A Platform for Integrating Physical Devices in the Internet of Things

Paulo F. Pires, Everton Cavalcante, Thomaz Barros, Flavia C. Delicato, Thais Batista, Bruno Costa

1Department of Computer Science, Federal University of Rio de Janeiro, Rio de Janeiro, Brazil
2Department of Informatics and Applied Mathematics, Federal University of Rio Grande do Norte, Natal, Brazil

{paulo.f.pires, evertonriandrielly, thomazavila, fdelicato, thaisbatista, bruncosta.dsn}@gmail.com

Abstract—The Internet of Things (IoT) has emerged as a paradigm in which smart things actively collaborate among them and with other physical and virtual objects available in the Web in order to perform high-level tasks. IoT environments are typically characterized by a high degree of heterogeneity, thus encompassing devices with different capabilities, functionalities, and network protocols. In such a scenario, it is necessary to provide abstractions for physical devices and services to applications and end-users, as well as means to manage the interoperability between such heterogeneous elements. In this context, we introduce EcoDiF (Web Ecosystem of Physical Devices), a Web-based platform for integrating heterogeneous physical devices with applications and users in order to provide services to support real-time data control, visualization, processing, and storage. In this paper, we present the main features of EcoDiF and detail its architecture and implementation, which is based on well-known Web technologies such as HTTP, REST, EEML, and EMML. Furthermore, we present a preliminary evaluation of an EcoDiF prototype through proof-of-concept applications from different domains as well as a performance analysis of the platform.

Keywords—Internet of Things; integration of physical devices; IoT middleware; EcoDiF; REST; EEML; EMML.

I. INTRODUCTION

The advances on electronic devices, wireless communications, RFID technology, and the explosive growth of the World Wide Web have contributed to leverage the development of the Internet of Things (IoT) paradigm [1]. In the IoT vision, every object on Earth can be identified, addressed, controlled, and monitored via Internet. IoT promotes the connection of the virtual and physical worlds by extending the existing interaction between men and machines provided by the Internet to new dimensions of human-to-thing (H2T) and thing-to-thing (T2T) communications.

IoT environments are typically characterized by a high degree of hardware and software heterogeneity, thus encompassing several devices with different capabilities, functionalities, and network protocols. Furthermore, the wide dissemination of the IoT paradigm has the potential to produce a considerable impact in the daily lives of the human beings motivated by the emergence of new applications from several domains that are based on interacting physical devices that can be engaged in complex relationships. In this scenario, there are several challenges to be addressed in order to concretize the IoT paradigm. One of them is related to the need of an architecture that is able to: (i) efficiently support the heterogeneity and dynamics inherent to IoT environments; (ii) provide abstractions over physical devices and services to applications and/or final users; (iii) provide search and discovery services for these elements; (iv) enable the connection among these elements through the network; (v) monitor location and state of the connected elements, and; (vi) provide means to manage the interoperability among such heterogeneous elements.

Aiming at proposing a solution for some of these challenges, in this paper we present EcoDiF (Web Ecosystem of Physical Devices) [2], a Web-based platform for connecting devices with applications and/or end-users in an IoT ecosystem [3] to enable the development of new ideas and products in an organic way. The main goals of EcoDiF are to integrate heterogeneous physical devices (and consequently connect them to the Internet) and to provide functionalities such as real-time data control, visualization, processing and storage, as well as to enable their use in several application domains, such as environmental monitoring, public urban infrastructure monitoring, health care, prevention of environmental disasters, etc.

EcoDiF is aligned with the Web of Things (WoT) paradigm, which uses existing Web technologies and protocols to enable the inclusion of physical devices in the digital world, so that their data and services are (re)used in different applications as any Web resource [4], thus leveraging the concretization of the IoT vision [5]. In this perspective, EcoDiF was designed and implemented based on REST (REpresentational State Transfer) principles [6, 7] and it relies on current Web standards and protocols, such as HTTP and URIs. Therefore, the HTTP protocol is not only used as a communication protocol to carry formatted data (as in Web services technologies, for example), but it is also used as the standard mechanism to support all interactions with smart objects. These interactions take place through the HTTP main operations (i.e., GET, POST, PUT, and DELETE), which provide a well-defined interface to expose the functionality of the objects in the Web. Such an interface complies with the REST principles due to its reduced complexity and loosely coupled operations [8], thus enabling devices to be accessed as resources in a Resource-Oriented Architecture (ROA) approach [9].

The adoption of the REST approach enables EcoDiF to standardize and simplify the application development processes, besides minimizing barriers related to compatibility issues between different manufacturers, proprietary protocols, and data formats [10] and addressing part of the interoperability issues in an IoT environment. Such interoperability and abstraction of devices heterogeneity are essential.
requirements to middleware platforms in the context of the IoT paradigm and consolidate one of the main goals of EcoDiF, which is to integrate heterogeneous physical devices. Therefore, EcoDiF is able to provide a unified access to data and services provided by integrated devices through high-level interfaces, as well as to make the development of applications in IoT environments easier.

In EcoDiF, data provided by devices (feeds) are structured by using the EEML (Extended Environments Markup Language) language [12], which provides a simple XML-based format for describing data obtained from devices in a specific context (environment). Once these data are made available at EcoDiF, they can be used by applications built as Web mashups [11], which are ad-hoc Web applications created from the composition of different types of information provided by several sources. In order to build and execute these applications, EcoDiF relies on EMMML (Enterprise Mashup Markup Language) [16], which is an open declarative language used as a standard for developing mashups.

In this paper, we describe the main features of EcoDiF, its architecture and implementation, as well as a preliminary evaluation of a prototype of the platform. The remainder of the paper is organized as follows. Sections II and III describe EcoDiF platform. Section IV presents evaluations of the current EcoDiF prototype through proof-of-concept applications from different domains and a performance analysis of the platform. Section V analyzes related work. Finally, Section VI contains final remarks and future works.

II. EcoDiF Architecture

In order to achieve its goals, EcoDiF defines several loosely coupled modules (middleware services) that compose its logical architecture, as illustrated in Fig. 1.

The Devices Connection Module aims to facilitate the connection of physical devices according to the EcoDiF’s API and by making use of customized drivers, which are developed for each specific type of device platform. These drivers play a very important role regarding the integration of devices with EcoDiF since the heterogeneity of such devices is abstracted away from users and applications use data provided by them. In EcoDiF, there are two types of drivers, namely active and passive drivers. Active drivers obtain data collected by the devices by making periodical requests to the device’s API even if the collected data values remain unchanged. On the other hand, passive (or event-driven) drivers wait for notifications from the device’s API that are triggered whenever there are changes in the data values. After obtaining data from the devices (the so-called feeds), the drivers structure them in the EEML format to be sent to EcoDiF through HTTP PUT requests in order to be registered at the platform by the Data Manipulation Module.

The Visualization and Management Module provides abstractions over physical devices through a Web interface that enables users to manage devices connected to EcoDiF. Via such an interface, users can monitor the state and the location of their devices and visualize historical data stored in the platform. Moreover, users can create triggers, which are event-based mechanisms that enable to notify them based on conditions defined in terms of the current values for the feeds. For instance, consider a feed associated with the monitoring of temperature in a given environment. In this case, users can create a trigger to notify them whenever the measured temperature is greater than 40 degrees Celsius.

The Collaboration Module aims to facilitate the collaboration among EcoDiF users, for instance enabling them to perform searches for devices and applications from their respective metadata (type, user, localization, etc.) through the EcoDiF’s Web interface. The Storage Module consists of two basic repositories: (i) a repository for storing data by using a relational database, and; (ii) a repository for storing application scripts in a file system. These repositories can use a Cloud Computing infrastructure [13] to store relational data and files, thus providing quality attributes such as robustness, reliability, security, availability, and scalability [14, 15]. The Common Services Module encompasses infrastructure services offered by the platform, such as security (in terms of user authenticity, confidentiality, and integrity), applications life cycle management, transactions, etc.

The Applications Module provides a model and environment for programming and executing mashups, which are applications that make use of data (feeds) available at EcoDiF and generate new information to be made available at the platform to be used by other feeds and other applications. For instance, consider a sensor that monitors the temperature in a given location, but the user wishes to combine this information with a map that informs the location of the collected measures. Therefore, a single mashup application can compose this temperature and location information. The programming and execution model adopted by EcoDiF is based on EMMML, which is an open declarative language used as a standard for developing mashups, despite the proposal of other languages in the literature [17, 18] with a smaller degree of maturity. Mashup applications are implemented as scripts written in the EMMML language and executed in an engine [16] that processes these scripts.

Given the features of its modules, EcoDiF envisions four user profiles (as depicted in Fig. 1), namely:

Figure 1. EcoDiF architecture.
(i) device manufacturers, which develop drivers to their devices in order to make them compatible with the EcoDiF’s API;
(ii) data providers, which are device owners that make data produced by their devices available at EcoDiF through the specific drivers for each device;
(iii) application developers, which build Web applications or services that take as input data available at EcoDiF (raw data provided by the connected devices or more refined data produced as output of other applications) or produced by any other Web resource, and;
(iv) data consumers, which are users that interact with EcoDiF in order to query information available at the platform, i.e., about devices and provided data, applications, etc.

The process of using EcoDiF follows a set of steps performed by users from the creation of drivers to the execution of applications in the platform. First, device manufacturers develop drivers in order to enable the integration of devices to EcoDiF. Next, the device type is registered at EcoDiF and the respective driver (developed to such a type) is uploaded to the platform to be used by data providers. In the next step, data providers connect their devices to EcoDiF by downloading and deploying available drivers. Afterwards, they can create feeds at EcoDiF to their connected devices to send data to the platform. Next, application developers create and execute applications that make use of the feeds previously registered at EcoDiF by data providers. Finally, data consumers can create triggers based on previously registered feeds, as well as query for triggers, applications, feeds, and connected devices.

III. ECODIF: IMPLEMENTATION

Fig. 2 illustrates the main technologies used for implementing the modules that compose the logical architecture of EcoDiF. The platform was implemented by using the Java programming language and deployed on a JBoss application server [19], which allows an easy management of distributed components and large data streams, as typically observed in IoT environments. Users can access the main functionalities offered by EcoDiF through the Web interface provided by the Management and Visualization Module, which was implemented by using the JavaServer Faces (JSF) [20] and Primefaces [21] open-source technologies.

After establishing the connection between EcoDiF and the integrated devices through drivers, data obtained from such devices are sent to the platform through HTTP PUT requests, thus providing a RESTful interface for clients (either human or applications). In order to support a RESTful approach in EcoDiF, we have adopted the RESTEasy implementation [22] for the REST architectural style and the Java API for RESTful Web services (JAX-RS) [23]. Data (feeds) produced by the devices are structured by using the EEML protocol, which is based on the XML language, and are handled by the Data Manipulation Module to be effectively registered in a MySQL relational database by using the Java Persistence API (JPA) [24] specifications implemented in the Hibernate framework [25].

Application developers can build mashup applications by making use of data provided by the devices (feeds) that are integrated to EcoDiF through the Applications Module. In EcoDiF, applications are implemented by using the EMMML language, so that when a feed is bound to an application, the Applications Module automatically includes it as an input variable in the EMMML script for the application, and users can codify the programming logic of such an application. On the other hand, users that have knowledge about the EMMML language can write the EMMML script on their own (outside EcoDiF environment by using and editor of their choice) and then import the script to EcoDiF. Applications registered at EcoDiF are executed by using an EMMML execution engine deployed on the JBoss application server in which the platform itself is deployed, and their respective results are shown at the EcoDiF’s Web interface to users. Finally, the Security Module allows controlling user authenticity, confidentiality, and integrity by using the Java Authentication and Authorization Service (JAAS) [26] specifications, which are implemented in the JBoss application server.

IV. EVALUATION

The current EcoDiF prototype was evaluated in two ways. In the first one (Section IV.A), applications from different domains were developed as proofs-of-concept aiming at demonstrating the functionalities and the use of the platform in practice. In the second one (Section IV.B), a quantitative evaluation was carried out in order to assess its performance regarding scalability and time spent when dealing with a large number of simultaneous requests.

A. Proofs-of-Concept (PoCs)

In order to validate EcoDiF and demonstrate its potential to integrate heterogeneous devices in real-world scenarios, different applications were developed as proofs-of-concept (PoCs). Due to space constraints, in this section we present two of the four developed PoCs. More details about these applications can be found at the following URL address: http://ubicomp.nce.ufrj.br/ecodif/poc-applications_en.

1) Monitoring Datacenters

In datacenters, it is necessary to monitor several assets (e.g., servers, network and storage devices, etc.) and variables related to the physical infrastructure (e.g., temperature,
humidity, etc.) so that managers can be notified when possible abnormalities in the operations arise. However, due to the lack of integration between the adopted monitoring solutions in this scenario, managers are not able to have a systemic view about the real situation of the datacenter and its physical structure and constituent assets. Moreover, they typically need to use specific proprietary software for each device by explicitly accessing them.

In this scenario, EcoDiF can enable the integration of the used devices in a datacenter and make data provided by them available on the Web. Therefore, managers are able to monitor what is happening in the datacenter and can be notified (through e-mail or messages sent to their mobile devices, for example) about meaningful events. As the assets are usually handled by using SNMP (Simple Network Management Protocol), a widely used application-layer protocol designed to manage network and storage assets, we have developed a driver to such a protocol in order to integrate the devices to EcoDiF and make the collected data available at the platform. This driver: (i) obtains information from the devices through the SNMP protocol; (ii) structures such data in EEML, and; (iii) sends the information to EcoDiF through HTTP PUT requests. It is noteworthy that any device that complies with the SNMP protocol can be easily integrated to EcoDiF by using the same developed driver. Furthermore, the information collected from these devices can be easily combined with temperature and humidity information collected by other sensors in order to verify if the current conditions of the datacenter may be harmful for its operations.

We have also developed a mashup application by using the EMML language in order to aggregate information (as feeds) from H3C S5500 switches at a datacenter in the Federal University of Rio Grande do Norte (Natal, Brazil). This application performs requests to EcoDiF to obtain such data and displays them to users through the EcoDiF’s Web interface. Therefore, managers can monitor the state of several variables such as: (i) temperature; (ii) CPU usage; (iii) RAM memory usage; (iv) operational state of the network interfaces; (v) number of input octets in the network interfaces; (vi) number of output octets in the network interfaces; (vii) number of input packages with errors, and; (viii) number of output packages with errors. In addition, triggers were created in order to notify users about warning and critical conditions regarding such monitored parameters, e.g., when the temperature is greater than 50°C (warning) or equal or greater than 55°C (critical).

2) Participatory Sensing

The Participatory Sensing concept [27] has emerged as means of retrieving information by involving users as active players for providing relevant data about the environment, weather, traffic, people, etc. By using their mobile devices and wireless and/or mobile networks, users are able to send information that may range from personal observations to the combination of data from thousands of individuals about the environment where they live. Through wireless and/or mobile networks, participants scattered across a given place, city, or even worldwide, can easily send data to servers that process and integrate them with other data, such as GIS map layers and weather reports. Participatory Sensing has shown to be very useful in several scenarios such as transportation planning, public health maintenance, and environmental sustainability, just to name a few.

One example of application from the Participatory Sensing domain consists in estimating the presence of people in given areas. For instance, in a conference, it may be interesting to check how many participants have attended to each event (workshops, tutorials, etc.). In our PoC scenario, volunteers have downloaded an EcoDiF driver that was made available as a mobile application to the Android operating system and they installed it on their mobile devices (smartphones and/or tablets) to read QRcodes (Quick Response Codes) fixed in the rooms in different locations of the building where a scientific conference was taking place. When executing this application, users have pointed the camera of their devices to the QRcodes, so that the location data were sent to EcoDiF through HTTP PUT requests as feeds structured in the EEML format. In order to make use of such data, we have also developed a mashup application that retrieves this information registered at EcoDiF and composes a Web presence map with the number of attendees that were present in the conference sessions.

B. Quantitative Evaluation

1) Methodology

In order to evaluate EcoDiF under a quantitative perspective, we have performed computational experiments aiming at verifying the performance of the platform in terms of scalability and time spent (latency) when dealing with different number of simultaneous requests and exchanging messages of different sizes.

The experiments were performed by using a dedicated server with Intel® Xeon™ 3.4 GHz processor, 32 GB of RAM, and Linux Ubuntu 12.04 LTS 64-bit operating system, thus representing a single computational node on which no load balancing techniques were applied. In this server, we have installed a JBoss AS 6.1.0 application server to deploy EcoDiF and a MySQL Server 5.5.34 relational database server to store information used by the platform. However, since the original configurations of JBoss server, the MySQL database, and the Java platform are not optimized to a production environment, we have done some modifications in order to extract the maximum of performance from the server. The first one was to enable the operating system to support larger memory paging, so that we have observed an improvement of 55% in the performance of the platform in terms of the response time to the requests when compared to the preliminary tests performed before such a modification. Another modification was an increase from 10 to 50 simultaneous connections of the TM 3.4 GHz processor, 32 GB of RAM, and Linux Ubuntu 12.04 LTS 64-bit operating system, thus representing a single computational node on which no load balancing techniques were applied. In this server, we have installed a JBoss AS 6.1.0 application server to deploy EcoDiF and a MySQL Server 5.5.34 relational database server to store information used by the platform. However, since the original configurations of JBoss server, the MySQL database, and the Java platform are not optimized to a production environment, we have done some modifications in order to extract the maximum of performance from the server. The first one was to enable the operating system to support larger memory paging, so that we have observed an improvement of 55% in the performance of the platform in terms of the response time to the requests when compared to the preliminary tests performed before such a modification. Another modification was an increase from 100 (initial configuration) to 1000 simultaneous connections allowed by the database in order to meet the demands of access. Moreover, in order to avoid a negative effect due to the connections between the JBoss server and the database, the application server was configured to optimize the connection pool, so that the application does not need to establish a new connection at each new query, thus reducing the time for accessing or inserting data. Finally, the JBoss server was configured to log only events related to errors that may compromise the
stability of the application instead of logging all events, which represents an expensive file writing operation.

In order to simulate simultaneous requests performed by users, we have made use of Apache JMeter [28], a Java-based open-source tool for performing automated tests in Web applications. By using three computer nodes with Intel® Core™ i5 3.2 GHz processor, 8 GB of RAM, and Microsoft® Windows™ 7 Professional 64 bits operating system, we have created request plans to address different configurations by varying the number of requests per second (created as threads) and the number of sensor data (datapoints) to be retrieved. For each configuration, 20 independent executions were performed. The server and the computer nodes were placed in the same Ethernet local network where the latency between them could be neglected. In each execution, we have used an initially empty feed (i.e., without any previously registered datapoint) due to the impact that was initially observed when requests were performed over feeds with a large number of data (greater than 1500 datapoints). In these cases, although the platform was able to answer all of the requests, the response times were significantly large when compared to the same tests performed over empty feeds or with few amounts of data.

With JMeter, we have assessed the latency for HTTP GET and HTTP PUT requests for respectively retrieving and sending data from/to EcoDiF. For HTTP GET requests, we have performed requests to a feed registered at EcoDiF to retrieve different numbers of datapoints. In turn, an EEML-based message with a length of 315 bytes and containing a single data collected by a sensor for updating a feed (Fig. 3) was sent through HTTP PUT requests.

![EEML representation of data collected by a sensor and sent to EcoDiF through HTTP PUT requests.](image)

Fig. 3. EEML representation of data collected by a sensor and sent to EcoDiF through HTTP PUT requests.

It is important to highlight that the quantitative evaluations were performed without considering the semantics of a specific application. For example, when changing from an application to another, the amount of datapoints retrieved through HTTP GET requests and sent through HTTP PUT requests would be the same (i.e., the semantics of the application would not affect the results) and the differences would be just in terms of the size of the exchanged messages.

2) Results

Table I presents the minimum, maximum, average, and standard deviation values in milliseconds for the latency of the HTTP GET requests for different numbers of simultaneous requests when retrieving 100 datapoints (about 6000 bytes of message payload). It is possible to notice that EcoDiF handles up to 150 requests per second with latency smaller than 1 second, whereas such a time increases to more than 4 seconds for 500 requests per second and to more than 12 seconds when reaching the maximum rate of 600 requests per second. It was observed an atypical behavior for the configuration with 1 request per second despite the expectation of increasing the latency time when increasing the number of simultaneous requests. A possible hypothesis is due to issues related to the internal processing of the JBoss application server, on which we do not have a fine-grained control.

![Table I. Minimum, maximum, average, and standard deviation for the latency of the HTTP GET requests for retrieving 100 datapoints under different numbers of requests per second.](image)

Table I. Minimum, maximum, average, and standard deviation for the latency of the HTTP GET requests for retrieving 100 datapoints under different numbers of requests per second.

<table>
<thead>
<tr>
<th>Requests per second</th>
<th>Min (ms)</th>
<th>Max (ms)</th>
<th>Avg (ms)</th>
<th>SDev (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>20</td>
<td>3134</td>
<td>91.61</td>
<td>256.37</td>
</tr>
<tr>
<td>5</td>
<td>20</td>
<td>3109</td>
<td>53.28</td>
<td>244.41</td>
</tr>
<tr>
<td>10</td>
<td>20</td>
<td>3104</td>
<td>66.20</td>
<td>236.27</td>
</tr>
<tr>
<td>15</td>
<td>20</td>
<td>3109</td>
<td>83.32</td>
<td>207.75</td>
</tr>
<tr>
<td>20</td>
<td>23</td>
<td>150</td>
<td>94.64</td>
<td>19.24</td>
</tr>
<tr>
<td>25</td>
<td>34</td>
<td>181</td>
<td>119.00</td>
<td>19.79</td>
</tr>
<tr>
<td>50</td>
<td>22</td>
<td>3101</td>
<td>258.20</td>
<td>79.58</td>
</tr>
<tr>
<td>100</td>
<td>26</td>
<td>3996</td>
<td>653.55</td>
<td>392.56</td>
</tr>
<tr>
<td>150</td>
<td>19</td>
<td>4377</td>
<td>884.60</td>
<td>216.16</td>
</tr>
<tr>
<td>200</td>
<td>21</td>
<td>4459</td>
<td>1202.76</td>
<td>280.51</td>
</tr>
<tr>
<td>250</td>
<td>22</td>
<td>5127</td>
<td>1459.22</td>
<td>2256.82</td>
</tr>
<tr>
<td>500</td>
<td>39</td>
<td>9293</td>
<td>4187.78</td>
<td>1036.62</td>
</tr>
<tr>
<td>600</td>
<td>23</td>
<td>44746</td>
<td>12541.11</td>
<td>7414.54</td>
</tr>
</tbody>
</table>

Fig. 4 presents the average values in milliseconds to the latency of the HTTP GET requests for different number of retrieved datapoints and requests per second. It is possible to note that when the difference between message payloads is about 27%, there are more outliers (i.e., a reduction of the latency time when an increase was expected), so that the latency times tend to be equivalent. However, when such a difference is about 60%, no outliers occur, thus leading to the conclusion that there is a slight increase in the latency time when increasing the message payload.

Finally, Table II presents the minimum, maximum, average, and standard deviation values in milliseconds for the latency of the HTTP PUT requests for the analyzed configurations. It is possible to notice that EcoDiF replies up to 250 requests per second with latency smaller than 1 second, whereas such time increases to more than 9 seconds for 500 request per second and to more than 23 seconds for 600 requests per second. It was possible to observe atypical behaviors for the configurations with 50 and 100 requests per second, also possibly due to issues related to the internal processing of the JBoss application server.

---

1 We have considered a maximum of 600 requests per second because EcoDiF has presented an unstable behavior for a larger number of requests, thus not being able to answer all of the submitted requests until stop working.
Although the platform is operating in a high workload, second do not negatively affect the end times reached by workloads greater than 500 requests per second. It is also important to highlight that EcoDiF maintained an average latency time in the order of 1 second even for rates near to 250 requests per second when the system does not provide any response. Therefore, we can conclude that the times reached by workloads greater than 500 requests per second do not negatively affect the end-user experience although the platform is operating in a high workload.

Finally, through the performed experiments, we have identified some adjustments in order to increase the scalability of EcoDiF and improve the end-user experience. As we have mentioned, the platform was entirely hosted in a single server. Therefore, load-balancing techniques through clustering to be applied to the JBoss application server seem to be promising for the scalability of the platform and improvements on the response times. Another possible alternative would be the physical separation of the database in different servers, as well as the replication of its contents in more computational nodes, thus distributing the loads of data input and output operations.

V. RELATED WORK

In this section, we briefly present some infrastructures for IoT proposed in the literature with respect to the integration of heterogeneous devices and composition of applications that take advantage of such an integration. It is possible to notice that most of them have not achieved a mature development state, thus revealing gaps that require additional research effort in the area.

The OpenIoT Project [15] proposes an open-source middleware platform for IoT applications in a Sensing as a Service model available at a cloud environment that can be transparently accessed and configured by the users. This middleware connects sensors to the cloud environment, so that cloud resources can be used for data processing and management, which represent useful functionalities that usually cannot be directly performed by the resource-constrained devices. In turn, the LinkSmart Project (formerly Hydra) [30] proposes a service-oriented middleware for embedded systems in order to support the development of applications composed of heterogeneous physical devices. This platform provides interfaces based on Web services for controlling any type of physical device and enables developers to incorporate physical devices into their applications.

Xively [31] (formerly Pachube and Cosm) is a cloud-based IoT platform for managing data provided by devices. It provides a RESTful API for sending data from the sensors, enables to visualize historical data, and provides mechanisms to trigger events based on the acquired sensing data. As in EcoDiF, feeds represent environmental data with their respective datastreams, which represent data provided by the sensors, whereas datapoints represent a specific value of a datastream in a given time instant. In addition, data available at Xively can be retrieved in JSON, XML (more specifically EEML), and CSV data formats.

RestThing [32] is a REST-based Web infrastructure designed to abstract away the heterogeneity among physical devices and to provide means of integrating devices embedded in Web applications. RestThing enables developers to create REST-based applications by combining physical and Web resources so that devices and information are both represented as resources and manipulated through a uniform RESTful interface via the conventional operations defined in the HTTP protocol.

Gao et al. [14] present another REST-based platform for integrating sensors and sharing data provided by them. These data are sent to a server called WoT Enabler, which is im-

### Table II. Minimum, Maximum, Average, and Standard Deviation for the Latency of the HTTP PUT Requests Under Different Numbers of Requests Per Second.

<table>
<thead>
<tr>
<th>Requests per second</th>
<th>Min (ms)</th>
<th>Max (ms)</th>
<th>Avg (ms)</th>
<th>SDev (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>42</td>
<td>231</td>
<td>65.74</td>
<td>25.01</td>
</tr>
<tr>
<td>5</td>
<td>44</td>
<td>3110</td>
<td>100.95</td>
<td>232.81</td>
</tr>
<tr>
<td>10</td>
<td>78</td>
<td>291</td>
<td>98.40</td>
<td>81.69</td>
</tr>
<tr>
<td>15</td>
<td>79</td>
<td>210</td>
<td>107.70</td>
<td>33.66</td>
</tr>
<tr>
<td>20</td>
<td>81</td>
<td>213</td>
<td>122.70</td>
<td>27.05</td>
</tr>
<tr>
<td>25</td>
<td>73</td>
<td>517</td>
<td>171.66</td>
<td>131.99</td>
</tr>
<tr>
<td>50</td>
<td>78</td>
<td>1652</td>
<td>87.42</td>
<td>453.97</td>
</tr>
<tr>
<td>100</td>
<td>48</td>
<td>3180</td>
<td>137.00</td>
<td>65.76</td>
</tr>
<tr>
<td>150</td>
<td>2</td>
<td>3176</td>
<td>273.10</td>
<td>162.80</td>
</tr>
<tr>
<td>200</td>
<td>45</td>
<td>3499</td>
<td>487.54</td>
<td>309.63</td>
</tr>
<tr>
<td>250</td>
<td>48</td>
<td>11574</td>
<td>945</td>
<td>990.12</td>
</tr>
<tr>
<td>500</td>
<td>49</td>
<td>58534</td>
<td>9178.5</td>
<td>11284.57</td>
</tr>
<tr>
<td>600</td>
<td>67</td>
<td>66713</td>
<td>23136</td>
<td>21049.46</td>
</tr>
</tbody>
</table>
implemented by using the Ruby on Rails technology. This server communicates with a database that stores data sent by the sensors (in the case of devices with direct Internet connection) or by gateways (in the case of devices without such a capacity). Therefore, these data can be accessed by applications and/or users through RESTful requests to the WoT Enabler server, which can return representations in the XML, JSON, and CSV data formats. As in Xively and EcoDiF, the proposed architecture also uses EEML for structuring data provided by sensors, but it is different in terms of the used data hierarchy: (i) a system represents a data collection regarding a given environment; (ii) a sensor represents an individual sensor in the context of a system, and; (iii) data are <key, value> pairs regarding measures gathered by a sensor in a given time instant.

Finally, S'OIA [33] is a service-oriented architecture for integrating several types of objects/devices in the IoT context by using a tuplespace approach [34] to semantically express information about the devices integrated by the platform and to enable the representation of each one. S'OIA has a component for discovering other devices in the environment and it provides mechanisms for integrating them to the platform. Moreover, users can interact with the platform by expressing their preferences and other relevant information through an intention mechanism. Conversely, the platform is able to provide feedback to users.

All of the aforementioned proposals have several features in common with EcoDiF regarding the integration and interoperability among heterogeneous physical devices. The EcoDiF architecture was designed for working both on a cloud environment (as in the case of the Sensing as a Service model of the OpenIoT Project) as on servers placed in local networks, thus enabling the platform to make use of cloud services as well as be used in private monitoring environments. In turn, service-based interfaces for embedded systems (as in the LinkSmart Project) are implemented in EcoDiF through the integration drivers, which can be easily developed and made available by device manufacturers. Among the presented proposals, the platform that has more features in common with EcoDiF is Xively because they use the same hierarchical concepts (feeds, datasets and datapoints) and they are based on a RESTful API for communication, thus following the best industrial and academic REST practices [35]. However, the main feature that distinguishes EcoDiF from the platforms presented in this section is related to the functionalities for developing and executing applications. In EcoDiF, applications are characterized as data mashups (technology that has shown to be suitable for the IoT paradigm [11]) from several devices and that can be directly manipulated through the platform according to the need of the application developer. For example, a developer can build a mashup and then aggregate data provided by temperature sensors, calculate averages, perform other complex calculations, etc. Only EcoDiF enables users to build applications by using such a technology, thus representing a first initiative for addressing the complexity associated to the development of loosely coupled applications in IoT.

VI. Final Remarks

In this paper, we have presented EcoDiF, a Web-based IoT middleware platform that aims to integrate heterogeneous physical devices and to provide real-time data control, visualization, processing, and storage functionalities in several application domains. The EcoDiF architecture and implementation are based on Web technologies in order to standardize and simplify the development of applications in the IoT context, thus minimizing compatibility and interoperability issues. The current prototype of EcoDiF was preliminary evaluated under qualitative and quantitative perspectives, thus illustrating its potential of use and enabling us to analyze its performance.

From the qualitative perspective, it was possible to observe the clear potential of EcoDiF for managing several types of devices, thus enabling the easy visualization and control of data provided by them and making our platform a very useful tool for monitoring activities. Moreover, due to the device integration capabilities provided by EcoDiF, it is possible to create applications built upon the integration of data provided by several heterogeneous devices and other Web resources. In this perspective, we can conclude that the range of applications and real contexts that can benefit from the functionalities provided by EcoDiF is very broad, thus potentially bringing contributions to applied research, industry, and society. For instance, EcoDiF can allow the combination of remote sensing for monitoring purposes with alert and action mechanisms in order to notify authorities and citizens about imminent disasters for taking the appropriate actions to avoid such an event.

In turn, the obtained quantitative results have shown that EcoDiF is also able to deal with a high number of simultaneous requests without major problems with respect to the time spent to answer them, thus making it possible to use it in real-world IoT scenarios that require this type of configuration with several users, devices, and applications. Through the performed experiments, we have also identified some limitations related to the scalability of the platform, some of them solved through additional configurations in the JBoss application server on which EcoDiF was deployed. However, other limitations still remain to be further solved through the use of load-balancing and clustering techniques to be applied to the server, as well as the physical separation of the database in different servers and the replication of its contents in more computational nodes in order to distribute the loads of data input and output operations.

Furthermore, the current prototype of EcoDiF have a limitation regarding the Applications Module, through which users must codify (by hand) the programming logic of the EMML scripts regarding the applications. Therefore, it is necessary to provide a development environment in EcoDiF in order to assist application developers in the tasks related to the building of mashup applications in EMML. Nevertheless, this limitation can be minimized since the adoption of the EMML programming model by EcoDiF allows using external tools (such as Presto [36]) that can be used as developed environments. It is also necessary to address issues related to the complexity of the applications and the programming
power of EMML, i.e., to assess if the EMML language and its elements are sufficient to develop the several types of mashup applications to be executed in EcoDiF.

Finally, we have noticed that CoAP [37] has emerged as a promising protocol in the IoT context. Due to its features that make it suitable for use with constrained capacity devices (such as the use of the UDP protocol and less complexity when compared to the standard HTTP application protocol), we envision providing support to such a protocol in future versions of the EcoDiF platform.

ACKNOWLEDGMENT

This work was partially supported by Brazilian funding agencies FAPERJ (under grant E-26/102.961/2012 for Flavio C. Delicato), CNPq, CAPES (under grant 11097/2013-2 for Everton Cavalcante), and RNP (Brazilian National Academic and Research Network).

REFERENCES