A programming model based on Interaction Nets

Joaquín F Sánchez
National University of Colombia.
Bogota.
Colombia
jofsancherzci@unal.edu.co

Jorge A Quiñones
National University of Colombia.
Bogota.
Colombia
jaquinonesg@unal.edu.co

Juan M Corredor
National University of Colombia.
Bogota.
Colombia
jumcorredorro@unal.edu.co

Abstract

In this article, we present the description of a programming language model based on Interaction Nets. With this base model, an explanation of its pertinence is made to model a programming language that helps the construction of decentralised systems (For the article, ad hoc networks are considered). Four interactions are presented, where the flexibility of the language is shown using native libraries and functions. The paradigm that is proposed is a multi-paradigm that combines the use of functions, software agents, and the use of libraries.

Keywords
Functional programming, ad hoc network, Network, Net Interactions

1. Introduction

The programming languages are tools developed along the computation history to interpret the reality of a situation toward a computing device and, in this way, to give solutions to problems and conditions that require the execution of quick and reliable operations (Grune, Van Reeuwijk, Bal, Jacobs, & Langendoen, 2012), (Sheng et al., 2014), (Fister Jr., Mernik, Fister, & Hrnčič, 2011), (Aho, 2003). Furthermore, with the passing of years, the networking computer systems have evolved. First, there were centralized systems, which have a robust and defined architecture. In the '90s, fixed telephony lost ground against mobile telephony, and the data transmission component over these networks was added (Brody & Pureswaran, 2014), (Ortiz, Hussein, Park, Han, & Crespi, 2014). In the 2000s, the evolution of the fixed network systems passed toward wireless and mobiles networks, where apps have been transforming the way of not only doing computation, but also of connecting devices and people (Nitti, Girau, & Atzori, 2014). Following this trend, where the computing devices are mobiles with significant processing features and memory, the current processing clouds are going to evolve to become mobile clouds with stochastic, dynamic, and distributed elements (Fitzek & Katz, 2013), (Loo, Mauri, & Ortiz, 2016).

In this way, a new paradigm in computing systems is coming, and the challenges to overcoming the system design and conception must generate new ideas and viewpoints to approach to the solution of the problems they face. The purpose of these will be to address the design challenges of a decentralized system of devices belonging to mobile clouds, and it is necessary to think about a tool through which the physical and logical setting of the system components are possible.

The proposed tool is a programming language that incorporates the fundamentals of the operation of a decentralized system and, in consequence, contains native functions that allow the adaptation of the components to provide required services to users. If attention is focused on wireless networks, specifically on an ad-hoc network, some challenges to guarantee the adequate network functioning need to be solved. In nodes management, aspects of transmission power control, frequency spectrum, the right use of the battery, the memory resource, and the processing capability should be managed, and this requires a suitable programming language that considers the dynamics of the operations of these factors (Loo et al., 2016), (Palos, Kiviniemi, & Kuusisto, 2014). When studying the dynamics of decentralized systems whose interactions define the behavior of the system (Zhang, Long, & Wang, 2013), (Prehofer & Bettstetter, 2005), the adaptation as a consequence of self-organization concept might be the way to design a programming language. The idea is to face the challenges of the startup of the required functions to make the system works correctly, and its functioning might depend on the services offered to the system users (Gershenson, 2007).
The contribution of this article is the presentation of a programming language prototype used in the construction of ad hoc networks, using the Interaction Nets as a form of inspiration for the design.
In section base model: Interaction Nets, the interaction nets model is introduced, which is the basis of section Programming model, which describes four interactions related to an ad hoc network. In section real work an interaction realisation with a prototype programming language is presented. Finally, the conclusions are given.

2. Base Model: Interaction Nets

The convenience of this computational model to represent local interactions in decentralised data networks is a function of the participating nodes of these networks. A node can have a number n of communication interfaces depending on the application for which it is intended, and each communication it makes with another node or other nodes is an action that generates a result (Fernández & Mackie, 1999), (Perrinel, 2014). For instance, a transmission of a packet burst, from node A to node B. The configuration is an IP address and a port number. Node A generates interaction with node B when the transaction has done. The net's interaction model is based on the idea of computing as an interaction. Yves Lafont in 1990 proposed this model (Fernández & Mackie, 1999), (Pinto, 2000).
The description of the system is as follows:

- A set Σ of symbols used to build networks.
- An set R of rules called interaction rules.

Network: A network N on Σ is a graph where the symbols of Σ give the labels of the nodes. A node is called an agent, and a link between two agents is called a connector. So networks are graphs that connect agents through connectors.
Active pairs: The active pairs are equal to a pair of agents (A, B) that belong to a set of symbols. An interaction rule is composed of an active pair on the left side and a network on the right side. The rules must meet two conditions:

- Both left and right side must have the same interfaces.
- In a rule set R, there is a maximum of one rule for each pair of agents.

The interaction nets model is similar to other formal models for the definition of programming languages, such as the lambda calculus for functional languages or the ***pi*** calculation for parallel processes. In this article, we show how we obtain the components of a programming paradigm oriented to decentralised networks by using process algebra tools. However, other aspects of decentralised networks must be considered in order to complete the paradigm.
The model can use several notations:

- $A \bowtie B \rightarrow N$: Active pair A and B produce an interaction in a network N.
- $foo(A, B)$: A function foo that acts on A and B creates an interaction.
- Graphic notation

The following describes four essential aspects to take into account for this type of network:
Self-configuration: in this context, configuration refers to the way the network is set up, and not only the configuration of an individual device. Nodes and applications can configure and reconfigure themselves automatically under any, predictable or unpredictable, condition with minimal or no human intervention. Self-configuration expects to reduce the effects of networks dynamics to users (Dressler, 2006), (Dressler, 2008), (Prehofer & Bettstetter, 2005). ***The language achieves this by building functions to monitor the state of the network nodes. The functions handle agents for these actions***.
Self-deployment: preparation, installation, authentication and verification of every new network node. It includes all procedures to bring a new node or applications into operation. Besides, self-deployment try to find strategies to improve both coverage and resource management in networking tasks (Dressler, 2006), (Prehofer & Bettstetter, 2005), (Dressler, 2008). ***The language has native functions that manipulate routing actions and management information.***

Self-optimization: utilisation of measurements and performance indicators to optimise the local parameters according to global objectives. It is a process in which the configuration of the network is adapted autonomously and continues the network environment regarding topology, resources, and users (Dressler, 2006), (Prehofer & Bettstetter, 2005), (Dressler, 2008). ***The language is a tool to build agents that measure the state of the network.***
Self-healing: Execution of routines that keep the network in the steady state and prevent possible problems from arising. These methods can modify configuration and operational parameters of the overall system to compensate failures (Dressler, 2006), (Prehofer & Bettstetter, 2005), (Dressler, 2008). ***The language creates verification scripts of the components of the network to perform healing processes***.

3. Programming model: language interactions

Based on the definition of Interaction Net and considering the nature of decentralised networks, the decision was taken to model interactions between agents in order to solve different challenges arising from the construction of the programming language. Since we want the scheme to generate local interactions, the designed interactions are:

- An agent sending a message to another agent, and being confirmed.
- An agent sending a ping message to another agent or a system element.
- Taking into account the communication specification between the FIPA agents, the proxy interaction is: a transmitter wants the receiver to make a selection of other agents to send a message or a service.
- Propagate: A transmitter wants the receiver to select other agents to send a message or service.

Below are the models:

3.1 An agent sending a message to another agent, and being confirmed.

This action defines two agents: A and B. Other agents are also used to complement the interaction. Figure 1 depicts this interplay.

- Agent A: send the message.
- Agent B: receive the message and send the confirmation.
- Mess is an agent with the action of the message.
- Ack is an agent with the confirmation action.

The set $\Sigma\{A, B, mess, ack\}$
The rules of interaction are:

1. $A \bowtie B \rightarrow mess$: sent a message.
2. $A \bowtie B \rightarrow ack$: an affirmative confirmation is received.
3. $A \bowtie B \rightarrow \sim(ack)$: received a negative confirmation.
4. $A \bowtie 0 \rightarrow \sim$: sent a message without a receiver.

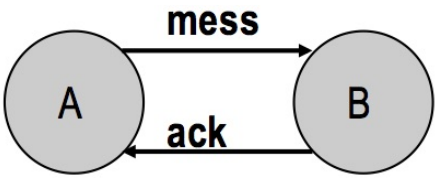

Figure 1: The interaction between agent A and agent B. A message is sent, and a confirmation is received.

3.2 An agent sending a ping message to another agent or a system element.

This action occurs between two agents: a transmitting agent which sends a ping to a receiving agent or a system device. Figure 2 represents interactions between them.

- Agent A: send the ping.
- Agent B: receive the ping and send the confirmation.
- $ping$: is an agent with the action of ping.
- ack is an agent with the confirmation action.
- X: is another element of the system.

The set $\Sigma\{A, B, ping, ack, X\}$

The rules of interaction are:

1. $A \bowtie B \rightarrow ping$: sent a ping
2. $A \bowtie B \rightarrow ack$: an affirmative confirmation is received
3. $A \bowtie B \rightarrow \sim(ack)$: received a negative confirmation
4. $A \bowtie X \rightarrow ping$: sent a ping another element of the system
5. $A \bowtie X \rightarrow ack$: an affirmative confirmation is received
6. $A \bowtie X \rightarrow \sim(ack)$: received a negative confirmation
7. $A \bowtie 0 \rightarrow \sim$: sent a ping without a receiver

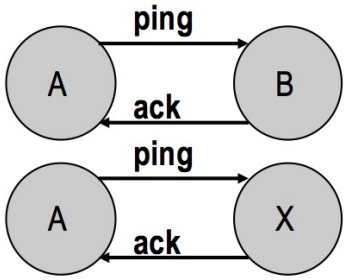

Figure 2: The interaction between agent A and agent B. A ping is sent, and a confirmation is received. It shows the possibility that A sends a ping to another device in the system, X is used to identify the other device.

3.3 Propagate

This interaction consists of Agent A sending a message to Agent B that aims to send the information to other agents in the system. The message is from an agent with the action of the message to n agents and represents the n agents of the system. Figure 3 shows that interaction.

- Agent A: send the message.
- Agent B: receive the message and send the confirmation.
- $mess$ is an agent with the action of the message.
- $mess_n$ is an agent with the action of the message to n agents
- $\bigotimes$ represents the n agents of the system.
- ack is an agent with the confirmation action.

The set $\Sigma = \{A, B, mess, mess_n, \bigotimes, ack\}$

The rules of interaction are:

1. $A \bowtie B \rightarrow mess$: sent a message
2. $A \bowtie B \rightarrow ack$: an affirmative confirmation is received
3. $A \bowtie B \rightarrow \sim(ack)$: received a negative confirmation
4. $A \bowtie 0 \rightarrow \sim$: sent a message without a receiver
5. $A \bowtie B \rightarrow mess_n \bowtie \bigotimes$: agent A sends a message to agent B and sends it back to the n agents of the system
6. $mess_n \bowtie \bigotimes \rightarrow ack$: there is a positive confirmation of the receipt of the message from the n agents of the system
7. $mess_n \bowtie \bigotimes \rightarrow \sim(ack)$: there is a negative confirmation of the receipt of the message from the n agents of the system
8. $mess_n \bowtie 0 \rightarrow \sim$: there are no agents in the system to send the message
9. $mess_n \bowtie B \rightarrow mess_n \bigotimes$: is the action of agent B of resends the message to the n agents of the system

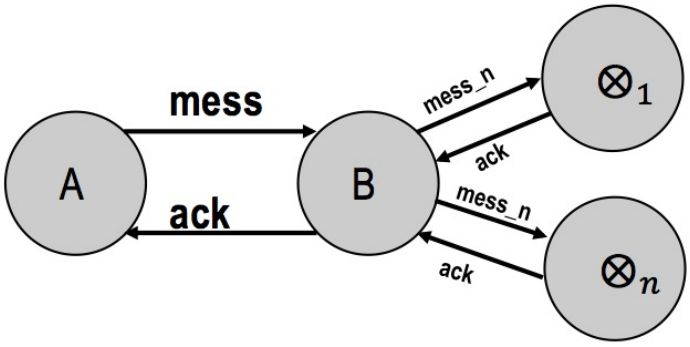

Figure 3: The base interaction is to send a message from A to B, but B has the option of propagating this message to the other agents of the system. Each agent of the system uses the interaction. The rules are similar for each interaction, and the purpose is to send the message that B received from A.

3.4 Proxy interaction

This interaction consists of Agent A sending a message to Agent B whose purpose is to send the information to other agents of the system who are authorised to receive the information. Figure 4 indicates that interaction.

- Agent A: send the message.
- Agent B: receive the message and send the confirmation.
- $mess$ is an agent with the action of the message.
- $mess_n$ is an agent with the action of the message to n agents.
- $mess_k$ is an agent with the action of the message to k authorized agents
- $\otimes$ represents the n agents of the system.
- ack is an agent with the confirmation action.

The set $\Sigma = \{a, B, mess, mess_n, mess_k, \otimes, ack\}$

The rules of interaction are:

1. $A \bowtie B \rightarrow mess$: sent a message
2. $A \bowtie B \rightarrow ack$: an affirmative confirmation is received
3. $A \bowtie B \rightarrow \sim(ack)$: received a negative confirmation
4. $A \bowtie 0 \rightarrow \sim$: sent a message without a receiver
5. $A \bowtie B \rightarrow mess_n \bowtie \otimes$: agent A sends a message to agent B and sends it back to the k agents of the system
6. $mess_k \bowtie \otimes \rightarrow ack$: there is a positive confirmation of the receipt of the message from the n agents of the system
7. $mess_k \bowtie \otimes \rightarrow \sim(ack)$: there is a negative confirmation of the receipt of the message from the k authorized agents of the system
8. $mess_k \bowtie 0 \rightarrow \sim$: there are no agents in the system to send the message
9. $mess_k \bowtie B \rightarrow mess_k \bowtie \otimes$: is the action of agent B of resends the message to the k authorized agents of the system.

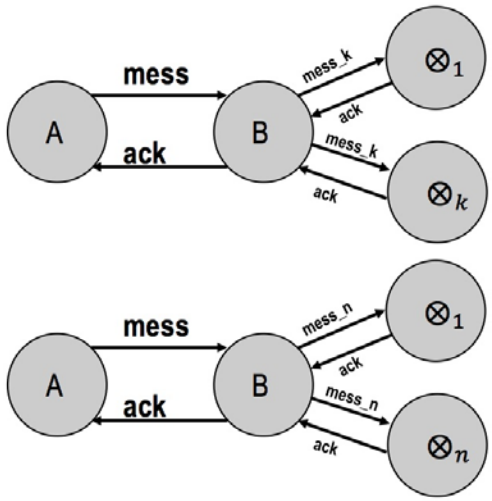

Figure 4: This interaction is similar to the propagation interaction but is modified depending on the target agents. Only the agents k, $k \in n$ receive the message of A. Which is classified by B.

4. Real work

The preliminary tests allow the establishment of an ad hoc network on embedded devices (Raspberry Pi). The network turns on, and the environment is switched on for communication between agents. Next, the following tests are carried out:

- Establish the ad hoc network.
- A pinging agent.
- An agent that sends information from sensors to a web service.
- Share the Internet from one node to the other nodes of the network.

4.1 Establish the ad hoc network

This function enables the option to establish the network quickly with the desired parameters for its operation and little physical manipulation of the nodes. The code that is used to make the power on the network, the environment, and the communication between agents are the following:

```
importar network.adhoc #import library
do_adhoc() # function call system
importar agent.environment #import library
a=Environment() # initialize environment
a=Environment(12345) # socket set up
```

The description of the interaction is:

1. $do\_ad\boxtimes oc(A, B) \rightarrow Net_{ad\boxtimes oc}$
2. $Enviroment(A, B) \rightarrow Enviroment(Socket)$

This interaction is simple since it handles four agents and it is represented in Figure 5. Agent A and agent B are the devices that help to turn on the network. The $do\_ad\boxtimes oc$ agent or function is responsible for enabling the requirements to be turned on by the ad hoc network. The environment agent creates the environment in the network and receives, as a parameter, the socket number for the communication between the agents.

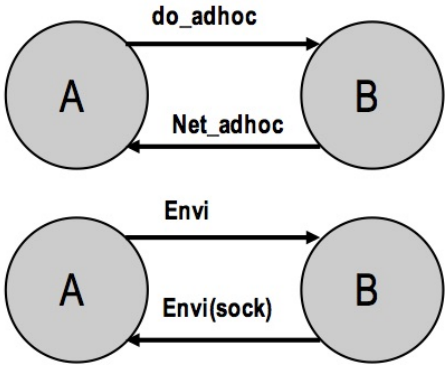

Figure 5: The interaction is divided in two, one happens when the ad hoc network is turned on, and the other is when the environment (*Envi*) is turned on for communication between agents. In the environment (*Envi*), it receives a socket (*sock*) parameter to receive the information.

4.2 A pinging agent

This function works as a tool to know if other agents or other devices are on in the network. The code used to create this tool is:

```
importar mas.__init__ #import library SMA

funcion ping(dir) #creation ping function
   x = PingAgent(dir)
   x.start()
end

log("test ping")
a = ping(192.168.2.1) #Execution of the function
```

This interaction is the realisation of the model described in section 3.2. The creation of a PingAgent is accepted as an IP address parameter. The internal function of this agent is to create the ping message and send it to the address as the parameter. Figure 6 represents the following interaction:

1. $ping(A, B) \rightarrow ack$
2. $ping(A, B) \rightarrow \sim(ack)$
3. $ping(A, X) \rightarrow (ack)$
4. $ping(A, X) \rightarrow \sim(ack)$

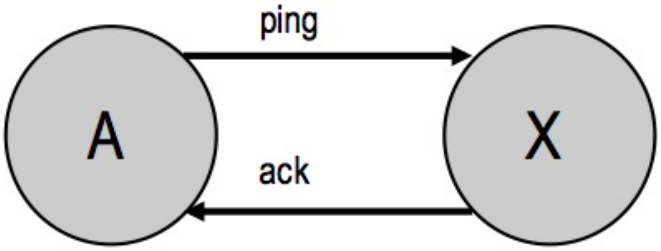

Figure 6: Interaction is a process of sending and receiving a ping message, from agent A to an element of the system. The response of the interaction is positive.

4.3 An agent that sends information from sensors to a Web service

This application is more elaborate than the previous ones. The data given by the sensors are humidity, temperature, and GPS position. The most convenient scheme is Agent A, which receives the data from the sensors, then sends it to Agent B, which is responsible for making the connection to the Web service. We must remember that the applications run on a network with raspberry pi; this facilitates the manipulation of the information of the sensors.

We have created native functions of the programming language, which serve as support for some procedures. The native function of this application is the reading of the sensors. It has been called the "Measurement Agent." The code for this interaction is:

```
importar mas.__init__

funcion dataSensor(identifier, description, times)
   x = ExecuteScript() # The agent that turns on the rutine
   x.start()
end

log("test Data Sensor")
a = dataSensor(1, MeasurementAgent, 10)
```

The *dataSensor* function receives an identifier as parameters, and an agent turns on the sensors sometimes to execute the transmission of data to the Web service, described in Figure 7. The description of the interaction is:

1. $dataSensor(A,B) \rightarrow dataSensor(B,SW)$
2. $dataSensor(B,SW) \rightarrow ack|\sim(ack)$
3. $dataSensor(A,B) \rightarrow dataSensor(B,SW)$
4. $ExecuteScript(A,MA) \rightarrow MA(device)$
5. $ExecuteScript(A,MA) \rightarrow ack|\sim(ack)$
6. $MA(device) \rightarrow data$
7.

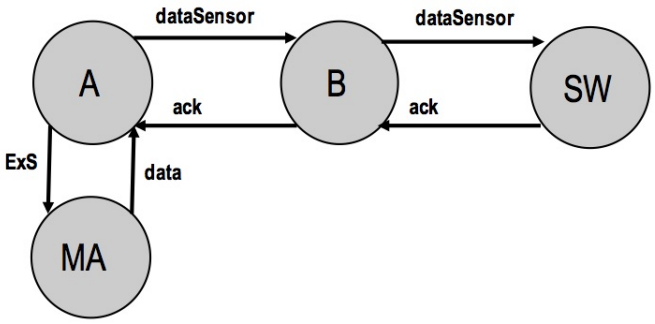

Figure 7: The interactions that occur are: between A and B, the data sensor function is used to send the data to the server we. Enter A and MA (*MeasurementAgent*) to take sensor measurements. The ExS (*ExecuteScript*) function takes care of the procedures at the node level. Between B and SW (*Web server*) which is the action for the transmission of the data.

4.4 Share the Internet from one node to the other nodes of the network

This interaction is the most powerful application in the ad hoc network. Several additional services can be offered on the internet channel to convert the network into a more flexible system to meet possible user requirements. The interaction described in section *propagate* is well suited to this application.
The code for this interaction is:

```
importar mas.__init__

funcion internet(identifier, description, times)
   x = ExecuteScript()
   x.start()
end

log("test Internet")
a = internet(1, CycleCallBash, 1000000)
```

For this application, we have used the *ExecuteScript* agent. In this case, however, the *CycleCallBash* agent executes the native function that modifies the configuration of the node to create a bridge interface and share the internet service. Figure 8 shows this kind of interaction.

The description of the interaction is:

1. $internet(A,B) \rightarrow CycleCallBas\boxtimes(A)|\ ack|\sim(ack)$
2. $internet(B,\otimes_n) \rightarrow internet(A,B)|ack|\sim(ack)$
3. $CycleBas\boxtimes(A,device) \rightarrow ExecuteScript(A,device)$
4. $ExecuteScript(A,device) \rightarrow data$

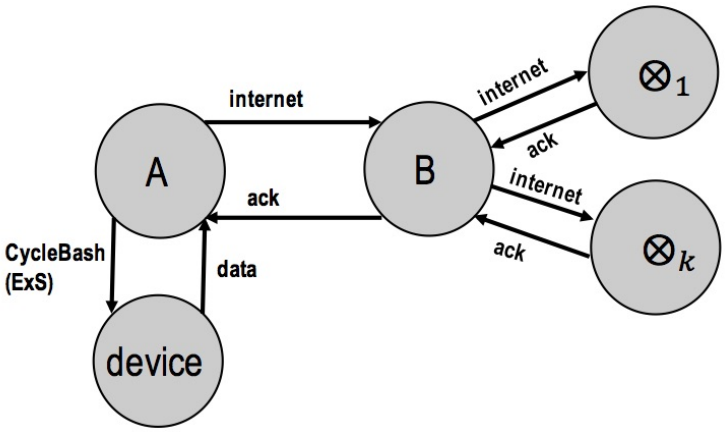

Figure 8: A shares internet with B . The interaction of B between the n nodes of the network, shares the internet that comes from A. The function of A is to execute the script as many times as necessary on the device to share the internet.

5. Conclusions

In this article, we have presented a programming language model with the intention of adapting it to the nature of decentralised networks. The nets interaction model is used as a basis for the programming paradigm. The advantage of the nets interaction computing model lies in the fact that interaction is seen as a form of computing. The new paradigm model views the characteristics of decentralised networks as a structural part of its conception. Thus, in the construction of the language, which is a DSL made in ANTLR, it has been thought to use functions as fundamental structures of language. On the other hand, the use of libraries and native functions that give the programming language the necessary flexibility in the construction of decentralised networks is also considered.

The description is made of the fundamental interactions of the programming language, and its application on the DSL in use is shown. The system of modules gives suitable results. We can build native functions of a sophisticated degree that are synthesised from the proposed programming language. The intention is the construction of a tool to streamline the process of building ad hoc networks, which are networks of a decentralised nature whose modules were built for the language.

On the other hand, the described models are simple and easy to implement. Thinking about the functioning of decentralised networks (for this work, ad hoc networks), the model considers interactions between two nodes (local interactions), and the agent schema can be seen with a control system for each node. Thus, the definition of the agent on a node must also be part of the programming paradigm.

In conclusion, the paradigm proposed in this article is a multi-paradigm that considers the use of the following elements: net interaction, functions, agents, and a library management structure. It is expected that in future tests on ad hoc networks, language as a tool will serves to manipulate elements at the node level, such as handling the data traffic load or bandwidth allocation for particular flows.

References

Aho, A. V. (2003). *Compilers: principles, techniques and tools (for Anna University), 2/e*. Pearson Education India.
Brody, P., & Pureswaran, V. (2014). Device democracy: Saving the future of the Internet of Things. *IBM, September*.
Dressler, F. (2006). Self-organization in ad hoc networks: Overview and classification. *University of Erlangen, Dept. of Computer Science, 7*, 1–12.
Dressler, F. (2008). *Self-organization in sensor and actor networks*. John Wiley & Sons.
Fernández, M., & Mackie, I. (1999). A calculus for interaction nets. In *International Conference on Principles and Practice of Declarative Programming* (pp. 170–187). Springer.
Fister Jr., I., Mernik, M., Fister, I., & Hrnčič, D. (2011). Implementation of the domain-specific language EasyTime using a LISA compiler generator. In *2011 Federated Conference on Computer Science and Information Systems, FedCSIS 2011* (pp. 801–808). Retrieved from http://www.scopus.com/inward/record.url?eid=2-s2.0-83155189061&partnerID=40&md5=56f97e8e3106dce012c77f442f8f72e8
Fitzek, F. H. P., & Katz, M. D. (2013). *Mobile clouds: Exploiting distributed resources in wireless, mobile and social networks*. John Wiley & Sons.
Gershenson, C. (2007). *Design and control of self-organizing systems*. CopIt ArXives.
Grune, D., Van Reeuwijk, K., Bal, H. E., Jacobs, C. J. H., & Langendoen, K. (2012). *Modern compiler design*. Springer Science & Business Media.
Loo, J., Mauri, J. L., & Ortiz, J. H. (2016). *Mobile ad hoc networks: current status and future trends*. CRC Press.
Nitti, M., Girau, R., & Atzori, L. (2014). Trustworthiness management in the social internet of things. *IEEE Transactions on Knowledge and Data Engineering*, *26*(5), 1253–1266.
Ortiz, A. M., Hussein, D., Park, S., Han, S. N., & Crespi, N. (2014). The cluster between internet of things and social networks: Review and research challenges. *IEEE Internet of Things Journal*, *1*(3), 206–215.
Palos, S., Kiviniemi, A., & Kuusisto, J. (2014). Future perspectives on product data management in building information modeling. *Construction Innovation*, *14*(1), 52–68.
Perrinel, M. (2014). On context semantics and interaction nets. In *Proceedings of the Joint Meeting of the Twenty-Third EACSL Annual Conference on Computer Science Logic (CSL) and the Twenty-Ninth Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)* (p. 73). ACM.
Pinto, J. S. (2000). Sequential and concurrent abstract machines for interaction nets. In *International Conference on Foundations of Software Science and Computation Structures* (pp. 267–282). Springer.
Prehofer, C., & Bettstetter, C. (2005). Self-organization in communication networks: principles and design paradigms. *IEEE Communications Magazine*, *43*(7), 78–85.
Sheng, W., Schürmans, S., Odendahl, M., Bertsch, M., Volevach, V., Leupers, R., & Ascheid, G. (2014). A compiler infrastructure for embedded heterogeneous MPSoCs. *Parallel Computing*, *40*(2), 51–68.
Zhang, Z., Long, K., & Wang, J. (2013). Self-organization paradigms and optimization approaches for cognitive radio technologies: a survey. *IEEE Wireless Communications*, *20*(2), 36–42.

Biographies

Joaquín F Sánchez is currently teacher at the Universidad San Mateo, in the area of telecommunications engineering. It is magister in telecommunications and their area of interest are mobile networks and congestion control in data networks. He is currently advancing doctoral studies in computer science, where research in implementing compilers.

Jorge A Quiñones: He is a Postgraduate student in the master's degree in systems and computer engineering at the National University of Colombia. His research interests are focused on the development of intelligent agents and blockchain technology.

Juan M Corredor: He is Student of Systems Engineering and Computing at the National University of Colombia. His research interest focuses on the development of intelligent agents, computer models for artificial intelligence and development of web applications.