

# Connecting Resource-Constrained Robots to Knowledge-Based Systems

Jukka Riekkı, Xiang Su, Janne Haverinen  
Intelligent Systems Group and Infotech Oulu  
FIN-90014, University of Oulu  
Finland  
{jukka.riekki, xiang.su, janne.haverinen}@ee.oulu.fi

## ABSTRACT

This paper focuses on connecting resource constrained robots to knowledge-based systems. These systems would enable various useful applications but on the other hand they present challenging requirements to the robots. The limited bandwidth, processing power, and storage capabilities do not allow the rich knowledge representations to be communicated, processed, nor stored by the robots. We suggest the Entity Notation for tackling this challenge. This representation is light-weight, so it can be handled by the robots. On the other hand, it can be transformed in an unambiguous fashion into the rich representations used by the knowledge systems. We present some preliminary experiments and discuss the future work.

## KEY WORDS

Entity Notation, light-weight representation, RDF

## 1. Introduction

The recent advances in electronics, sensor technology and material technology have made it possible to implement multi-robot systems, robot swarms [21][22][23], that can consist of even hundreds of autonomous robots. Robot swarms can be utilized in numerous interesting application domains like space exploration, distributed environment modeling, cleaning, and search and rescue operations. In comparison to a single and complex autonomous robot, robot swarms provide increased robustness to individual failures and the power of parallel operation.

In this paper, we focus on knowledge representations for connecting resource-constrained devices to knowledge-based systems. How could we utilize the rich, state-of-the-art knowledge representations in swarm systems? These representations would allow describing knowledge explicitly in a format that could be processed by other systems, e.g. reasoners could deduce actions that would proceed the swarm towards its goal; they could coordinate and adapt the operation of the swarm of unsophisticated and resource bounded robots. Semantic Web technologies would offer lots of possibilities to implement functionality for the swarm. The main challenge here is that the individual robots most often have constrained resources and cannot communicate, store, or process these advanced representations, not to mention running the advanced

reasoners and other applications utilizing the representations.

We suggest the Entity Notation (EN) to connect the resource-constrained robots to knowledge-based systems. The key idea is that the robots handle the light-weight Entity Notation, the knowledge-based systems handle the advanced knowledge representations – and that an unambiguous, lossless transform can be performed between these two representations when necessary.

Generally, EN is a light-weight representation for exchanging information between nodes of a distributed system. We started to develop EN as a general representation that can be used by any resource-constrained device but now we are focusing on mobile robots. The EN sets so small requirements to the robots that in practice any robot capable of communication can use this representation. The EN packets can be as short as some tens of bytes and still the packets can be utilized by knowledge-based applications, e.g. a reasoner, in a straightforward way.

The rest of this article is organized as follows. Section 2 introduces the Entity Notation. Section 3 introduces a simulator for testing EN. Section 4 presents a gesture recognition system utilizing EN. Section 5 introduces the related work and Section 6 concludes the paper.

## 2. Entity Notation

Entity Notation is a light-weight representation for exchanging information between nodes of a distributed system. It is not a protocol; hence it does not offer any routing mechanisms, checksums, etc. An EN packet is assumed to be the payload of some protocol delivering the packets, for example HTTP, TCP/IP, or some sensor network protocol.

We assume a protocol that offers at least operations to send and receive messages that contain data in EN format as payload. Also publish/subscribe functionality, or even content-based routing might be available. We do not focus on the protocol delivering the information in EN format in this paper – although we suggest a couple of operations for negotiating the EN packet type later in this paper. The first goal in developing EN is that it can be used over simple communication links by resource-constrained

devices. The second goal is that the packet content can be used in a straightforward fashion by advanced knowledge-based systems. For example, it should be possible to transform the packets into RDF (Resource Description Framework) or OWL (Web Ontology Language) format in an unambiguous fashion. The third goal is that EN is general; it can carry any kind of information between two nodes. The fourth goal is that the information can always be identified in a unique fashion. This is an important feature when all the possible usages of the information cannot be specified in advance. Furthermore, unique identification is an important prerequisite for the second and third goals.

Our aim is to achieve these goals as follows. To support resource-constrained devices and simple communication links, EN offers short packets that in their simplest form can be less than 40 bytes long. To allow the packet content to be utilized by knowledge-based systems, the EN offers straightforward and unambiguous transformations to advanced representations like RDF and OWL. Hence, packet content can be delivered to e.g. a reasoning engine processing knowledge in OWL format. Furthermore, the basic EN format is general and it does not constrain the type of information. Finally, EN uses UUIDs (Universally Unique Identifiers) and URLs for unique identification.

When two nodes exchange a message, the role of the Entity Notation is the following: The sender builds an EN packet and delivers it as the payload for the protocol performing the actual delivery. The receiver extracts the EN packet from the message and transforms it into the native format it uses. The EN software offers the tools for transforming the packet from the EN format to the general formats like RDF and OWL and vice versa. During the communication, the lower layer checks that the received packet is not corrupted, routes the packets to the recipient, and performs all the other tasks familiar for communication protocols.

The rest of this section explains Entity Notation in detail. We describe first the complete packet format and then the short format and the related templates. Furthermore, we discuss shortly how the sender and the recipient can together decide whether the complete format or some short format should be used in the communication.

## 2.1 The Complete Packet Format

The complete EN packet format follows closely the RDF format. A basic RDF document contains a sequence of (subject, predicate, object) triplets. These statements describe resources. The subject identifies the thing the statement is about. The predicate identifies the property or characteristic of the subject, while the object identifies the value of that property. [1] A complete EN packet is of the form:

```
[subject    predicate    object]
[subject    predicate    object]
...
[subject    predicate    object]
```

Subjects, predicates, and objects are identified in a unique fashion by URIs (when the objects are not literals). In addition to these three values, type information is included in the packet.

The Entity Notation, as its name implies, describes entities. The first subject is interpreted to specify the primary entity the EN packet is about. Any number of statements can be made about this subject. Furthermore, some objects can be other entities, and additional statements can give information about those entities. An entity is some identifiable whole, physical or digital. For example, a robot, a sensor, an RFID tag, a sending node, a recipient, a person, a measurement, a message, a book, and a user terminal are entities.

An EN packet is a sequence of entity descriptions. Each description specifies the type of the entity, a unique entity identifier, and a number of property triplets (name, type, value). XML Schema types are used. An entity description is of the form:

```
[EntityType EntityId
  PropertyName PropertyType PropertyValue
  ...
  PropertyName PropertyType PropertyValue
]
```

An entity description contains a set of triplets about a single entity. *EntityId* specifies the subject, *PropertyName* is a predicate, and *PropertyValue* is an object. *EntityType* and *PropertyType* specify the types of the corresponding subject and object.

As discussed above, some objects can be other entities. In this case the value of a property is an identifier of the other entity. This allows relations to be represented. The relations can be between physical entities, e.g. a user is in a room. Or, the relations can be between digital entities. For example, the first entity of a message can specify a list and refer to all list member entities. The rest of the message can then describe these entities.

Here is an example in which a room entity description refers to a temperature sensor in that room:

```
[http://ee.oulu.fi/o#Room
  http://ee.oulu.fi/o#room1879
  http://ee.oulu.fi/o#hasTempSensor
  http://ee.oulu.fi/o#EntityId
  http://ee.oulu.fi/o#tempSensor234
]
```

```
[http://ee.oulu.fi/o#TempSensor
  http://ee.oulu.fi/o#tempSensor234
  http://ee.oulu.fi/o#tempValue
  http://www.w3.org/2001/XMLSchema#float
```

```

23.5
http://ee.oulu.fi/o#timestamp
http://www.w3.org/2001/XMLSchema#dateTime
2002-10-10T12:00:00-15:00:00
]

```

The type *EntityID* is an exception in the type system as all the other types are XML Schema types. *EntityID* specifies the corresponding property value to be an entity identifier – not a literal URI (that would be the XML Schema type *anyURI*).

A complete packet can be transferred to an RDF description in a straightforward fashion because of their natural relationship. The entity type can be represented as a resource class (using the *rdf:type* property) and the entity identifier is the identifier of the resource. Property names and values can be mapped directly, and the property type can be represented by adding an *rdf:datatype* attribute to the property element. To maintain a straightforward mapping between the EN and RDF formats and to keep the RDF documents simple to process, we do not use striping. However, we do use the shorthand for resource classes, that is, we replace “*rdf:Description*” with the resource type and omit the *rdf:type* property. We also use identifiers, i.e. the *rdf:ID* attribute instead of *rdf:about*.

Here we present the RDF document of the example above (due to the narrow text columns, we have replaced in this and all the following examples “http://ee.oulu.fi/o” by “EE” and “http://www.w3.org” by “W3”; the description has been indented as well):

```

<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF xmlns:rdf="W3/1999/02/22-rdf-syntax-ns#"
  xmlns:e="EE#" xml:base="EE">
  <e:Room rdf:ID="room1879">
    <e:hasTempSensor rdf:resource="#tempSensor234"/>
  </e:Room>
  <e:TempSensor rdf:ID="tempSensor234">
    <e:timestamp rdf:datatype="W3/2001/XMLSchema#dateTime">
      2007-10-22T12:00-17:00
    </e:timestamp>
    <e:tempValue rdf:datatype="W3/2001/XMLSchema#float">
      23.5
    </e:tempValue>
  </e:TempSensor>
</rdf:RDF>

```

The naming convention is the following: entity types start with an initial capital letter, property names start with initial lowercase letters, when the property value is an entity the property name contains a verb and a noun, when the value is a literal the name contains only a noun, identifiers are otherwise identical with entity type names but they start with a lowercase letter and end in a number. URI references are used, i.e. fragment identifiers are used heavily.

The RDF graph corresponding with the RDF document is shown in Figure 1.

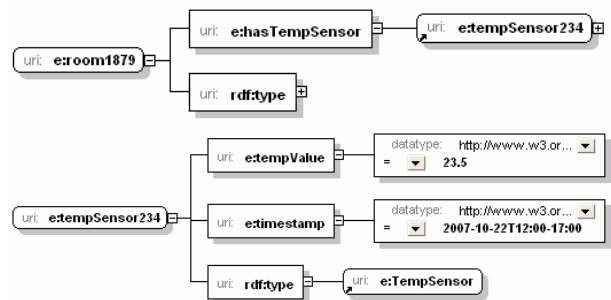


Figure 1. RDF graph representing a room and a sensor.

## 2.2 The Short Packet Format

Complete Entity Notation packets can be quite long because of meaningful type names and URIs. We suggest templates to shorten the packets. A template contains a description of the constant part of a message and placeholders for the values that differ in each packet. The packet sent over the communication link needs to contain only the template identifier and the changing information. A complete packet can then be assembled by replacing the template’s placeholders with the values contained in the message.

For example, if a robot sends temperature values to the local network, the message might contain just a template identifier, the temperature value, and a timestamp. The receiver can assemble the complete packet without any information loss as long as it has the template.

We identify templates using the UUID scheme. UUID is an identifier that is guaranteed to be unique across space and time. UUIDs can be created independently, in a distributed fashion, without any central registry (even then they are unique at a very high probability). Each UUID is unique over the whole network – this is a critical feature when a general solution is required. The basic UUID is 128 bits long, and in URN format looks like this:

```
urn:uuid:f81d4fae-7dec-11d0-a765-00a0c91e6bf6
```

For example, if we want to send temperature values, only the temperature value and the timestamp differ from message to message. If we have for the packet the template:

```

[EE#TempSensor EE#tempSensor234
 EE#tempValue W3/2001/XMLSchema#float ?1
 EE#timestamp W3/2001/XMLSchema#dateTime ?2
]

```

and that template has the identifier `urn:uuid:5e76af`, we can represent the packet as:

```

[urn:uuid:5e76af
 23.5
 2002-10-10T12:00:00-15:00:00]

```

Nodes can have predefined templates for each packet type they can send. Usage of short packets can either be decided by the engineer at design time or the nodes can use complete packets by default and agree the short format during communication. Negotiating the packet type is discussed next.

### 2.3 Negotiating Short Packets

The default mode of communication is to use complete EN packets. A node can suggest to its peer a short message format, i.e. to use a template. This request *useTemplate* can be part of the interface that a node implements. If the peer replies with an agree reply, the node can start using the short format. But if the peer replies with a denial, the node has to continue using the complete packet type.

## 3. Simulator

As the first step to test the EN, we developed a simulator to simulate the transfer of the EN packets between different devices. We created three types of sensors in the simulator: temperature sensor, a wireless device designed for reporting the patient's well-being to nurses [2] and a location sensor.

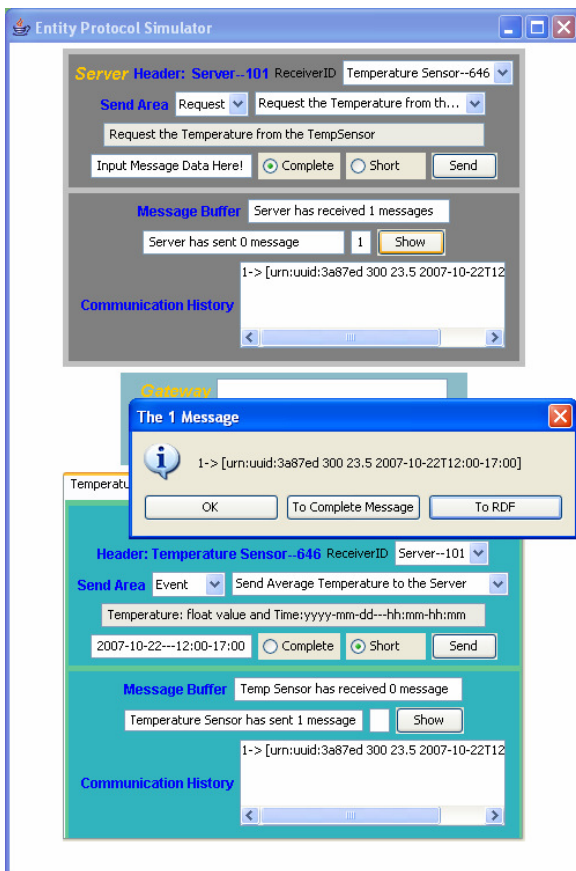


Figure 2. The EN Communication Simulator.

The user can specify messages for any device, send them to the specified recipients, assemble short packets into complete packets and transform packets into RDF format.

In this simulator, we added one entity in front of each message to describe the message itself: the type (request, reply, or event), the sender, the receiver, and a sequence number. This entity was included in the template. In a real implementation such an entity would not be needed if the same information is carried by the protocol.

Figure 2 shows a situation in which the temperature sensor has sent a short packet to the server. The user has selected the received message and is about to decide whether the message is to be assembled into a complete message or transformed into RDF format.

In the temperature sensor window, we can specify the details of the packets: the ReceiverID, the type of the packet, and the data content. In Figure 2, a short packet including the current temperature and the timestamp can be seen. The short packet is the following:

```
[urn:uuid:3a87ec 300
 23.5
 2007-10-22T12:00-17:00]
```

In this packet, the identifier *urn:uuid:3a87ec* determines the template, *300* is the sequence number, and the last two values are the current temperature and the timestamp. The complete packet structure and the RDF document follow the guidelines presented above and the temperature sensor entity is identical with the one presented above.

The server and the devices can send 12 different packets in total: the server can send requests to the devices, the devices can reply to the server's requests, and send asynchronous events to the server. Compared with complete packages and their RDF representations, short packets contain on the average only 11.50% of the characters that a complete package contains and 7.30% of the characters that the corresponding RDF document contains.

## 4. Gesture Recognition System

Our first prototype utilizing the EN controls a Web Browser based on the gestures performed by a user. We utilize the UPnP (Universal Plug and Play) [3] technology in this prototype. This system consists of distributed nodes – resource-constrained sensors, knowledge-based applications, browsers and displays – that send EN packets to each other. User's gestures are recognized from data produced by an acceleration sensor that is held by the user. Up to 12 different gestures can be recognized.

As shown in Figure 3, the architecture of the gesture recognition system includes three main parts: physical accelerometers and access points, acceleration sensor

applications and the gesture recognition control point application. Physical acceleration sensors send acceleration data at 50Hz over radio link to a base station connected to a PC's USB port. As the sensor is quite resource-constrained and challenging to program, we implemented on the PC an acceleration sensor application that acts as a virtual sensor. This virtual sensor is an UPnP device that holds in its state variables the most recent acceleration vectors.

The gesture recognition control point is an UPnP control point. When an acceleration sensor moves inside the operation range of the base station, the acceleration sensor application starts an instance of a virtual acceleration sensor device, and registers it to the control point application. The virtual device sets its power on and starts delivering data packages to the subscribers. When an acceleration sensor moves out the range for a while (5 seconds), acceleration sensor application stops the instance of the virtual device and unregisters it from the control point.

The gesture recognition control point subscribes to the changes in the virtual sensor's state variables. It uses hidden Markov models to recognize the movement of sensor's holder [4]. The control point sends the names of the recognized gestures (for example, "clockwise") to its client, that is, to the browser. The browser maps the gesture name to an application-specific action, for example, a "clockwise" gesture to the browser's "forward" command.

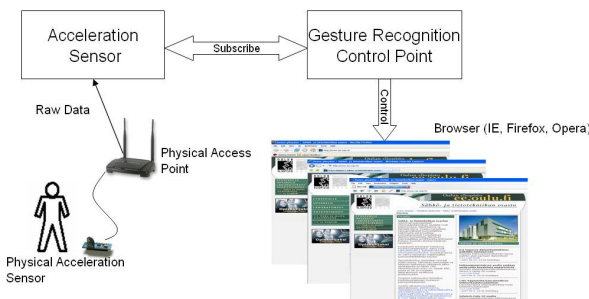


Figure 3. Architecture of the UPnP Gesture Recognition System

The state variables holding acceleration vectors are represented as short packages as follows:

```
[urn:uuid:e232-a113-e440-e062 0 AccMessage
3 544 -484 590 25.1175 125 101648 1978
...
3 591 144 1315 25.1175 101 101648 2007
]
```

In this packet, the number after the UUID is the sequence of the packet, and the `AccMessage` is the name of the state variable, which is followed by the value of the state variable.

## 5 Related Work

The key feature of the Entity Notation is that it has a lightweight data representation and it supports Semantic Web technology at the same time. In this section we compare EN with other data representations. We adopt UUID in EN, as UUID is a good solution for identifying distributed system's resources. It is also used in other protocols [5] and peer-to-peer systems [6].

XML is widely used, and RDF can be described using XML. But XML is a heavy-weight representation. To reduce the payload of XML messages, many methods and tools for compressing XML have been suggested, for example, XMill [7], XMLPPM [8], SCA [9], Millau [10], XGrind [11], XMLZip [12], XPress [13] and XML Skeleton Compression [14]. Most of them use some structure compression algorithms or some homomorphic transformation strategy to compress XML. According to Wilfred Ng et al's report [15], XMLPPM achieved the best compression ration in their test datasets. The compression percentage can reach 8.25% of the original XML file size. Some methods loose content during the compression. EN has the advantage that it is a lossless solution.

On the other hand, some markup/data serialization languages are also good solutions for lightweight data exchange, like UBF [15], SSDP [16] and JSON [17]. For example, JavaScript Object Notation (JSON) is a text-based, human-readable format for representing objects and other data structures and is mainly used to transmit structured data over a network connection (in a serialized format). JSON is a subset of the object literal notation of JavaScript and is commonly used with JavaScript. The basic types and data structures of most other programming languages can also be represented in JSON, and the format can therefore be used to exchange structured data between programs written in different languages. It is more lightweight and easier to parse and generate than XML. Currently, JSON finds its main application in Ajax web application programming, as a simple alternative to using XML. But these serialization languages define only the message structure and allowed basic data types, they do not present any general data formats that could be transformed in a straightforward fashion into knowledge representations.

## 6. Discussion

This paper presents Entity Notation, a light-weight representation for exchanging information in a distributed system. We described the complete packet format, the short packet format, and discussed how the packet type can be negotiated between the communicating peers. We presented the preliminary experiments using a simulator and a gesture recognition system.

The Entity Notation can be used to connect resource-constrained robots to knowledge-based systems. EN allows very simple and short messages to be sent by the robots; on the other hand these messages can be transformed into advanced knowledge representations in an unambiguous fashion. For example, when the robot does not even contain any OS but all functionality is programmed directly using C, it is straightforward to use UUID values as constants that are used to identify received messages and placed at the beginning of the sent messages.

The knowledge-based system receiving data from the robots can transform EN packets into RDF or OWL format and use the information to produce new task relevant knowledge, and to adapt the operation of the swarm to perform given tasks efficiently. Furthermore, the knowledge representations can be utilized to automatically create status reports about the operation of the swarm in human readable form.

The length of short packets is less than 10% of the corresponding RDF document's length for simple messages. These results are comparable with the XML compression results reported by others. However, when short EN packets are used, a resource-constrained device does not need to run any algorithm to process (i.e. compress) the packet before sending or after receiving it. Furthermore, we expect much better results for complex messages.

The future work includes building the first prototype where robots use the EN notation. We will use robots that are modified version of the iRobot Create mobile robots. These robots will be able to read and write information from/to RFID tags, as well as communicate with the central server and other robots. The prototype will contain OWL descriptions and simple reasoning. The Entity Notation will be used in communication and to deliver information to the reasoning engine. The aim of this work is to verify the Entity Notation more thoroughly and to demonstrate its capabilities. We will modify the EN based on this work, for example, to offer as straightforward transformation to OWL as possible. We will also consider alternative representations for complete packets, for example, Notation 3 and N-Triples. We will experiment with more complex data structures and binary data as well.

## Acknowledgements

This work was funded by Infotech Oulu. We would like to acknowledge Mikko Kauppila and Susanna Pirttikangas for implementing the gesture recognition component.

## References

[1] W3C, RDF Primer, <http://www.w3.org/TR/REC-rdf-syntax/>

[2] J. Riekkilä, I. Alakärppä, R. Koukkula, J. Angeria, M. Brockman & T. Saloranta (2007) Wireless Pain Monitoring. *The 2nd International Symposium on Medical Information and Communication Technology (ISMICT'07)*, Oulu, Finland, 2007.

[3] UPnP Forum. <http://www.upnp.org/about/default.asp>.

[4] M. Kauppila, S. Pirttikangas, X. Su & J. Riekkilä (2007) Accelerometer Based Gestural Control of Browser Applications. *Int'l Workshop on Real Field Identification (RFId2007)*. In conjunction with *Fourth International Symposium on Ubiquitous Computing Systems (UCS 2007)*. Nov. 25-28, Tokyo, Japan.

[5] S. Avancha, A. Joshi & T. Finin (2002) Enhanced Service Discovery in Bluetooth, *Computer*, 35(6), 96-99.

[6] K. Aberer, A. Datta, and M. Hauswirth, Efficient, Self-Contained Handling of Identity in Peer-to-Peer Systems, *IEEE Transactions on Knowledge and Data Engineering*, 16(7), 2004, 858-869.

[7] H. Liefke and D. Suciu, XMill: An Efficient Compressor for XML Data. *Proceedings of the ACM SIGMOD International Conference on Management of Data*, Dallas, Texas, 2000, 153-164.

[8] J. Cheney. Compressing XML with Multiplexed Hierarchical PPM Models. *Proceedings of the IEEE Data Compression Conference*, Snowbird, Utah, 2000, 163-172.

[9] M. Levene and P. T. Wood. XML Structure Compression. *Proceedings of the Second International Workshop on Web Dynamics*, Honolulu, Hawaii, 2002.

[10] N. Sundaresan and R. Moussa. Algorithms and Programming Models for Efficient Representation of XML for Internet Applications. *Proceedings of the 10th International WWW Conference*, HongKong, China, 2001, 366-375.

[11] P. M. Tolani and J. R. Haritsa. XGRIND: A Query-friendly XML Compressor. *IEEE Proceedings of the 18th International Conference on Data Engineering*, San Jose, CA, 2002, 225-234.

[12] XMLZip - XML Solutions. <http://www.xmls.com/>.

[13] J. K. Min, M. J. Park, and C. W. Chung. XPRESS: A Queryable Compression for XML Data. *Proceedings of the ACM SIGMOD International Conference on Management of Data*, San Diego, CA, 2003, 122-133.

[14] P. Buneman, M. Grohe, and C. Koch. Path Queries on Compressed XML. *Proceedings of the 29th International Conference on Very Large Data Bases*, Berlin, Germany, 2003, 141-152.

[15] Armstrong J (2002) Getting Erlang to talk to the outside world. *Proceeding of ACM SIGPLAN Erlang Workshop 2002*, Pittsburg, PA, 2002, 64-72.

[16] Yaron Y. Golland, Ting Cai, Paul Leach, Ye Gu, and Shivaun Albright, Simple Service Discovery Protocol, IETF Draft draft-cai-ssdp-v1-03.txt, <http://quimby.gnus.org/internet-drafts/draft-cai-ssdp-v1-03.txt>.

[17] JSON, <http://www.json.org/>.

[18] N-Triples, <http://www.w3.org/2001/sw/RDFCore/ntriples/>.

[19] Notation 3, <http://www.w3.org/DesignIssues/Notation3>.

- [21] G. Beni and J. Wang. Swarm intelligence in cellular robotic systems. In *Proc. NATO Advanced- Workshop on Robotics and Biological Systems*, I1 Ciocco, Tuscany, Italy, 1989.
- [22] J. Haverinen, M. Parpala & J. Röning (2005), A Miniature Mobile Robot With a Color Stereo Camera System for Swarm Robotics Research, Proc. IEEE International Conference on Robotics and Automation (ICRA2005), Apr 18-22, Barcelona, Spain, p. 2494-2497.
- [23] F. Mondada, A. Guignard, M. Bonani, D. Bar, M. Lauria & D. Floreano (2003) SWARM-BOT: from concept to implementation. In. Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003), Oct 27-31, Las Vegas Nevada, USA, p. 1626 – 1631.