A Platform-as-a-Service for in-situ Development of Wireless Sensor Network Applications

Yong Ding, Martin Alexander Neumann, Dawud Gordon,
Till Riedel, Takashi Miyaki, Michael Beigl
Karlsruhe Institute of Technology, TECO
Karlsruhe, Germany
Email: [firstname.lastname]@kit.edu

Wenzhu Zhang, Lin Zhang
Tsinghua University
Beijing, China
Email: [zhwz,linzhang]@tsinghua.edu.cn

***Abstract*—In this paper we present a Platform-as-a-Service (PaaS) approach for rapid development of wireless sensor network (WSN) applications based on the dinam-mite concept, i.e. an embedded web-based development environment and run-time platform for WSN systems integrated in a single information appliance. The PaaS is hosted by a cloud of dinam-mite nodes which facilitates the on-demand development, deployment and integration of WSN applications. We introduce the dinam Cloud architecture and focus, in this paper, on the PaaS layer established by the dinam-mite nodes. In addition to the description of this so-called dinam PaaS, a performance analysis of the dinam-mite node towards its applicability to forming a dinam PaaS layer is demonstrated. We then present the MASON mobile vehicular network as an example of such a WSN which delivers spatially and temporally fine-grained environmental measurements within the city of Beijing, and illustrate how to utilize the dinam PaaS for integrating the data from the MASON network into its back-end business system. Finally, we discuss the five essential properties of the Cloud Computing stack, according to the NIST definition, with respect to the dinam PaaS and illustrate the benefits of the dinam PaaS for system integration as well as WSN application development.**

***Index Terms*—Sensor systems and applications, Wireless sensor networks, Distributed computing, Distributed information systems, Quality of service.**

I. Introduction

The concept of Software-as-a-Service (SaaS) allows companies to purchase subscriptions to software systems which run at remote locations, rather than purchasing the software and its hosting infrastructure itself. Furthermore, it allows companies to leverage from the technological and economical advantages of cloud computing for their needs in business software. As with utility computing, cloud computing promotes the delivery of computing resources (e.g. processing and storage) to clients remotely, based on a distributed computing and networking infrastructure [1], [2]. Cloud computing extends this idea by the delivery of concrete software applications (SaaS) hosted on an abstract platform established by the distributed fabric. Furthermore, it is a characteristic of the service delivery architecture defined in cloud computing that there are no relevant concerns on the concrete nature and locations of the infrastructure (i.e., the systems and networks that comprise it). This is enabled by the infrastructure (which the service delivery architecture is built on) which is required to provide constant and location-independent quality of service (QoS), constant reliability and on-demand scalability in terms of processing, storage and networking resources.

Instead of offering bare computing resources or concrete software applications, the concept of Platform-as-a-Service (PaaS) describes to sell application hosting as a service. Analogous to SaaS, in which customers purchase usage rights to software services, PaaS allows clients to purchase services for deployment of consumer-created or acquired applications. Deployed applications immediately benefit from the technological guarantees established by the entire infrastructure: QoS, reliability and scalability. PaaS implementations are often web browser-based and provide comprehensive application development, deployment, monitoring and configuration capabilities [3], [4].

This work proposes to facilitate the concept of an integrated embedded development and run-time appliance (called the *dinam-mite* [5]) to establish an embedded PaaS approach (called the *dinam PaaS*). The approach provides in-situ development, deployment, monitoring and configuration of WSN applications, and transfers the technological benefits of the service delivery architecture of cloud computing to WSNs by designing an analogous architecture that provides analogous guarantees to hosted WSN applications, called the *dinam Cloud*. We demonstrate the benefit of our approach by an exemplary case study to integrate a sensing system into a business system.

Sensing systems in this context primarily refer to wireless sensor networks (WSNs) and business systems refer to service-based systems at enterprise scale, especially Service-Oriented Architecture (SOA) [6]-based, i.e. web services-based, systems, as for example Enterprise Resource Planning (ERP) or Supply Chain Management (SCM) systems. As these kinds of business systems are more and more dependent on contextual information on their planned resources and the entities in their supply chains for optimized execution of business processes, the integration of systems to capture these contexts, i.e. sensing systems, has become important [7], [8].

The guarantees provided by the dinam Cloud fundamentally simplify development and seamless integration of sensing systems into today's business systems. To enhance business systems by contextual information gathered from sensing systems easily and reliably, the architecture provides QoS, reliability, and it provides an abstract platform on a heterogeneous and distributed WSN fabric to enable on-demand scalability of resources. In this paper, we focus on the architecture of the dinam Cloud as a key enabler to easy and seamless integration of sensing systems into business systems. We do not discuss communication architectures, as for example publish-subscribe for the SensorCloud as introduced in [7].

Our concept is not the first PaaS-style approach for the development and deployment of embedded software. The MBED project from ARM [9] provides a cloud-based and PaaS-like approach for their micro controller-based rapid prototyping platform. The primary difference between our approach and MBED is that the dinam PaaS is built on a cloud architecture that provides an abstract platform and guarantees to hosted applications. Furthermore, in contrast to MBED, the dinam PaaS concept is built on a set of stand-alone information appliances that do not require a central infrastructure. To the authors' knowledge, the concept of an embedded PaaS which provides guarantees to applications which are vital for business integration, and which does not depend on a central infrastructure, is being discussed here for the first time.

This paper will begin by introducing the dinam PaaS concept, and our implementation of it, in the context of the dinam Cloud architecture. Afterwards, our PaaS approach will be evaluated: (1) in terms of our implementation's processing scalability, (2) in terms the concept's applicability to a use case, (3) and by a discussion on its general applicability to integration into back-end business applications. In the context of the use case evaluation, a real-world application based on a city-wide sensing system in Beijing, as well as a business application to reduce the cost of energy, will be introduced. We will hypothetically evaluate to integrate the data from the city-wide sensing system into the business application. Finally, we conclude by summarizing our discussion.

II. Dinam Cloud Architecture

The most notable of cloud computing definitions is that published by the U.S. National Institute of Standards and Technology (NIST) [3], whose so-called technical cloud stack consists of three layers, i.e. *Software as a Service (SaaS)*, *Platform as a Service (PaaS)* and *Infrastructure as a Service (IaaS)*, see Figure 1.

According to these service delivery layers of cloud computing, we present in the following the proposed dinam Cloud stack. Figure 1 schematically depicts the Cloud Computing stack as well as the dinam Cloud stack:

Dinam IaaS: The infrastructure services delivery layer of the proposed dinam Cloud architecture provides fundamental computing resources (e.g. processing, storage, networking, etc.) in the context of WSNs, i.e. using sensor nodes. Conceptually, these infrastructure services could be marketed as a fully outsourced service for consumer's own WSN application

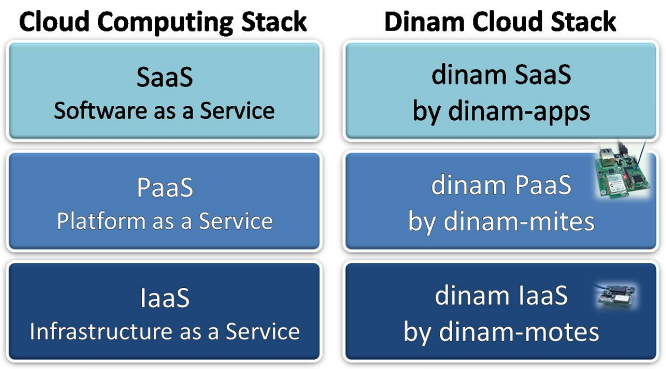


Fig. 1. The dinam Cloud Architecture

needs. Our reference implementation of this layer is based on the uPart low-power wireless sensor nodes [10].

Dinam PaaS: The platform services delivery layer of the proposed dinam Cloud architecture provides computing platform services for the *rapid* development, deployment, monitoring and configuration of WSN applications. Furthermore, the dinam PaaS facilitates an abstract platform to WSN applications in the dinam SaaS layer. The layer presents a unified layer of networking, storage, database and service interfaces to applications, as shown in Figure 2. And applications are essentially enabled to run on top of a potentially heterogeneous infrastructure, i.e. to run on the dinam IaaS layer which is formed by potentially heterogeneous wireless sensor nodes.

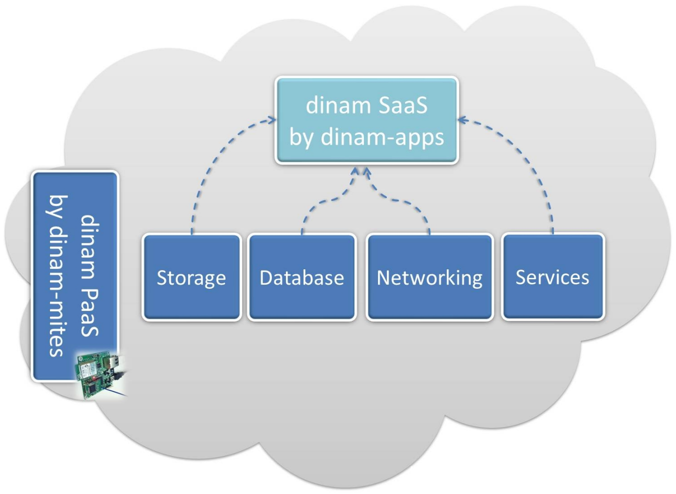


Fig. 2. The dinam PaaS Layer

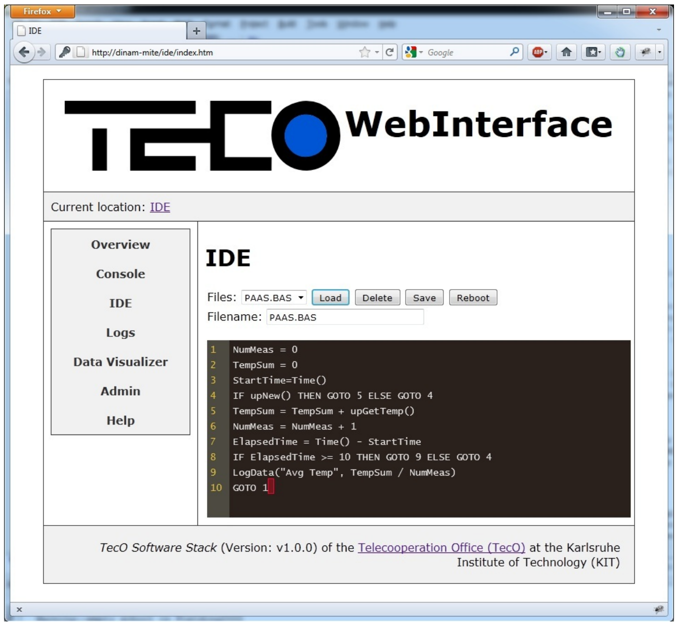
The dinam PaaS layer hides the actual underlying WSN hardware and software and its inherent complexity from running applications and customers. Customers are protected from buying and managing these complex ecosystems physically, and hosted applications benefit from prominent Cloud technology features, such as the easy migration of hosted applications between systems in the infrastructure. All of which, in general, results in synergetic effects to reduce cost of customers [11]. Our reference implementation of this layer is based on the dinam-mite embedded information appliance (and wireless sensor node) [5].

Dinam SaaS: The software services delivery layer of the proposed dinam Cloud architecture is composed of the WSN applications hosted on the dinam PaaS layer, i.e. the concrete service implementations for integration of concrete wireless sensing systems into concrete business systems form the dinam SaaS layer. In our application study in section III, we provide an example application for integrating an environmental monitoring system into an energy consumption prediction business system.

Conceptually, applications are enabled to offer (arbitrary WSN-related) services which are accessible from various client devices based on a broad network accessibility. But the dinam SaaS layer is dependent on the service access provided by the PaaS layer. Our reference implementation is based on a run-time library on the dinam-mite node which provides the ability to register web service end-points to hosted applications. These services can be consumed by clients using standardized REST web services technology.

A. *Dinam PaaS Implementation*

Our reference implementation of our novel PaaS concept primarily provides a programming and run-time environment for rapid development and deployment of services between *complex* sensing systems and *complex* business systems. The programming environment provides access to a web-based IDE (shown in Figure 3) that developers can use to implement services which integrate sensing systems into business systems. The services are based on a service API which is available to the programming and run-time environment.



Current location: IDE

IDE

Files: PAAS.BAS

Filename: PAAS.BAS

```
1  NumMeas = 0
2  TempSum = 0
3  StartTime=Time()
4  IF upNew() THEN GOTO 5 ELSE GOTO 4
5  TempSum = TempSum + upGetTemp()
6  NumMeas = NumMeas + 1
7  ElapsedTime = Time() - StartTime
8  IF ElapsedTime >= 10 THEN GOTO 9 ELSE GOTO 4
9  LogData("Avg Temp", TempSum / NumMeas)
10 GOTO 1
```

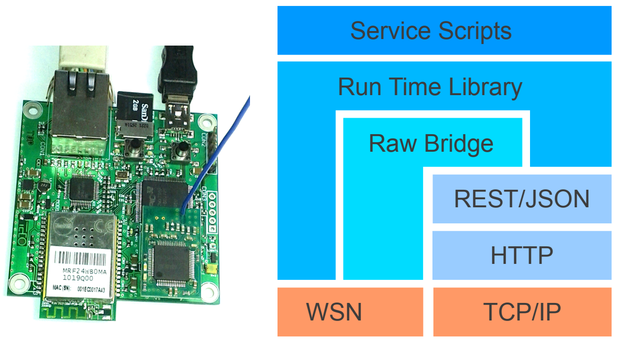
TecO Software Stack (Version: v1.0.0) of the Telecooperation Office (TecO) at the Karlsruhe Institute of Technology (KIT)

Fig. 3. Dinam-mite Web-Based IDE

The implementation is based on the dinam-mite HW and SW platform (shown in Figure 4) which is an embedded system for rapid sensing system applications development, whereby our reference implementation provides the introduced service capabilities of the Dinam PaaS layer.

The hardware platform (shown in Figure 4(a)) provides physical networking interfaces to communicate with WSNs and IP-based Ethernet or Wi-Fi networks.

The software architecture of the dinam-mite platform relevant for communications between sensor nodes and business systems is depicted in Figure 4(b). On its bottom, the system provides access to WSNs and IP-based networks. Both of which may become rather complex, for example in case of the presented Urban WSN in Beijing (as illustrated in our application case study in section III), or ERP enterprise systems in general.



(a) HW Platform (b) SW Platform

Fig. 4. Dinam PaaS Reference Implementation: The dinam-mite HW and SW Platform

Our technology resides on top of both communication stacks to implement services for integration of WSNs and business systems. These services are primarily founded on a dedicated bridging layer which natively provides *simple* bridging services, and on a BASIC [12] interpreter which provides *simple* and *complex* bridging services using user-defined programs in BASIC. The interpreter is equipped with an according run-time library that provides an API (also called *bridging stubs*) for access to both network ends which represents our programming framework for services connecting both network ends with each other. Simple bridging refers to bridging of raw incoming data (e.g. raw sensor values) and complex bridging to bridging of incoming data after they have been processed, e.g. filtered and/or recombined. BASIC scripts running on the interpreter implement services that perform either simple or complex bridging.

In general, the bridging of the dedicated layer and the stubs is in a *push* fashion, meaning that any packets coming in from either network end can be pushed into the other network end, i.e. based on REST [13] over TCP or UDP in IP-based networks. But in case of IP-based networks, also a *pull* fashion has been implemented which provides the ability to pull data which have come in a dinam-mite node via REST to remote IP-nodes. The dedicated bridging layer is only able to push any raw incoming data from sensor nodes into IP-based business systems and vice versa. The bridging stubs provide pull and push functionality to both network ends – BASIC programs may either push incoming data into the other network end or store them locally until an IP-node pulls them. Developers may use the web-based IDE to implement services (based on either simple or complex bridging) in BASIC using our provided bridging framework to integrate WSNs and business systems.

B. Performance Evaluation of Dinam PaaS Implementation

Before we evaluate the dinam PaaS in the context of a use case in section III, we will analyze the performance of our dinam PaaS implementation (which is based on the dinam-mite node). This analysis primarily determines recommendations on how to configure the BASIC run-time environment to preserve an adequate overall performance of the dinam-mite when multiple BASIC programs are executed concurrently. The configuration recommendations are affected by the application scenario the dinam-mite node is used in, i.e. recommendations are deduced from the number of programs concurrently run on the dinam-mite and the processing power each one requires. The processing power is defined by the budget which the scheduler allocates to each script.[1]

This performance analysis' primary goal is to quantitatively answer the question how the round robin-based scheduler's budget configuration and the number of concurrently executed BASIC programs affects the dinam-mite's overall performance, in the current interpreter implementation. Therefore, the assessed variables in this benchmark are (1) the budget that the scheduler accounts to each script and (2) the number of concurrently running BASIC programs. (Note: The configured budget directly affects the execution performance of a BASIC program on the dinam-mite node, as it defines how many operations a script is allowed to perform in one iteration of the scheduler.)

On the one hand, this analysis will demonstrate the applicability of our reference implementation to the dinam PaaS concept from a processing scalability perspective. And on the other hand and more importantly, the following results have established the ground, in terms of background knowledge, for monitoring and run-time governance components on the dinam-mite node, i.e. our reference implementation of the dinam PaaS concept. The combination of both components allows a dinam-mite node to provide essential Cloud Computing characteristics as discussed in the following section V. In particular, the ability to monitor and react dynamically to processing resources demands of hosted services is provided.

The overall performance of the dinam-mite can for example be measured by downloading a file from the microSD card via the web server. This seems to be a deliberate choice, as downloads involve data processing in the web server and I/O operations in the file system driver, all of which runs concurrently to the interpretation of BASIC programs. Therefore, a file download is used as overall performance indicator in this analysis.

[1]In the current implementation of the scheduler (round robin), all script have equivalent budgets.

Additionally, the analysis tries to qualitatively interpret the results of the performance assessment. It is expected that the benchmark reveals a trade-off between budget configuration (script performance) and the number of concurrently running BASIC programs, with respect to the overall performance of the dinam-mite node. Figuratively speaking, a "sliding bar controller" will hopefully become evident. The slider's two extremes are: (1) high budget (script processing power) and a low number of scripts and (2) low budget (script processing power) and a high number of scripts. Meaning that, given a required overall performance of the dinam-mite node, the slider tells how many scripts a given budget may serve while preserving the required overall performance level of the dinam-mite node, or how the budget should be configured to preserve the overall required performance level, when the number of scripts is given. This information in consequence could for example be employed to give recommendations on how to configure the budget when a set of BASIC programs to be run on a dinam-mite node is given.

```
1 temp = Time()
2 GOTO 1
```

Listing 1. BASIC Code for Benchmark Sripts

The BASIC script listed in Listing 1 is used as the benchmark program in this analysis. The script performs an endless loop, whereby each loop iteration uses the run-time library to fetch the dinam-mite node's current local time. This script models commonly expected behavior of BASIC programs on a dinam-mite node. Common BASIC scripts are expected to have infinite behavior which is dominated by conditional tests, arithmetic calculations, (fast) run-time library requests and a few I/O operations (i.e. relatively slow run-time library requests), for example in case logs are written to permanent storage or remote UDP/TCP connections are initiated.

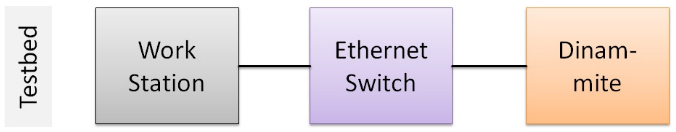


Fig. 5. Testbed for Performance Evaluation

The testbed for this section is illustrated in Figure 5. In essence, the work station issues a file download request to a dinam-mite node using the GNU wget [14] command line tool which outputs average download speeds. The command used for benchmarking is depicted in Listing 2.

```
$ wget --delete-after -v
http://169.254.1.1/test.dat
```

Listing 2. Console Command on Workstation for Benchmarking

The test routine was divided into 6 iterations. In each of the iteration, the dinam-mite node's software stack has been compiled using a different static budget in the (round robin) scheduler, ranging from 10^0 to 10^5. Additionally, 6 performance tests have been executed, whereby each performance test had a different number of concurrently running BASIC program instances of the program listed in Listing 1 in place, i.e. from 0 to 5 instances. After setting up the proper number of scripts in one performance test, a *file of 5 MiB in size* has been downloaded from the dinam-mite node using the wget command on the work station. The test results are depicted in Table I in numbers.

TABLE I
PERFORMANCE RESULTS

Budget \ #[2]	0	1	2	3	4	5
10^0	157	157	156	156	155	155
10^1	157	154	151	157[3]	156[3]	154[3]
10^2	157	147	133	122	113	104
10^3	157	77	50.3	37.5	29.9	24.6
10^4	157	13.4	7	4.73	3.58	2.87
10^5	157	1.44	0.743	0.496	0.363	0.295

There are two remarkable aspects regarding the information in the table. On the one hand, there is an unexpected sudden increase in file download performance for the 10^1 budget with 3 concurrent BASIC scripts. At the moment it is not clear to us what causes this behavior. One plausible explanation might be that inter-communication timings of TCP/IP/Ethernet between the work station, the Ethernet switch and the dinam-mite node are "nicely" matched/served in this setting. On the other hand, the figure shows which budgets are impractical for properly most application scenarios. The lowest possible budget of 10^0 imposes a low run-time performance for the BASIC program on the dinam-mite node which is probably too slow for most transformational systems and also reactive systems. We expect budgets equal to or lower than 10^1 to be hardly used in practical scenarios. Also, budgets equal to or higher than 10^5 are probably impractical, as the overall performance tremendously drops, even if only a single BASIC program is run. In addition, e.g. with a budget of 10^5, the tests showed that the interpreter's scheduler becomes very "uncooperative in multitasking". With a budget of 10^4 or 10^5 the scheduler spends long periods of time in script execution which blocks the cooperative multitasking scheduler of the dinam-mite node's core software stack for long periods.

The information in the previous table are also plotted in Figure 6, whereby the overall performance is illustrated on the y-axis and the number of scripts on the x-axis. All values that belong to a specific row in Table I are highlighted using a distinct color. Besides the bare marks, the rows of the table (equi-colored marks) have been fitted into linear, exponential or polynomial functions, as depicted too in Figure 6. Each function serves to approximate the overall performance decay when adding more and more concurrent BASIC program instances to the interpreter which schedules the scripts using a fixed budget.

[2]Number of concurrent programs
[3]This value has been double-checked.

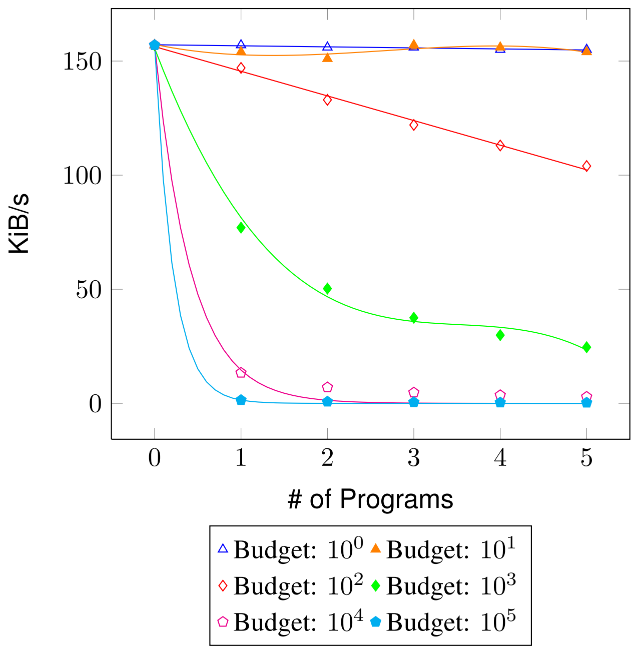


Fig. 6. Function-Fitted Plots of Performance Test Data

The figure illustrates that a budget of 10^2 seems to offer an adequate trade-off between BASIC program performance (budget) and the number of scripts that can concurrently be run, whereby the overall system performance is kept above 100 KiB/s in all tests for the 10^2 budget. For this reason, the interpreter's scheduler by default is set to a budget of 10^2 which probably serves a wide range of potential applications running on the dinam-mite node in BASIC. But the budgets of 10^1 and 10^3 might also be interesting. The run-time performance for BASIC scripts offered by the 10^3 budget is 10 times higher than with 10^2 and the overall system performance for one running BASIC script is still moderate (77 KiB/s). This might be very interesting if only one BASIC script is hosted on the dinam-mite which requires higher processing power than the 10^2 budget could possibly offer. Analog deliberations hold for the 10^1 budget which might be very interesting if only processing resource-friendly scripts are hosted and the application scenario requires other modules on the dinam-mite node to be fast and highly responsive.

III. Application Example for Dinam PaaS

In this section a real WSN application will be introduced, followed by a business process which could be improved by integrating the sensed data from the WSN. First, a real WSN implementation for city-wide environmental sensing, called MASON (Metropolitan Area Sensing and Operating Networks) [15], [16], will be introduced, which collects fine-grained environmental data within the city of Beijing. Afterwards, a business process within an enterprise system of power production companies in and around Beijing is illustrated, whereby focus is put on their prediction process for future energy consumption in Beijing. In the next section an application example for the dinam PaaS is presented to demonstrate the potential of the PaaS for rapid development of WSN applications.

A. *The MASON Urban Sensing Platform*

MASON is a mobile vehicular network created by Tsinghua University which provides sensing coverage for urban areas. The network has been deployed within the city of Beijing, and constantly samples environmental parameters and location information using nodes attached to taxis which travel through the city. Each sensor node is equipped with the following sensors: GPS, temperature, humidity, carbon-monoxide, and 3-axis accelerometer. Figure 7 displays GPS locations of sensor measurements generated by the MASON prototype over the course of one day.

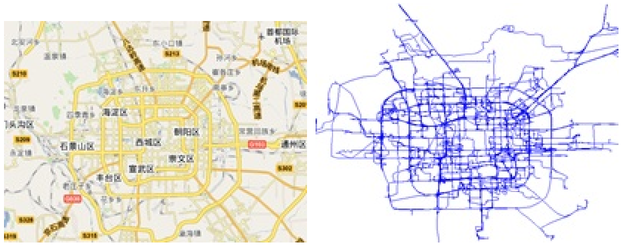


Fig. 7. GPS Traces Generated by the MASON Prototype in Beijing

The nodes use IP-based routing, and can be accessed directly when they are in range of a base station. When out of range of a base station, a MASON node stores the sampled data locally. When a connection has been established, a node uploads its data to a central system (through the just connected base station).

B. *Business Process*

In general, an energy utility process [17] of an energy utility company consists of a specifically ordered set of activities across time and location. The activities have clearly defined inputs and outputs, and they are performed by various computing systems inside of a complex enterprise system. Each system provides specialized applications according to its role in the business process.

The prediction of the energy demand for customers is an essential part of such energy utility processes. But currently, the majority of prediction implementation is purely based on long-term statistics which does not take the dynamically changing real-time power consumption into account. But it has been shown that various environmental factors, such as location, temperature and humidity have significant impact on the accuracy of power demand prediction. Furthermore, the analysis of urban-scale information on improvement of the energy consumption performance [18], [19] and the development of Business Intelligences [20] both proved that different aspects of urban information can be an effective support to derive a more accurate prediction of urban energy consumption [21].

Estimating these factors in real time enables spatially fine-grained energy consumption prediction. In consequence, spatially fine-grained and short-term (intra-day) control of power consumption is fostered, which is an interest of both, energy suppliers and energy consumers. The intra-day control translates directly into reduced the monetary loss, e.g. for a big industry (consumer) that could be 500 € every 15 min [22].

IV. System Integration Example Using the Dinam PaaS Implementation

In this section, the usefulness of the dinam PaaS for rapid development of WSN applications will be demonstrated by an example based on the energy prediction services envisaged in the previous section. We will integrate a wireless sensing system into an energy utility (business) process by a simple application on our dinam PaaS reference implementation.

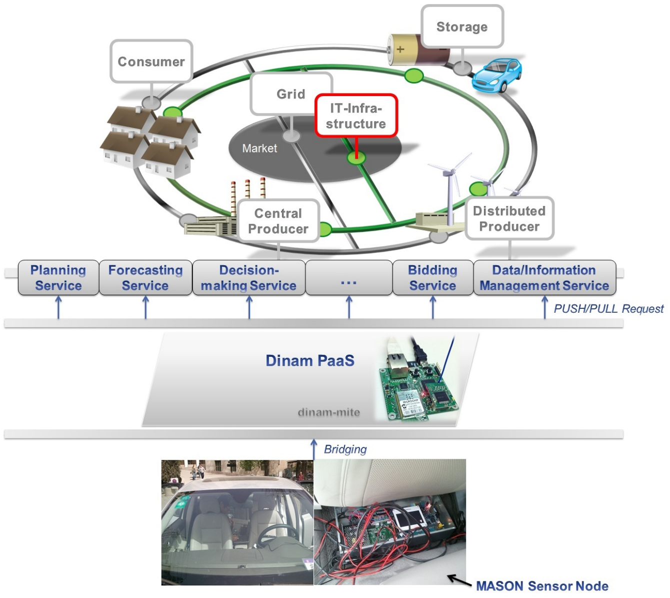


Fig. 8. Integration Architecture of Energy Business Process

At its top layer, Figure 8 shows the schematic structure of an urban-scale energy ecosystem, consisting of energy producers, consumers, transporters and supporting IT infrastructure. Using a dinam PaaS implementation on the middle layer, a power demand forecasting service is provided and enabled for business integration. The services encapsulate existing forecasting mechanisms that provide short-term energy consumption prediction using monitoring of environmental factors. The environmental monitoring functionality is provided by the MASON network whose seamless integration into the service system is enabled by the Cloud characteristics (discussed in section V) of the dinam PaaS.

Based on the WSN application description in the previous section, we assume that the PaaS devices are placed at strategic locations throughout the city, and that passing vehicles upload sampled data to nearby devices when in range. We also assume that a fine-granular real-time temperature map of different areas of the map would increase the prediction accuracies of the energy consumption by allowing them to better estimate how many inhabitants will use AC or heating devices.

The example in Listing 3 shows a BASIC script which calculates the average temperature in 10 second intervals for the immediate area around the dinam PaaS base station. The computing platform gathers local temperature readings from local taxis in its proximity. Every 10 seconds the average is the calculated and uploaded to a energy producer's database using a REST API.

```
1 NumMeas = 0
2 TempSum = 0
3 StartTime = Time()
4 IF NewPacket() THEN GOTO 5 ELSE GOTO 4
5 TempSum = TempSum + GetTemp()
6 NumMeas = NumMeas + 1
7 Period = Time() - StartTime
8 IF Period>=10 THEN GOTO 9 ELSE GOTO 4
9 HTTPput("http://domain.cn/CouchDB/",
          "Average␣Temp␣for␣Region␣17",
          TempSum / NumMeas)
10 GOTO 1
```

Listing 3. BASIC Code for Temperature Application

In the example, lines No. 1 through 3 initialize variables for holding the number of measurements received, the sum of all measurements and the timestamp for the start of the measurement period. Every time a new measurement packet arrives, lines 4 through 6 extract the temperature from that packet, add it to the sum of all of the temperature measurements and increase the count of received measurements by one. Lines 6 through 8 check to see if 10 seconds have passed, at which point the average temperature for that region over that period of time is then uploaded to a database. Line 10 then returns to the first line, resetting the program.

This simple example demonstrates the effectiveness of the dinam-mite as a PaaS for the rapid development of WSN applications. In this case, the energy producer would be able to create a computing platform which calculates a regionally and temporally fine-grained average temperature and place it on their servers. There it can now be integrated into their power consumption prediction processes to improve the efficiency of their systems, thereby lowering their costs.

As demonstrated previously by Gordon et al. [5], application construction requires little or no understanding of the technical details of the underlying system and can be created in 5 minutes or less. The applications are generated by accessing the IDE served by the PaaS with nothing more than a web browser, minimal abstract understanding of the underlying application and an introduction lasting only a few minutes [5]. The resulting applications can combine WSN communication, network bridging and data processing operations to integrate valuable data generated by WSN applications into business processes which can make use of that data.

V. Discussion

In WSN applications, each sensor node is wireless and usually has to operate for long periods of time, it has therefore inherently strict energy resource constraints. To guarantee low power needs, sensor nodes have then a limited processing power, storage capacity and communication bandwidth. As the consequence of offsetting these limitations, the deployment of sensor nodes has to be highly dense and has to have a high degree of interaction among them [23]. Furthermore, as Akyildiz et al. [24] point out: the cost of WSN deployment and maintenance must be low. That's why we introduced the dinam Cloud as a new type of service to enable rapid development of WSN applications with minimum cost.

Similar to the NIST definition of Cloud Computing [3], an instance of the proposed dinam Cloud could also be rapidly provisioned and released with minimal management effort. Therefore, the dinam Cloud stack explicitly addresses the five essential characteristics [3] of cloud computing in the following ways:

- *On-demand self-service.* A consumer can provision computing capabilities by deploying additional instances of the dinam-mite node. The processing power of each script can be configured in a node's web interface. The overall provisioning of resources runs automatically and in a context-aware manner, without requiring any interaction with the dinam-mite service provider. This process is aware of the context (e.g. geographic location) of a dinam-mite node to accommodate for contextual constraints of hosted application, which, for example, may only be executed in a certain geographic area.
- *Broad network access.* As described in Sec. II-A, the dinam-mite platform provides networking interfaces to communicate with WSNs and IP-based networks. Using IP-based networks, the applications forming a set of services hosted on the dinam PaaS are accessible via REST [13] over TCP or UDP. Based on this, the services hosted on the dinam PaaS are accessible from various client devices, for example using a web browser. Furthermore, applications on dinam-mite nodes are able to establish connections (reliable and unreliable) to both networking domains, i.e. WSN and IP-based networks. For example, applications may establish a connection to an external REST interface to transmit data into a database.
- *Resource pooling.* In principle, multiple consumers can upload arbitrarily many BASIC scripts to the same dinam-mite node. As immediate consequence, all BASIC scripts will be run concurrently on one dinam-mite node. In case that the main memory of a node is exhausted, the BASIC scripts will be reassigned to another node.
- *Rapid elasticity.* In one network, consumers can deploy arbitrarily many dinam-mites to form an entire computing platform. As mentioned above, the BASIC scripts can be automatically reassigned to other dinam-mites.
- *Measured service.* A monitor creates profiles of running applications on a dinam-mite node. The profiles track how

scripts behave with respect to processing and input/output. If scripts do not perform adequately from a processing or IO perspective (as requested by a customer), the allocated service qualities for processing and IO can be adapted using a node's web interface.

VI. Conclusion

In this paper, the concept of an embedded Platform-as-a-Service for in-situ development of WSN applications was introduced. The approach is based on the dinam-mite concept which propagates the use of integrating development and runtime environment into a single information appliance to rapidly develop embedded software. The system's implementation is based on the dinam-mite node, which is a wireless sensor node with an embedded development environment which is served by an embedded web server. The IDE provides users with the ability to combine data, processing and networking commands to programs using a version of BASIC which has been adapted for these purposes.

Besides conducting a performance evaluation of the dinam PaaS with respect to its scalability, a real world scenario for WSN system integration using the dinam PaaS was introduced. Using instrumented taxis, gathered wireless sensor data are integrated into the business processes of energy suppliers. Based on related research, it was illustrated that energy consumption forecasting processes within these companies can incorporate this data to improve forecasting, especially short-term prediction. Afterwards, the dinam-mite concept and node were hypothetically evaluated as a PaaS for application development within the example application. In this context, a simple computing platform program was developed which provides temperature averages to the forecasting process on a highly frequent base, every 10 seconds. The results indicate that the dinam PaaS is a promising tool for integrating WSNs into business processes, allowing users to develop applications rapidly through the provided platform services.

Acknowledgment

The research reported in this paper has been partially supported by the Robert Bosch Stiftung for the initiation funding of a cooperative research project "Urban Sensing of City Dynamics and Energy Use in the Beijing Metropolitan Area" (32.5.8003.0102.0). In addition, the authors would like to acknowledge the funding by the European Commission under the ICT project "TIMBUS" (Project No. 269940, FP7-ICT-2009-6) within the 7th Framework Programme.

References

[1] M. N. Huhns and M. P. Singh, "Service-oriented computing: Key concepts and principles," *IEEE Internet Computing*, vol. 9, no. 1, pp. 75–81, Jan. 2005. [Online]. Available: http://dx.doi.org/10.1109/MIC.2005.21

[2] A. Dubey and D. Wagle, "Delivering software as a service," *McKinseyQ*, vol. Web exclusive, no. May, 2007.

[3] P. Mell and T. Grance, "Nist definition of cloud computing," National Institute of Standards and Technology, Tech. Rep., October 7 2009.

[4] G. Lawton, "Developing software online with platform-as-a-service technology," *Computer*, vol. 41, no. 6, pp. 13 –15, june 2008.

[5] D. Gordon, M. Beigl, and M. A. Neumann, "dinam: A wireless sensor network concept and platform for rapid development," in *Proceedings of the Seventh Internation Conference on Networked Sensing Systems*. Kassel, Germany: IEEE, 2010, pp. 57 – 60.

[6] T. Erl, *Service-Oriented Architecture: Concepts, Technology, and Design*. Upper Saddle River, NJ, USA: Prentice Hall PTR, 2005.

[7] M. M. Hassan, B. Song, and E.-N. Huh, "A framework of sensor-cloud integration opportunities and challenges," in *Proceedings of the 3rd International Conference on Ubiquitous Information Management and Communication*, ser. ICUIMC '09. New York, NY, USA: ACM, 2009, pp. 618–626. [Online]. Available: http://doi.acm.org/10.1145/1516241.1516350

[8] C. Decker, P. Spiess, L. M. S. D. Souza, and M. Beigl, "Coupling enterprise systems with wireless sensor nodes: Analysis, implementation, experiences and guidelines," in *In Pervasive Technology Applied @ PERVASIVE*, 2006.

[9] A. R. M. Ltd., "The MBED Rapid Prototyping Platform for Microcontrollers," www.mbed.org, 2010.

[10] M. Beigl, C. Decker, A. Krohn, T. Riedel, and T. Zimmer, "uparts: Low cost sensor networks at scale," *Demo at Ubicomp 2005, Tokyo, Japan*, Sept. 11-14 2005.

[11] P. Mathur and N. Nishchal, "Cloud computing: New challenge to the entire computer industry," in *Parallel Distributed and Grid Computing (PDGC), 2010 1st International Conference on*, oct. 2010, pp. 223 –228.

[12] T. E. Kurtz, "Basic," *History of programming languages I*, pp. 515–537, 1981.

[13] R. T. Fielding, "Architectural styles and the design of network-based software architectures," Ph.D. dissertation, 2000, aAI9980887.

[14] H. Niksic *et al.*, "Gnu wget 1.13.4," The non-interactive download utility, Tech. Rep., 2011. [Online]. Available: http://www.gnu.org/software/wget/manual/wget.pdf

[15] X. Yu, H. Zhao, L. Zhang, S. Wu, B. Krish-namachari, and V. O. Li, "Cooperative sensing and compression in vehicular sensor networks for urban monitoring," in *Proceedings of IEEE International Communications Conference (ICC'10)*. IEEE, 2010.

[16] L. Zhang, W. Zhang, X. Mao, J. Jiao, S. Zheng, L. Li, Y. Liu, T. Wang, and M. Gu, "Nomad - networked-observation and mobile-agent-based scene abstraction and determination," in *Proceedings of the 8th ACM Conference on Embedded Networked Sensor Systems (SenSys'10)*. ACM Press, 2010.

[17] S. Neumann, K. Shankar, and A. Vojdani, *Applying Workflow Technologies to Integrate Utility Business Processes*, Utility Integration Solutions, Inc., 2005.

[18] C. Ratti, N. Baker, and K. Steemers, "Energy consumption and urban texture," *Energy and Buildings*, vol. 37, no. 7, pp. 762 – 776, 2005.

[19] W. Fong, H. Matsumoto, Y. Lun, and R. Kimura, "System dynamic model for the prediction of urban energy consumption trends," in *Proceeding I of the 6th international conference on indoor air quality, ventilation & energy conservation in buildings (IAQVEC 2007)*, Sendai: Tohoku University, 2007, pp. 762–769.

[20] M. Golfarelli, S. Rizzi, and I. Cella, "Beyond data warehousing: what's next in business intelligence?" in *Proceedings of the 7th ACM international workshop on Data warehousing and OLAP*, ser. DOLAP '04. New York, NY, USA: ACM, 2004, pp. 1–6.

[21] Y. Ding, W. Zhang, T. Miyaki, T. Riedel, L. Zhang, and M. Beigl, "Smart beijing: Correlation of urban electrical energy consumption with urban environmental sensing for optimizing distribution planning," in *Work in Progress (Energy 2011), 1st International Conference on Smart Grids, Green Communications and IT Energy-aware Technologies*. Venice/Mestre, Italy: Think Mind, May 22-27 2011, pp. 98 – 101, iARIA Conference.

[22] Y. Ding, H. R. Schmidtke, and M. Beigl, "Beyond context-awareness: context prediction in an industrial application," in *Proceedings of the 12th ACM international conference adjunct papers on Ubiquitous computing*, ser. Ubicomp '10. New York, NY, USA: ACM, 2010, pp. 401–402. [Online]. Available: http://doi.acm.org/10.1145/1864431.1864457

[23] D. Culler, D. Estrin, and M. Srivastava, "Guest editors' introduction: Overview of sensor networks," *Computer*, vol. 37, no. 8, pp. 41 – 49, aug. 2004.

[24] I. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci, "A survey on sensor networks," *Communications Magazine, IEEE*, vol. 40, no. 8, pp. 102 – 114, aug 2002.