A Service Infrastructure for the Internet of Things based on XMPP

Sven Bendel, Thomas Springer, Daniel Schuster, Alexander Schill
Computer Networks Group,
Faculty of Computer Science, TU Dresden
01062 Dresden, Germany
{*sven.bendel, thomas.springer*}*@tu-dresden.de*
{*daniel.schuster, alexander.schill*}*@tu-dresden.de*

Ralf Ackermann, Michael Ameling
SAP Next Business And Technology Dresden
SAP AG
Walldorf, Germany
{*ralf.ackermann, michael.ameling*}*@sap.com*

***Abstract*—Following the vision of an Internet of Things (IoT) real world objects are integrated into the Internet to provide data as sensors and to manipulate the real world as actors. While current IoT approaches focus on the integration of things based on service technologies, scenarios in domains like smart cities, automotive or crisis management require service platforms involving real world objects, backend-systems and mobile devices. In this paper we introduce a service platform based on the Extensible Messaging and Presence Protocol (XMPP) for the development and provision of services for such pervasive infrastructures. We argue for XMPP as protocol for unified, real-time communication and introduce the major concepts of our platform. Based on two case studies we demonstrate real-time capabilities of XMPP for remote robot control and service development in the e-mobility domain.**

***Keywords*-service infrastructure, Internet of Things, mobile devices, XMPP**

I. Introduction

Technologies for the Internet of Things (IoT) enable the integration of real world objects into the global infrastructure of the Internet. In this way things can participate in virtual processes by providing real world data as sensors and allow to control and manipulate the real world as actors. Current approaches for IoT mainly focus on communication protocols to integrate things with internet protocol standards considering limited computation and memory resources as well as restricted bandwidth and energy availability. Special focus is set on the integration of things at service layer. EncDPWS [1] examplifies the efforts to use optimized message encoding for the use of SOAP as application layer protocol for resource constrained devices. In [2] an approach for accessing real-world objects as RESTful services in order to provide a Web of Things is presented.

However, upcoming scenarios in the domains of smart cities, connected cars, crisis management or health care have a much broader scope than considered by these projects. They require user-centric services involving real world objects, backend-systems and mobile devices to enable users to consume real world data and interact with the physical environment in real-time. Thus, the underlying communication infrastructure has to support efficient, pub/sub-based data provisioning for services and mobile apps at the one hand and a request/response-based interaction for service consumption on the other. Moreover, it should be available on a large set of device platforms including resource restricted sensors and actors.

The Extensible Messaging and Presence Protocol (XMPP) [3] gains more and more attention as communication protocol in the Internet of Things. It is a family of protocols standardized by the IETF and well established in the Internet. Protocol stacks are available for major programming languages and device platforms including Android, iOS, Java and JavaScript. Even more, projects like uXMPP [4] and the mbed XMPPClient[1] implement lightweight XMPP protocol stacks which allow deployment even on tiny sensor nodes.

According to the findings in [5] and [6] major characteristics of XMPP address the requirements of IoT scenarios quite well. Designed as a protocol for near real-time communication it supports small message footprint and low latency message exchange. JIDs provide a scheme for globally unique addresses similar to e-mail addresses. Different types of messages (called stanzas) allow bi-directional communication based on publish/subscribe as well as request/response interactions enabling push-based data provisioning and pull-based service access. While its decentralized client-server architecture ensures high scalability, XEP-0174 also specifies an extension protocol enabling serverless messaging without any infrastructure components. Thus, XMPP-based infrastructures can scale from simple infrastructureless settings to complex configurations with multiple federated XMPP servers.

Furthermore, XMPP is highly extensible by allowing the specification of extension protocols (XEPs). Of high value for the domain of IoT are XEP-0045 for efficient m:n communication based on multi-user chats, XEP-0030 specifying a protocol for service discovery and XEP-0060 defining generic publish/subscribe funtionality.

In this paper we introduce a service platform[2] based on XMPP for the development and provision of pervasive services interconnecting real world objects, backend systems

[1] http://mbed.org/cookbook/XMPPClient
[2] http://mobsda-dev.inf.tu-dresden.de/coca

978-1-4673-5077-8/13/$31.00 ©2013 IEEE

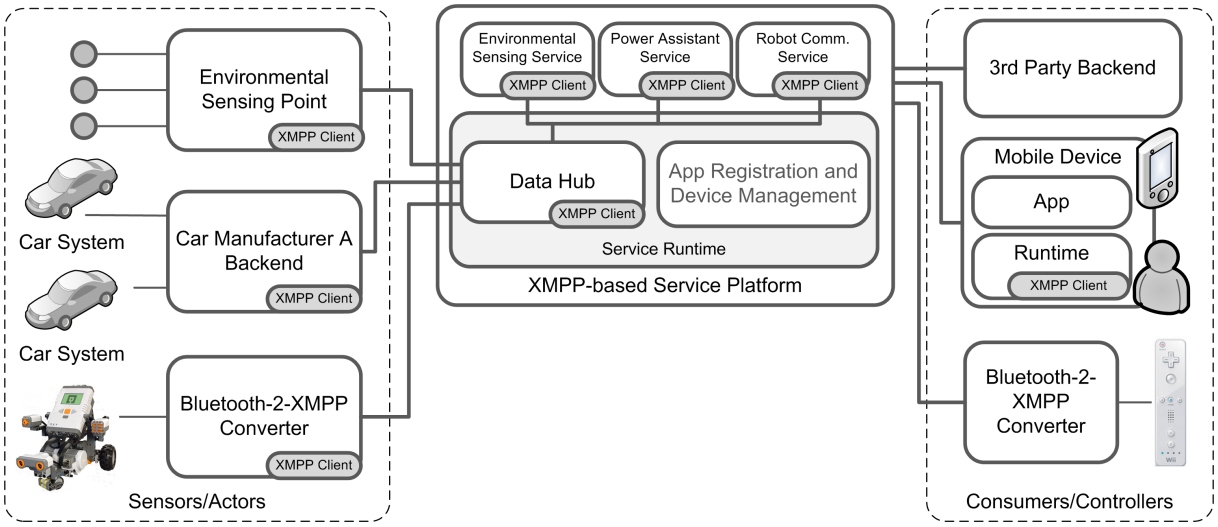

Figure 1. Architecture overview.

and mobile devices in seamless manner. We introduce the architecture and concepts of our platform in Section II. Based on two case studies in Section III we demonstrate real-time capabilies of XMPP for robot control and service development in the domain of e-mobility.

II. Approach

The design goals of our platform are threefold. First, the platform should enable a simplified and cost-efficient development and deployment of pervasive services, especially for 3rd parties. Considering pervasive service infrastructures, 3rd parties usually consume data provided by multiple organizations. For example, in the automotive domain, 3rd parties usually have to integrate floating car data from multiple car manufacturers to provide their services to any customer independent from its car brand. This implies the requirement of a secure, reliable and scalable data exchange, which is the second design goal. Third, following a user centric approach the platform should support the integration of apps on (mobile) devices as part of pervasive services.

The architecture of our approach is shown in Fig. 1. The heart of the proposed system is the Service Platform which provides a runtime environment for multiple, dynamically deployable services. Based on the Data Hub, part of the service runtime, the platform connects the services with various real world objects. On the one hand these are sensor and actor devices, depicted on the left side. On the other hand, 3rd party backends, controller devices and (mobile) end user devices can be connected to the service runtime. A second major building block of the service runtime is the App Registration and Device Management component. It handles the provisioning of apps to mobile devices as well as user and device management.

XMPP is used for the communication in the proposed system. As shown in Fig. 1 all entities including the services are XMPP clients which can be identified by a JID (e.g. alice@example.com/home) in a system wide unique manner. Exceptions are backend systems and specific controlling devices which might be integrated by specific adapters provided by the Data Hub. Not shown in Fig. 1 are XMPP servers which are usually necessary to mediate communication between XMPP clients.

Every piece of data pushed into the system by a data source has a unique data type. These data types are registered at the DataHub and can then be requested by services. The DataHub receives all the data produced by data sources registered at this particular server. Prospectively the incoming data shall be aggregated and pushed to the services being registered for this data type via XMPP. As of now the DataHub is not yet implemented and the services handle the most needed functionalities themselves which means they are directly addressed by the generators as data receivers. This works due to the fact that services are fully functional XMPP clients with their own JIDs.

Data sources push data based on the publish/subscribe principle. When a data consumer (e.g. a smartphone application) wants to be notified of new data of a specific type it registers at it's appropriate service for this particular data type. The service then adds the consumer to this data type's list of interested clients. Same goes for services which register for data types at the DataHub although this usually happens only once during their setup phase. As soon as new data with this specific type arrives at the DataHub the data gets pushed to the service which in turn pushes it to the data consumer. Due to the fact, that the connection between the data generator and the DataHub as well as the connection between the DataHub and the service and the connection between the service and the data consumer are

long-lived XMPP connections, the data is transmitted nearly instantaneously (see section III) from a generator via the DataHub and the appropriate service to all consumers.

III. Evaluation

To demonstrate the feasibility of our approach, we implemented two show cases, namely RemoteBot and PowerAssistant. Both systems share the proposed platform as means of implementation.

A. Show Cases

RemoteBot demonstrates the real-time capabilities for message exchange in the implemented platform. The movements of a Lego NXT Robot[3] can be controlled by an Android app and/or a Nintendo Wii Remote (see left part of Fig. 2). The devices are interconnected via a service residing on the Service Platform. The service is responsible for storing robot configurations, matching control and monitoring devices with particular robots as well as for routing the data flows between control devices and robots. Even multi-control/monitoring device setups (i.e. robot is controlled by Wii Remote and monitored by Android device) are supported. Using the Wii Remote, continuous control data is pushed to the assigned robot to allow simple robot control. Advanced robot control is possible based on the Android app. The app is able to visualize information about the robot movement, for instance the current speed based on values provided by the rotation counters of the NXT motors or movement direction based on magnetometer data. Dependent on that data the user may now control the robot's motors in nearly real-time via touch screen controls.

All communication except from the communication with the Wii Remote and the robot is done via XMPP, even during the configuration and setup phase. Unfortunately the Wii Remote as well as the NXT cannot be directly addressed by XMPP, however a simple Bluetooth2XMPP converter takes care of the "last mile" to the Wii Remote and the NXT. Please refer also to the demo video[4].

PowerAssistant implements an e-mobility scenario. Electric car drivers can use an Android app to search for charging station in proximity (see right part of Fig. 2). Dependent on current car position and remaining battery load reachable charging stations are listed or presented on a map. Part of the charging station data are pricing information for charging and special offers like free Wifi access during charging (as charging can take half an hour or even more). Drivers can easily select the most appropriate offer and book a recharge.

The scenario involves cars of various brands accessible via the manufacturers backend systems, backend systems of charging stations and Android devices hosting the PowerAssistant app. Interaction between these components is mediated by the PowerAssistantService running on the Service Platform. Transmission of floating car data is requested by the Android app. On startup the app subscribes for car position and battery level of a particular car at the PowerAssistantService which itself subscribes to the appropriate manufacturers backend. When receiving the battery level, the app calculates the remaining cruising range and presents the current value to the user. If the remaining cruising range drops below a defined treshold, the app subscribes for charging station information at the PowerAssistantService. Charging stations push new and changed offers to the service. If these stations are within cruising range, the server forwards these offers to the Android app. Drivers can finally book one of the available offers. The usage of XMPP as a protocol for all communication links in this system again allows instant delivery of data changes.

B. Implementation

We have chosen the Mobilis platform [7] as the foundation for our implementation. The Mobilis platform is based on XMPP and facilitates the implementation of collaborative real-time applications for Android, Java and Javascript. The Mobilis server written in Java allows the dynamic deployment of services which can be discovered by clients and then exchange XMPP messages and XMPP IQs in a bidirectional manner.

The communication between services and servers is standardized by so called Beans (high-level abstractions of XMPP messages and IQs) which are generated from a service API description document (similar to WSDL).

C. First performance measurements

We evaluated our first implementations based on the PowerAssistant use case by measuring the time between data generation and receiving by 20 data consumers (i.e. event propagation latency). To make sure, that the clocks at data generator and data consumers are absolutely synchronous both were deployed on the same physical machine (a 2012 Apple MacBook Pro with 2,3 GHz quad core processor and 8 GB RAM). On the server side a Openfire XMPP server handles communication between the MobSDA server an the clients. Both the Openfire and the MobSDA server were deployed on a VM with 2,4 GHz single core processor and 2 GB RAM. The machine running the data generator and consumers was deployed a few (approx. 10) hops away from the server. The route includes various different network types like WLAN, DSL (cable) and Ethernet. Regarding the networks we think this is a rather realistic scenario, however we did no measurements on mobile devices yet. Additionally we just looked at a "1 generator : 20 consumers" situation. For all these reasons the measurements provide us with a first estimation of a lower boundary for event propagation latencies in our system under somewhat realistic circumstances.

[3] http://mindstorms.lego.com/en-us/Default.aspx

[4] http://www.youtube.com/watch?v=cPCDRsmcSKA

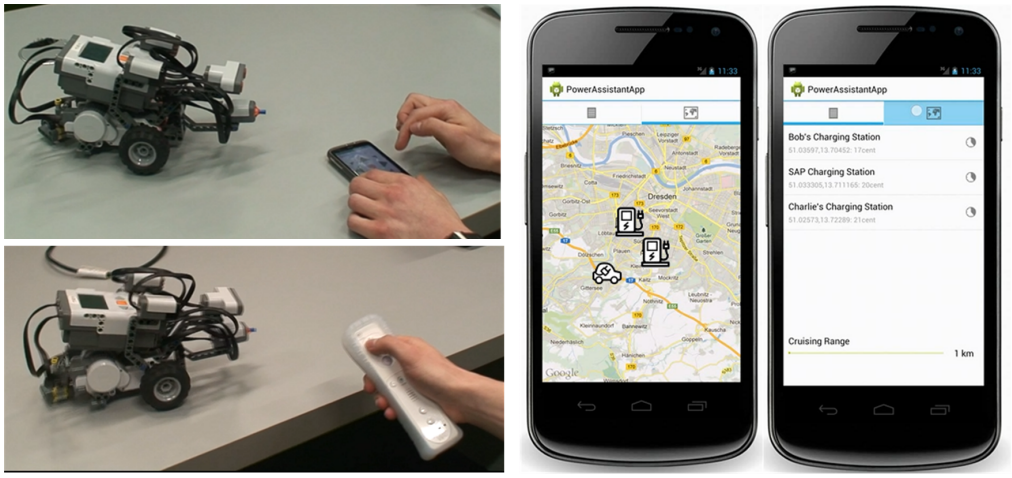

Figure 2. Show cases: RemoteBot (left), PowerAssistant app (right).

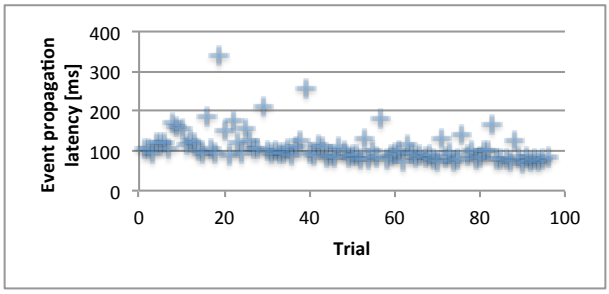

Figure 3. Mean event propagation latency over 100 trials with one generator and 20 consumers.

First results are quite promising. Using the 95th percentile for each trial we measured averagely 108 ms delay between data generation and arrival at the consumers (see Fig. 3, 1 generated data value per second with totally 100 data values) which means nearly instant delivery. The chart also shows fairly constant values encouraging us to continue usage of XMPP as our main communication protocol. We are currently setting up a larger automated test infrastructure to be able to run more sophisticated tests to confirm our first results as well as to find out how many generators and consumers our current implementation supports and what impacts mobile devices (as generators and consumers) have.

IV. Conclusion

In this paper, we introduced a service infrastructure for the Internet of Things which seamlessly integrates real world objects, backend-systems and mobile devices. The platform relies on XMPP as communication protocol to support efficient and highly scalable communication between all building blocks. The platform simplifies service development by supporting dynamic service deployment, convenient access to data streams from multiple external services and the involvement of apps running on mobile devices. Although work is still in progress, the potential of our approach is demonstrated with two show cases. Especially, first performance measurements are quite promising. Next steps will be deep investigations of concepts for aggregation and adaptation of data streams inside the Data Hub, extension of concepts and implementation of mobile device integration and management as well as performance and scalability evaluation in large system setups.

References

[1] G. Moritz, D. Timmermann, R. Stoll, and F. Golatowski, "encDPWS - message encoding of soap web services," in *Pervasive Computing and Communications Workshops (PERCOM Workshops), 2010 8th IEEE International Conference on*, 29 2010-april 2 2010, pp. 784 –787.

[2] D. Guinard, V. Trifa, F. Mattern, and E. Wilde, *From the Internet of Things to the Web of Things: Resource Oriented Architecture and Best Practices*. New York Dordrecht Heidelberg London: Springer, 2011, ch. 5, pp. 97–129.

[3] P. Saint-Andre, "Extensible messaging and presence protocol (XMPP): Core," IETF RFC 6120, 2011.

[4] A. Hornsby and E. Bail, "uXMPP: Lightweight implementation for low power operating system Contiki," in *Ultra Modern Telecommunications Workshops, 2009. ICUMT '09. International Conference on*, oct. 2009, pp. 1 –5.

[5] P. Saint-Andre, "XMPP: lessons learned from ten years of XML messaging," *Communications Magazine, IEEE*, vol. 47, no. 4, pp. 92 –96, april 2009.

[6] M. Kirsche and R. Klauck, "Unify to bridge gaps: Bringing XMPP into the internet of things," in *Pervasive Computing and Communications Workshops (PERCOM Workshops), 2012 IEEE International Conference on*, 2012, pp. 455 –458.

[7] A. Schill, D. Schuster, T. Springer, R. Lübke, and S. Bendel. (2012) Mobilis - pervasive social computing using XMPP. [Online]. Available: http://mobilis.inf.tu-dresden.de