Computer Science • 13 (3) 2012 http://dx.doi.org/10.7494/csci.2012.13.3.139

Michał Nieć
Paweł Pikuła
Aleksander Mamla
Wojciech Turek

ERLANG-BASED SENSOR NETWORK MANAGEMENT FOR HETEROGENEOUS DEVICES

Abstract *The paper describes a system designed to manage and collect data from the network of heterogeneous sensors. It was implemented using Erlang OTP and CouchDB for maximum fault tolerance, scalability and ease of deployment. It is resistant to poor network quality, shows high tolerance for software errors and power failures, operates on flexible data model. Additionally, it is available to users through an Web application, which shows just how easy it is to use the server HTTP API to communicate with it. The whole platform was implemented and tested on variety of devices like PC, Mac, ARM-based embedded devices and Android tablets.*

Keywords Erlang, mobile devices, sensor networks

1. Introduction

Recent years brought significant increase in computational power of mobile devices. It seems that such growth of cheap microcontrollers performance may create new principles of building large scale sensor networks.

Typically the responsibilities of geo-distributed devices, called sensors, are limited to simple, data collecting tasks [6]. A sensor network is usually a system with distributed data acquisition and centralized processing. One of the most significant problems arising in such systems concerns amounts of transferred data. Reduction of required throughput can be achieved by using on-demand data gathering. The TinyDB system [11], which is a well recognized solution, aims at creating a query processing system for selective gathering of information from a network of sensors. It provides the programmer with high level interface that hides logic responsible for connecting, collecting, filtering and merging it together.

Slightly different approach has been proposed in the SwissQM system [13]. It runs a virtual machine on each sensor. The query language responsible for obtaining data from sensors is translated to bytecode and then sent to nodes. Furthermore, SwissQM architecture enables push data model, which allows sensors to push data to server if readings change significantly.

These advanced solutions still perform all data processing and analysis in centralized manner. A different approach could be proposed if each sensor was equipped with a computational unit with significant performance. It could perform complex data processing and actively notify the system when important situation is detected. The basic function of a sensor would change: instead of providing raw data it could provide information. The sensor network would become a distributed data processing system.

Such approach can be utilized in many new areas, where centralized processing could fail. An active sensor with accelerometer could be installed in a car to detect damaged road surface or immediately notify about accidents. It could also be used for monitoring of vehicle fleet. Reports about cars violating traffic regulations or moving in unexpected areas could be generated.

The security industry could utilize such systems to create specialized networks of intelligent cameras able to recognize people or to detect movement in restricted areas. The user of such network would be provided with information about particular events instead of series of pictures from many different cameras.

In this type of geo-distributed system several important problems have to be addressed. Execution of complex algorithms rises the risk of runtime errors, therefore the software on the sensors should provide advanced failover mechanisms. Moreover, methods for updating software executed by the sensors should be provided. These requirements can be relatively easily met by utilizing Erlang language and technology [5].

1.1. Erlang technology

Erlang is a declarative language developed by Ericsson for programming concurrent and distributed systems. It aims at providing several unique features, required in such systems, like:

Fault Tolerance. Erlang has various primitives which have been proven to guarantee high fault tolerance level [7]. It also supports unified monitoring and automatic process failover after a crash. This features are critical for systems consisting of many devices communicating with each other. It is crucial that a failure of a single node does not cause whole system breakdown. Monitoring and testing of sensor networks are considered important issues of the domain [8].

Message-Based Concurrency. The sole data exchange method for Erlang processes is message passing. This is strongly supported by the language design and Erlang Virtual Machine. This solution makes a small number of Erlang code lines sufficient to implement communication between processes even if they are running on different machines. Such features are very helpful when developing distributed, concurrent applications.

Hot Code Swapping. The feature is most desired in constantly running systems, which may require software updates. Erlang allows updating the whole application or chosen modules without having to stop it.

1.2. Assumptions of the erlang-based monitoring system

The exemplary system described in this paper was designed to manage and collect data from large number of heterogeneous, active sensors. The conceptual diagram of the system is shown in Figure 1.

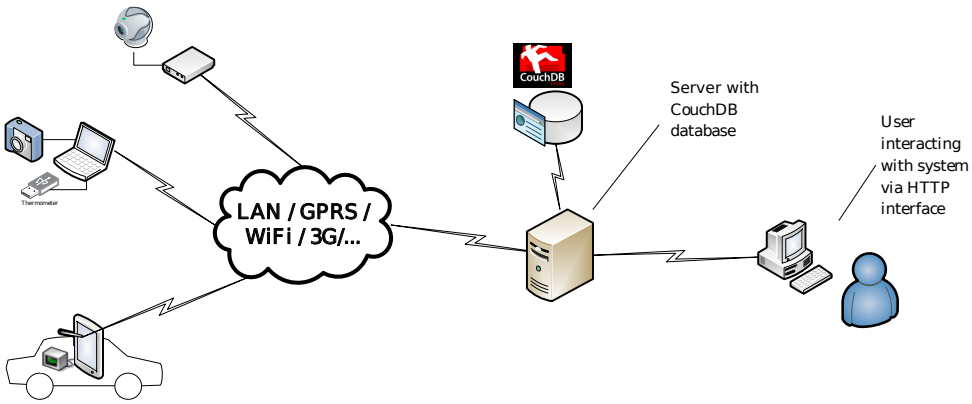

Figure 1. The conceptual diagram of the system. Variety of devices collecting different data are available in common infrastructure.

Basic assumptions, which were considered during the development, are:

Multiplatform. Building an application on top of the Erlang VM enables deploying modules to almost all devices running Unix-like operating systems (Linux, VxWorks, Solaris) or Windows. This feature empowers reusing existing systems as well as new powerful devices. For example, the same application could be deployed on regular old PCs, devices running Intel Atom or ARM processors.

Flexibility. The system aims to provide a flexible, easy to handle format of acquired and stored data. The sensors return data in form of a property list which can be effortlessly transformed to a JSON object, Erlang list or SQL record. This enables quick integration with other services and easy replacing modules responsible for sending and persisting data.

Resistance to poor network quality. Current solutions often assume that connection between sensor and the server is stable and available. The system described in this paper, implements mechanisms for caching data when the connection is unreliable or lost.

Expandability. One mobile device can run many processes responsible for gathering different data. The processes are written in Erlang programming language so that they are capable of implementing complicated logic, using operating system's resources and extending its functionality by natively compiled code extensions.

Availability. Thanks to modularized structure and using Erlang OTP components, the system should be resistant to software errors in sensors implementation and hardware failures. Restarting or turning the devices off (for example during power failures) is no longer a threat to the system.

Transparency. The whole system was designed to provide a single, easy to implement HTTP interface to manage sensors and browse data from all the sensors. Thanks to that, other systems and higher level logic can be integrated without changing the sensor network.

In the following section details of the system architecture are described. Next sections present implementation details followed by tests results and conclusions.

2. Architecture of the monitoring system

The Erlang-Based Monitoring System (called **CollectE**) is composed of three basic subsystems: graphical user interface (GUI), Server and Node with Plugins. As shown in Figure 2, each of them can be run on separate machine and should be considered a separate service with well defined interface.

- Server – is a single program responsible for communicating with Nodes, collecting data from them, monitoring their state and handling all external requests.
- Node – is an application run on each mobile device connected to the system. It manages and monitors Plugins, collects data from them, sends it back to server and also handles all requests sent by server.

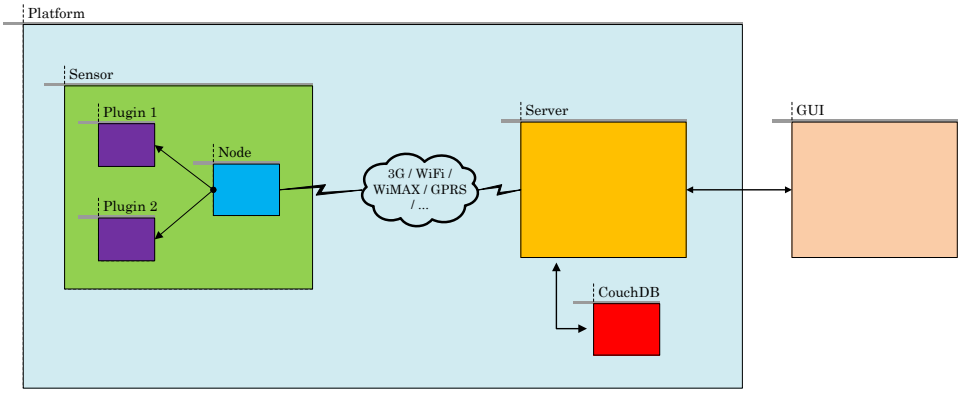

Figure 2. Main components of the system. The GUI is an application which communicates with whole platform via HTTP API provided by the server. The server maintains connection witch multiple Nodes and collects data from Plugins run by the Nodes.

- Plugin – is a simple Erlang module executed by the Node as a separate Erlang process. It collects data from sensors and classifies it. It is also responsible for transforming data to proper format.
- Graphical User Interface (GUI) – is a *Ruby On Rails* web-application that communicates with the server. It enables performing management tasks on the platform as well as examining collected data.
- Server with Nodes – creates a single platform that can be accessed via an interface provided by the server. The platform can be used as a part of a larger system.

2.1. The server

Server is a central part of the system. It is an application written in Erlang responsible of collecting and preserving data sent by the Nodes. It monitors the state of all connected Nodes. The server also delivers a special HTTP interface, allowing interaction via GUI.

The architecture of the server application is presented in Figure 3. Basic elements of the server are:

- NetLink – responsible for communication with Nodes
- DataManager – receives data sent by Nodes and stores it
- NodeManager – registers new Nodes or Plugins, monitors Nodes state and sends requests to Nodes
- DataAccessLayer – elements responsible of proper communication with database
- HTTPInterface – receives user requests sent by GUI and delivers them to NodeManager

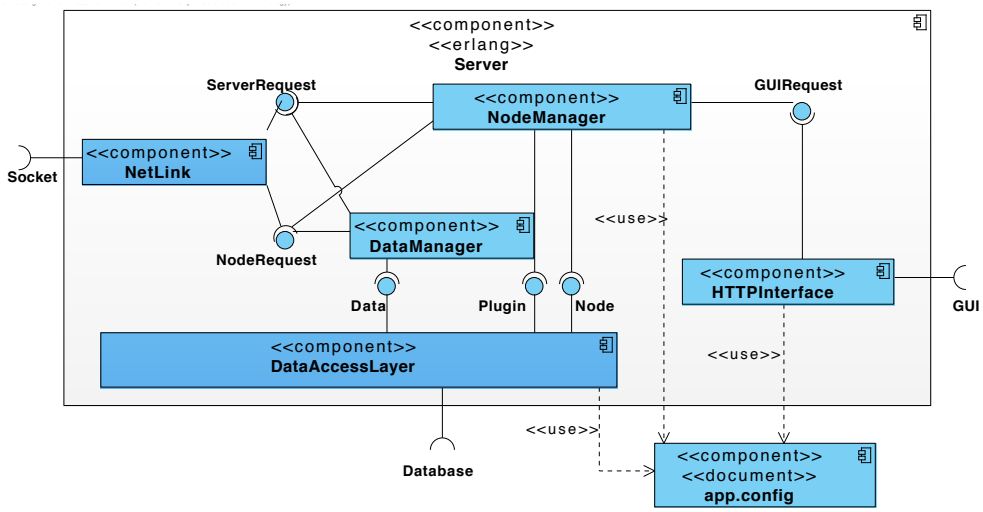

Figure 3. Server internal architecture. Three interfaces are provided: for GUI, for a database and for Nodes.

- app.config – server configuration file, configures settings such as CouchDB database address, database connection options or time interval of Node availability check.

Every new Node connected to the system needs to register itself at the server. It sends the information about itself and its Plugins to the server, where it is stored in a database. If a particular Node was previously registered and reconnects to the system, e.g. after re-establishing internet connection, it just updates its state at the server. This update takes place every time the state of the Node or any of its Plugins changes.

The server regularly monitors the state of all the registered Nodes. If the connection with any of the Nodes is lost, the information about that Node is updated with status *unavailable* and saved in a database.

To allow easy and universal interaction with GUI, server delivers a simple HTTP interface.

2.2. Node

A Node is an Erlang application that controls Plugins installed on a particular hardware device, collects data provided by them and delivers it to the server. It is composed of modules that run as separate Erlang processes. The design follows *OTP Design Principles*[10] therefore all processes are linked together in supervision tree as shown in figure 4. Such architecture protects the whole application from crashing down if there is an error in one of the child processes. This approach protects the Node application from failures of hardware devices or drivers as well as from errors in data

processing algorithms. It also makes it possible to execute many independent data gathering or processing algorithms.

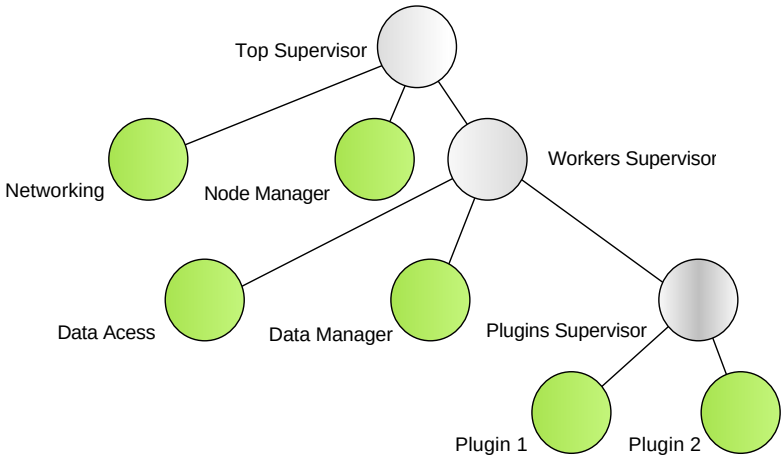

Figure 4. Node's supervisor structure. Each *Supervisor* starts all processes one level below itself. All Node component processes are bound in a single supervision tree.

When the Node application is launched, top level supervisor spawns networking, managing and worker supervisor processes. The following operations are performed:

- The networking process connects to the server and obtains a modified configuration for the Node.
- The managing module handles all requests from server and delegates them to appropriate workers.
- The worker supervisor initiates and controls Plugin manager, data manager, Plugins supervisor and a few other supporting processes.
- The Plugin manager is the most important module which controls life cycle of Plugins. It mediates between the Node manager, the Plugins and the data access layer. The Plugin manager stores information about all active Plugins in local database. Therefore if the device is unexpectedly rebooted, all the running Plugins are restarted with the configuration from before the shutdown.
- The data access layer process is responsible for controlling the connection to the database, marshalling data and all other operations on DB.
- The data manager handles the replication of data to the server from the local database.

The communication with the server is done via TCP socket. All requests and data are transmitted using this channel. Node tries to keep connection alive for the whole time. If the connection is closed, the Node restarts the networking module immediately. When the communication is not available, the data from Plugins and their state are stored in local *Mnesia* [12] or *CouchDB*[9] database (depending on configuration).

When the connection becomes available the *Data manager* synchronizes stored data with the server and cleans the cache.

2.3. Plugin

The Plugin is a single application controlling particular physical sensor or a group of them. It processes the collected data and delivers the output to the Node via well-known interface.

Each Plugin is responsible for processing data collected by controlled physical sensors. If the processing results in valuable information, it is sent to the Node and further to the Server. If the information is urgent, a special marker can be set for it. This kind of information is provided directly to the system user via the GUI.

The Plugin structure is described by an Erlang behaviour. In Erlang, a behaviour is a design pattern implemented in a module, which provides functionality similar to interfaces in other programming languages. They are essentially a required set of callback functions. For example in the CollectE system all of the Plugins must implement callbacks defined by beevree_gen_plugin behaviour such as: $init, set\_configuration, on\_resume, on\_suspend, get\_state, stop$.

Erlang OTP does not provide functions that give uniform access to devices connected via USB or serial port. CollectE Plugins use native applications (written in C/C++) that communicate with EVM using Erlang ports and *Erlang_Interface* library (see [5] for more details). *Erlang_Interface* provides a set of functions for decoding/encoding to Erlang data format.

The ports provide a basic mechanism for communication with the external programs. Erlang can communicate with a port by sending and receiving lists of bytes. This means that the custom encoding and decoding scheme can be used. The actual implementation of the port mechanism depends on the platform. In the Unix case, pipes are used and the external program should read from the standard input and write to the standard output. This means that any programming language can be used to develop a port application.

Erlang Ports have been chosen in order to meet the reliability requirement. An error in external program will not cause the entire Erlang runtime system to leak memory, hang or crash. Detection of hardware driver failure can be easily performed in the Plugin process, which can restart the driver or notify the Node application.

2.4. CouchDB database

The data collected by the server is stored in Apache CouchDB. It is a document oriented, schema-free database. CouchDB stores data in documents. Each document is an object identified by unique id, containing named fields (represented with use of JSON standard) and optional attachments.

Documents in CouchDB database are stored in a flat address space. That is why a new way of data filtering and reporting (the view model) was needed. Views are special functions, written is JavaScript and saved in special *design documents*. Each

view acts as the "map" part in a MapReduce system. By taking CouchDB document as an argument it determines whether the document should be available in view result. Views don't affect the stored documents, they are only the way to create dynamic representation of database content.

The CollectE system was designed to collect and store data from various types of sensors. Each sensor can deliver data in a specific format and because of that it would be almost impossible to design relational database without limiting system flexibility.

CouchDB features like semi-structured documents and views model were used by our system to retain intended flexibility and extensibility. The data collected by Plugins is formatted into JSON (when it is text-based) or saved as an attachment (when it is e.g image, sound or binary file etc.). Special views defined in database allow easy document filtering (e.g. obtaining data with high priority, latest data, results from specific Node or Plugin).

2.5. User interface

The graphical user interface allows user-friendly system management and data representation. It is a Ruby on Rails web application. GUI interacts with the server sending HTTP request to its HTTP interface. Most important features of the GUI are:

Displaying information about the sensors – the user can see a list of all registered sensors with brief information about their state. If sensors provide their GPS location they are shown on the map. For each sensor it is also possible to show its more detailed information including state of sensor's Plugins and data it has recently collected.

Data representation – the user can view the data collected by the system. The data is organized by its origin, that is by the id of the sensor and the Plugin which collected it. The data is displayed with the help of a tree-like structure. Any data that cannot be represented in text form (images, sound files etc.) is displayed in a form of an attachment link and can be downloaded by user. Figure 5 presents a screenshot of the result screen with sample data.

Sensors management – the user can manage the sensor Plugins: delete, suspend or restart work of single or multiple sensor's Plugins.

Notifications – GUI notifies the user about urgent data sent by the sensors.

Users management – GUI extends the system with a possibility to create, edit or delete the system user accounts. The accounts are arranged in user groups. Each user group can be created with specific permissions and access restrictions.

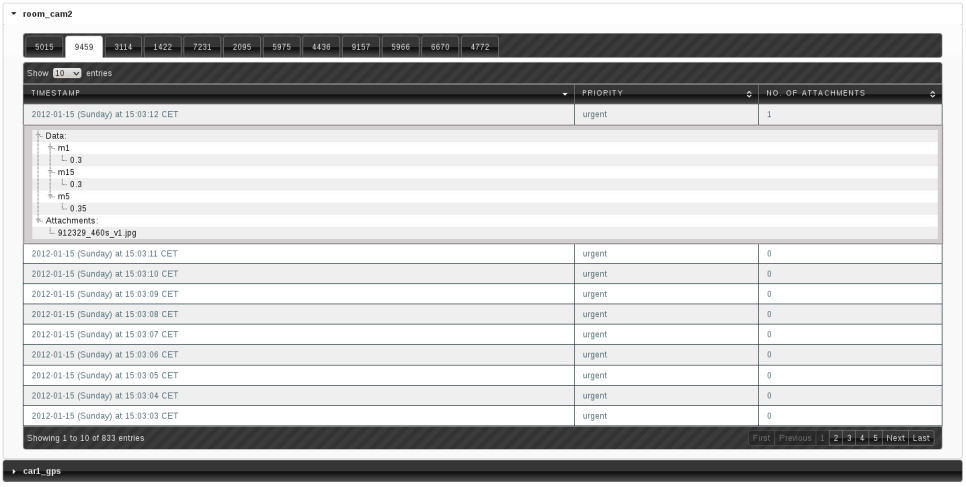

Figure 5. Graphical user interface, collected results screen. Data is organized in sections by Node id. Within Node section different Plugins can be selected.

3. Implementation and test results

3.1. Implementation

CollectE is built by rebar[4]. Rebar is an Erlang build tool that makes it easy to compile and test Erlang applications. It creates application skeleton that uses standard Erlang/OTP conventions for the project structure. Rebar also provides dependency management, that gives a possibility to easily re-use common libraries from variety of locations, like *git* or *hg*. Generating release with rebar is an easy task. It ends with creating a self-contained environment including Erlang VM, required OTP libraries and external dependencies, ready to deploy on the target machine.

In order to test the system, several Plugins have been developed:

- **LoadAvg** – reports current loadavg of the sensor CPU and sends urgent notifications when it is higher than defined threshold.
- **Temp** – gives information about air temperature and location of the device. It was tested on a device connected by USB which was visible as a serial port device. Plugin reads the most common sentence transmitted by GPS devices in the GPRMC format. The position is described as a float pair (*longitude*, *latitude*) compatible with the Google Maps service. The thermometer has no serial port interface, it was accessed with use of *libusb-1.0* library. Temperature data is represented as a float value(number of Celsius degrees). The Plugin sends urgent notification when read temperature exceeds given value.

- **Motion** – captures frames from a camera. If motion is detected, the Plugin sends a frame to the server. Frames are sent to the server as a sequence of *jpeg* compressed images. It uses v4l2 library (Video for Linux) for grabbing frames and *libjpeg* for decoding them before they are processed. In the end the frames are back compressed.

CollectE module responsible for communicating with USB devices could be easily extracted as a standalone library. Its port application uses libusb 1.0 [2] library which gives access to USB devices across many different operating systems. Serial module could also be extracted, but it is platform dependent (it uses Linux API).

3.2. Tests

The CollectE system was successfully tested on multiple PC running Linux (Arch, Fedora, OpenSUSE) on x86 and x64 architecture. It was also run on Mac OS X.

The system was also deployed on *BeagleBoard*[1] computer, which is low power OMAP3530 based platform. It has *600MHz ARM processor, 512 MB RAM, SD card slot and USB Ports*. BeagleBoards had Debian Squeeze for ARM architecture installed. External devices (webcam, thermometer, gps receiver) were tested on PCs and on BeagleBoards.

The tests focused mostly on trying to run multiple Plugins simultaneously on BeagleBord and observe how devices are handling in such scenario. All Plugins worked smoothly, without any problems, which is a significant achievement. It proves, that the Erlang technology is suitable for this kind of applications and that the architecture of the CollectE system is correct.

What is more CollectE was successfully run on Android tablet Eee Pad Transformer. Eee Pad Transformer is ARM based device with *NVIDIA Tegra 2 1.0 GHz dual-core CPU, 1 GB of RAM* and it is controlled by *Android 3.2 Honeycomb*. To fully use Android tablet devices we decided to run Debian distribution in android system using *chroot* technique. This enabled easy deployment of Erlang on this tablet and also get other dependencies required to compile and run CollectE Node and Plugins.

Additionally CouchDB replication system was tested by *netem* tool [3] to simulate possible network problems which can be encountered by moving sensor device. Netem emulates properties of wide area networks such as variable delay, loss, duplication and reordering. CouchDB handled poor connection scenarios without any problems. All data, collected by a sensor, eventually reached the server.

4. Conclusions and further work

The implemented system links all the sensors with the server and provides a convenient API to manage them. Plugins executed on sensors can perform complex data processing algorithms and provide information to the server, without sending large amounts of data. Moreover, important information can be urgently presented to the system user. These features can help solving a variety of problems.

With the Erlang OTP it was possible to achieve flexibility and durability of the system. Software executed on sensors is resistant to hardware failures, which is the main advantage of the system. Geo-distributed processing of data increases scalability and reduces requirements concerning central server.

There are several directions of further development of the system. For example, mechanisms for updating Plugins should be provided. This task is relatively simple using Erlang hot code swapping mechanisms.

Current system is not secure enough. The transferred data is not encrypted and the platform does not authenticate users, so using it in a limited trust environment is not an option. Erlang OTP offers cryptographic mechanisms, which could be used in the future to enable such capabilities.

The tests showed that CouchDB on Node is not a safe option. It consumed a lot of resources during its work (especially the disk space). In the future the Node could take advantage of Mnesia term storage to provide data caching.

Acknowledgements

The research leading to this results has received founding from the AGH grant no. 15.11.120.075.

References

[1] Beagleboard official website. `http://beagleboard.org/`, July 2012.

[2] libusb – official website. `http://www.libusb.org/`, July 2012.

[3] Network emulation functionality for testing protocols. `http://www.linuxfoundation.org/collaborate/workgroups/networking/netem`, July 2012.

[4] Rebar: Erlang build tool. `https://github.com/basho/rebar/wiki`, July 2012.

[5] Armstrong J: *Programming Erlang: Software for a Concurrent World.* Pragmatic Bookshelf, 2007.

[6] Dargie W., Poellabauer C: *Fundamentals of wireless sensor networks: theory and practice.* John Wiley and Sons, 2010.

[7] Earle C., Fredlund L., Derrick J: Verifying fault-tolerant erlang programs. In *ERLANG '05 Proc. of the 2005 ACM SIGPLAN workshop on Erlang*, 2005.

[8] Kapłoniak R., Kwiatkowski L., Szydło T: *Environment emulation for wsn testbed. Computer Science*, 13(3), 2012.

[9] Lennon J: *Beginning CouchDB.* Apress, 2009.

[10] Logan M., Merritt E., Merritt R: *Erlang and OTP in Action.* Manning Publications, 2010.

[11] Madden S., Franklin M., Hellerstein J., Hong. W: *Tinydb: an acquisitional query processing system for sensor networks. Transactions on Database Systems (TODS) – Special Issue: SIGMOD/PODS*, 2003.

[12] Mattsson H., Nilsson H., Wikstrom C: *Mnesia – a distributed robust dbms for telecommunications applications.* In *Proc. of the First International Workshop on Practical Aspects of Declarative Languages, London, UK*, pp. 152–163, 1998.

[13] Mueller R., Alonso G., Kossmann D: *Swissqm: Next generation data processing in sensor networks.* In *Proc. of Third Biennial Conference on Innovative Data Systems Research, Asilomar, CA, USA*, 2007.

Affiliations

Michał Nieć
Erlang Solutions, Kraków, Poland, michalniec@gmail.com

Paweł Pikuła
AGH University of Science and Technology, Krakow, Poland, ppikula@gmail.com

Aleksander Mamla
AGH University of Science and Technology, Krakow, Poland, alek.mamla@gmail.com

Wojciech Turek
AGH University of Science and Technology, Krakow, Poland, wojciech.turek@agh.edu.pl

Received: 28.03.2012
Revised: 19.06.2012
Accepted: 9.07.2012