A Model Driven Internet of Things

Till Riedel*, Dimitar Yordanov*, Nicolaie Fantana**, Markus Scholz*, Christian Decker*
*TecO, Karlsruhe Institute of Technology **ABB AG Corporate Research Germany

***Abstract*—The presented work proposes a model driven development approach employing flexible code generation strategies to overcome the technological gap between networked embedded objects and enterprise back-end system. We design a modeling and generator toolchain based on state of the art technologies to support efficient data exchange between hybrid resource constrained systems. We start by mapping different high level message description languages to a common meta model. From those models we generate visual pushdown automata that are the basis for our code generation and message encoding strategy. On the basis of this representation we present a practical and efficient approach to generate binary representation.**

I. Introduction

The term "*Internet of Things*" [1],[2] alludes to the Internet as we know it in two different aspects: the network connectivity and the resulting business impact. By bridging the last barrier of information technology, towards and between things, we hope to support business processes as they were not possible before. We are at a point where a huge number of networked things exist on a local scale. More than 98% of all microprocessors are part of an embedded system [3] most of which have some means of communication. Information from them is used at a very local scope and mostly between a managed set of homogeneous systems, while the biggest business opportunity lies in the interconnection and communication of such information sources [1]. Networking objects and devices is the real added value [4].

A. Coping with Diversity

To enable diverse interaction between all nodes in an internet a common language is needed. A system that can include wireless sensor networks and RFID technology would have to meet a variety of technical requirements. Specialized protocols for *Internet of Things* systems (including WSN and both passive and active RFID) address common resource constraints like size, energy, communication bandwidth and computation. Optimizing a system in this regard has proven rather complex because all factors are interlinked. (As an example, bigger batteries will relax energy constraints and thus give more options for higher bandwidth protocols and faster processors, but will have a negative effect on the size.) This means that changing a requirement slightly in one dimension might give a totally different degree of freedom in another.

This may explain the limited impact of standardization efforts on the inter-networking of things. Recently used standards for wireless personal area networks like 802.15.4 give tribute to application specific requirements by giving a number of options to choose from. However, to picture the consequence, one may consider the difference in the notion of using a 802.11a or a 802.15.4a compliant device. The first one will at least guarantee you network access on the PHY and MAC layer, the second one specifies six different physical layers that are supported by a MAC layer that can be driven in a number of different modes and configurations that all enable different applications. On top of this diverse set of layers a number of standards for the higher network layers like Zigbee(Pro), 6lowpan, Wireless HART and ISA 100.11a have recently emerged. Additionally a company that is serious about harvesting a maximum amount of data from an Internet of Things will likely have to deal with a diversity of data formats and protocols. This ranges from Web Services to Wireless LAN and a diversity of proprietary WSN Protocols to the EPC Gen2 (over the air) protocol in addition to commonly used industrial networks like Fieldbus, Profibus or DeviceNet.

Other sources of diversification are different programming models and tool support, different operating systems, concurrency models and programming languages leading to different collaboration and data exchange themes. The surge for highly efficient implementations often translates to even more protocols and data formats. The result is a world of things talking in a Babylonian number of languages. Well-intentioned integration efforts may even leads to an ever increasing number of different middleware products and customized gateway solutions that are added to a system landscape.

From the previous analysis follows that diversity is an enabling factor for an *Internet of Things* at a *local scope*. However, at a *global scope* diversity contradicts the vision of an *Internet of Things*, where few protocols should allow access to all worldly resources. As a way out of this dilemma we propose a paradigm shift from a *protocol centric* architecture towards a *data centric* architecture. This means the central specification of a system or application is always a description of the content of communication rather than the means. Optimally, the final technology mapping can be done independently, ideally automatically based on the specific requirements. This means that on a common abstract level every thing can talk with another. Such development approach takes into account different necessities when mapping the *Internet of Things* vision to an application and its efficient technological implementation. This can be done with a model-driven software development (MDSD) process [5] that starts from an abstract model and derives an implementation. Analog to human languages, we describe how to build sentences but neither limit what can be said nor how we communicate such as using spoken or written words via any available technical means. The goal is to allow communication across any technical barrier while obeying resource or contextual constraints. While we have motivated on an abstract level how a MDSD process can help to cope with hardware and system diversity, our prime motivation as application driven researchers and engineers is that we want to do things efficiently in practice. In the paper we show how a MDSD approach contributes to an efficient encoding of messages in a wireless sensing application and enables conversion into a verbose XML representation (i.e. document-based Web Services) for backend systems.

II. Related Work

Active messages [6], which are employed by the NesC language, are a different approach which however is by design coupled to a certain execution and platform model. Domain specific abstractions and ecodings for communication are common in wireless communication. uMiddle [7] or the CoBIs UPnP gateway architecture [8] try to overcome this by providing manual message mappings between systems. Another line of development during the last decades was focused on using XML Technology for interoperability of heterogeneous systems. Binary encoding, especially those based on XML Schema grammars like BiM [9], paved the way for efficiently using XML-equivalent encodings from DVB-T to networked embedded systems [10]. Those systems, however, rather focus on making XML usable in resource constrained devices rather than providing tools to map between structurally equivalent message representations. This can be provided by a MDSD approach. Using model driven techniques for message interchange is not entirely new. [11] describe a system based on Ada for heterogeneous military application. In recent years, however, much progress has been made in terms of standardization and advances of model driven software development tools and techniques. This paper combines those techniques with advances in the formal modeling and parsing of structured data formats such as XML and applies it to the domain of wireless sensing systems. It provides a well-defined models and meta-models for describing both data-structure of messages and the execution structure for de-/encoders and builds a code generator toolchain on top.

III. Model Driven Framework

Figure 1 outlines a model driven software development workflow we have implemented. As depicted in the vertical transformation process, we want to allow a number of textual and graphical domain specific notations to describe the contents of the communication. From those domain specific languages we generate a computational independent model (CIM), which serves as a common meta model for the code generation part. The code generation is based on a platform independent model (PIM). This model specifies how messages can be consumed by the application. We use an automata representation for this purpose. Abstract automata and type annotations (e.g. data type, minimum/maximum value) are used to generate code for the execution of the automata on the target platform. Because we can use full code generation we do not have to specifically define a platform specific model (PSM). On the horizontal axis of figure 1 we sketch a meta-model hierarchy that starts with the meta-meta model, which is the Essential Meta-Object Facility (EMOF), that defines separate meta-models for both communication messages and the acceptor automata (M2). They describe the abstract message content and the acceptor for a message format (M1). At the end there is a communication message instance and a communicating “thing”(M0).

With a common abstract representation of messages in EMOF we have build a foundation for defining a technology

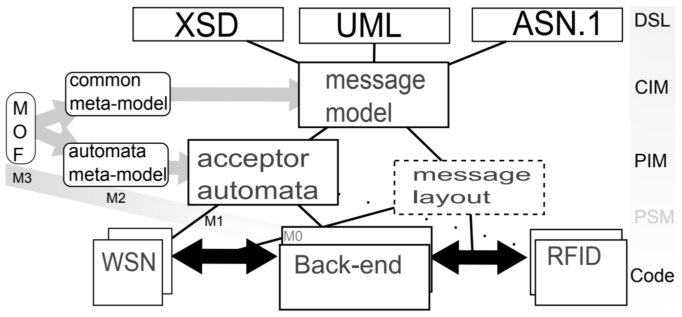

Fig. 1. Model driven development and message exchange scheme

mapping for target code generation. However, EMOF is a superset of context free grammars[12], which are traditionally difficult to handle especially when resources are limited. Therefore we have to restrict the model to support a subset of context free languages known as regular nested word languages [13]. (We specifically demand that all elements are reachable via attribute and containment relations). This class of languages can model many context-free aspects and prove to be as robust as regular word languages. Generally deterministic nested word automata have the same expressiveness as non-deterministic ones. In contrast to regular languages they can additionally represent both linear as well hierarchical relations between language elements. Furthermore an important aspect of modularity is given as the class of nested word languages, which is closed under intersection and union.

For implementing communication on a channel a linear encoding of messages (i.e. nested words) is important rather than the abstract representation. Any such encoding can be accepted by a special class of pushdown automata called visibly pushdown automata. It turns out that visibly pushdown languages are suited to model most structured document formats that are used for data exchange. Especially the expressiveness of schema-based XML can be represented. The visibly pushdown automata (VPA) are the basis to develop a model to text transformation that implement accepting and emitting messages. In a first step the structure of the automata can be directly translated to control flow. In a second step the specific encoding is implemented by specifying templates for accepting or emitting concrete encoding symbols based on the transitions in the automata. The runtime system thus consists of interconnected communicating automata that can be used to accept and translate any concrete encoding that was defined as a derivation of the abstract message model.

Automata are a good compromise between an abstract and formal representation of data formats and a low-level, imperative solution to describe a computer system. They can be easily constructed on the basis of the abstract grammar of the language they should accept and can be mapped efficiently to soft and hardware. Especially embedded designs are traditionally affine to automata because of their efficiency on non-parallel, pipeline-less MCU and their deterministic behavior. Finite state machines are especially well understood. The problem is that they are restricted to regular languages, which are generally not fit to describe many data models. Push-down automata provide an alternative for context-free grammars, but only in the non-deterministic form. The decision to restrict ourselves to nested word languages as outlined, gives us beyond other properties the possibility to use deterministic visibly pushdown automata. Those can provide the basis for code generation. Information exchange and encoding is done via the bisimulation of the acceptor automata: One automata accepts a linear encoding from the program and emits symbols that are transmitted and thus drive the remote automata.

We modeled the visual pushdown automata (VPA) as an Ecore meta model in accordance to [13]. It is not necessary to model the stack alphabet Γ explicitly. Γ and the alphabet of symbols Σ is defined implicitly by references and attributes of the original common data model that are *referenced* directly by the callTransition, localTransition and returnTransition. This allows to apply data type restrictions (such as integer type or min/max values) to the accepted symbols that are not captured by the VPA itself. The VPA is constructed using a model-to-model transformation from the original Ecore Model. The models are both based on EMOF and can be interchanged using XMI, which allows us to use various transformation frameworks. The current implementation uses plain java code using Eclipse Modeling framework (EMF) bindings to perform a depth-first search on the model. We transform the Ecore graph by performing a depth-first search on the graph starting at the root element of the message. We follow all containment references and attributes in the Ecore Model creating a callTransition when we progress down a reference link or a attribute relation and a returnTransition as we return. We further construct additional looping callTransitions for Ecore classes and attributes, that can have multiple occurrences, and epsilonTransition for elements, that can be skipped. Figure 2 shows a VPA for a simple sensor message . One can see that the call and return symbols match the references and attributes in the model.

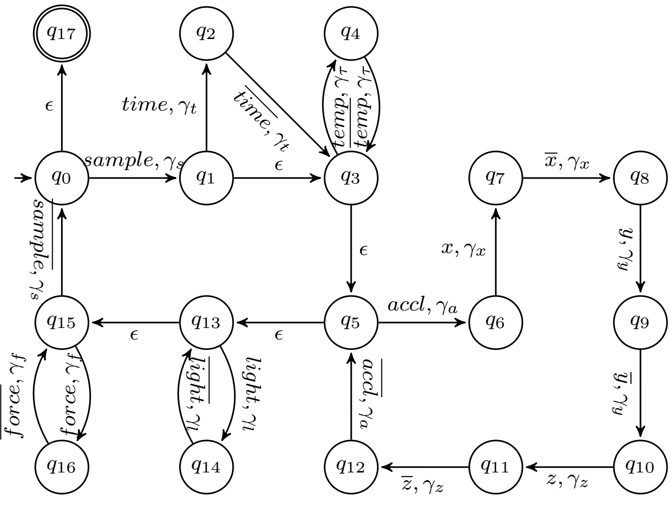

Fig. 2. VPA that accepts linear encoding of simple Data Model

Not all semantics of the message meta model can be captured by the VPA. In order to minimize the automata for code generation we currently create a loop for elements within the automata, if the maximum cardinality of a reference is bigger than 1. This means that for bounded relations the VPA currently accepts a superset of the data model. However, we do not lose this information for later code generation steps, as the original model is linked via references in all automata transitions. If it is e.g. necessary to do a runtime validation of messages this information can be exploited if needed. There are other cases, where additional information is of critical value to design optimal encodings of data. We e.g. exploit the data type (integer, timestamp, string, ...) information to select a value encoder/decoder that is called to consume/produce values (in contrast to pure structure). Because this scheme is extensible and only needs to be exploited for specific encoders, we can add various value restriction to the meta-model. Currently we support arbitrary minimum and maximum values as well as precision information, e.g. a minimal step of 0.1 points. We use annotations to reflect the restriction of real sensor values: a temperature value with a range from -20 to $80\,^\circ\mathrm{C}$ with a resolution of $0.1\,^\circ\mathrm{C}$ can be encoded in 10 bits.

A. Aspect-oriented Code Generation

From the given model the code generator should be able to produce efficient code for any number of platforms. This implies optimizing both code and data structure automatically under given requirements and goals. Theoretically this means that code generation has to be handled differently for each platform. While code generation always requires a one time effort per target, it eliminates the need for re-implementation for different message protocols. The advantages of removing such redundancies are manifold beyond saving routine work. Especially code maintenance can be done much better: possibilities for bugs are reduced and improvements propagate more quickly.

[14] shows how different software configurations can be efficiently generated using an aspect oriented code generation. We adopt this approach using the xPand template language to implement the code generator. All refinements are encoded as aspects of the basic automata control flow. Different input and output aspects can be added to the transitions of the pushdown automata via a configuration file. This allows the automatic creation of eclipse code generator plug-ins for any combination of model source and target platform. Compiler and framework support is offered by eclipse builders that can be customized via the plug-in work flow. Triggering code generation will automatically create or update jar or self-contained binary libraries, that can be linked into the target system. An encoding is accepted by reading symbols from an input. Input aspects implement this reading operation. Generally we implemented three different kinds of readers, a bitwise stream reader, a event reader and a depth-first tree reader for in-memory representations. The reader code generation is done locally for each automata state, code generation solely relies on information associated with the current automata transition. Output aspects are added in an analogous way to write either to a stream or to a structured output. This way the automata can perform transcoding between any number of formats, that can be constructed based on the abstract model.

IV. Efficient Binary Encoding

Choosing an encoding that fits our needs we exploited the fact that pushdown automata are a good means to compress data efficiently. We only transfer a minimum amount of data needed to encode the change in the acceptor automata.

Encoding [1]	XML	GZip	XMill	VTD	ConCom	**generated**
size (byte)	391	198	210	840	32	**20**
time (ms)	24.44	12.38	13.13	52.50	2.00	**1.25**
energy (μJ)	200	101	108	430	16	**10**

TABLE I
COMPARISON OF ENCODING PERFORMANCE
CALLING GZIP 1.3.12, XMILL 0.7, VTD-XML 2.4 WITHOUT ADDITIONAL ARGUMENTS

We further use the data type restriction connected with the automata state to choose an efficient encoding for a value. A binary 1 is emitted, if a path in the automata is taken and 0 otherwise. Transitions are ordered by the number of their target state. If only one choice is left no binary symbol is emitted. If an element contains a value it is encoded with the least possible bit size according to the minimum, maximum and stepping restrictions. All values are bit packed. Our current encoding resembles many parts of the BiM payload encoding for XML [15]. We, however, considerably extended the data type encodings exploiting existing and newly defined data type restrictions such as minimum and maximum as well as precision based value restrictions, that can reduce the payload even more drastically.

In order to confirm the efficiency of our binary encoding, we compared the resources needed to transmit an encoded packet. We compared the document produced by the generated encoder to a "naive" encoding like XML as well as packed XML documents using GZIP and the XML-aware XMill compressor. Furthermore we added a comparison to VTD-XML that provides support for effective processing of received data for embedded systems. The comparison in table I includes the size of the ConCom encoding, which is used by the native application convergence layer (ACL) of the Particle Computer [16]. The format uses binary encodings together with a human readable token format (ConCom tuples). While the results for the XML formats could only be gathered from simulation due to the lack of code and data memory for a target implementation, we could compare the generated target code to the ConCom code for the pPart 2/32. This platform is a typical resource constrained networked embedded system with a 5 MIPS pic18f6720 8bit MCU and a TR1001 869MHz radio. We also compared the implementations based on the resources needed on the target platform. We could confirm that the generated code added no overhead to the code size. Both programs use 40 kByte of the available 128 kByte of code memory. Concerning memory usage we could see a slight shift towards statically allocated memory in the program using the generated code, which needs 604 instead of 560 byte of RAM. At the same time the maximum amount of memory allocated dynamically on the stack during runtime was reduced from 118 to 80 byte in comparison to the manually optimized code.

V. CONCLUSION AND FUTURE WORK

In this work we proposed a model driven development approach to overcome the diversity of communication in an *Internet of Things*. We have shown how the principle steps of such an approach can be applied to the domain of message exchange between networked embedded system such as wireless sensor nodes and classical back-end systems. The work described in this paper shows how by using a model to model transformation we derive a abstract acceptor model. Based on visibly pushdown automata that are the basis for code generation we can build resource efficient encoders and decoders. Furthermore we can automatically translate between different encoding schemes that can be freely chosen by domain specific requirements.

We designed a model driven development framework using this approach that generates a common model representation of data formats from diverse sources, which we will report on in future papers. Based on this code generation framework we have already implemented a gateway system, which provides a highly efficient communication interface to arbitrary wireless sensor network services via a Web Service interface. We are also working on different cross-layer optimization based on this MDSD toolchain. Using the proposed scheme we can provide both optimization and better platform integration while retaining a top-down domain specific view on message exchange. We believe that this a key aspect in developing future proof wireless sensor applications and to accelerate a real *Internet of Things* in general.

VI. ACKNOWLEDGMENT

This work was partially funded by the German Federal Ministry of Education and Research (BMBF) as part of the Aletheia project.

REFERENCES

[1] I. T. Union, "The internet of things." *ITU Internet Reports*, 2005.

[2] N. Gershenfeld, R. Krikorian, and D. Cohen, "The internet of things." *Scientific American*, vol. 291, no. 4, p. 7681, 2004.

[3] J. L. Turley, *The essential guide to semiconductors*. Prentice Hall PTR, 2003.

[4] J. Buckley, "From RFID to the internet of things: Pervasive networked systems," in *Final Report*. Brussels: CCAB, 2006.

[5] T. Stahl, M. Voelter, and K. Czarnecki, *Model-Driven Software Development: Technology, Engineering, Management*. John Wiley \& Sons, 2006. [Online]. Available: http://portal.acm.org/citation.cfm?id=1196766

[6] T. V. Eicken, D. E. Culler, S. C. Goldstein, and K. E. Schauser, "Active messages: a mechanism for integrated communication and computation," *ACM SIGARCH Computer Architecture News*, vol. 20, no. 2, p. 256266, 1992.

[7] J. Nakazawa, H. Tokuda, W. K. Edwards, and U. Ramachandran, "A bridging framework for universal interoperability in pervasive systems," in *Proceedings of the 26th IEEE International Conference on Distributed Computing Systems*, 2006, p. 3.

[8] T. Riedel, C. Decker, P. Scholl, A. Krohn, and M. Beigl, "Architecture for collaborative business items," *Lecture Notes in Computer Science*, vol. 4415, p. 142, 2007.

[9] U. Niedermeier, J. Heuer, A. Hutter, W. Stechele, and A. Kaup, "An MPEG-7 tool for compression and streaming of XML data," in *IEEE International Conference on Multimedia and Expo*, 2002, p. 521524.

[10] C. Werner, C. Buschmann, and S. Fischer, "WSDL-driven SOAP compression," *International Journal of Web Services Research*, vol. 2, no. 1, p. 1835, 2005.

[11] C. Plinta, R. D'Ippolito, and R. V. Scoy, "A specification and code generation tool for message translation and validation," in *Proceedings of the 1998 annual ACM SIGAda international conference on Ada*. Washington, D.C., United States: ACM, 1998, pp. 276–286. [Online]. Available: http://portal.acm.org/citation.cfm?id=289649

[12] M. Alanen and I. Porres, "A relation between Context-Free grammars and meta object facility metamodels," Mar. 2003.

[13] R. Alur and P. Madhusudan, "Adding nesting structure to words," 2009.

[14] M. Voelter and I. Groher, "Product line implementation using Aspect-Oriented and Model-Driven software development," in *Software Product Line Conference, 2007. SPLC 2007. 11th International*, 2007, pp. 233–242.

[15] J. Heuer, C. Thienot, and M. Wollborn, "Binary format," *Introduction to MPEG-7: multimedia content description interface*, p. 61, 2002.

[16] C. Decker, A. Krohn, M. Beigl, and T. Zimmer, "The particle computer system," *Proceedings of the 4th international symposium on Information processing in sensor networks*, 2005.