information in the processing pipeline. It results in the display of the visualization. Following visual analytics principles, the visualization can be zoomed and panned. Additionally, the processing pipeline can be parameterized by human input in terms of textual or numerical input, as the example of the *Year* shows in the figure.

6.1. Analytics pipeline editor

The analytics pipeline editor is a collaborative tool for designing a data processing graph. This data pipeline defines the data flow from data sources, user input over processing functions to the visualization. The data structure of the pipeline is a directed acyclic graph (DAG). The nodes of the graph are modules, the edges consist of directed data flows from source to target. The modules are either processing nodes or user input nodes. A processing node can be anything that retrieves, transforms or generates data, for example retrieving data from a RESTful service. A user input node stands for input that the user can manipulate during runtime. For convenience, the data format of an user input node can be defined, e.g. time, a number or a range. Also, default values can be specified. This allows the rendering engine to display adequate user input boxes, e.g. a calendar for a time, etc.

The interaction with the pipeline editor is synchronized across all devices and users who are simultaneously using the SWEVA Web application. For this reason, we employ the Yjs library.[6] It is a JavaScript library that uses a CRDT algorithm for synchronizing a data structure across devices. By default, it exchanges data from one client to the other via a Websocket connection to a server. Following a peer-to-peer architecture, the server part only has the role of forwarding messages between clients, no conflict resolution is performed. Besides the exchangeability of the communication layer, the library can work on various data types. SWEVA uses hashmap and array data structures to save value-key pairs as well as lists.

6.2. Embedding hardware resources

Even though devices in the Internet of Things expose APIs that can be used by developers, it is a very complex tasks to access these APIs. Protocols need to be followed and parameters need to be set correctly. As explained earlier, documentation specifications allow to rapidly generate code to access the APIs. We leverage these formats to create data retrieval modules for the SWEVA pipeline editor. For this, the devices need to offer the documentation specification. We achieve this by attaching a barcode to the IoT node, which points to the documentation file. It is either stored on the device itself via a tiny webserver, or the link shows to a server hosting the documentation. Parameters that are represented in the specifications with curly brackets, are appended to the URL via GET notation. This way, the same specification file can be reused by various Internet of Things sensors. For example, the specification file may contain a variable for specifying the id of the device: {id}. In this case, the QR code or NFC tag would expose the URL https://example.com/api.json?id=312 where 312 is the ID of the specific IoT node. SWEVA reads in the specification file and generates modules based on heuristics. It replaces the variables with the actual parameter values.

A typical generated module for an asynchronous API creates a connection to the message queue and subscribes to the specified topic. Upon message arrival, the module is informed and continues to run through the pipeline. The pipeline is explained in detail in the next section.

6.3. Visual analytics engine

After the data processing pipeline is created collaboratively in the pipeline editor, the model is formalized into a JavaScript object which is then handed over to the SWEVA visual analytics engine. Instead of directly passing the objects locally, the data structure can also be serialized into a JSON string to be shared amongst users. The visual analytics rendering engine is responsible for showing the visualization and create user interface elements for influencing the visualization. For the user input fields it uses the definitions of the user input nodes defined in the pipeline. The data processing nodes are run following the directed acyclic graph. For running the nodes, the visual analytics engine infers SWEVA Core, the JavaScript task runner performing the calculations. SWEVA Core can either run locally in the browser, or on a server. The latter option is particularly useful for low-end mobile phones that are not capable of running expensive data transformation

[6] http://y-js.org.

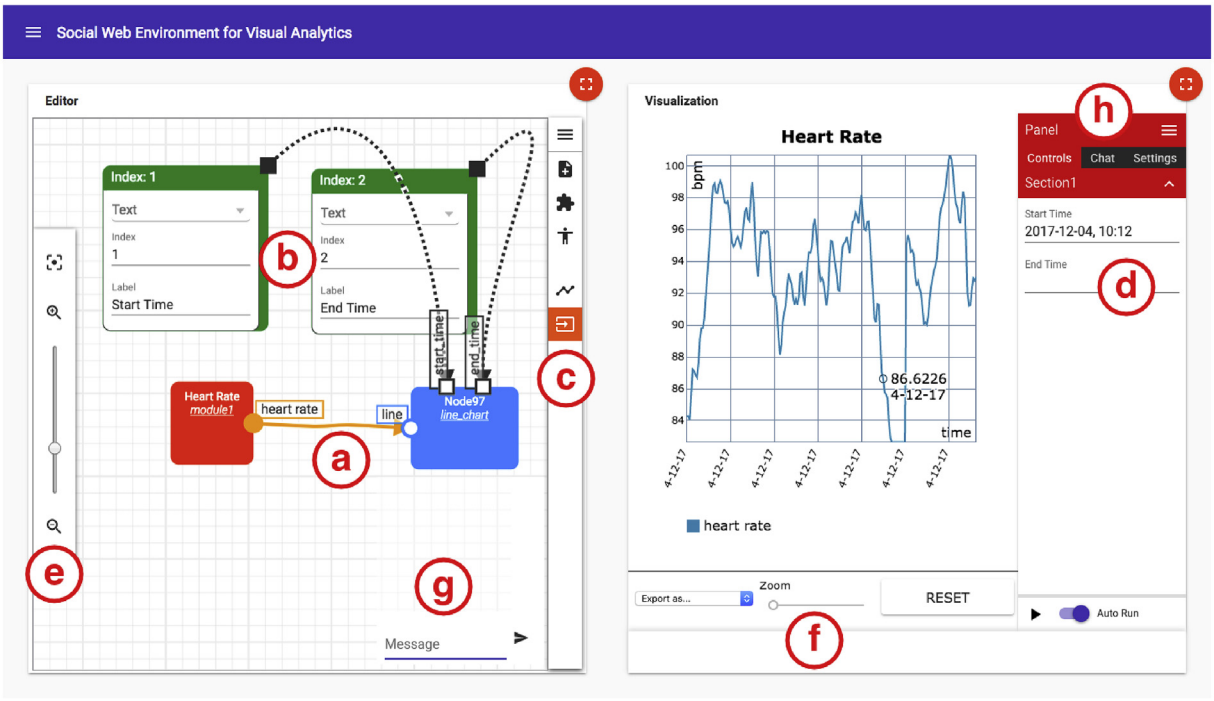

Fig. 3. SWEVA pipeline editor (left) and visualization engine (right).

tasks due to a restrained processor and memory.

Currently, there are already multiple visualization options available: raw, line, and bar charts. The raw visualization only shows the resulting JSON document that is the output of the SWEVA core task runner. The line and bar chart visualizations are developed using D3.js, a library for creating dynamic engines. The main intention of using D3.js was to create the visualization locally, without having to send the data to a third-party website like Highcharts or Google Charts. This way, we keep the data safe on the device of the user.

However, the visualization engine is extensible. New chart options can be created and provided as Web components on the Web. They are embeddable by pointing SWEVA to their URL. The application then reads in, displays and invokes the custom visualization.

6.4. Resulting Web application

A screenshot of the final product is depicted in Fig. 3. On the left, it shows the SWEVA pipeline editor, on the right, the resulting visualization. Here, they are side-by-side, but they can be independently embedded into arbitrary Web pages. First, we explain the pipeline editor in detail. The processing graph is marked with an (a). In this case, the heart rate node on the left provides the data to the node on the right that prepares the data for the line chart. The output of the heart rate source is connected to the input of the line chart. The line chart node contains two user input connection points; one for the start time and one for the end time of the data's time period (b). On the right side of the pipeline editor (c), the toolbar contains buttons for adding new processing nodes (either empty or predefined) or new input nodes, and to select a visualization. The last button (orange) deploys the processing pipeline to the visualization engine. The visualization is shown on the right of the screenshot. Mainly it contains the line chart visualization of the heart rate data. On the right (d), the user input nodes of the pipeline are transformed into text boxes that the user of the visualization can change. Upon change, the visualization updates the displayed timeline. In this case, the end is open, so the chart continues displaying new incoming values. The content of both widgets can be zoomed (e) and (f). For working together remotely, a chat functionality has been integrated for easier coordination of the collaborating users (g), and (h, hidden behind a tab).

The system can be embedded in third-party websites by using Web components. For this, the visualization engine is developed as a custom HTML tag < sweva-visualization > . By loading the Web component definition and adding the tag as a child node, any website can display visualizations. For example, we successfully embedded visualizations in a third-party site based on the WordPress content management system.

7. Evaluation

To validate our prototypical implementation of SWEVA, we performed a technical evaluation and a moderated usability test with volunteers. The results are presented in this section. First, we do a comparison between the envisioned and the delivered functionality. Both evaluations described here were performed not with concrete Internet of Things devices, but with comparable scenarios, stressing individual aspects. However, we implemented an Internet of Things scenario where we connected an accelerometer node with an MQTT broker. A SWEVA module was developed that accessed the broker to display orientation information in real-time. As SWEVA Core, which runs the data pipeline, is protocol-agnostic, we do not see major technical differences in the scenarios, except for possible use cases in training, which are discussed in the next section.

7.1. Requirements

As described in Section 5, the most important requirement estimated by the target community is capturing the time of the performed learning actions. We tackled this by allowing real-time visualizations, as well as historical visualizations. The time frame can be influenced by user inputs, e.g. by selecting the start and end dates of the analytical content. Our developed timeline visualization allows linking specific points in time with further data. Together with a service providing attention tracking information, it can be combined to show the learner's attention over a specific period of time. Our visual analytics component allows zooming in and selecting subsets of data. Comparing performance between learners is enabled through the access to historical data sets of comparable situations. While our framework is in principle

embeddable into arbitrary (wearable) devices, we have not yet implemented direct real-time feedback within applications. In this respect, the demanded gaze tracking in the concrete augmented reality context is out of scope for our prototype, however, the technical infrastructure to capture and replay gaze tracking features is set up in our visual analytics infrastructure.

7.2. Technical evaluation

The goal of our technical evaluation was to measure the load of the execution of the visualization pipelines on current client systems with varying processing pipelines. The machine running the evaluation was run by an Intel i5-3210M CPU with 2.50 GHz. Therefore, we set up two visualization scenarios that involved querying data from a third-party webservice and performing data transformations. Even though these scenarios are not embedded in wearable-based learning with IoT devices, we consider our measurements to be representative for similar visualization pipelines in the described settings. As first visualization, we choose to display the network of contributors to Requirements Bazaar, our continuous innovation platform used for the requirements gathering. The graph-based visualization shows the selected project in the center. Other nodes are the project's categories, its requirements, and the respective contributors of new ideas, votes and comments. The second visualization consists of our analytics dataset gathered by all the nodes of our las2peer[7] application framework, running the learning backend infrastructure of the Learning Layers informal learning European project.[8]

To measure the execution speed, we performed the visualization request three times with disabled caching and took the average value. The Requirements Bazaar visualization required 1.11 s to process and render the visualization, including the download of data from the webservice. The overall time required for the actual data transformation was on average 0.87 s. While this value does neither satisfy real-time requirements from the embedded systems community, nor the human reaction time of around 150 ms, we are positive that it is fast enough to be considered real-time in terms of analytics. Additionally, it only counts the initial processing; real-time chart updates, e.g. when adding a value coming from an MQTT node, are added faster. Memory-wise the total heap memory consumption was 5.3 MB. We consider this reasonable compared to other websites (YouTube front page: 7.7 MB; Wikipedia front page: 3.2 MB).

7.3. Usability

We tested the usability of the system with 15 users in total. They were recruited on voluntary basis from the pool of students at our university; most of them were from the computer science department. While we acknowledge the limited expressiveness of that computer science test group, as end users are our main target, we wanted to make sure that developers in particular understand the concepts of our visual pipeline editor. Also, a focus was on the perceived value of the built-in collaborative scripting editor to create even more complex pipelines. We held three sessions with three participants and one session with five. A printed document contained an introduction and the task descriptions. To not burden the users with a complex wearable analytics scenario, we decided to test the software with a basic movie rating setup. For this, the users first had to rate some popular movies on a five-level Likert-type scale. Then, the task was to create a visual analytics visualizations. To achieve this, a webservice had to be queried that returned the voting results. The data then had to be wired together and visualized in a bar chart with the help of the SWEVA platform. After the evaluation session, the participants were asked to fill in a questionnaire. It consisted of 31 five-level Likert scale questions and two open text questions.

The usability was asked on the basis of *EN ISO 9241-110*,[9] covering aspects such as task suitability and conformity with user expectations. The participants mostly agreed, that the presented tools offer the necessary functionalities to fulfill the tasks. All user interface elements were rated as purposeful, though we received recommendations to create tooltips and further visual aid. This might be also rooted by the fact that on purpose we did not explain users the interface before by providing videos or oral instructions. Customizability was perceived as sufficient, while error feedback needs to be improved. Collaboration support was also asked for in the questionnaire. As discussed in Section 2, we consider collaboration as an essential means when multiple stakeholders come together and add a social dimension to visual analytics. The results indicate that the usefulness is perceived as neither overly useful nor detrimental. We explain that with the study design and expect the usefulness value to rise when employed by end users, as they may rely on collaborating expert-users to reach their goals. However, we observed some destructive behavior during the collaborative session: some users deleted nodes that have just been created by another users, or new nodes were obstructing the view of another. We deduce from this, that awareness functionalities could enhance the results significantly. Most users were able to learn something from one another; this is even more valid for the non-developer users, as all three of them rated that they had learned something new in the collaborative session. All participants acknowledged that collaboration encourages sharing of experience and knowledge.

The technical evaluation showed that we fulfilled our goal of creating a light-weight visual analytics engine that can be easily embedded into third-party Web applications. Based on the usability evaluation, we can conclude that all users were sufficiently able to use the tools for the intended purpose. However, we see a clear need for awareness functionality. Finally, we want to study the appropriation of the platform for real wearable and Industry 4.0 use cases. Therefore we plan to implement the exact requirements mentioned in Section 5.1.

8. Conclusion

In this article we presented the extended SWEVA framework for embedding Industry 4.0 and wearable devices into cross-device Web applications for the reason of Visual Learning Analytics. For this reason, we showed how the functionalities of cross-device application development, first of all the easy embedding of third-party devices, can be easily accomplished using commodity functionality like QR codes. A particular highlight of our prototype is using API documentation specifications for automatically generating code to access Industry 4.0 and wearable devices. API documentation is a popular instrument for developers to make their APIs usable by other developers. In our case, we use the documentation to make the APIs usable for end user communities.

8.1. Collaborative learning analytics

Another main contribution of our work is adding a social dimension to the visual analytics workflow. We achieved this by embedding real-time collaboration between stakeholders into our method. Technically, this is realized using a state synchronization across browsers. Embedding different stakeholder views and interests is a powerful means. In particular, we envision experts manipulating and refining analytics pipeline during training sessions. Displaying these visualizations in real-time for instance in augmented reality scenarios may further support self-assessment.

[7] https://las2peer.org.
[8] http://learning-layers.eu/.
[9] https://en.wikipedia.org/wiki/ISO_9241.

8.2. Future work

The extensible architecture of SWEVA allows future updates in various regards. First of all, we envision a proper user management with layered restrictions based on the collaborator's expertise. This is particularly important since we also include end users in the collaboration who might not know what a specific module is doing. On the other hand, developers may remove certain user interface elements that are important for the end users. Together with user management, we want to introduce awareness functionalities to ensure that all participants know what's going on in the collaborative workspace. Then, we are working on more intuitive ways to allow all users to participate in the session. We could e.g. introduce textual or graphical drawing annotations. Besides, we are currently working towards embedding near real-time feedback in the head-up display of augmented reality glasses like the Microsoft HoloLens to allow in-situ feedback.

Acknowledgements

We would like to thank our student Alexander Ruppert for his contributions towards the implementation of SWEVA and we are grateful for the feedback received in our evaluation. We are especially thankful for the useful and supportive remarks expressed by our reviewers; they helped positioning our approach in the wider context of visual learning analytics. The work has received funding from the European Commission's FP7 IP Learning Layers under grant agreement no. 318209 and from the European Research Council under the European Union's Horizon 2020 Programme through the project "WEKIT" (grant no. 687669).

References

Anderson, C. (2006). *The long tail: Why the future of business is selling less of more.* New York: Hyperion.

API Blueprint (2017). https://apiblueprint.org/.

Atzori, L., Iera, A., & Morabito, G. (2011). Siot: Giving a social structure to the internet of things. *Communications Letters, IEEE, 15*(11), 1193–1195.

Atzori, L., Iera, A., & Morabito, G. (2014). From 'smart objects' to 'social objects': The next evolutionary step of the internet of things. *IEEE Communications Magazine, 52*(1), 97–105. https://doi.org/10.1109/MCOM.2014.6710070.

Atzori, L., Iera, A., Morabito, G., & Nitti, M. (2012). The social internet of things (SIoT) – when social networks meet the internet of things: Concept, architecture and network characterization. *Computer Networks, 56*(16), 3594–3608. https://doi.org/10.1016/j.comnet.2012.07.010.

Carretero, J., & García, J. D. (2014). The internet of things: Connecting the world. *Personal and Ubiquitous Computing, 18*(2), 445–447. https://doi.org/10.1007/s00779-013-0665-z.

Eisenhauer, M., Rosengren, P., & Antolin, P. (2009). A development platform for integrating wireless devices and sensors into ambient intelligence systems. *2009 6th annual IEEE communications society conference on sensor, mesh and ad hoc communications and networks workshops* (pp. 1–3). .

Fielding, R. T. (2000). *Architectural styles and the design of network-based software architectures*Ph.D. thesis. Irvine, Irvine, CA, USA: University of California. http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm.

Freitas, A. d., Nebeling, M., Chen, X. A., Yang, J., Ranithangam, A. S. K. K., & Dey, A. K. (2016). Snap-to-it: A user-inspired platform for opportunistic device interactions. *Proceedings of the 34th annual ACM conference on human factors in computing systems (CHI'16)*.

Galiegue, F., Zyp, K., et al. (2013). *JSON schema: Core definitions and terminology.* Internet Engineering Task Force (IETF).

Gubbi, J., Buyya, R., Marusic, S., & Palaniswami, M. (2013). Internet of things (IoT): A vision, architectural elements, and future directions. *Future Generation Computer Systems, 29*(7), 1645–1660. https://doi.org/10.1016/j.future.2013.01.010.

Guinard, D., Trifa, V., Mattern, F., & Wilde, E. (2011). From the internet of things to the web of things: Resource-oriented architecture and best practices. In D. Uckelmann, M. Harrison, & F. Michahelles (Eds.). *Architecting the Internet of Things* (pp. 97–129). Berlin, Heidelberg: Springer Berlin Heidelberg.

Hadley, M. (2009). *Web application description language*.

Keim, D. A., Andrienko, G., Fekete, J.-D., Görg, C., Kohlhammer, J., & Melançon, G. (2008). Visual analytics: Definition, process, and challenges. In A. Kerren, J. Stasko, J.-D. Fekete, & C. North (Vol. Eds.), *Information visualization: Vol. 4950 of LNCS*, (pp. 154–175). Springer Berlin/Heidelberg.

Keim, D. A., Kohlhammer, J., Ellis, G., & Mansmann, F. (2010). *Mastering the information age: Solving problems with visual analytics.* Goslar: Florian Mansmann and Eurographics Association.

Klamma, R. (2013). Community learning analytics – challenges and opportunities. In J.-F. Wang, & R. W. H. Lau (Vol. Eds.), *Advances in web-based learning: ICWL 2013: Vol. 8167 of LNCS*, (pp. 284–293). Berlin: Springer. https://doi.org/10.1007/978-3-642-41175-5_29.

Koren, I., & Klamma, R. (2018). The exploitation of OpenAPI documentation for the generation of web frontends. In P.-A. Champin, F. Gandon, F. Gandon, M. Lalmas, & P. G. Ipeirotis (Eds.). *Companion of the web conference 2018 - WWW '18* (pp. 781–787). New York, New York, USA: ACM Press. https://doi.org/10.1145/3184558.3188740.

Kovachev, D., Renzel, D., Nicolaescu, P., Koren, I., & Klamma, R. (2014). DireWolf: A framework for widget-based distributed user interfaces. *Journal of Web Engineering, 13*(3&4), 203–222.

Krawiec, P., Sosnowski, M., Batalla, J. M., Mavromoustakis, C. X., Mastorakis, G., & Pallis, E. (2017). Survey on technologies for enabling real-time communication in the web of things. In J. M. Batalla, G. Mastorakis, C. X. Mavromoustakis, & E. Pallis (Eds.). *Beyond the Internet of Things, Internet of Things* (pp. 323–339). Cham: Springer International Publishing.

Labrinidis, A., & Jagadish, H. V. (2012). Challenges and opportunities with big data. *Proceedings of the VLDB Endow, 5*(12), 2032–2033. https://doi.org/10.14778/2367502.2367572.

Lasi, H., Fettke, P., Kemper, H.-G., Feld, T., & Hoffmann, M. (2014). Industry 4.0. *Business & Information Systems Engineering, 6*(4), 239–242. https://doi.org/10.1007/s12599-014-0334-4.

Lieberman, H., Paternò, F., & Wulf, V. (2006). *End user development. Human-computer interaction series, Vol. v. 9*. Dordrecht: Springer.

Lin, Z., & Dong, L. (2017). Clarifying trust in social internet of things. *IEEE Transactions on Knowledge and Data Engineering.* https://doi.org/10.1109/TKDE.2017.2762678.

McEwen, A., & Cassimally, H. (2013). *Designing the Internet of Things.* Hoboken, N.J.: Wiley.

Mikusz, M., Clinch, S., Jones, R., Harding, M., Winstanley, C., & Davies, N. (2015). Repurposing web analytics to support the IoT. *Computer, 48*(9), 42–49. https://doi.org/10.1109/MC.2015.260.

MuleSoft, I. (2017). *RAML.* https://raml.org/.

Orlikowski, W. J., & Robey, D. (1991). Information technology and the structuring of organizations. *Information Systems Research, 2*(2), 143–169.

Perera, C., Zaslavsky, A., Christen, P., & Georgakopoulos, D. (2014). Context aware computing for the internet of things: A survey. *IEEE Communications Surveys & Tutorials, 16*(1), 414–454. https://doi.org/10.1109/SURV.2013.042313.00197.

Pipek, V., & Wulf, V. (2009). Infrastructuring: Toward an integrated perspective on the design and use of information technology. *Journal of the Association for Information Systems, 10*(5), 447–473. http://aisel.aisnet.org/jais/vol10/iss5/1/.

Renzel, D., & Klamma, R. (Vol. Eds.), (2014). *Large-scale social requirements engineering: Vol. 2*IEEE Special Technical Community on Social Networking (IEEE STCSN).

Rogers, E. M. (2003). *Diffusion of innovations* (5th ed.). New York: Free Press.

Satyanarayanan, M., Simoens, P., Xiao, Y., Pillai, P., Chen, Z., Ha, K., et al. (2015). Edge analytics in the internet of things. *IEEE Pervasive Computing, 14*(2), 24–31. https://doi.org/10.1109/MPRV.2015.32.

Schuler, D., & Namioka, A. (2009). *Participatory design: Principles and practices.* Boca Raton and London: CRC.

Swagger (2018). https://swagger.io/.

Thomas, J. J., & Cook, K. A. (2006). A visual analytics agenda, computer graphics and applications. *IEEE, 26*(1), 10–13.

W3C (2018). *HTML living standard: WebSocket.* https://html.spec.whatwg.org/multipage/web-sockets.html.

Waher, P. (2015). *Learning Internet of Things: Explore and learn about Internet of Things with the help of engaging and enlightening tutorials designed for raspberry Pi.* Birmingham, England: Packt Publishing.

Walli, S. R. (1995). The POSIX family of standards. *StandardView, 3*(1), 11–17. https://doi.org/10.1145/210308.210315.

Wenger, E. (1998). *Communities of practice: Learning, meaning, and identity, learning in doing.* Cambridge, UK: Cambridge University Press.