Hindawi Publishing Corporation
International Journal of Distributed Sensor Networks
Volume 2012, Article ID 860864, 11 pages
doi:10.1155/2012/860864

Research Article

Resource Description Language: A Unified Description Language for Network Embedded Resources

André C. Santos,[1,2] Luís D. Pedrosa,[1,3] Martijn Kuipers,[1,2] and Rui M. Rocha[1,3]

[1] *IST, Technical University of Lisbon, Avenida Professor Dr. Aníbal Cavaco Silva, 2744-016 Porto Salvo, Portugal*
[2] *INESC-ID, Instituto de Engenharia de Sistemas e Computadores Investigação e Desenvolvimento em Lisboa, Rua Alves Redol 9, 1000-029 Lisboa, Portugal*
[3] *Instituto de Telecomunicações, Avenida Rovisco Pais 1, 1049-011 Lisboa, Portugal*

Correspondence should be addressed to André C. Santos, acoelhosantos@ist.utl.pt

Received 20 April 2012; Revised 10 July 2012; Accepted 10 July 2012

Academic Editor: Jianhua He

As machine-to-machine networks become larger and more pervasive, manual configuration and discovery of resources will become intractable. It is in this context that we propose the RDL, a Resource Description Language that represents a uniform way of describing embedded resources, allowing them to be shared and enabling a new class of resource-aware applications. The RDL can describe a wide range of resources, characterizing individual nodes or entire networks. It can contribute to overcome performance issues in dense networks or mobility-driven problems in highly dynamic machine-to-machine topologies by providing the means for self-adaptability and manageability, as well as opportunistic resource sharing in context-aware embedded applications. The main goal for the RDL is to define a reusable and extensible resource description specification, which can only be reached if the resources are described in a standardized format. To illustrate the feasibility of our approach, we have also developed a Java implementation of the RDL framework, as well as a TinyOS implementation targeting resource constrained platforms. Furthermore, we have developed Modulus, a modular middleware for the development of resource-aware distributed applications.

1. Introduction

Over the past few years, the increasing demand for "always on" communications has been imposing a big pressure on network architectures that not only have to provide efficient and sustainable support to an increasing range of services but also have to cope with scenarios where heterogeneity and mobility are of key importance. In parallel, the emerging smart objects concept brought along new challenges in architectures, protocols, and services where scalability, heterogeneity, and mobility are also issues to consider, perhaps even more important as they are supposed to interact with the physical world. Commonly, these systems are not well prepared for changes in the surrounding environment as with changes in the evolving resources, resulting in problems and malfunctions, which are especially critical in such resource-constrained devices that support resource-aware applications.

Machine-to-machine (M2M) systems are also experiencing a decisive evolution in this last decade. Indeed, as the first years of the new millennium went by, one could notice how solutions used in typical control applications for reaching remote embedded devices from central control hubs, using a one-to-one interconnection approach, were progressively turning into rich network-based architectures encompassing a multitude of network resources and converging towards a global mobile Internet paradigm. This Internet of embedded devices—the so called Internet of Things—relies on a multi-tier architecture mixing already operating wired and wireless networks with emerging embedded networks specially developed and deployed for M2M applications.

Beyond the traditional challenges associated with wide, local, or personal area communication networks, there are a number of challenges specifically related with M2M settings. Challenges like coping with device heterogeneity, "zero-touch" manageability, standardized plug-n-play capability,

augmented sensing support based on resource-constrained devices are among those that should be tackled efficiently in future M2M networks [1].

A number of usage models have been identified as promising for M2M applications. Examples more often cited are smart homes that use embedded systems integrated in home appliances for automation, security, entertainment, and energy management; vehicular networks used for safety, traffic information, navigation, and entertainment; healthcare in a variety of scenarios based on interconnected body area networks capable of gathering vital information from biomedical sensor networks and triggering appropriate responses. These examples are just the "tip of the iceberg" in a world of interconnected smart objects. However, as the complexity of this world increases, in terms of scale and popularity, the system's overall reliance on human intervention to determine network roles and policies in the presence of complex dynamics is no longer sustainable.

Adapting to new environments is indeed an important task for true M2M networks, involving the collection of information about neighboring nodes and the negotiation of available resources to communicate on, as well as available services [2].

Highly dynamic topologies, where new nodes can appear or existing ones can move around and shut down, are also envisaged scenarios in the networked embedded system's world. Thus, a set of mechanisms providing autoconfiguration and self-organization of network nodes are mandatory to handle topology changes. Also, given the heterogeneous nature of modern networks, interoperability issues are of extreme importance as they define how adaptable a system can be. Furthermore, due to the hierarchical nature of M2M systems architecture, the existence of network nodes where information should be aggregated and processed is practically mandatory for efficiency reasons, leading to the need to determine and configure optimal aggregation points.

Besides several tentative approaches (e.g., IEEE 1451 [3], UPnP [4]), a suitable standardized solution to describe embedded resources in a network is still lacking. In fact, solutions for network nodes autoconfiguration tend to be extremely complex given the heavy resource constraints of small embedded nodes.

In this paper, we define a unified approach for the description of resources, called the Resource Description Language (RDL), which can be used by various entities, such as resource management algorithms. In short, the RDL concept is about describing objects that do not exist as content, such as temperature sensors and queues. The RDL, for instance, can be used for cross-layering information from the physical layer, so that actions can be triggered from events happening at the lowest layer, such as a fading of the wireless channel in case of a wireless link. Another example is the use of the RDL to describe the resources in a Wireless Sensor Network (WSN) or in M2M networks. As the RDL is a neutral specification, it allows M2M communication between different devices that adhere to the RDL. The specification is robust in a way that machines can easily extract known and necessary information from the RDL and ignore or (if needed) forward the unknown parts, without the need to implement all possible uses of the specification in each participating device.

The main goal for the RDL is to define a reusable resource description specification, which can only be reached if the resources are described in a standard, interchangeable form. Since the RDL can be input to management algorithms, there is a need to separate the definition of the resources from the functionality of the algorithms that use them. This way, the algorithms can support the heterogeneity of resources, which is beneficial for opportunistic networking, where each node might try to use whatever resources that are available at any given time.

To better illustrate the benefits of using the RDL in M2M environments, we consider two use cases where the RDL may represent a significant improvement on their functionality allowing an efficient and flexible network operation.

The first use case relates to cost-effective solutions in highly dense structures comprising a large number of embedded devices providing information that should be uploaded to monitoring centers. In such situations, network capacity and devices' energy resources are at a premium, demanding for mechanisms capable of saving precious network resources. One efficient mechanism is the aggregation of information produced by the sensor devices right at the lower levels of the sensor subnetwork tree as depicted in Figure 1.

A common situation in sensing systems is often characterized by sensor devices that operate in overlapping sensing regions. In these cases, their sensor readings will have practically the same values, resulting in a waste of resources since many sensor messages are just duplicated copies of previously sent ones. For optimization purposes, a data aggregation mechanism is used.

In order to help the system to achieve optimal efficiency, namely, in the case of a large sensor field, the aggregator nodes should be configured automatically, without user intervention, whenever two streams of the same resource type pass through these nodes. This is where the RDL comes into play conveying information about resource types and requirements, and allowing aggregator nodes to configure themselves to operate over data streams upon requests (also adhering to the RDL semantics) received from upper level nodes. What is not at all obvious is whether aggregated values should be aggregated with raw data or if aggregated values should be aggregated with other aggregated values when both regard the same data type. This multilevel aggregation limits message forwarding even further, increasing energy efficiency. It is precisely the situation illustrated in Figure 1. In it, the straight red lines are raw data readings, the black dotted lines are aggregated readings, and the final small dotted line regards are the hypothetic aggregation of previously aggregated values.

The second application scenario concerns a mobility setting, commonly found in vehicular networks or in mobile healthcare applications, where M2M subnetworks may become in contact occasionally with a fixed or mobile infrastructure. We chose a road system scenario to demonstrate the usefulness of the RDL in opportunistic situations. Maintenance and safety assurance on extensive road

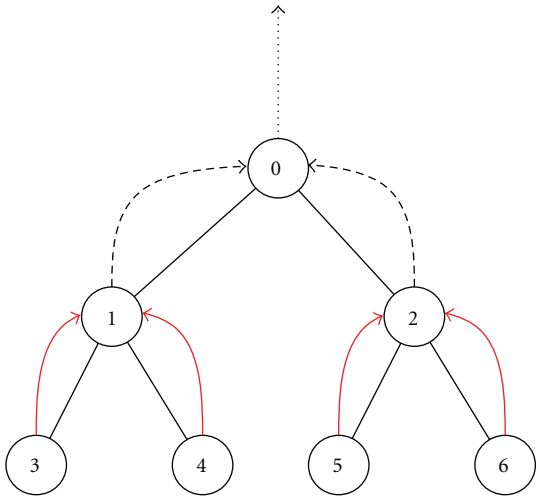

Figure 1: RDL used in multilevel autoaggregation.

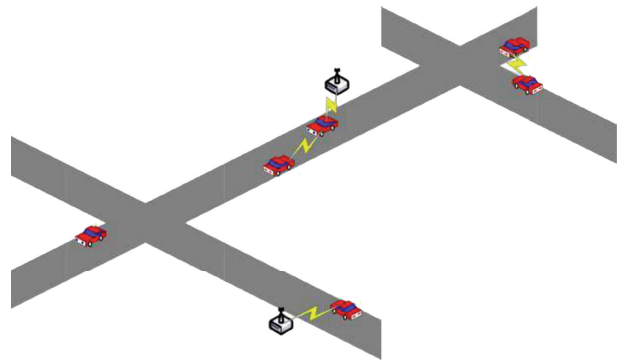

Figure 2: A road system scenario.

networks present relevant challenges for the organizations responsible for them. Highways are usually equipped with expensive monitoring equipment. Secondary and regional roads, on the other hand, are of too great extent for such constant monitoring to be cost-effective or even sustainable.

A M2M system could be the right answer for these scenarios as small, low-cost embedded sensor devices can be freely deployed wherever needed, allowing information collection with an acceptable investment strategy. There is, however, the problem of how to transmit the captured information to the monitoring center. Using public cellular communication infrastructures is certainly a possibility but carries significant operational costs. An opportunistic M2M network, as the one depicted in Figure 2, using cheap, plug-n-play On Board Units (OBUs) on regular vehicles would present a less expensive alternative. Such OBUs are already commonly deployed by governments and transit companies for vehicle identification for billing purposes (e.g., for electronic toll collection) and in many places are even mandatory. These systems could be easily repurposed to allow some degree of vehicle-to-vehicle and vehicle-to-infrastructure communications, while also increasing their processing and storage capacity. By using such enhanced OBUs as mobile carriers, sensor nodes would only have to perform local message transfers using built-in short-range radios.

As drivers do not all follow the same routes, there would be opportunities for vehicle-to-vehicle communications which could positively impact the network performance. However, there are several problems associated with such opportunistic strategies. On the one hand, when the density of vehicles becomes too high, the impact on the network performance is affected by the multiple secondary nodes within interference range of each other, all trying to gain access to the same spectrum. On the other hand, as users expect the OBUs batteries to last for several years, even with the aid of an unobtrusive energy harvesting scheme (e.g., a solar panel), the minimization of energy consumption, through a series of mechanisms (e.g., carefully choosing the next nodes, based on limited available information), is of prime importance.

This is where the RDL can play a significant role. By allowing vehicles and road-side sensor nodes to negotiate available resources in a standard way, the underlying application or middleware systems (as discussed in Section 6) can easily ensure that data is only passed onto a node that has sufficient storage and power resources to carry it. Furthermore, some OBUs (e.g., those on public transit or road maintenance vehicles) could be equipped with sensors of their own and could perform more enhanced sensor data fusion and processing. The ability to allow such systems to be incrementally developed and deployed is the RDL's main strength. Indeed, all three of the above described systems (road-side sensors, common and enhanced vehicle OBUs) could be developed independently and in a decoupled fashion.

As for the interference mitigation, common channel sensing policies for tracking white space in the spectrum are seen as an important part of opportunistic spectrum access [5]. Interfering nodes must collaborate in order to reduce the collisions, which would affect the efficiency of the spectrum utilization. In such cases, the RDL can be used as an effective method for the nodes to convey channel information to others, as the RDL can give a common and very compact description of their resources. Although the RDL itself is not an algorithm to solve the spectrum access among secondary nodes, it can be of great benefit as a sharing mechanism of information for the negotiation game in opportunistic channel access.

The RDL can be specified in two supported formats, that is, XML or KLV, and we further contribute with the development of a Java implementation for the RDL framework which embodies the translation of the RDL concepts for platforms which support the Java environment. For user interaction and development of the RDL language, a Java applet tool has been developed (RDL applet tool available at http://web.ist.utl.pt/acoelhosantos/rdl/). Within the tool, using a graphical user interface, it is possible to build an RDL graphically using a tree interface, having a clear perception of the hierarchical structure of the resource definitions and respecting the RDL element structure; import external RDL

files to be viewed as a resource tree which allows further development of already defined RDL's; export the RDL created by generating it to XML and/or KLV formats. For more computational limited devices, a TinyOS implementation of the RDL framework is currently being developed. For a stable establishment as a language, the RDL specification will not be closed, always aiming at addressing and tackling new and arising resource description necessities. Moreover, we have developed Modulus, a modular middleware for the development of resource-aware distributed applications.

The structure of the remainder of the article is as follows. In Section 2, an overview of related work is presented. Section 3 describes the RDL concept and framework. Next, Section 4 presents the RDL specification, presenting the XML and KLV format representations; followed by the resource matching algorithm in Section 5. An application case study for the RDL language, using the Modulus middleware, is shown in Section 6. Conclusions are drawn in Section 7, being complemented with future work remarks.

2. Background and Related Work

Relevant background and related work focuses on research for the description of specific resources, mainly sensors, actors, and services. These research efforts mostly concentrate on concrete areas or applications and are used to define only a subset of items regarding resource description. The RDL differentiates itself by defining a broader resource scope, enabling therefore the production of multiple focused resource-based applications. In addition to the main related works mentioned, it is also important to refer research work accomplished on the topic of resource awareness (e.g., [6]) and network policy languages (e.g., [7, 8]). Both topics focus on the need for monitoring the level and quality of resources in the operating environment and on the need of being able to control them and to react upon changes, aiming for better network resource management and security.

Relatively to sensors, attention has been given to the description of their characteristics for interoperability, sharing, and discovery. Within this topic, we point out the Semantic Specification of Sensors [9], Sensor Web Enablement [10], and IEEE 1451 [3], whose summary follows.

The work by Compton et al. [9] reviews the state of the art of semantic specification of sensors. Semantic sensor networks use declarative descriptions of sensors to promote reuse and integration and to help solve the difficulties of installing, querying, and maintaining complex, heterogeneous sensor networks.

The Sensor Web Enablement (SWE) is the definition of web service interfaces and data encodings to make sensors discoverable, taskable, and accessible through the Internet, thus enabling a standardized communication and interaction with arbitrary types of sensors and sensor systems [10]. SWE makes use of Sensor Model Language (SensorML) [11] which provides models and encodings to describe any kind of process in sensor or postprocessing systems and metadata descriptions. SensorML has been applied in research, such as in the work of Aloisio et al. [12], to build an information structure to integrate sensor networks in grid environments.

The IEEE 1451 [3] is a set of standards that describe open, common, network-independent communication interfaces for connecting both sensors and actuators to other systems or entities. The IEEE 1451 depends on the definition of the Transducer Electronic Data Sheet (TEDS) for each transducer (sensor or actuator) which contains identification, calibration, correction data, and manufacturer-related information. This allows systems to automatically identify sensors and obtain their calibration and operating parameters in an opportunistic manner [13]. The IEEE 1451 standard is already being used for several sensor-related applications, such as for networking systems of intelligent vehicles [14].

Considering services, great importance goes to the Service Location Protocol (SLP), which is a scalable framework for the discovery and selection of network services formalized as an IETF standard [15, 16]. This protocol provides a dynamic configuration mechanism that allows applications to find and use services based on required attributes and not by name or address. The objective is that internet-enabled devices become less dependent on static and manual configuration of network services. This is extremely important as computers become more portable and networks larger and more pervasive, the need to automate the location and client configuration for network services also increases [17]. Considering mobile users, who frequently change network service environments, the SLP is essential in supporting user mobility [18].

3. RDL Framework Overview

By defining the Resource Description Language as a framework, we provide an abstraction that provides the general concepts, relationships, and functionalities that can be translated and specialized for specific systems. The RDL is designed to provide a flexible resource description framework, capable of not only describing the resources per se (e.g., printers, printing services, sensors), but also of contextualizing them as capabilities or requirements. Capabilities define which resources incorporate certain characteristics, whilst requirements define which necessary characteristics the resources need to possess. Requirements may also consider alternatives to the resources it is requesting, allowing a sort of requirement priority requests. A visual overview of the framework is presented in Figure 3.

The resources in the RDL are either scalar or service resources. Scalar resources represent the most fundamental aspects of the described objects, that is, describing what the objects are and characterizing them both qualitatively and quantitatively. These scalar resources describe such things as node hardware capabilities, as well as any specific metrics or state information provided by the network layers. Service resources describe coherent packages of scalar resources that export some sort of functionality. For example, whereas a network router can describe its queues and protocol capabilities as scalar resources, it can export its gateway capabilities as a service for its peers to route packets through it. In addition, it is also possible to specify additional constraints to characterize resource functionality.

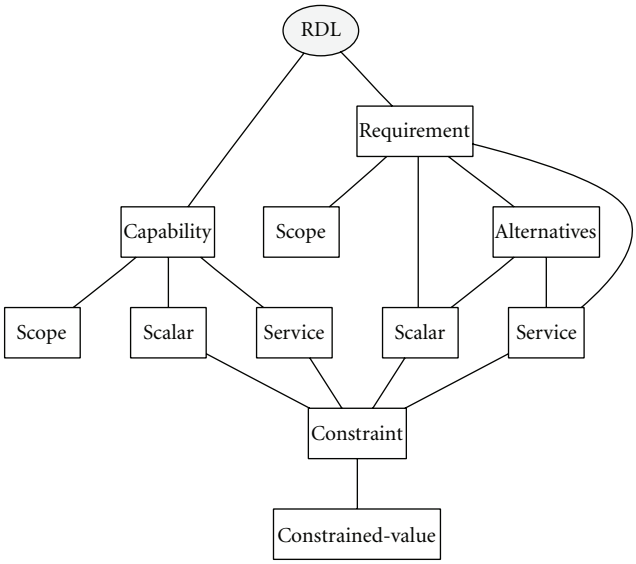

Figure 3: RDL framework diagram depicting main concepts and their relations.

Resources, whether they are scalar or service ones, belong to nodes in the network. Scalar resources that are, most likely, found in nodes in the network, describing what resources the node has. Service resources can be available either in a single node, a group of nodes, or an entire network. The RDL defines two scopes in order to define where the resources exist, or where they can be used. These scopes are the node-scope, where the scope indicates a single node, or a network-scope, where it indicates a group of nodes or a network.

4. RDL Format Specification

Given the heterogeneous nature of modern networks, interoperability issues come into play. As such, the RDL must be built upon standard formats with clear and unambiguous specifications, assuring that different nodes can successfully manage their resources cooperatively.

For the RDL specification, two formats are proposed, an official Extensible Markup Language (XML) [19] format and an alternative, compact binary representation, using a Key-Length-Value (KLV) [20] format. Whereas the XML format is suited for most applications, enabling the use of a vast library of XML-based tools and APIs that have already been developed, the KLV representation can be used in networked embedded systems and wireless sensor networks where the limited network characteristics and processing power as well as the need for energy efficiency advise against the use of a more computational demanding XML format.

The format specification concretely specifies the element structure of the concepts defined in the framework. The RDL describes resources as either capabilities or requirements. These resources, in turn, can be specified as one of two types: scalars or services. Additionally, scopes are used to define the context within which these resources are defined, as well as to limit the reach of the queries that look them up (e.g., a resource may be described as belonging to "node 1," and one may search for all of the resources that belong to "node 1" to find it). Constraints, on the other hand, are used to further define any limitations that apply to these resources by describing intrinsic properties (e.g., the printer is only capable of printing in black and white).

Having the intention of not limiting the expressiveness of the language, we acknowledge the need for it to accept extensions that further augment its reach and adaptability, while at the same time maintaining its logical structure.

4.1. XML Format. Following the general RDL element structure model, the official format describes capabilities, requirements, resources, and their scopes and constraints using XML elements and attributes. An example of a sensor node RDL specified in this format is illustrated on Algorithm 1.

The first lines need to show the XML declaration (version and encoding) and the schema which it obeys to. At the root, a top level "`<rdl>`" element is defined. Within this element, multiple capabilities and requirements can be described through the elements "`<capability>`" and "`<requirement>`", respectively. The capability or requirement element, in turn, contains two main sections, the scope description and the resource description. The scope description, specified within a "`<scope>`" subelement, describes the context within which the capability/requirement is needed or can be used. This element has a mandatory "`type`" attribute, indicating whether the scope describes a "`node`" or a "`network`," as well as any number of optional descriptive subelements.

The resource description section, within a capability or requirement element, is specified through the subelements "`<scalar>`" or "`<service>`". Each of these subelements is uniquely identified by an "`id`" attribute and described by a mandatory "`type`" attribute, that specifies what kind of resource they represent. Optional descriptive subelements can be used to further better describe these resources, while the "`used-resource`" subelements may be used to identify any additional resources used.

Finally, constraints are specified through "`constraint`" and "`constrained-value`" subelements. The "`constrained-value`" subelements indicate a limitation on the values a particular variable may take, through the use of "key"-"`operator`"-"`value`" attributes, specifying an inequality that applies to a property (e.g., pages < 20), according to the semantics specified in the optional "`operator`" attribute (i.e., the default values "`equals`," "`different`," "`greater`," "`less`," "`greater-equal`," "`less-equal`," "`contains`," "`does-not-contain`," "`is-contained`," "`is-not-contained`"). The constraint subelement clusters and groups multiple "constrained-values", performing a composition. The constraint composition follows different semantics whether it is being used in a capability or a requirement which will be visible when matching resources.

Using a mechanism similar to the one used to describe composite constraints, simple composite requirements can also be built. Using the additional "`<alternatives>`"

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE s1 PUBLIC "<location>/rdl.dtd" "rdl.dtd">
<RDL>
  <capability>
    <scope type="node">
      <nodeAddress>123456789</nodeAddress>
    </scope>
    <scalar id="5" type="module-sensor-light">
      <constraint>
        <constrained-value key="sensor-max-value" operator="equals" value="100"/>
        <constrained-value key="sensor-min-value" operator="equals" value="0"/>
      </constraint>
    </scalar>
    <scalar id="6" type="module-sensor-temperature">
      <constrained-value key="sensor-precision-ppm" operator="equals" value="1"/>
    </scalar>
  </capability>
</RDL>
```

Algorithm 1: RDL KLV description of a sensor node with two sensor modules (light and temperature).

subelement, further "`<scalar>`" or "`<service>`" subelements can be specified which consist of alternative requirements to the ones stated outside the "`<alternatives>`" element.

4.2. KLV Format. The XML-based format proposed gives a complete and human readable form of the description of resources. However, in some scenarios, such as WSNs, there is the need for a more compact representation, due to node limitations with respect to processing power and storage. As such, a KLV-based format is proposed. KLVs are tuples that associate a variable length value with a fixed size integer key identifier, while also explicitly specifying the size of the value. The new format can be built using a structure similar to the one used in the XML format, as illustrated through the example on Algorithm 2. The example is presented in a more understandable human-readable format, since the KLV format represents information in a binary form, not directly human readable, but optimized for machine interpretation.

The KLV format allows the creation of a hierarchal system by recursively encapsulating entire children KLV tuples within their parents value. The main advantage of this system is the fact that if a node is not familiar with a given key, while it will not be able to use its value, it will still be capable of skipping over it and using the remaining, otherwise usable, KLV tuples. Additionally, for a small set of mandatory key identifiers with fixed size values, the format can be further compacted by omitting the length value, as the nodes preexisting knowledge can be used to calculate it.

4.3. Performance Metrics. In this section, we compare XML and KLV descriptions in terms of memory to justify the existence of the two available formats. The XML format is human-readable and easily understandable but comes at a higher memory cost in comparison to the more lightweight and simple KLV format.

Comparing the RDL specification examples presented in Algorithms 1 and 2, the memory ratio between the XML and the KLV version is tenfold. For the same description, the XML version used around 546 bytes of memory whilst the KLV version used only around 55 bytes. This difference can be understood by the fact that the binary representation of the KLV is more compact, which although not directly human-readable has a greater memory advantage for machine processing and RDL message interchange.

Further experimentation on the memory difference between the two formats is depicted in Figure 4. The figure, presents four example cases (A, B, C, and D) which specify different RDL descriptions that increase in complexity. Case A describes a single capability with scope and a single scalar resource. Case B describes a single capability with two scalar resources with constraints associated. Case C describes three capabilities and three requirements, each with one scalar resource and constraints. Case D describes four capabilities and four requirements, with scalar resources, constraints, and requirement alternatives.

In Figure 4, the differences for the individual cases can be seen, as well as the general memory trend when increasing the complexity of RDL specifications. It is interesting to notice that, as the RDL specification increases in complexity, the XML format has a more marked slope increase in comparison with the KLV format, making XML growth surpass KLV's as more and more RDL resource descriptors are defined.

5. Resource Matching Algorithm

The RDL format is intended to ease the verification of the fulfillment of requirements with the available capabilities.

```
<capability> = {
  <scope> = {
    <type> = <node>;
    <node-address> = 123456789;
  }
  <scalar> = {
    <type> = <modulus-module-sensor-light>;
    <constraint> = {
      <constrained-value> = {
        <key> = <modulus-sensor-max-value>;
        <operator> = <equals>;
        <value> = '0 0 0 100';
      }
      <constrained-value> = {
        <key> = <modulus-sensor-min-value>;
        <operator> = <equals>;
        <value> = '0 0 0 0';
      }
    }
  }
  <scalar> = {
    <type> = <modulus-module-sensor-temperature>;
    <constrained-value> = {
      <key> = <modulus-sensor-precision-ppm>;
      <operator> = <equals>;
      <value> = '0 0 0 1';
    }
  }
}
```

Algorithm 2: RDL KLV description of a sensor node with two sensor modules (light and temperature).

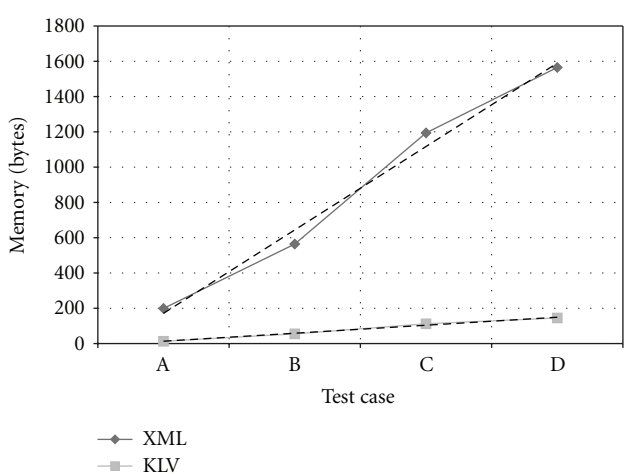

Figure 4: Memory comparison for the XML and KLV formats as RDL descriptions increase in complexity.

To verify such fulfillment, a resource matching algorithm was developed which matches the known requirements with capabilities, or vice versa. Pseudocode for the algorithm can be seen in Algorithm 3.

When matching resources, a match occurs when at least one requirement has one capability that satisfies it. In order to check if a requirement is provided by a capability (or if a capability satisfies a requirement), both the scope and the scalar/service resources need to be evaluated. Scopes must be compatible, and for all scalar and service resources, constraints must be checked. Considering the logic behind the resource matching and defining n as the number of requirements to match against m capabilities, complexity of the algorithm can be generally defined as $O(n \times m)$.

The algorithm executes and is implemented as follows. Within capabilities, all constraints and constrained-values are logically OR-ed at the resource description root-level and logically AND-ed within the constraint subelements, in other words, if any of the root conditions is met, the capability is considered to satisfy the requirement. On the other hand, all of the constrained values within a constraint must be met to satisfy the requirement. Requirements, in contrast, follow the opposite logic: constraints and constrained-values are AND-ed at the root level and OR-ed within subconstraints, that is, all root-level constraints must be met and subconstraints represent alternative choices.

Whereas capabilities can only be logically OR-ed at the root-level, that is, a requirement need only matches one capability to be satisfied, a requirement can describe a richer relationship. The idea is that, like in capabilities, any of the root-level requirements can be met to achieve a match between two RDL descriptors but, within each requirement

```
Input : Resource Descriptor (RDL-A), Resource Descriptor (RDL-B)
Output: True (match) or False (mismatch)
foreach Requirement in RequirementsList of RDL-1 Ado
    foreach Capability in CapabilitiesList of RDL-Bdo
        if Requirement.Scope matches Capability.Scopethen
            foreach ResourceR in ResourcesList of Requirementdo
                foreach ResourceC in ResourcesList of Capabilitydo
                    if (ResourceR.Type equals ResourceC.Type) and (ResourceR.Constraints
                    matches ResourceC.Constraints)then
                        return true;
                    end
                end
            end
        end
        return false;
    end
end
```

Algorithm 3: High-level pseudocode for the match between resources.

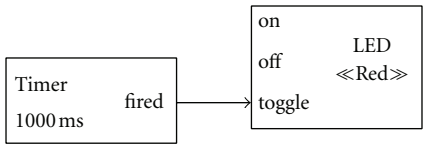

Figure 5: LED blinking modulus application diagram.

element, all scalar and service resources must be met (i.e., they are logically AND-ed). This description can be further enhanced through alternatives, containing additional scalar or service resources, allowing one to have a third level of depth, now following an OR-ed logic.

6. Application Scenario: Modulus-Modular Middleware Solution

In this section, to better exemplify how the RDL can be used in a real context and to show that the definition of the resources in a interchangeable format assumes a very important role, we present a modular middleware solution that we have been developing, called Modulus, that enables rapid application adaptation to available resources.

Modulus is a modular middleware solution for M2M systems and wireless sensor networks. The concept is loosely borrowed from the way systems are designed in electrical engineering. Electrical engineers do not go about reinventing the transistor every time they project a new amplifier. An assortment of prefabricated reusable components can be looked up in a catalogue and complex systems are built up by interconnecting these components in the right way. Modulus brings this sort of approach to the design and development of embedded software, focusing on rapid application development through intensive component level reusage. As before, large and complex applications are built from smaller building blocks, and the concept of application is redefined as being an interconnected set of simple reusable components.

More concretely, in Modulus, modules are instantiated, each instance sharing the same implementation, but keeping its own state and configuration. These modules communicate with each other in an abstract way using interfaces, allowing one component instance that uses a given interface to be wired to another that provides the same interface. With such a system in place, and with applications designed according to this paradigm, the runtime system is built upon a message passing framework that pushes data from module to module, according to the specified wiring.

One of the key advantages of Modulus lies in how it leverages the RDL to describe its applications. In a sense, Modulus applications do not connect predefined modules to each other, but rather connect their RDL descriptions. This small difference is a key factor for runtime adaptability and makes even more sense from the developer's point of view. Indeed, in an abstract sense, a module's implementation is not as important as the functionality that it exports. As long as the interface by which such functionality is provided is known to all parties involved, each module can be seen as a little black box. Furthermore, through the use of RDL constraints, a higher level of matching can be performed, by which wiring is not only constrained to interfaces of the same type, but also to other application level measures of compatibility or quality.

Figure 5 illustrates this concept in action. The LED blinking application is one of the simplest applications that can be implemented on an embedded system and is frequently used to quickly test tool-chains and build automation systems, much like Hello World applications are used on traditional computing systems. The application is simply built by connecting two components: a Timer and an LED. The LED component uses three interfaces of type Event; one to turn it on, one to turn it off and one to toggle

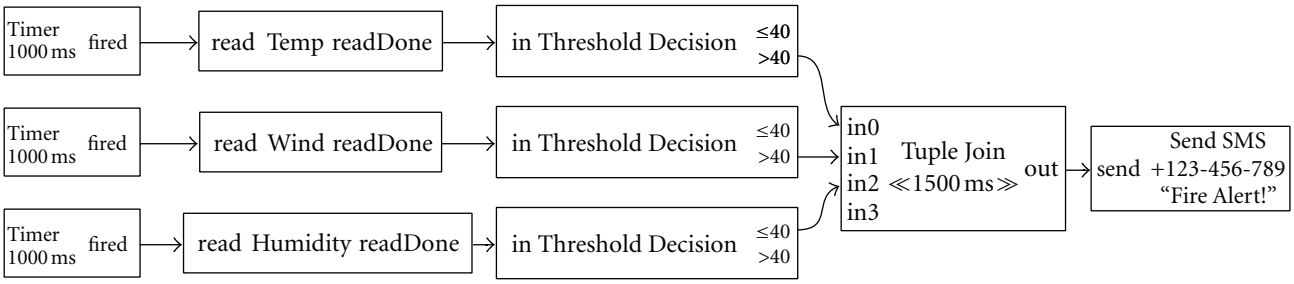

Figure 6: Fire alert modulus application diagram.

its current state. The Event interface is a generic interface used to signal that something happened. The Timer module provides this interface to periodically generate this signal at the configured rate and, by wiring these two components together, the periodic LED toggling is achieved, as expected.

Following this basic concept, the Modulus application is simply encoded as the component configurations (only the timer periodicity, in this case), and the interface wiring based on RDL descriptors. The entire application is described in KLV as shown in Algorithm 4. Furthermore, the inherent flexibility of the RDL encodes both static constraints such as the module specifications shown here (Timer and LED), as well as any additional application specific constraints, such as the LED color.

The power of this approach is even more clear in networked distributed systems. By transparently allowing wiring to traverse the network, Modulus allows functionality to be exported between nodes, enabling the development of complex distributed systems, using the same building blocks as before. Under these circumstances, application developers might not even have direct access to knowledge of which resources will be available in the deployed system. So long as an adequate description of the resource is given, the application will be adapted accordingly, at run-time.

A good example of this is the hypothetical application illustrated in Figure 6. In this example, it is determined that there is a higher risk of forest fires when all of the following conditions are met: the temperature is higher than 40°C, the wind speed is faster than 40 km/h, and the relative humidity is less than 40%. As such, park rangers want an application to automatically send an SMS to a predetermined phone number when such conditions are met.

To this end, each of the three sensors must be sampled every second and their values compared against the given threshold. If all of the preconditions are met within a certain time window (1500 ms in this case), then the SMS module is signaled and the message sent. What is interesting, however, is that such an application would work equally well whether all three sensors and the GSM module were all on the same system or in four networked nodes. Modulus abstracts away such details and transparently marshals the data to where it is needed.

From the user's perspective, the RDL can be used to provide a powerful insight into the networks built-in capabilities, allowing them to create better informed policies. From within Modulus, the additional information provided by the RDL can be used to enable automated application optimizations, or even to decide when it would be wise to migrate a particular application or functionality to another, more capable, node. Under these circumstances, the RDL can be a key element in enabling opportunistic resource sharing and creating more powerful, context-aware, embedded applications.

7. Conclusions

This paper describes the initial steps in the specification of the RDL. The main goal for the RDL is to define a reusable resource description specification for different algorithms, which can only be reached if the resources are described in a standard form. By separating the actual resources from the functionalities behind the algorithms that use them, new systems can take advantage of a pool of available resources. The RDL language is explicitly specified in the official XML format or the alternative KLV format. The KLV is a more memory compact representation as evaluation shows.

The RDL defines two layers of resources, that is, scalar resources and services. Scalar resources describe the objects and characterize them both qualitatively and quantitatively. Scalar resources are, for example, node hardware capabilities or packet error rate of a wireless link, and so forth. Services, on the other hand, describe packages of scalar resources that export some sort of functionality. For example, whereas a network router can describe its queues and protocol capabilities as scalar resources, it can export its gateway capabilities as a service, allowing its peers to route packets through it. Resources are bound to a scope, and there have been two scopes defined, that is, node and network scope. Resource matching is accomplished through an algorithm that analyzes resource descriptors and checks if their requirements and capabilities match.

In order to further develop the RDL framework, we also proposed a Java translation, which respects the language concepts and provides a core implementation for language usage. A Java applet tool was also developed to provide a means of interacting with the RDL language, allowing easy graphical creation of RDL descriptors as well as their generation to XML or KLV code.

To further illustrate the power of the RDL, we developed Modulus, a modular middleware solution for networked

```
<module-instance> = {
  <rdl-description> = {
    <requirement> = {
      <scalar> = {
        <type> = <modulus-module-timer-periodic-milli>;
      }
    }
  }
  <configuration> = 1000;
}
<module-instance> = {
  <rdl-description> = {
    <requirement> = {
      <scalar> = {
        <type> = <modulus-module-actuator-led>;
          <constrained-value> = {
            <key> = <modulus-module-actuator-led-color>;
            <operator> = <equals>;
            <value> = <red>;
          }
        }
      }
    }
  }
}
<wiring-data> = '0 0 0 0 1 2';
```

Algorithm 4: LED Blinking Modulus Application KLV Encoding.

embedded sensing systems. Modulus uses the RDL to describe software components and resources available on sensor nodes, allowing the development of applications to be decoupled from the actual resources they will use. This allows applications to automatically adapt, at runtime, to available resources, thus increasing the systems overall flexibility.

In short, this initial specification represents an initial effort to aid resource-based application development. Further improvements are planned for continuous development of the language. Main future work additions include

(i) further development of the specification to include more necessary properties and values that need to be defined to better describe resource nodes;

(ii) implementing additional functionality to the RDL Applet tool to fully respect the specification;

(iii) proposing the RDL as an IETF RFC.

Acknowledgments

Initial work on the description and specification of the RDL was supported by the Instituto de Telecomunicações, Lisbon, in the context of the EU-FP7 NEWCOM++ NOE [21] by M. Kuipers, L. D. Pedrosa, and R. M. Rocha. Further development took place at GEMS as part of L. D. Pedrosa and A. C. Santos' doctoral research. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the Instituto de Telecomunicações. This work was partially supported by national funds through *Fundação para a Ciência e a Tecnologia* (FCT), under project PEst-OE/EEI/LA0021/2011 and Doctoral Grant no. SFRH/BD/47409/2008.

References

[1] G. Wu, S. Talwar, K. Johnsson, N. Himayat, and K. D. Johnson, "M2M: from mobile to embedded internet," *IEEE Communications Magazine*, vol. 49, no. 4, pp. 36–43, 2011.

[2] J. Zander and O. Queseth, *Radio Resource Management for Wireless Networks*, Artech House, Norwood, Mass, USA, 2001.

[3] D. Wobschall, "IEEE 1451—a universal transducer protocol standard," in *Proceedings of the 42nd Annual IEEE AUTOTESTCON Conference*, pp. 359–363, September 2007.

[4] M. P. Bodlaender, "UPnP 1.1—designing for performance & compatibility," *IEEE Transactions on Consumer Electronics*, vol. 51, no. 1, pp. 69–75, 2005.

[5] H. Liu, B. Krishnamachari, and Q. Zhao, "Negotiating multichannel sensing and access in cognitive radio wireless networks," in *Proceedings of the 6th IEEE Annual Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks Workshops (SECON '09)*, pp. 1–6, June 2009.

[6] A. Acharya, M. Ranganathan, and J. H. Saltz, "Sumatra: a language for resource-aware mobile programs," in *Proceedings of the 2nd International Workshop on Mobile Object Systems—Towards the Programmable Internet (MOS '96)*, pp. 111–130, Springer, London, UK, 1997.

[7] N. Damianou, N. Dulay, E. Lupu, and M. Sloman, "The ponder policy specification language," in *Proceedings of the 9th*

IEEE Workshop on Policies for Distributed Systems and Networks (POLICY '01), pp. 18–38, London, UK, 2001.

[8] G. N. Stone, B. Lundy, and G. G. Xie, "Network policy languages: a survey and a new approach," *IEEE Network*, vol. 15, no. 1, pp. 10–21, 2001.

[9] M. Compton, C. Henson, H. Neuhaus, L. Lefort, and A. Sheth, "A survey of the semantic specification of sensors," in *Proceedings of the 2nd International Workshop on Semantic Sensor Networks at the 8th International Semantic Web Conference*, vol. 522, pp. 17–32, October 2009.

[10] A. Bröring, K. Janowicz, C. Stasch, and W. Kuhn, "Semantic challenges for sensor plug and play," in *Proceedings of the 9th International Symposium on Web and Wireless Geographical Information Systems (W2GIS '09)*, pp. 72–86, Springer, Berlin, 2009.

[11] M. Botts and A. Robin, "OpenGIS R Sensor Model Language (SensorML) Implementation Specification," OpenGIS Implementation Specification OGC 07-000, Open Geospatial Consortium Inc., Version: 1.0.0, July 2007.

[12] G. Aloisio, D. Conte, C. Elefante, G. P. Marra, G. Mastrantonio, and G. Quarta, "Globus monitoring and discovery service and sensorML for grid sensor networks," in *Proceedings of the 15th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE '06)*, pp. 201–206, June 2006.

[13] D. Wobschall, "Networked sensor monitoring using the universal IEEE 1451 standard," *IEEE Instrumentation and Measurement Magazine*, vol. 11, no. 2, pp. 18–22, 2008.

[14] K. C. Lee, M. H. Kim, S. Lee, and H. H. Lee, "IEEE 1451 based smart module for in-vehicle networking systems of intelligent vehicles," in *Proceedings of the The 29th Annual Conference of the IEEE Industrial Electronics Society*, pp. 1796–1801, November 2003.

[15] J. Veizades, E. Guttman, C. Perkins, and S. Kaplan, "Service Location Protocol," RFC 2165 (Proposed Standard). Updated by RFCs 2608, 2609, June 1997.

[16] E. Guttman, C. Perkins, J. Veizades, and M. Day, "Service Location Protocol, Version 2," RFC 2608 (Proposed Standard). Updated by RFC 3224, June 1999.

[17] E. Guttman, "Service location protocol: automatic discovery of IP network services," *IEEE Internet Computing*, vol. 3, no. 4, pp. 71–80, 1999.

[18] C. E. Perkins, "Service location protocol for mobile users," in *Proceedings of the 9th IEEE International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC '98)*, pp. 141–146, September 1998.

[19] T. Bray, J. Paoli, C. M. Sperberg-McQueen, E. Maler, and F. Yergeau, "Extensible Markup Language (XML) 1.0 (5th Edition)," W3C Recommendation, November 2008.

[20] "The KLV Standard—Data Encoding Protocol Using Key-Length Value," 2007.

[21] L. Galluccio, A. Leonardi, G. Morabito et al., "EU-FP7 NEWCOM++– WPR11: Opportunistic Networks—Intermediate Report on Resource Management Issues and Routing/Forwarding Schemes for Opportunistic Networks," Tech. Rep. Bilkent/KHAS, CNIT-CT, CNITBO, CNIT-PD, CNIT-TO, CNRS-LAAS, ISMB, IST-TUL, KAU, NKUA/IASA, PUT, UPC, RWTH, 2009.