Towards Optimising the Data Flow in Distributed Applications

Felix Leif Keppmann
Karlsruhe Institute of Technology (KIT), Germany
felix.keppmann@kit.edu

Maria Maleshkova
Karlsruhe Institute of Technology (KIT), Germany
maria.maleshkova@kit.edu

Andreas Harth
Karlsruhe Institute of Technology (KIT), Germany
andreas.harth@kit.edu

ABSTRACT

Networked applications continuously move towards service-based and modular solutions. At the same time, web technologies, proven to be modular and distributed, are applied to these application areas. However, web technologies have to be adapted to the new characteristics of the involved systems – no explicit client and server roles, use of heterogeneous devices, or high frequency and low latency data communication. To this end, we present an approach for describing distributed applications in terms of graphs of communicating nodes. In particular, we develop a formal model for capturing the communication between nodes, by including dynamic and static data producing devices, data consuming client applications, as well as devices that can serve as data produces and consumers at the same time. In our model, we characterise nodes by their frequencies of data exchange. We complement our model with a decision algorithm for determining the pull/push communication direction to optimise the amount of redundantly transferred data (i.e., data that is pushed but cannot be processed or data that is pulled but is not yet updated). The presented work lays the foundation for creating distributed applications which can automatically optimise data exchange.

Categories and Subject Descriptors

H.1.0 [**Models and Principles**]: General; D.2.11 [**Software Engineering**]: Software Architectures; H.3.5 [**Information Storage and Retrieval**]: On-line Information Services

Keywords

distributed applications; services; interaction patterns; communication model

1. INTRODUCTION

Recent years are marked by the widespread use and adoption of mobile devices and smart sensors, accompanied by the Internet of Things (IoT) vision and service-based modularisation and distribution of systems. In parallel, new web-based application scenarios emerge, which no longer involve mainly central servers and various clients but include several heterogeneous devices. Devices become smaller and smarter and reveal at the same time new and different characteristics and trade-offs, e.g., in terms of energy consumption, computing power, network connection, bandwidth, dynamic or static data provision. Web-based modularisation and distribution of systems, such as in the area of video sensors, tracking, augmented reality and virtual reality, are only some of the recent developments in industry and research[1]. The integration of local and remote, mobile and stationary devices in one distributed system has become achievable and at the same raises new challenges.

In particular, the characteristics of new mobile and sensor-based application areas differ from traditional web scenarios, since they rely to a greater extent on communication and data integration between several nodes in a network. Systems are more complex in terms of distribution, modularisation or integration, and the classical server and client roles have become obsolete. For instance, it is common that the nodes in a network of a distributed system may own one or more roles (e.g., data source, data sink, controller, or processor) complemented by diverse characteristics (e.g., standalone or embedded devices, energy-efficient low-cost single-board computer or high performance server, mobile or network attached device). The distributed system is composed of these devices to provide a functionality of higher value, while individual devices may be part of one or more distributed systems.

Web services provide means for the development of open distributed systems, based on decoupled components, as required in these emerging application areas. Web services overcome heterogeneity and enable the publishing and consuming of functionalities and data of application and devices. Recently the world around services on the Web, thus far limited to traditional Web services based on Simple Object Access Protocol (SOAP) [4] and Web Services Description Language (WSDL) [2], has been enriched by the proliferation of services with Web Application Programming Interfaces (APIs) conforming to the architectural principles of the Representational State Transfer (REST) [3]. Web APIs are characterised by their relative simplicity and their natural suitability for the Web, relying on the use of Uniform Resource Identifiers (URIs) for resource identification,

[1]For example, the ARVIDA (`http://www.arvida.de/`) and i-VISION (`http://www.ivision-project.eu/`) projects.

WWW 2015 Companion, May 18–22, 2015, Florence, Italy.
ACM 978-1-4503-3473-0/15/05.
http://dx.doi.org/10.1145/2740908.2743044.

and Hypertext Transfer Protocol (HTTP) as protocol for resource access and manipulation.

However, emerging complex distributed application systems composed of nodes with heterogeneous hard- and software characteristics introduce new challenges. The current shift away from classical server and client roles has implications on the communication between nodes. Traditional web architecture assumes a request/response model, where clients pull data from servers. In many scenarios, however, the active role to establish the communication is no longer mainly assigned to clients. New protocols and APIs, e.g., WebSockets or WebRTC [9], have been developed for these specific use cases, providing bi-directional communication. While both push and pull enable the data flow between nodes, one may be less efficient in terms of optimal data transmission, latency times or bandwidth use. These new characteristics have implications on the traditional web architecture based on clear client and server roles.

We exemplify the new challenges with an example scenario based on a simple distributed house monitoring system. The system monitors a single room via video images, position tracking of people, and temperature measurements. The video camera, the position tracking device and the thermometer provide access to their data and functionality via Web APIs. This scenario includes devices and applications with a diverse set of characteristics to illustrate the problem and our approach.

To enable networks based on push/pull communication, with no strict server/client role assignment to the participating devices, this paper makes the following contributions:

- We introduce a formal model to describe a distributed application in terms of a graph of nodes to capture communication dependencies. Nodes are annotated with direction of data flow and update frequencies.
- We present a basic algorithm to determine interaction patterns (i.e., pull vs. push communication) to avoid transfer of redundant or outdated data, thus optimising the data transfer.

The remainder of the paper is structured as follows: in Section 2 we further motivate the problem and specify the details of the scenario that serves as our running example. We introduce our approach in Section 3.1, followed by implied preliminaries in Section 3.2 and the network model in Section 3.3. In Section 3.4 we present the algorithm to determine interaction patterns and in Section 3.5 we show how to apply the network model and the optimisation algorithm to the scenario. We discuss in Section 4 enhancements to extend the model, provide an overview of related work in Section 5 and conclude in Section 6.

2. SCENARIO

We now introduce the scenario of a house monitoring system in more detail.

Our distributed scenario application is build upon a number of devices and systems, all connected to the same network. In particular, a panning video sensor, a thermal sensor and a low-cost computer, providing a tracking service, are composed in the application. A mobile app, a map at a website, and a display at a screen visualise the information for a human user. Each device provides data to or processes

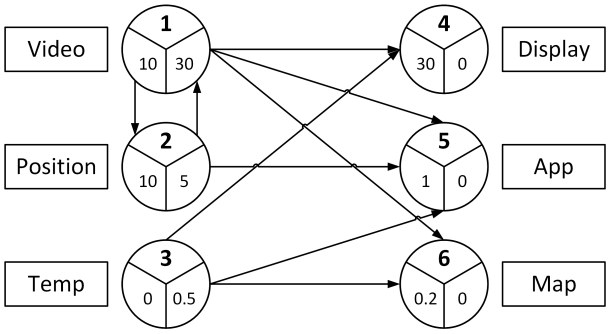

Figure 1: Room monitoring scenario with video camera, position tracking device and thermometer as data sources, and monitor display, mobile app and web-based map as data sinks.

data from other devices in the network at its own frequency. Figure 1 visualises the distributed application as a network or nodes representing the participating devices.

Node 1 – Video (N_1) A panning video camera provides a colour and a depth image thirty times per second, i.e., at a frequency of 30 Hz. At the same time, the video camera processes tracking information and follows the position of a tracked person in the room. In order to follow a person, the camera reacts every 100 ms (i.e., at a frequency of 10 Hz) to the new position coordinates by adjusting the angle of its electric motor.

Node 2 – Position (N_2) A low-cost single-board computer is connected to the network and provides body tracking and position coordinates of recognised people. It is capable to analyse depth images ten times per second, i.e., at a frequency of 10 Hz. Furthermore, the applied algorithm processes these depth images and provides every 200 ms (i.e., at a frequency of 5 Hz) tracking information about the people in the depth image stream.

Node 3 – Temperature (N_3) A thermal sensor measures the temperature in the room once every two second, i.e., at a frequency of 0.5 Hz. Due to the high precision, the sensor usually provides a slightly different value at each measurement. No further data is required by the sensor.

Node 4 – Display (N_4) A monitor display, updated thirty times per second, i.e., at a rate of 30 Hz, visualises the current colour video and temperature. No data is provided by the monitor to other systems.

Node 5 – App (N_5) A mobile app on a smartphone or tablet shows new colour video and temperature data once per second, as trade-off between energy consumption and update rate, i.e., at a frequency of 1 Hz. As a background service the app informs the user with notifications whenever a person is tracked or particular temperature limits are hit. The colour video is only shown if the app is activated and the tracking service tracks a person in the video in order to save bandwidth, e.g., at a paid mobile connection. No data is provided by the app to other systems.

Node 6 – Map (N_6) A web-based map provides an overview of the system, including the temperature and the current colour video. Due to a low user-driven access rate the websites is updated every ten seconds, i.e., at a rate of 0.1 Hz. No data is provided by the map to other systems.

3. NETWORK MODEL

We now introduce our model to describe the data flow in distributed applications, including the dynamics in terms of devices' input and output frequencies.

3.1 Approach

In our approach we treat a distributed application as a graph of nodes. A single node in the graph takes over specific tasks and may provide means to access and modify the state of underling objects, i.e., objects which are either required as input or are output of the tasks. These nodes are integrated in a specific way to provide the higher-level functionality of the distributed application. The integration is established by transferring the state of objects between nodes, i.e., a node may require the state of objects from remote nodes, remote nodes may require the state of local objects, or both at the same time. Which state transfers are required to provide a higher-level functionality is heavily dependent on the specific distributed application.

During runtime, a node repeatedly executes tasks, accesses the state of required objects and modifies the state of influenced objects. Depending on the task, the rate at which objects are accessed may differ from the rate objects at which they are modified, e.g., a node may aggregate data over time. We take this into consideration by distinguishing an in-frequency, at which a node accesses the state of objects for the execution of a task, and an out-frequency, at which a node modifies the state of objects. In addition, a node may not require any states to execute its tasks or it may not modify any states while executing tasks (i.e., the in- or out-frequency is zero).

The integration requires state changes to be transferred to the corresponding connected nodes. We consider two interaction patterns – push and pull, in order to perform this transfer. A node may either pull states from a remote node, or get this states pushed from a remote node. Analogously, states of objects, which are modified by the node, can be pushed to connected nodes, or be pulled by these nodes.

In- and out-frequencies of nodes, in combination with the different interaction patterns, raise the question when should state changes be transferred between nodes via pull and when via push. In a pull interaction outdated data may be pulled, while in a push interaction data may be pushed that cannot be processed until the next push to the remote system. Given these characteristics, we provide a basic algorithm to improve the interaction between nodes by calculating the optimal pattern, i.e., push or pull, for each data flow, based on the in- and out-frequencies of the involved nodes. By deploying the interaction according to the results of the algorithm, we prevent nodes from transferring messages containing data that will be discarded or was not changed and thus minimise the overall data flow in terms of transferred messages. The incorporation of further parameters, e.g., the trade-off between bandwidth and latency, or processing and bandwidth limitation, is addressed in the discussion section.

It should be noted that this approach aligns well with the resource-oriented concepts of REST. In the context of REST, systems are integrated as distributed applications in a resource-oriented manner to provide a higher-value functionality. For the transfer of object states these systems expose a REST API on the network. Nodes represent such an API, collection of resources in an API, or a resource, depending on the particular use case. In a basic case, a complete REST API represents a node as long as the tasks of accessing and modifying the objects exposed by the resources adhere the same in- and out-frequencies. A REST API enables remote nodes to pull states from or push states of object to a node. Thus, nodes may exist in the network, which do not provide a REST API but only pull or push states.

3.2 Preliminaries

In this section we discuss a number of preliminaries used as the basis for the model. In particular, we do not consider limitations or constraints of the network architecture, e.g., bandwidth limitations, transport effects like technically introduced latency, or topological restrictions like Network Address Translation (NAT). While these have a reasonable impact on the deployment of distributed applications, we abstract from these constraints in this simple version of the model.

Dependencies in the application, expressed by a data flow between nodes, are assumed to be stable at runtime or at least for a sufficient amount of time. One-time dependencies, i.e., a single required state of another node without the need for further updates, are not included and are usually handled by a single pull.

Only active nodes are considered, i.e., nodes timing the execution of their tasks. These nodes provide in- and out-frequencies independently from other nodes in the graph. Passive nodes provide functionality on demand and are driven by the frequencies of depending nodes. In this version of the model, these may be handled transparently but must be considered if further characteristics are introduced.

No variable frequencies are considered. These may appear if a task of a node is triggered by an external event (e.g, temperature changes trigger the task of a temperature sensor) or are caused by technical inaccuracies or influences (e.g., network load). For integration cases that require a guaranteed transfer of all states, output queues for states in case of pull and input queues in case of push may overcome this limitation and handle the variances of frequencies.

3.3 Model

$$\begin{aligned} \{F\} \times \{I\} \times \{O\} \times \{P\} \\ F \in \{\text{true}, \text{false}\}^{n \times n} \\ P \in \{\text{push}, \text{pull}, \text{both}, \text{none}\}^{n \times n} \\ I, O \in \mathbb{R}^n; \quad n \in \mathbb{N} \end{aligned} \tag{1}$$

Model. Our model of the network, as shown in Equation (1), includes information about the number (n) of involved nodes, the data flow (F) between these nodes, their in- (I) and out-frequencies (O), and the interaction patterns (P) these node should execute in order to establish the data flow.

$$F = \begin{pmatrix} \text{false} & f_{1,2} & \cdots & f_{1,j} \\ f_{2,1} & \text{false} & \cdots & f_{2,j} \\ \vdots & \vdots & \ddots & \vdots \\ f_{i,1} & f_{i,2} & \cdots & \text{false} \end{pmatrix} \tag{2}$$
$$f_{i,j} \in \{\text{true}, \text{false}\}; \quad i, j \in \mathbb{N}$$

Data Flow. We model the data flow between nodes in the graph as a matrix, as shown in Equation (2). Each entry

in the matrix represents the data flow between node i and node j, i.e., a data flow between node i and j exists ($true$) or not ($false$). The size of the matrix is the number of nodes n in the graph. By default, a loop data flow from a node to itself is not allowed, i.e., the diagonal is set to $false$.

$$F_s = \begin{array}{c} N \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{array} \begin{array}{c} \begin{array}{cccccc} 1 & 2 & 3 & 4 & 5 & 6 \end{array} \\ \begin{pmatrix} & \text{true} & & \text{true} & \text{true} & \text{true} \\ \text{true} & & & & \text{true} & \\ & & & \text{true} & \text{true} & \text{true} \\ & & & & & \\ & & & & & \\ & & & & & \end{pmatrix} \end{array} \tag{3}$$

We apply the modelling of data flows to our scenario, shown in Equation (3). For readability, the table lists only $true$ entries indicating a data flow between two nodes. The instance of the matrix depends on the integration of nodes in a distributed application, that is, every arrow in Figure 1 leads to a $true$ entry in the matrix.

$$I = \begin{pmatrix} i_1 \\ i_2 \\ \vdots \\ i_i \end{pmatrix} O = \begin{pmatrix} o_1 \\ o_2 \\ \vdots \\ o_j \end{pmatrix} \tag{4}$$
$$i_i, o_j \in \mathbb{R}; \quad i, j \in \mathbb{N}$$

Frequencies. We model the in- and out-frequencies of nodes as vectors, shown in Equation (4). Each pair of entries in the vectors I and O, indexed by the node number, are the in- and out-frequency of a node as describe in Section 3.1. Frequencies are measured in hertz (Hz), defined as events or cycles per second.

$$I_s = \begin{array}{c} \text{N} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{array} \begin{array}{c} \text{Hz} \\ \begin{pmatrix} 10 \\ 10 \\ 0 \\ 30 \\ 1 \\ 0.2 \end{pmatrix} \end{array} O_s = \begin{array}{c} \text{N} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{array} \begin{array}{c} \text{Hz} \\ \begin{pmatrix} 30 \\ 5 \\ 0.5 \\ 0 \\ 0 \\ 0 \end{pmatrix} \end{array} \tag{5}$$

We apply the modelling of frequencies to our scenario, shown in Equation (5). The instances of the vectors depend on the properties of the nodes, i.e., every pair of in- and out-frequencies in the circles of Figure 1 is a pair of entries in the vectors.

$$P = \begin{pmatrix} \text{none} & p_{1,2} & \cdots & p_{1,j} \\ p_{2,1} & \text{none} & \cdots & p_{2,j} \\ \vdots & \vdots & \ddots & \vdots \\ p_{i,1} & p_{i,2} & \cdots & \text{none} \end{pmatrix} \tag{6}$$
$$p_{i,j} \in \{\text{push}, \text{pull}, \text{both}, \text{none}\}; \quad i, j \in \mathbb{N}$$

Interaction Patterns. We model the interaction patterns between nodes in the graph as a matrix, shown in Equation (6). Each entry in the matrix represents the interaction pattern a node i should execute with regard to node j, i.e., if node i should *pull*, *push*, pull and push (*both*), or execute no (*none*) interaction pattern with regard to node j. The size of the matrix is the number of nodes n in the graph. By default, a node does not execute an interaction pattern on itself, i.e., the diagonal is set to *none*. The interaction patterns for a specific scenario are calculated by the basic algorithm presented next.

3.4 Algorithm

Algorithm 1 Basic Algorithm

Require: n, inf, outf, flow, NONE, PULL, PUSH, BOTH
Ensure: interaction

```
function INTERACTION_PATTERNS(n, inf, outf, flow)
    for r ← 0, n do
        for c ← 0, n do
            if flow[r][c] ∧ outf[r] ≤ inf[c] ∧
               flow[c][r] ∧ outf[c] > inf[r]
            then
                interaction[r][c] ← BOTH
            else if flow[r][c] ∧ outf[r] ≤ inf[c] then
                interaction[r][c] ← PUSH
            else if flow[c][r] ∧ inf[c] > outf[r] then
                interaction[r][c] ← PULL
            else
                interaction[r][c] ← NONE
            end if
        end for
    end for
    return interaction
end function
```

We use a basic algorithm to calculate the optimal interaction patterns for each combination of nodes – if a node should pull data from another node, push data to another node, or execute both pull and push patterns to send data in both directions. It requires as input the number of nodes in the network, all in- and out-frequencies and the data flow matrix between nodes. The constants NONE, PULL, PUSH, and BOTH are placeholders for interaction pattern. The algorithm returns as output the interaction pattern for each node in relation to each other node.

Flow	Condition	R	C
$R \to C$	$out_R \leq in_C$	$PUSH$	
$R \to C$	$out_R > in_C$		$PULL$
$R \leftarrow C$	$out_C \leq in_R$		$PUSH$
$R \leftarrow C$	$out_C > in_R$	$PULL$	
$R \leftrightarrow C$	$out_R \leq in_C \wedge out_C \leq in_R$	$PUSH$	$PUSH$
$R \leftrightarrow C$	$out_R > in_C \wedge out_C > in_R$	$PULL$	$PULL$
$R \leftrightarrow C$	$out_R \leq in_C \wedge out_C > in_R$	$BOTH$	
$R \leftrightarrow C$	$out_R < in_C \wedge out_C \leq in_R$		$BOTH$

Table 1: Decision Table of the Algorithm

Table 1 lists all decisions made by the algorithm with their conditions and the derived interaction patterns. For readability, $NONE$-entries are not shown.

$$P_s = \begin{array}{c} N \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{array} \begin{array}{c} \begin{array}{ccccccc} 1 & 2 & 3 & 4 & 5 & 6 \end{array} \\ \begin{pmatrix} & & & \text{push} & & \\ \text{both} & & & & & \\ & & & \text{push} & \text{push} & \\ & & & & & \\ \text{pull} & \text{pull} & & & & \\ \text{pull} & & \text{pull} & & & \end{pmatrix} \end{array} \tag{7}$$

We apply the algorithm on our scenario and calculate the interaction patterns P_s, shown in Equation (7). The algorithm is executed with $n = 6$, implementations of I_s, O_s, and F_s as inf, $outf$, and $flow$. The constants $PULL$, $PUSH$, $BOTH$, and $NONE$ are set to *pull*, *push*, *both*, and *none*. For readability, *none*-entries are not listed.

3.5 Scenario Solution

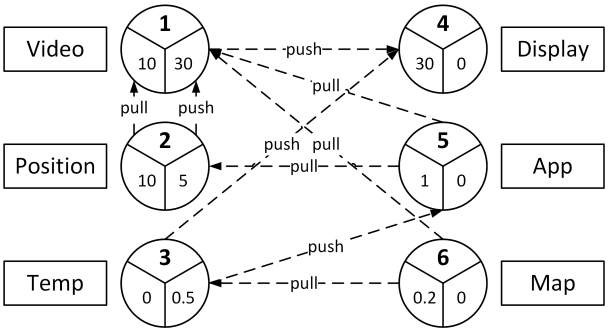

Figure 2: Room monitoring scenario with determined direction of data flow.

Figure 2 visualises the solution calculated by the algorithm in Section 3.4. Dashed arrows between nodes imply, in contrast to data flows in Figure 1, the direction of interaction and are labelled by the corresponding interaction pattern. By deploying the interaction between nodes, the data flow is established and the amount of redundantly pulled or discarded pushed data is minimised.

For example, the data flow from node 2 to node 5 is realised by a pull of the data from node 2 by node 5. Node 5 pulls data one time per second (in-frequency) and will not receive redundant data, since the data at node 2 is changed five times per second (out-frequency). During a push of data from node 2 to node 5 four out of five messages would be discarded by node 5.

	Frequency	Redundant	Discarded
Solution	48.4	0	0
Push-only	131.5	0	83.1
Pull-only	83.4	35	0

Table 2: Comparison of the solution to push- and pull-only interaction.

Table 2 lists aggregated estimate figures for the comparison of the solution to a push-only and a pull-only interaction. Included are the accumulated overall frequencies in the node graph, the accumulated redundantly pulled packages and the accumulated discarded pushed packages. By adapting the derived optimised interaction patterns the solution provides the lowest overall frequency and no messages are transferred, which are redundant or discarded. In comparison, push-only causes the highest overall frequency and the most discarded messages. Pull-only, provides in this particular scenario an overall frequency, which is in between the two previous cases, and thus some messages with redundant data are pulled. The frequency serves also as an indicator for bandwidth consumption between the different approaches. Nevertheless, the payload of messages has to be taken into account in order to be able to make statements about the technical bandwidth usage.

4. DISCUSSION

We now discuss further relevant criteria that can be added to enhance our basic communication model. In particular, some further characteristics of distributed applications can be of importance in deployment scenarios and can have impact on the communication between the nodes. For instance, we plan to integrate bandwidth and latency as additional factors. While the number of transferred messages already have an indirect effect on the used bandwidth in different interaction scenarios, the bandwidth limitations set by the network connection are not considered so far. In a deployed network, the size of transferred data makes a difference (for example, the small size of positioning data of a tracking device compared to relatively large image data of a video sensor). By including information about the message payload (i.e., the size of data transferred between two nodes) we are able to assess the required bandwidth and to take bandwidth limitations into account.

The importance of latency, the second characteristic we aim to include in the model, largely depends on the use case. In classical web scenarios the latency plays a minor role – the technically introduced latency of a request to a Web API should be small enough, but the latency with regard to the last state change is not important. In emerging integration scenarios, such as in the area of depth sensors, tracking and visualisation, latency is much more relevant. For example, the time lag between a movement in reality and the reaction time of an augmented reality impacts the usability of the system. Since the decision between pull or push interaction has an impact on latency, we aim to integrate latency in our model.

The data flow between nodes is so far treated independently – the decision of whether to push or pull is calculated for each connection separately. In a graph of nodes, in particular if we consider further characteristics such as bandwidth, the communication between two nodes may not be independent from others. For example, connecting one depth sensor compared to connecting dozens of depth sensors to a tracking device makes a difference. The example points out two interdependencies, network topology and processing power. We consider to evaluate the integration of interdependent nodes, which e.g., share a single bandwidth limitation, or latency constraint. Furthermore, the processing power of a node may be limited and thus expose restrictions on the amount and frequencies of in- and outgoing data flows.

The algorithm we presented in Section 3.4 implements a decision function that aims to minimise the number of transferred messages between two communicating nodes. Introducing further characteristics, such as bandwidth or latency, allows for assessing different trade-offs. For example, we may reduce latency by using more bandwidth through extensive push. In contrast, the reduction of latency may depend on the type of network (e.g., a local network or a paid mobile connection). We evaluate the possibility of introducing costs for different parameters, which can serve as weights in decision function.

5. RELATED WORK

Related work has already been conducted in several domains, including economics and business administration. In relation to this work, we take into consideration existing research on IoT, and on scheduling and optimisation in sensor networks.

In the context of IoT [5, 6] propose the use of established web technologies, i.e., the integration of smart things following the REST architectural style. The main target of IoT is the connection and combination of various network-enabled devices and virtual artifacts – smart things – to distributed applications. This integration is primarily based on network connectivity between the objects but may lead to incompatible digital islands, i.e., distributed applications in IoT that are closed environments By opening these digital islands via common web technologies, i.e. REST, the authors introduce the Web of Things (WoT) vision. The WoT approach supports our viewpoint on distributed applications consisting of a network of relatively independent nodes and the data flow between this nodes, enabled by push or pull interaction. In [6] the authors evaluate the impact of HTTP and the interaction direction in a purely resource-oriented scenario and a scenario with syndication of things. In particular, they measure the effect on latency in different interaction scenarios.

The authors in [1] discuss different deployment strategies to avoid the decrease of sensor network lifetimes caused by high energy consumption of specific important network nodes. They propose a general model to asses and compare these strategies under certain restrictions as well as to calculate the deployment costs with a specific strategy. For the scenario of a sensor network deployed over an area for surveillance, [8] propose a cost model approach to determine an optimal distribution of sensing nodes and nodes, which act as cluster heads. Factors taken into account are the energy per sensor type and intensity of distribution per node type, to guarantee a specific lifetime and equal power consumption. For large wireless ad-hoc networks [7] propose a combination of pull- and pushed-based strategies to optimise the routing for specific information needs. Thereby, the query frequencies are take into account. Compared to our work these approaches focus more on including factors specific to the deployment of sensors, e.g., power consumption, wireless strength or equipment costs, while we focus more on the optimisation of general issues in a network of data producing and consuming nodes.

6. CONCLUSION

Our approach is a first step towards capturing the dynamics of distributed applications in terms of devices' input and output frequencies. While our model already supports the description of the data flow in a distributed application, we still see different options for extensions, leading to a more detailed representation and, in return, to a more optimised application. In particular, possible extensions include the modelling of further characteristics, interdependencies between nodes and data flows, and the consideration of costs.

In summary, we introduce a way for describing distributed applications based on frequencies of integrated nodes, an approach to formalise this description in a model and an algorithm for optimising the interaction patterns by utilising the model. Part of our future work is to evaluate which extensions can be integrated, either in the model or in the algorithm. With our view on distributed applications we provide the means to deal with emerging application scenarios that pose new requirements on the integrations of systems, thus far unsupported by traditional client/server-based communication patterns.

7. ACKNOWLEDGMENTS

This work was supported by the German Ministry of Education and Research (BMBF) within the project ARVIDA (FKZ 01IM13001G).

8. REFERENCES

[1] Z. Cheng, M. Perillo, and W. B. Heinzelman. General Network Lifetime and Cost Models for Evaluating Sensor Network Deployment Strategies. *IEEE Transactions on Mobile Computing*, 7(4):484–497, Apr. 2008.

[2] R. Chinnici, J.-J. Moreau, A. Ryman, and S. Weerawarana. Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language. Recommendation, W3C, June 2007. `http://www.w3.org/TR/2007/REC-wsdl20-20070626`. Latest version available at `http://www.w3.org/TR/wsdl20`.

[3] R. T. Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, University of California, Irvine, USA, 2000.

[4] M. Gudgin, M. Hadley, N. Mendelsohn, J.-J. Moreau, H. F. Nielsen, A. Karmarkar, and Y. Lafon. SOAP Version 1.2 Part 1: Messaging Framework (Second Edition). Recommendation, W3C, Apr. 2007. `http://www.w3.org/TR/2007/REC-soap12-part1-20070427/`. Latest version available at `http://www.w3.org/TR/soap12-part1/`.

[5] 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. In *Architecting the Internet of Things*. Springer Berlin Heidelberg, 2011.

[6] D. Guinard, V. Trifa, and E. Wilde. A Resource Oriented Architecture for the Web of Things. In *Proceedings of the Internet of Things Conference*, 2010.

[7] X. Liu, Q. Huang, and Y. Zhang. Combs, Needles, Haystacks: Balancing Push and Pull for Discovery in Large-Scale Sensor Networks. In *Proceedings of the Conference on Embedded Networked Sensor Systems*, 2004.

[8] V. P. Mhatre, C. Rosenberg, D. Kofman, R. Mazumdar, and N. Shroff. A Minimum Cost Heterogeneous Sensor Network with a Lifetime Constraint. *IEEE Transactions on Mobile Computing*, 4(1):4–15, Jan. 2005.

[9] A. Narayanan, C. Jennings, A. Bergkvist, and D. Burnett. WebRTC 1.0: Real-time Communication Between Browsers. Working draft, W3C, Sept. 2013. `http://www.w3.org/TR/2013/WD-webrtc-20130910/`. Latest version available at `http://www.w3.org/TR/webrtc/`.