A model-driven concept for the automatic integration of legacy protocols to distributed component-oriented software systems

Ralf Vandenhouten, Thomas Kistel

Zusammenfassung

Die Implementierung von Kommunikationsprotokollen zu externen Systemen ist eine wichtige Aufgabe, welche häufig in Softwareprojekten zu realisieren ist. Dieser Artikel ist ein Ideenpapier, welches die Komponenten bisheriger Implementierungsstrategien und deren Probleme beschreibt. Der Artikel führt die wesentlichen Techniken zur Protokollimplementierung ein und stellt diese im Kontext der modelgetriebenen Softwareentwicklung dar. Zum Schluss wird eine Methode mit ASN.1 und SDL vorgestellt, welche die automatische Generierung von Manager-Schnittstellen für die Protokolle von Geräte-Agenten ermöglicht und dabei den Einsatz in verteilten komponentenbasierten Systemen erlaubt.

Der Artikel beschreibt die Zwischenergebnisse des vom Bundesministerium für Bildung und Forschung (BMBF) geförderten MOSES-Projektes (Modellgetriebene Software-Entwicklung von vernetzten Embedded Systems). In dem Projekt konzentrieren sich die Autoren auf die Bereiche Medizin- und Gebäudetelematik, worin sie besondere Erfahrungen besitzen.

Abstract

The implementation of communication protocols is an important development task that appears frequently in software projects. This article is a vision paper that describes the components of the currently available implementation strategies and problems that arise. The article introduces the main existing protocol engineering techniques and puts them into the context of model driven software development. At the end a methodology is introduced for the automatic generation of manager interfaces of Device Agent protocols for the use in a distributed component oriented environment, using ASN.1 and SDL.

This article describes the preliminary results of the MOSES project (model-driven software engineering of distributed embedded systems) which is funded by the German ministry of Education and Research (BMBF). In this project the authors are concentrating on medical and facility management areas where they have particular experiences.

1 Introduction

The communication between systems is a frequent requirement of development projects. In different vertical industry sectors the interconnection to different external devices (Device Agents) is often requested. These Device Agents have more or less proprietary communication protocols and are sometimes rather old. An example in the hardware area is the serial connector standard RS232 (EIA 232) which was originally developed in the 1960s and is still widely used in many industry systems, even though successor technologies like USB or FireWire (IEEE 1394) have technical advantages. This situation is also applicable to the software application protocols (Legacy Protocols) that are used. There are various reasons for the long life-cycle of communication systems in the industry sector. Some of these are:

- The systems need to be compatible with older systems.
- The product life-cycle of industry systems is relatively long, e. g. an installed fire alarm system in an office building cannot be easily replaced.
- The hardware environment of the embedded systems has limitations for the support of newer protocol stacks.
- The development of a protocol extension (i.e. a new feature) is sometimes much cheaper than introducing a new protocol.

There are two reasons that custom protocols will continue to be used. The first is that manufacturers produce specific protocols for their applications and the second

is that specific protocols are often technically advantageous.

On the other hand it is necessary to effectively develop connectors to Device Agents in a cost-effective way and to integrate them into existing distributed systems. Also there exists much research about the automatic protocol definition and implementation, but they are less or non-integrable in the context of distributed component-oriented software development. In this article we introduce an approach for the model-driven development of manager software connectors to external Device Agents. Therefore we first give a short overview on the history of protocol engineering and highlight the concepts that are useful for a model-driven approach. In section 3 we examine how current developments in the MDD area have to be appraised in our context and draw an outlook in section 4 how these concepts can be adapted to component oriented software engineering.

2 Concepts of Protocol Engineering

The area of protocol engineering has been extensively researched. The first efforts of Protocol Engineering were already done in the late 1960ies and '70ies, so Protocol Engineering looks back at a relatively long history. The article of Bochmann (Bochmann et al. 2010: 3197–3209) provides a good overview of the history of Protocol Engineering. It summarizes the different development stages of the current protocol stacks and formalism techniques that were successfully introduced and those that were not widely accepted. In this section we give a short summary of this development and outline important aspects for the current development in this area.

The first link protocols that were developed in the late 1960ies were redesigned at the beginning of the '70ies. In this redesign bit-oriented framing and sequence numbering were introduced. This work concludes with the development of the X.25 protocol. Another aspect of several research projects at this time was packet switching which resulted in the development of the well-known ARPANET, which were later adopted for the TCP/IP protocol. The development of many application protocols during the '70ies leads to the initiative of harmonizing the interworking between such systems. This initiative was the OSI standardization project at ISO, which was started in 1977. One of the main outcome of the OSI project was the layered architecture design of protocol descriptions.

The most important OSI-Layer for Device-Agent protocol implementation is the application layer 7, because most external devices describes their communication protocol on that layer and rely on specific lower layer protocols (e. g. IP, Serial). Generally a communication protocol can be seen as a set of digital data that is transferred on different states. As of (Grimm 2009) digital data are constituted by the following parameters:

1. Logical structure
2. Raw data
3. Presentation
4. Digital Encoding

For a formal description of these parameters the Abstract Syntax Notation One (ASN.1) was developed. ASN.1 was developed during the OSI standardization process for the description of the OSI protocols and allows the definition of primitive types and data structures of application layer protocols. A key advantage of ASN.1 is that it not only describes the logical structure, raw data and presentation of protocol messages, but also the encoding rules for transforming the messages into transferable byte data. ASN.1 describes a set of common encoding rules which are used for different purposes. The most frequently used examples of these encoding rules are:

- BER – The Basic Encoding Rules uses the TLV (Type, Length, Value) approach to encode the data. An important advantage of Basic Encoding Rules is that they allow extension and different versions of protocols, as each protocol item is identified by its type and length. For example Google Protocol Buffers, that we introduce later, uses the TLV approach to encode the data and to guarantee protocol compatibility. This advantage comes at the cost of an higher overhead required by the Encoding rules. Therefore the PER were introduced.
- DER – The Distinguished Encoding Rules together with CER (Canonical Encoding Rules) are basically the same as BER with some restrictions on the encoding. DER is mainly used in cryptography such as for the encoding of the X.509 Certificates.
- PER – Packed Encoding Rules were developed for producing compact transfer syntax. They are mainly used in the mobile communications area.
- XER – XML Encoding Rules produce XML output for the specified data.

The encoding rules BER, DER, CER are the standard encoding rules of ASN.1, PER and XER are additional encoding rules. However, other encoding rules for ASN.1

can be defined by using the Encoding Control Notation (ECN). ECN is particular useful to describe the Encoding Rules of legacy protocols. ASN.1 and their Encoding Rules are supported by many software tools of different manufacturers. A list of ASN.1 tools (mostly commercial) can be found on the ITU-T website (ITU 2011).

Another important ITU standard for protocol engineering is the Specification and Description Language (SDL). It allows the specification of the behavior of external systems and can be used together with ASN.1. Both, ASN.1 and SDL, have gained much acceptance in the telecommunications sector.

3 Model Driven Technologies

Besides the formal protocol engineering techniques, mainly driven by the telecommunications sector, which led to the development of standards like ASN.1, SDL and others, there exist other technologies that allow an easy development of individual protocol implementations. Those technologies enable the efficient encoding and decoding of structured data into a specific format that can be used to implement a protocol (e. g. for a client-server application). Two popular examples are Google Protocol Buffers (Google 2011b) and Apache Thrift (Apache 2011). These »message definition languages« can also be used for message generation in an RPC environment. They allow the abstract definition of protocol messages (PDUs) and the automatic generation of executable programming code (e. g. Java, C#, C++). Both technologies are widely used. Google Protocol Buffers is used for many internal Google Web protocols and files, the Thrift project that is now hosted by Apache was formerly developed by Social Network Provider Facebook.

In this article we want to compare different implementation aspects of ASN.1 tools and other technologies like Google Protocol Buffers. Therefore we created a »simple Employee example« entity. The Employee entity can be used by different applications (e. g. a web application for managing the vacation accounts, a time logging system) and their respective protocols. The Employee entity has the following attributes:

1. Employee number
2. Title and name
3. Date of hire
4. Name of spouse
5. List of children

```
Employee DEFINITIONS ::= BEGINEXPORTS;

PersonnelRecord ::= [APPLICATION 0] IMPLICIT SET {
   name          Name,
   title         [0]IA5String,
   number        EmployeeNumber,
   dateOfHire    [1]Date,
   nameOfSpouse  [2]Name,
   children      [3]IMPLICIT SEQUENCE OF ChildInformation
}

ChildInformation ::= SET {
   name          Name,
   dateOfBirth   [0] Date
}

Name ::= [APPLICATION 1] IMPLICIT SEQUENCE {
   givenName     IA5String,
   initial       IA5String,
   familyName    IA5String
}

EmployeeNumber ::= [APPLICATION 2] IMPLICIT INTEGER

Date ::= IA5String

END
```

Listing 1: Description of the data structure in ASN.1

The listings show the defined data structure of this Employee entity a) with an ASN.1 tool (Listing 1) that was taken from Objective Systems and b) with Google Protocol Buffers (Listing 2).

```
package employee;

message PersonnelRecord {
   optional Name                name = 1;
   optional string              title = 2;
   optional EmployeeNumber      number = 3;
   optional Date                dateOfHire = 4;
   optional Name                nameOfSpouse = 5;
   repeated ChildInformation    children = 6;
}

message ChildInformation {
   optional Name                name = 1;
   optional Date                dateOfBirth = 2;
}

message Name {
   optional string              givenName = 1;
   optional string              initial      = 2;
   optional string              familyName = 3;
}

message EmployeeNumber {
   optional int64               value = 1;
}

message Date {
   optional string              value = 1;
}
```

Listing 2: Description of the data structure with Google Protocol Buffers

The listings show the definition of a `PersonnelRecord` that contains different subtypes like `EmployeeNumber`, `Name` or `ChildInformation`. The corresponding tools allow the generation of Java source code from these abstract message definition.

The figures below show the class diagram of the generated source code in Java.

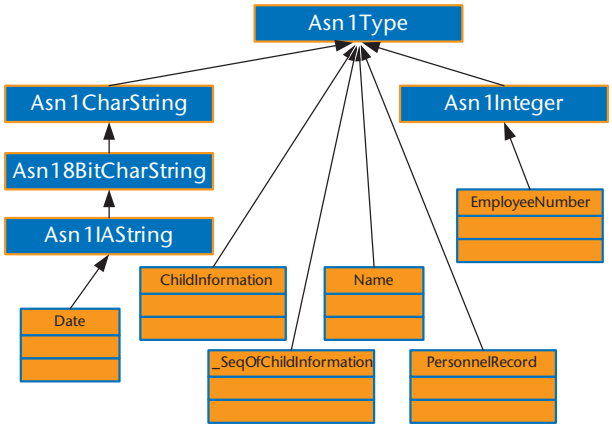

Fig. 1: Description of the data structure in ASN.1

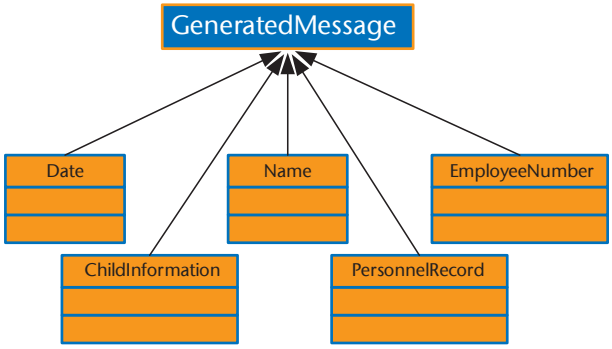

Fig. 2: Description of the data structure with Google Protocol Buffers

Aside from the encoding aspects, Figure 1 and Figure 2 show that the class structures of the generated classes are similar. The classes are derived from super classes which provide basic functions for encoding and decoding the messages. These super classes (blue color in the class diagram) are part of a Jar-Library of the corresponding generation tool. The polymorphism here is a fairly strict coupling of object oriented data representation and their encoding/decoding. This coupling makes the classes less flexible and their reuse as data objects rather limited. To use these classes in the business logic, adapter classes or other code generation techniques are required. A better reuse can be reached by generating POJO-classes. A POJO is an acronym for Plain Old Java Object and is a class that does not have dependencies to external libraries or other conventions. This is a key issue which is not addressed properly in currently available tools.

In the next section we want to highlight some aspects that are relevant in component-oriented environments.

4 Component-Oriented Development

Beyond object-oriented design patterns (Gamma et al. 2009) Component Systems have gained much more acceptance in the software industry in the past decade. Many of these component-oriented concepts are explained in (Szyperski 1999). One pioneer in this area is the Java Component Framework OSGi which is standardized by the OSGi Alliance (OSGi Alliance 2011). OSGi specifies techniques to separate software modules into different bundles that communicate through defined services. Each bundle has its own lifecycle which makes OSGi systems very flexible. The defined services can be accessed locally or from a remote system (Vandenhouten et al. 2009: 142-146).

Another important paradigm is the »Inversion of Control« (IoC) by Dependency Injection (DI) design principle. Dependency Injection allows for the separation of object definition and its creation/wiring at runtime. In the context of OSGi different DI-Frameworks like Spring (Spring Source 2011), Google Guice (Google 2011a) or OSGi Declarative Services are used.

These technologies have led to more flexible, scalable and better unit-testable software systems. However, this flexibility, especially the lifecycle behavior of OSGi Bundles and Services, comes at the cost of more complex system integrity. This means that although the single components are better unit-testable, the whole system has more integrity states and is more difficult to test. In the past years these concerns have been addressed by research projects in the modeling area. SDL plays an important role, as it provides methods to model state behavior. However the automated generation of component oriented application code for distributed systems from model description is an open challenge task. In the context of the automated generation of protocol implementations, as introduced in section 1, the research task can be divided into three main categories:

1. Define a way for the description of data structures and encoding of application layer protocols. ASN.1 is important as it exactly provides methods for this; however, it does not make any assumptions about the structure (such as patterns in Gamma et al. 2009) of the generated code since it generally depends on the implementation of the ASN.1 compiler. These assumptions and transformations must be defined.
2. Describe the communication behavior of the external Device Agent systems using finite state machine models. This can mainly be done with SDL and UML.

A research question concerns modeling the business aspects of the protocol. This means that the real business application that communicates with the device does not need to know all internal aspects and states that the protocol defines. A method to hide these aspects must be developed.

3. Integrate the generated application code into component systems so that it can be accessed via defined services. To achieve this, OSGi provides several features (e. g. bundling and service definition). The result should be a Device Bundle that exposes its business logic via Services and Connectors. For example, the Generation of Software Connectors is described in (Bures et al. 2008: 138-147). Another aspect is the use of those Components in a distributed environment. For these implementations restrictions are required in order to deploy these Components in a distributed environment.

To realize these tasks, it is not generally necessary to completely re-implement existing ASN.1- or SDL-Tools, but to provide extensions for the model transformation and code generation.

5 Example Implementation

An example how this methodology can be applied, is the implementation of a manager device for an IEEE-11073 compatible agent device. The ISO/IEEE 11073 is a family of standards for medical device communication. Two important standards are described in (Health Informatics 2004a) (Domain Information Model – i.e. the data model of the IEEE-11073 standard) and (Health Informatics 2004b) (Application profile and Communication). In the following we will put the implementation of an IEEE-11073 manager device in the context of the three main tasks listed above. Step 1 is to describe the Domain Information Model of this standard with ASN.1. These data description can also be used to generate business logic objects. These business objects should be POJO's if possible. In Step 2 the communication behavior of the IEEE 11073 standard has to be modeled. The Communication Model of this standard defines the sub-protocols ACSE (Association Control Service Element), CMDISE (Common Medical Device Information Service Element and extension of CMISE – Common Management Information Service Element) and ROSE (Remote Operation Service Element) that are all OSI protocols.

The communication behavior of the IEEE-11073 is an aggregation of these sub-protocols. Also these sub-protocols can be hided to the business application layer. Business functions are exposed via Services in a Component System (Step 3). Communication errors whether or not they internally rely on ACSE, CMDISE, ROSE or lower layer protocols, are exposed via Service errors that can result in a service deregistration from the component system.

Figure 3 shows the modules of the proposed solution. Here the protocol logic is modeled with existing ASN.1 and SDL tools. An upper Service layer will integrate this protocol logic into the Business logic and Middleware. The Service Layer also maps the Business logic objects to the protocol objects and ensures the interaction.

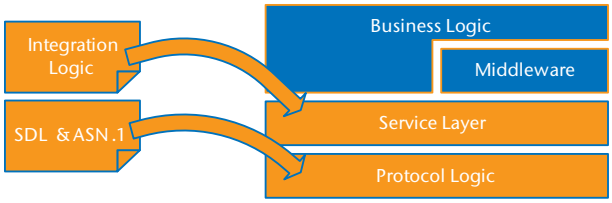

Fig. 3: Modules of the proposed solutions

The composition of these steps provides a more effective methodology for the implementation of the relatively complex IEEE-11073 standard. The availability of better modeling tools, especially in the Eclipse area (Eclipse Foundation 2011), provides a good basis to achieve this. For example, this enables the development of domain specific languages (DSL) that allow for additional descriptions of transformation procedures etc.

6 Conclusion

The development tools that are available do not allow for the automated development of software connectors to industrial systems (Device Agents). In this article we introduced a basic concept to refine the development on the basis of standards that are widely used in the telecommunications sector. The goal of these research efforts is a more simplified and natural development process for engineering the protocol implementation of manager connectors to external Device Agents.

Bibliography

Apache (2011): Apache Thrift Project Website, http://thrift.apache.org, cited: 31.05.2011.

Bochmann, G.v., Rayner, D., West, C.H. (2010): Some notes on the history of protocol engineering. In: Comput. Netw 54, 3197–3209.

Bures, T., Malohlava, M., Hnetynka, P. (2008): Using DSL for Automatic Generation of Software Connectors. In: Composition-Based Software Systems, 2008. ICCBSS 2008. Seventh International Conference on Composition-Based Software Systems, 138-147.

Eclipse Foundation (2011): Eclipse Project Website, http://www.eclipse.org, cited: 31.05.2011.

Gamma, E., Riehle, D. (2009): Entwurfsmuster. Elemente wiederverwendbarer objektorientierter Software. Addison Wesley, München.

Google (2011a): Google Guice Project Website, http://code.google.com/p/google-guice, cited: 31.05.2011.

Google (2011b): Google Protocol Buffers Project Website, http://code.google.com/p/protobuf, cited: 31.05.2011.

Grimm, R. (2009): Digitale Kommunikation. Oldenbourg Verlag, München, Wien.

Health Informatics (2004a): Health Informatics –Point-of-Care Medical Device Communication –Part 10201: Domain Information Model (2004). In: ISO/IEEE 11073-10201:2004(E).

Health Informatics (2004b): Health Informatics –Point-of-Care Medical Device Communication –Part 20101: Application Profile –Base Standard (2004). In: ISO/IEEE 11073-20101:2004(E).

ITU (2011): ITU ASN.1 Tools, http://www.itu.int/ITU-T/asn1/links/index.htm, cited: 31.05.2011.

OSGi Alliance (2011): Official Website, http://www.osgi.org, cited: 31.05.2011.

Spring Source (2011): Spring Framework Website, http://www.springsource.org, cited: 31.05.2011.

Szyperski, C. (1999): Component software. Beyond objetc oriented programming. Addison Wesley, Harlow, England.

Vandenhouten, R., Kistel, T., (2009): Aus der Entfernung. Verteilte Dienste mit R-OSGi. In: iX Magazin für professionelle Informationstechnik (12), 142–146.

Authors

Prof. Dr. rer. nat. Ralf Vandenhouten
Fachgebiet Telematik
Fachbereich Ingenieurwesen/Wirtschaftsingenieurwesen
Technische Hochschule Wildau [FH]
T +49 3375 508-359
ralf.vandenhouten@th-wildau.de

Thomas Kistel, M. Eng.
Fachgebiet Telematik
Fachbereich Ingenieurwesen/Wirtschaftsingenieurwesen
Technische Hochschule Wildau [FH]
T +49 3375 508-615
thomas.kistel@th-wildau.de