Reusable and flexible design of communication gateways

Ingmar Kühl
pdv Technische Automation + Systeme GmbH
Dorotheenstrasse 64
22301 Hamburg, Germany
kuehl@pdv-tas.de

Alexander Fay
Helmut-Schmidt-Universität
Holstenhofweg 85
22043 Hamburg, Germany
alexander.fay@hsu-hh.de

Abstract
Communication gateways have an indispensable role in the architecture of distributed automation systems. Numerous communication gateways have been implemented to address particular devices, data exchange requirements, programming languages, and communication protocols, and further will have to be implemented in the future. Typically, these communication gateways have been implemented from scratch; with little reuse being made, in terms of software reuse and knowledge reuse as well.

In this paper, an approach is described to design communication gateways in a way to maximize the reuse even across different implementation technologies. This approach allows adapting communication gateways (instead of reprogramming them) in case of increasing requirements during operation.

1. Introduction: communication gateways

Within automation systems, communication gateways fulfill the need to transform data at the interface of different communication technologies, i.e. at the interface between different communication protocols, or at the hardware interface of different vendors. The term “gateway” can be defined as follows [4]:

“Communication requires very precise, rigidly defined rules for successful communication. Even slight variations can render communication impossible. In order for two computers using different protocols to communicate, some kind of translation must take place. The device that performs this translation is called a gateway.”

As given by Tanenbaum [46], a gateway does interconnect different networks or different network segments. From this perspective, the gateway implements the mapping of two semantically different models of an application domain, i.e. different communication protocols.

In SCADA (supervisory control and data acquisition) systems and DCS (distributed control systems), data from the field is usually transmitted via marshalling terminals and RTUs (remote terminal units) to a gateway, where it is transformed into a different format for further distribution and data processing.

1.1. State of the art

Usually, communication gateways are specific for a certain pair of communication protocols, being implemented in a certain programming language and running on a dedicated hardware and operating system. Therefore, numerous communication gateways are required to address the particular requirements.

Although the basic task of a communication gateway has always been the same, most gateways have been developed from scratch, and most of them contain a hard-coded implementation of the algorithm which translates the communication protocol commands, i.e. the communication protocols which are defining the domain model are normally not defined at a semantic level. Thus, a change in a communication protocol requires a change in the source code of the gateway.

State of the art to implement communication gateways, taking into account the need to adapt the gateway software to changes in communication protocols, is to implement a gateway using imperative or object-oriented programming languages and the use of external configuration files. In these configuration files, parameters are defined which are taken into account by the gateway software during configuration, i.e. at start-up. A change in the configuration file of a communication gateway requires a re-start of the communication gateway to become valid, i.e. the normal operation has to be interrupted. A change of the data model requires a restart as well. This is a deficiency which is no longer acceptable for users, see e.g. [16].
1.2. The challenge
   Ideally, it should be possible to modify the application domain model on either side of the gateway, to redefine the mapping between the domain models, and to automatically reflect these changes in the communication gateway without reprogramming and without even interrupting the communication gateway. This would maximize the possible reuse of a communication gateway. But further, the whole software architecture of a communication gateway has to be designed for reuse, i.e. a flexible architecture is needed that take future technology changes into account and doesn’t focus on specific technology.

1.3. Overview
   In the following chapter, various means to foster reuse in software development in general and in the development of communication gateways in particular are discussed. Additionally, research activity for flexible communication systems is reflected. In chapter 3, the authors present a programming approach to design communication gateways, which is inspired by the object model of Common Lisp: the data abstraction is clearly separated from the information flow. Furthermore, the modeled data artifacts are given in terms of an expert system. This approach is elaborated in more detail in chapter 4. Chapter 5 gives a conclusion, whereas chapter 6 presents need for future work.

2. Development technologies
   In this chapter different technologies are reflected which define the state-of-the-art in software engineering for communication gateways.

2.1. Object-oriented design
   Several communication gateways, such as some based on a technology like OPC (Openness, Productivity and Collaboration) DA (Data Access), have been implemented using the object-oriented programming paradigm. According to [12], “designing object-oriented software is hard, and designing reusable object-oriented software is even harder”. Many attempts to increase the software reuse, which is defined in [36] as the “process of implementing and updating software systems using existing software assets”, have not resulted in the desired effect [44]. As pointed out by Griss [14, 15], software must especially be designed for reuse to be able to benefit from the advantages of reuse later.

   Gabriel has pointed out that developing software just and only with the object-oriented programming paradigm has produced poor software reuse [10, 11]. To come closer to a systematic software reuse, he proposes developing software by combining various programming paradigms. This doesn’t exclude the object-oriented programming paradigm. In addition, this does contain paradigms like the functional programming paradigm, the logic programming, expert systems, aspect-oriented programming, adaptive programming, intentional programming, etc.

2.2. Service-oriented architectures (SOA)
   Currently, significant efforts are devoted to transfer the ideas of service-oriented architectures (SOA) to the factory floor and into embedded devices. One approach is to incorporate the service infrastructure in the device itself [6, 43], i.e. SOAP-based web services or XMLRPC (XML remote procedure calls) based functions. A different approach is the use of an IPC upstream [21] to host the service infrastructure.

   Another interesting approach is the ITEA SIRENA [3] project, which does provide a generic service framework based on OSGi (OSGi Alliance, former Open Services Gateway initiative). Though the idea of a loose coupling by means of services has its merits, it does not solve the problem of being dependant on the chosen particular implementation technology. This means, that no abstract modeling is existing in a manner which does permit to use different technology without huge changes.

   However, SIRENA and OSGi define a method to change code in service.

2.3. Adaptive field bus technology
   In [7], an interesting approach to transform different communication data into each other has been reported. In this approach, XSL is used to translate different field bus data formats into each other. However, the approach is dependant on the chosen technology, i.e. XSL and XSLT, and doesn’t abstract from this technology to support other solutions in terms of a domain-oriented framework.

2.4. OPC UA
   OPC UA (Unified Architecture) [20, 28] is intended to serve as a communication platform in industrial automation [17]. For the definition of transformations between different communication protocols and also for transformation to higher (business) levels, section 5 [29] of the OPC UA specification is of relevance, the “Information Model”. By using an information model, the data structure can be modified in service, though the access to the data model must be defined inside the gateway code, and this isn’t supported by OPC UA to do while the gateway is in service. Unfortunately, the information models and the framework of interfaces are rather rigid, i.e. there exists extensive APIs, which must be complied. This has the negative impact of huge technology dependence. Less code is reusable for other implementation solutions not based on OPC UA.

   Nonetheless, OPC UA is a big step forward, unifying the different technology chunks of OPC, like
DA, HDA (Historically Data Access), AE (Alarm and Events), etc, and provides support for specifications like EDDL [26], MIMOSA [32], or FDI [2].

2.5. Other approaches
A different approach of transforming communication data was proposed by the company Command and Control Technologies (CCT). CCT has defined a so-called “Semantic Gateway” that assigns communication data a meaning [34]. This should allow to get rid of communication data specific code. A similar approach has been reported by Lynch et al. [19] in the healthcare business. The authors have proposed that requests towards a data server can be defined semantically.

The idea of formulating concepts on a semantic level, above the implementation level, and to modify these concepts without interfering with implementation code, reflects the concept of expert systems, where domain information is stored in rules, constraints, or frames, thus forming a domain model which can be changed or extended while the implementation system is continuously in service. The concepts of expert systems are therefore the basic idea of the concept for highly reusable communication gateways, as explained in detail in the next chapter.

3. A basis for reuse
Reuse is the key to profit. Therefore many researchers, engineers and developers are reaching for reuse. As it was pointed out by Griss [15] there are different levels of (software) reuse. Most often ad-hoc reuse of code fragments is understood as reuse. For this article the level of systematic reuse is requested.

3.1. Expert systems for reuse
The concept of expert systems has been widely used to support the design, operation, and maintenance of technical systems. Mostly, when expert systems have been applied, the user has been confronted with rules, constraints, or frames. This required always a dedicated training and had, thus, negative impacts on the acceptance of the expert systems.

Independent of whether the user is in contact with the rules, constraints, or frames, these concepts can be employed to formalize a domain model. In the approach favored in this paper, an expert system to drive the reusability of code and communication knowledge is described. To describe the domain model regarding communication needs in the automation concept, Minsky’s frame theory [23] is used.

Frames can be perceived as a network of nodes and relations of these nodes. Protégé employs four types of frames: classes, slots, facets, and instances. A class can be viewed as a container of the modeling artifacts, the slots, which have a name and a value (the value can be a class, an instance or a scalar data value as for example integer or string). The slots can have some distinct properties, the facets, for e.g. a minimum and a maximum to limit a slot value. The instance can be imagined as a concrete instantiation of a class, which corresponds to instances of a class in an object-oriented programming language.

Furthermore, constraints can be defined to assert specific states for frames, classes and instances. This is inspired by the methodology of design by contract [22].

Data can be retrieved by delegating the retrieval task to a logic calculus. These building blocks form an expert system.

To capture the domain model, the tool “Protégé” is used. As pointed out in [13], “Protégé is neither an expert system itself nor a program that builds expert systems directly; instead, Protégé is a tool that helps users build other tools that are custom-tailored to assist with knowledge acquisition for expert systems in specific areas.”

This means, that Protégé can offer a platform to maintain the data model of a communication gateway. To capture the knowledge of communication tasks, Protégé employs frames.

This will be used in the following sections to establish a software architecture for communication gateways, to foster software reuse and, more importantly, knowledge reuse, i.e. the reuse of meta data for communication protocols. But, before doing this, some basic requirements to motivate the need for an expert system for communication data will be presented.

3.2. Basic requirements
To foster software reuse and knowledge reuse as well, the proposed software architecture must fulfill some basic functional requirements. They are given as follows:

- The possibility to incorporate and switch device profile standards like FDCML [33], FDT/DTM [27], EDDL [26], FDI [2], GDI [24], CANopen device profiles [25] etc. without affecting the whole system, i.e. reducing rework costs at a maximum;
- A way to add new datagrams of a communication protocol to an existing communication system, in service;
- The possibility to support higher level communication protocols, e.g. specific diagnostic protocols like ODX [1, 42] for automotive applications;
- Seamless change of communication protocols between communication gateway and higher level servers, e.g. Data Access Server;
- Domain specific modeling, so changes of technical libraries don’t affect the reusable components. Furthermore, some quality attributes must be addressed:
  - Defining the architecture for a software product line for different requirements in security, scale and communication domain;
  - Taking into account, that technology does shift permanently, and this for ever decreasing budgets, i.e. for ever decreasing time slots.

3.3. The information flow

In addition to the frame-based data model the information flow is the second corner stone of the proposed expert system for building communication gateways.

The vision behind the information flow of the proposed design is the concept of “generic functions”. Programming languages can offer object-orientation in two different ways: as a message-based object-orientation, where the objects of classes receive messages which methods (of a corresponding class) are to call, as in languages like JAVA, C#, Smalltalk, etc., or an object-orientation based on a clear separation of functional information flow and data modeling. This type of object-orientation is implemented by the Common Lisp Object System (CLOS) [5, 45], the object-orientation added to Common Lisp, and the object-orientation which is based on the frame theory. In CLOS, the functional information flow is given by an abstract definition, the “generic function”, which can be implemented by methods, but which is not part of a class. Data modeling, by classes, and the information flow, by using generic functions, is clearly separated.

The idea of “generic functions” in CLOS is just the metaphor for the proposed system: a clear separation of the basic information flow on the one hand and the data modeling (which in turn also defines the data chunks that build up the information flow) on the other hand. A clear interface does provide the access from the methods to the data model. Again, it is just a metaphor, the system is developed using JAVA and Protégé, no development was carried out in CLOS, but the idea of a clear separation of the different concerns is adapted from CLOS and Common Lisp.

3.3.1. Services

The information flow is defined by services. This provides a loose coupling of the parts of the whole system, so a refactoring of parts of the system (different services) does not affect any other part. The grouping of the services is two-fold: there are the domain services [47], which capture the information flow from the viewpoint of a domain expert, and there are the technical implementations, the technical services, which implement the domain services by a dedicated technology. Every domain service is defined as an abstract class, so no domain service can be used as is. The technical service does inherit from the domain service and defines class variables and private methods to use a specific technology to fulfill the tasks of the domain services. For every technical service, there exists exactly one domain service, but for a domain service there can be a set of technical services.

3.3.2. Logic calculus

To request data which is stored using Protégé easily by using a service, the Protégé Axiom Language (PAL) is used. PAL offers a logic calculus to define constraints for a class or to query important information about the knowledge base like e.g. a specific class name or a required scalar value. But furthermore, PAL can also be used for complex data retrieval, this is e.g. for complex telediagnosis applications of interest. PAL is also used to define specific restrictions on classes and slots. Taking into account, that frames are the building blocks of Protégé, but also of CLOS, such a restriction defines a design by contract approach to develop communication gateways.

A logic calculus is quite different to other data management languages like SQL, HQL, OQL or even QVT. A logic calculus is not just for data management, but rather for building constraints or reasoning on a knowledge base, like that one of Protégé.

Especially, the use of PAL has the advantage that data structures can change without affecting the JAVA code, just the queries (or the constraints) have to be adapted. This is guaranteed by the plugin concept of Protégé which permits PAL to plug into Protégé.

A very simple example of a query is given in Figure 1. The pictured query retrieves tuples which consist of commands for an embedded device and IP-based device ids. The Range defines the data types on which the query works. The Statement defines the query itself. The Name is a unique id which can be used from within a computer program to execute a query.

Figure 1. Example of a query.
3.3.3. Flexible data access

Furthermore, the domain services must provide a way that they remain reusable in case of a modified data model. If the access to slots and facets of a specific Protégé instance were hard wired to a domain service, little would be reusable. The renaming of only one slot name would corrupt the access to the data. Therefore, the design pattern “visitor” from Gamma et al. [12] is applied. With such a “visitor”, a data structure can be processed in such a way that the data structure is independent from the visitor.

The intent of a visitor is the following. “Represent an operation to be performed on the elements of an object structure. Visitor lets you define a new operation without changing the classes of the elements on which it operates” [12]. If a domain service has to be fixed, a visitor can encapsulate all access methods to frames from Protégé which can vary in time. Therefore, a class using a visitor takes either a visitor from a typed dynamic data structure in terms of a class variable, or the visitor is given as a parameter argument to a method. The class which calls the visitor methods has no information which visitor is to be used, but knows the actual argument list for the visitor. In JAVA, this can be an argument list of variable length.

As an example, take the framework commons collections [38] of the Apache Foundation, which defines collection data structures not given in the JAVA Collections. To iterate over the different data structures, like LinkedMap or MultiSet, specific visitors are defined, like Iterator or MapIterator. Using such a loose coupling of collection and iterator has the benefit that the collection does not have to be modified when a new iterator is required, and vice versa.

But just only with the visitor pattern, no reusability for the communication gateway can be guaranteed. Furthermore, a possibility to inject the dependency of a service from a visitor is required, i.e. a dependency from a specific implementation of an interface. This can be realized by the Dependency Injection Pattern (DIP), proposed by Fowler [9]. Another name of this pattern is “Inversion of Control”, because a class A which does need an object of class B does not call an object of B directly, rather A has an accessor setDependency(A a) to publish the dependency of A from B. So, at any time a service can have a visitor object which will be injected, i.e. the actual implementation is not known at compile time. The delegation of responsibility from the service to the visitor is captured by another design pattern, a behavioral pattern, from Gamma et al. [12]: the “Chain of Responsibility”. The responsibility of processing data (i.e. JAVA objects or Protégé data) is delegated from the service to the visitor. Indeed a visitor itself can have sub-visitors, which take over the responsibility to process data.

With such a structure for the information flow, domain services are decoupled from their technology implementation and also from the access to the data structures. This can be used to modify the data structure independent from the structures which build the information flow.

After the general design guideline of a reusable communication gateway is described, the details of the design will be presented in the next section.

3.4. Design details

This section starts with the introduction of the data model and general architecture for a reusable communication gateway. Subsequently, the authors explain the necessary services and how the reusability will be achieved by the use of services, visitors, and PAL.

3.4.1. The general architecture

The data model is implemented by using Protégé to define ontologies. According to Gruber, an ontology is a formalization of a concept [17]. In Protégé, three ontologies have been defined for the data model: a central ontology for all communication-related data modeling, for e.g. commands, datagrams, device IDs, etc. In two other ontologies, helper classes have been defined.

Protégé is on the one hand a sophisticated graphical tool to develop and maintain ontologies, but can on the other hand be used as a library to access frames (i.e. elements of an ontology) from within a computer program. The Protégé library can be used stand alone or as a Client-Server system. The authors have decided to use Protégé in client-server mode, though this is changeable to a standalone mode.

The general architecture of the communication gateway is pictured in Figure 2. This architecture has been implemented in “pdvRemote” [41], which is a development effort of pdv Technische Automation + Systeme GmbH. It defines a reusable framework of the common components of a telediagnosis system. “pdvRemote” is a service-oriented framework and written using JAVA, though all data modeling is done by using Protégé.

Figure 2. Architecture of the gateway.
3.4.2. Main services

The framework for the communication gateway as a reusable asset offers two services for accessing the communication gateway from remote computers, the CommandService and the AddressService. The CommandService defines the highest abstraction level for a communication with field bus devices, devices using Industrial Ethernet or devices having wireless access. The CommandService offers methods to retrieve command ids, to get commands (i.e. Protégé instances) by command IDs or to send commands. A command can be a field bus command, but also can be a request to the gateway to doing some specific.

Up to now, all offered communication is synchronous communication, but other communication types are under active development, mainly asynchronous communication and listener concepts (publish/subscribe).

The AddressService defines methods to receive registered destinations. In most applications, this is a device ID, a MAC ID, an IP, a LON id, or similar.

The given domain services are defined as abstract classes and have technical implementations assigned (in a specific reference technology, though these are changeable). As a reference implementation, the Apache open source project Hivemind [39] is used as a service registry to provide the services. Hivemind uses an XML-based file configuration to define service interfaces and their implementations. The corresponding classes are “plain old JAVA objects” (POJO) [8]. This technology is only a reference implementation; every other JAVA-based technology can be used to provide the services, too. The support of other service registries, like OSGi, is delegated to future work.

To access services remotely provided by Hivemind a technology like JMX (Java Management eXtension) [30], RMI (Remote Method Invocation) [35], XML-RPC [37] or SOAP [40] can be used. To simplify matters, RMI was chosen in “pdvRemote”. Again, this is just a reference implementation. Any other of the described technologies could be used as well.

As described in section 3.3, to establish a systematic software reuse from the beginning, the domain services must be freed from dependencies of specific slots, facets, or classes to access the data model. It was pointed out that, to foster reusability, visitors must be defined. Almost all services have visitors to access some data stored in Protégé.

3.4.3. Internally used services

The OntologyService encapsulates (almost) all ontology-related tasks of “pdvRemote”, e.g.: getting classes and instances, getting slots, creating new instances and cloning instances at the server side and locally as well. All other Protégé related actions can be directly accomplished by using the methods of the JAVA class DefaultKnowledgeBase of the Protege.jar library.

An especially interesting service is the CommunicationGatewayService. This service, called by the CommandService to communicate with a machine or a device, does every communication with devices or machines. For an IP-based communication this is solved by the technical implementation using SUN NIO [31]. For other purposes, such as serial port interaction different implementations can be used.

A very important service, which is pictured in Figure 3, is the SemanticMapperService. This service is responsible for mapping between communication data and higher level data, e.g. between a Protégé instance of a class ExampleDatagram and the communication datagram of the CommunicationGatewayService. The SemanticMapperService does not hold any Protégé-related data access capability; instead, this task will be accomplished by an injected visitor, the DataTypeVisitorService. This visitor object carries out all Protégé-related data access for data transformation between communication data and higher level data objects stored in Protégé without hardwiring to the SemanticMapperService.

Figure 3. The service layers of “pdvRemote”

The mentioned services are pictured in Figure 3, which defines the core functionality of “pdvRemote” in terms of a service layer architecture. The business layer comprises the CommandService and the AddressService and defines the domain specific abstraction of the whole communication framework, though, it’s by sure not the complete business service stack. The data layer, consisting of the OntologyService and the QueryService,
which is responsible for executing PAL queries, defines access functionality to the data storage. The QueryService defines methods to retrieve queries and to execute them. The mediation layer, based on the SemanticMapperService, is responsible for mapping between low level communication data and higher abstractions for the business layer, i.e. mapping between JAVA communication objects and Protégé instances. The communication layer implements the communication tasks, based on the CommunicationGatewayService and subsequently services like the CommunicationSelectorService, which is responsible for the selection of the right communication service. One of the communication services used is the NIOInternetCommunicationService, which communicates IP-based via SUN NIO. Another service is the GenericSOAPCommunicationService, which handles web service calls to service-enabled embedded devices. The CommunicationGatewayService is the central service for the communication in “pdvRemote”. This service delegates all requests and responses from higher level services or from embedded devices to the appropriate service. Such a service is the NIOInternetCommunicationService. This technical service does maintain the connections, sends data to embedded device or receives data from an embedded device using an IP-based communication.

3.5. Example communication

The core services of “pdvRemote” have thus been described statically. In addition, for a deeper understanding, a workflow of the different services is given in Figures 4-6.

Figure 4. The sequence of method calls for “sendCommand” of the service CommandService – the data handling.

Figure 4 shows the call of the method “sendCommand” from the CommandService. The communication is done via RMI. The request is delegated from the RMI wrapper to the technical service CommandService. This service decides to query the needed parameter (the command corresponding to an id) from Protégé and to transform the parameters. The SemanticMapperService transforms the command to low level communication data. After the communication is done, the communication data are transformed to data appropriate to the business layer, i.e. appropriate for the Java RMI Client.

Figure 5. The sequence of method calls for “sendCommand” of the service CommandService – the overview of the communication.

After the transformation of the data has been accomplished, the communication is done, for e.g. synchronously. This can be seen in Figure 5. If an embedded device has received the request and has responded, the return result will be thrown back via the chain of responsibility of the different services to the Java RMI Client.

Figure 6. The sequence of method calls for “sendCommand” of the service CommandService – the details of the communication.

Figure 6 shows the service chain for IP-based communication by SUN NIO. The method “sendCommand” causes the CommunicationSelectorService to select a service which does handle the communication data appropriately, i.e. the service selects the right communication service. This is the NIOInternetCommunicationService in Figure 6.

4. Conclusions

In this paper problems of a classical development of communication gateways have been identified. Practical used definitions of the term “communication gateway” were reflected. It was pointed out that developing software focusing just and only on one programming paradigm has its drawbacks against a multi-paradigm approach.

A knowledge-based method to implement communication gateways, integrating different programming paradigms like expert systems, logic calculus and object-oriented software development,
was sketched, using JAVA, PAL and the data acquisition system Protégé.

It was shown that the separation of the technical representation from the domain services can foster reusability of the used methods. Furthermore, a clear separation of information flow and data modeling was investigated and implemented, which has significant benefits compared to a merge of data abstraction and information flow by means of methods, procedures and functions.

5. Future work

The next steps will be to include other types of communication in the framework, like OSGi services. But also different communication types like asynchronous communication and listener concepts are needed.

Another important research need is the mapping of the communication data to business data and vice versa, but also between different communication protocols. This should be realized using ontology mappings and adaptive object-oriented programming [18], so no hand-crafted code should be required. This will complete the presented concept, and guarantee that communication gateways developed following the given concept don’t have any downtimes for the data management. Only the management of new services requires a downtime of parts of the system (OSGi services included) or for the whole system.

A thoroughful discussion about performance issues, ease of development and security concerns are also delegated to future work.

Acknowledgement

This work was partially funded by the pdv Technische Automation+Systeme GmbH.

References


[12] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, Design Patterns, Elements of Reusable Object-Oriented Software. Addison-Wesley, 1995.


