Pattern Language for Specification of Communication Protocols

Juha Pärssinen, Markku Turunen

1 Abstract

This paper presents the pattern language for specification of communication protocols. The pattern language contains four levels which are used to specify communication protocol and its messages. These four levels of this pattern language are: high-level protocol specification, protocol structure specification, message specification, and detailed message specification. Several existing patterns, e.g. the Layers, are used together with new ones to specify protocol structure. Patterns for communication protocol messages are most important part of this pattern language, and also most important part of any communication protocol specification.

2 Introduction

This pattern language concentrates on specification of communication protocols. Specification and implementation of communication protocols are two different sides of a coin. Specification of communication protocol explains meaning of communication messages sent between protocols and tells practically nothing about system structure. Implementation explains static and dynamic structure of protocol system and its layers, but messages are merely referred as events or payloads.

The authors of this language have participated during recent years several in projects in the area of protocol engineering, including protocol implementation projects, protocol specification and development method research projects, and protocol structure and message design and implementation tool development projects. They have also written other pattern languages which also concentrate to communication protocols [2][3]. These languages have took lots of influence from [1][6][7][9], and they are under slow, but continuous, development.

The pattern language presented in this paper contains four levels. This paper explains first briefly patterns related to high-level protocol specification, and then patterns related to protocol structure. After this protocol message related patterns are presented. These patterns are the most important part of this pattern language. Last and most detailed level of patterns is used to support message specification.

Road maps of these patterns are shown before each section. In these road maps and in the body text of this paper all patterns explained in detail in this paper are marked using ***bold italics***. Patterns explained elsewhere are marked using *italics*. All road maps are collected to Figure 15, which is included as an appendix at the end of this paper.

In this paper we use as a running example a simple high-level protocol called TeleChess. This protocol is used to play chess remotely. This protocol is used also as an example in protocol engineering courses held by the authors in the Helsinki University of Technology.

3 Pattern Language

3.1 Patterns for High-Level Protocol Specification

Pattern language presented in this paper proposes a solution to problem raised by contradiction between needs to communicate and means to communicate. These two concepts and contradiction between them are discussed in next sections. Contradiction and related high-level patterns are presented in Figure 1. In this figure there are three patterns. Two of them, *Community of Nodes* and *Conversation between Nodes,* are patterns from [3]. *Transmission Media* has currently "patlet" status. These patterns and their context are explained next briefly.

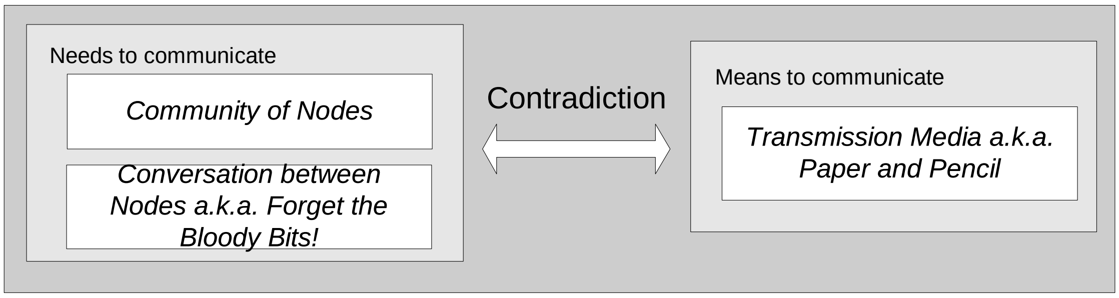

Needs to communicate
Community of Nodes
Conversation between Nodes a.k.a. Forget the Bloody Bits!
Contradiction
Means to communicate
Transmission Media a.k.a. Paper and Pencil

Fig. 1. Contradiction between needs and means in communication. In this figure there are patterns from [3]. This figure presents highest-level road map to patterns in this and related languages.

3.1.1 Needs to Communicate

Communication is one of the basic needs in human society. This can be seen how communicating related systems, like cellular system and internet, have been successful during the last decade. If we take a high level view on communication, then we can see that in communication there are two or more parties, here called as nodes. To facilitate communication between two nodes there has to be a communication channel, or connection, between them, as shown in Figure 2. In the modern society there are also other kind of communication types than just between humans. For example, there are communication between humans and different kind of automatic services, and communication between autonomous systems without human interference. All these nodes are part of *Community of Nodes*, pattern from [3].

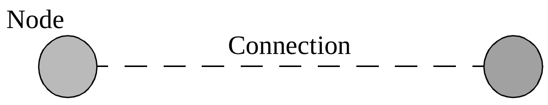


Fig. 2. Two nodes and connection between them.

A typical communication session might have the following sequence. First, one node initiates communication with other, or it responds to the initiation of communication from other. Second, during the communication session there are messages transmitted between parties, as shown in Figure 3. During message exchange different problems need to be solved e.g. there can be a variety of errors that need to be handled. Finally communication session is finished.

However, there are variations about this basic scheme. For example in message-based communication there is no explicit connection establishment and disconnection phases.

To make meaningful communication possible rules for communication, a communication protocol, between nodes are needed. These rules define what are the messages which nodes can send to each other and what is the meaning and format of each message, what kind of message exchange forms a meaningful dialog between the nodes and what kind of functionality is implied to happen in the nodes due to the message exchange. Messages are part of *Conversation between Nodes*, pattern from [3]. These rules of messages are main and most important part of communication protocol standards.

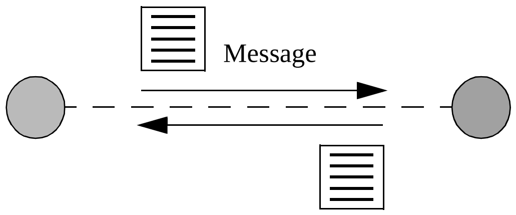


Fig. 3. Messages transferred between nodes.

Running example

In the TeleChess protocol all nodes are chess applications. These players form *Community of Nodes*. Messages between these nodes contain for example challenges to play, moves, and checkmates and are part of *Conversation between Nodes*.

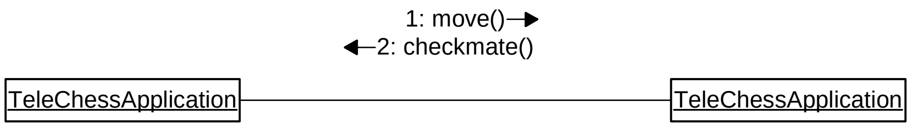


Fig. 4. Two TeleChess applications and message exchange between them.

3.1.2 Means to Communicate

The nodes have connection between each other via some existing physical media. The physical media between nodes offers only low-level communication service, moving bits from one point to another. This can be achieved by using photons, radio frequencies, carrier pigeons or other whatever available *Transmission Media*. Physical media can be considered as a real connection, compared to connection between nodes which can be considered as a virtual connection.

However, nodes are interested only in very high-level aspects of communication, they want to send message to another node and take no any interest in technical details. For this reason, adaptation is needed between node and physical media. Also, if there are changes in physical media, then the node doesn't want to know anything about them.

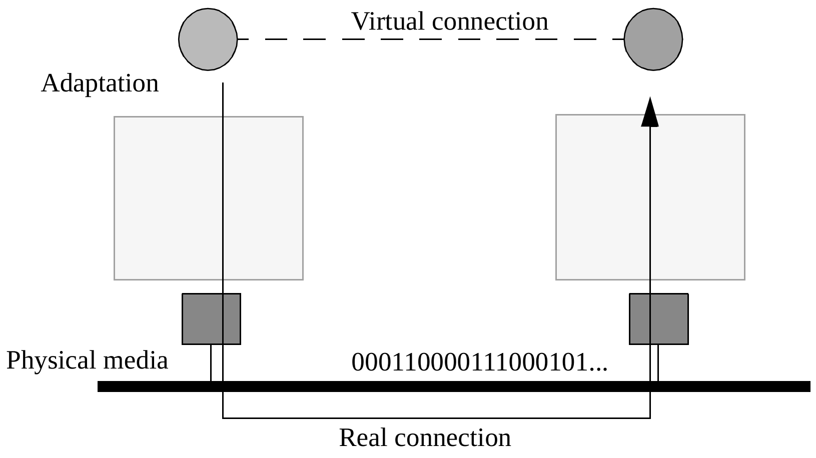


Fig. 5. Bits transmitted between nodes via physical media, adaptation is used to fill gap between node and physical media.

Running example

TeleChess protocol uses one common email program to send messages between players, or nodes. Email program uses other protocols to transmit messages between parties, e.g. TCP/IP protocol stack, and these protocols uses some physical hardware to transmit bits, e.g. modem line or ethernet.

3.2 Patterns for the Structure of Communication System

Patterns related to the structure of communication are shown in Figure 6. *Elements of a Node, Two Roles of Nodes,* and *Interfaces of an Entity* pattern are explained in [3]. ***From Service to Protocol*** pattern is presented in this paper. These patterns can be used after *Community of Nodes* pattern is used.

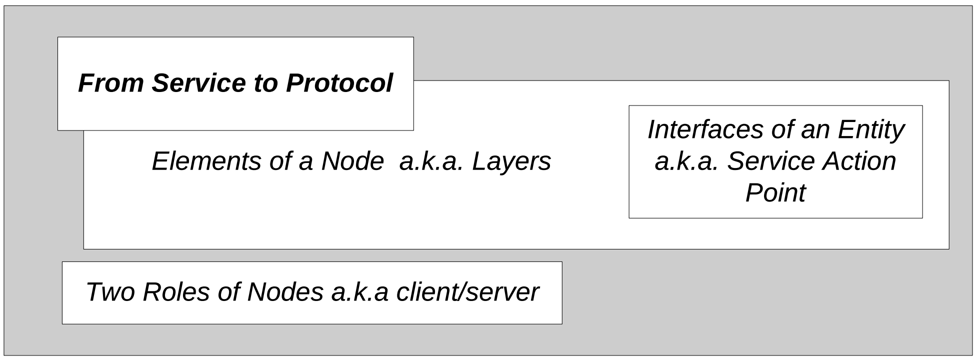

From Service to Protocol
Elements of a Node a.k.a. Layers
Interfaces of an Entity a.k.a. Service Action Point
Two Roles of Nodes a.k.a client/server

Fig. 6. Patterns related to the structure of communication system.

From Service To Protocol

Context

A communication system is specified using layered architecture and each layer, called typically as an entity, communicates with layer which is in the same level in architecture, but in different instance of a system. These entities, called as peer entities, use a well-defined communication protocol between them, e.g. (N) Service Protocol in Figure 7. Entities also offer Service Access Points (SAP) to entities above them. In Figure 7 (N Service Entity) provides services, as a SAP, to it’s user.

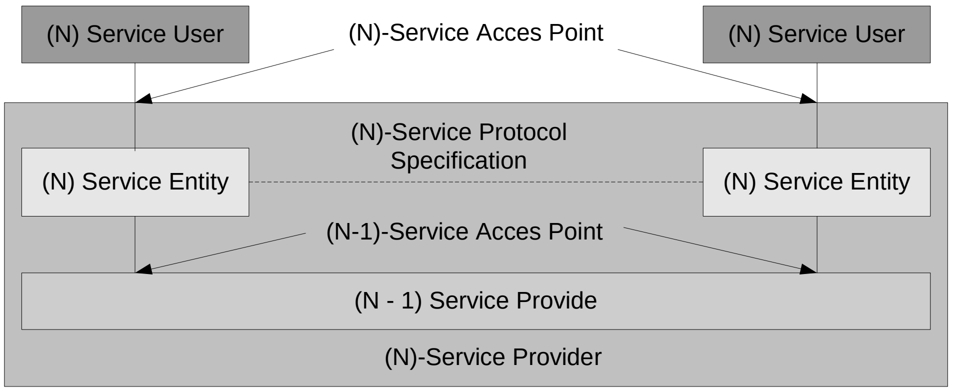


Fig. 7. Service Layering in OSI [6]

Problem

Entities in same level but in different instances of system don’t have direct connection to each other, but they want to send messages to each other. Typical communication protocol specification explains only messages sent between peer entities.

Solution

Communication messages are routed as shown in Figure 8. This route, when going from up to down, looks little bit like a greek character ζ (zeta). Messages coming from service user are sent to peer entity, but actually they are routed using the services of lower entity. This works also to opposite direction: an entity receives messages from peer entity, without knowing that they are actually coming from lower entity. One interesting point related to this pattern is that it is used recursive when a stack of entities is designed: every entity offers service to higher entity and uses services offered by lower entity.

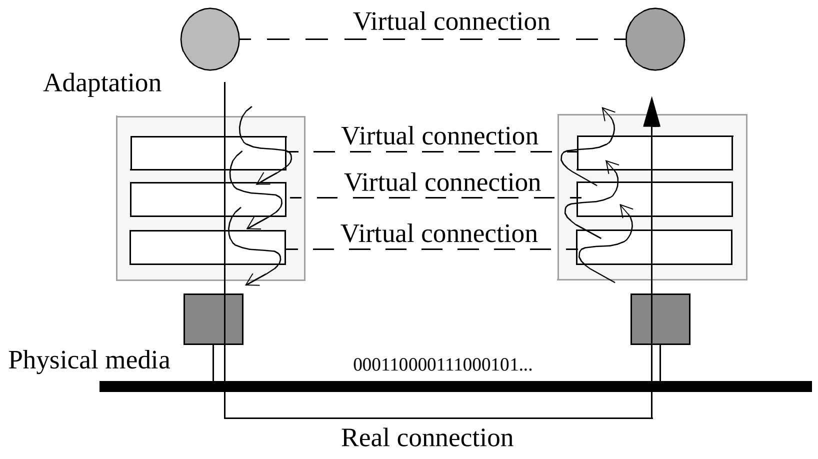


Fig. 8. Message flows in ***From Service to Protocol*** pattern.

Running example

TeleChess protocol stack contains two entities, or layers. TeleChess entity offers services to TeleChessUI, an user interface, and it is responsible e.g. starting a game, manage the state of current game, and ending a game. TeleChess protocol specification defines set of messages send and received between two instances of protocol, and how these messages should be react. Specification don't contain any information how messages are actually delivered to other party. There are other protocol for this, called Mail, which offers a transmission service to TeleChess layer. In Figure 9 messages there are shown message flows from TeleChessUI to Mail and in opposite direction. TeleChessUI uses service offered by TeleChess entity. To send message to its peer entity TeleChess entity uses services offered by Mail entity. Mail itself uses other services, offered by lower entities, not shown here.

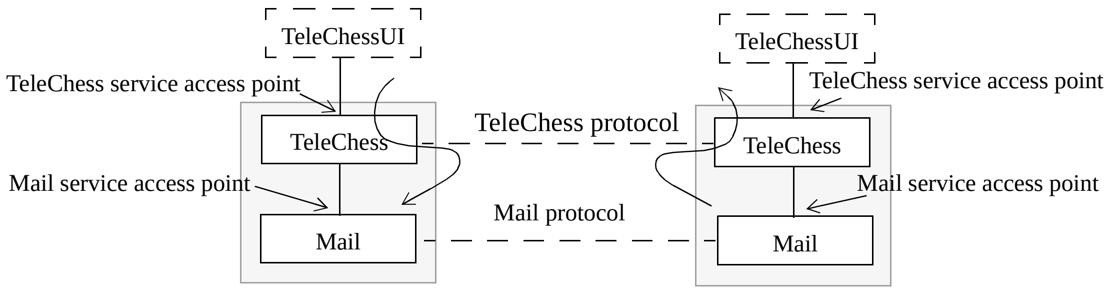

TeleChessUI
TeleChessUI
TeleChess service access point
TeleChess service access point
TeleChess protocol
TeleChess
TeleChess
Mail service access point
Mail service access point
Mail protocol
Mail
Mail

Fig. 9. Services offered by TeleChess and Mail protocols, and message flows in TeleChess protocol.

Related Patterns

This pattern is used together with the *Layers* pattern [9] and *Patterns for Generating a Layered Architecture* [5] when almost any communication protocol is specified. This pattern is typically implemented with *Protocol Behavior* pattern [2]. Implementation issues of those two patterns can be found from [2].

3.3 Patterns for the Communication Message Specification

Patterns of this language for communication messages specification are shown in Figure 10. There are four patterns: ***Message Exchange***, ***Content of a Message for Humans***, ***Content of a Message for Machines***, and ***Message Transfer Syntax***. In this pattern language there are also more detailed patterns which are used together with ***Message Transfer Syntax***. Those patterns are presented in section 3.3.1 "Supporting Patterns for Communication Message Specification".

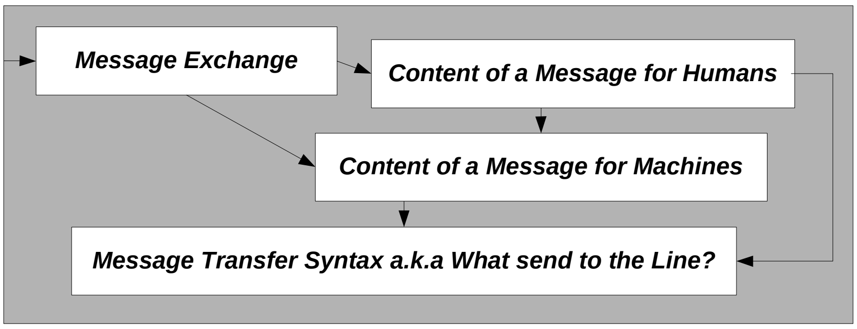

Message Exchange
Content of a Message for Humans
Content of a Message for Machines
Message Transfer Syntax a.k.a What send to the Line?

Fig. 10. Patterns related to communication message specification.

Message Exchange

Context

You have identified communicating entity, or layer, which communicate, and you are ready to think about messages between peer entities. You might also have considered messages between entities with *Conversation between Nodes* pattern.

Problem

You want to provide an easily understandable view to messages between entities.

Forces

- Rules for message exchange between entities are needed to make co-operation possible. These rules define what are the messages which entities can send each other, what is meaning of each messages, and which are possible sequences of messages.
- When messages are specified, experts from several fields must communicate with each other. Some might be expert on the behavioral part, knowing what information a message must carry so that participants of a communication procedure can have a common understanding of a state, what is requested and what are expected actions. Others might be experts on technical areas, like on methods of message specification.
- It might tempting to start message encoding with bits and bytes because they are “concrete” in a sense that when one sees a bit table one can have an impression that one understands what a message contains.

Solution

Define messages between entities, give them meaningful names, and show their sequences using e.g. UML Sequence Diagram.

Running Example

In TeleChess protocol one of the most important message is move. In Figure 11 there is shown simple message exchange between two TeleChess entity. The names of messages are meaningful: first one move a chessman, second one moves a chessman and checkmates.

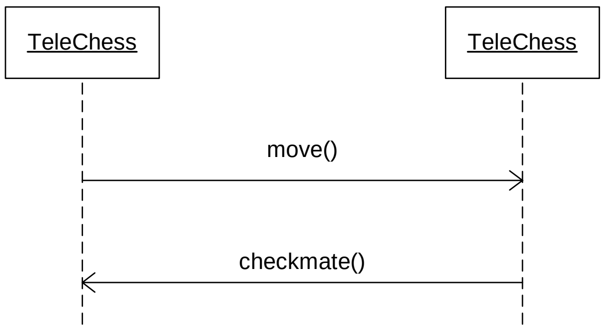


Fig. 11. TeleChess UML Sequence Diagram

Next Pattern

Now you have defined messages with their name and relations between them as message sequences. Next, you will define content of your messages. To define message content, you will use ***Content of a Message for Humans*** pattern or ***Content of a Message for Machines*** pattern.

Content of a Message for Humans

A.k.a ***High-level Abstract Syntax***

Context

You have specified messages and their relationships between entities, but not their content.

Problem

You want to provide a simple and easily understand view about messages for humans.

Forces

- The higher the level of protocol, the more complex the message structure is likely to be.
- Your specification should have enough information so that bit level representation can be derived from your description.
- In addition to message information content there may be additional required message properties for messages, such as priority or optionality.
- The use of formal methods makes description less ambiguous, but people might be uncomfortable with them. From someone's point of view formalism hides the idea of concepts, be-

cause notations might look too much like programming languages. From another's point of view, a formal notation hides the bits, and thus removes control from their hands.

- It might tempting to start message encoding with bits and bytes because they are “concrete” in a sense that when one sees a bit table one can have an impression that one understands what a message contains.
- You have messages, which are complex, and you have selected those message definitions that will be robust and future-proof. This can result complex message definitions where the actual contents are hidden behind mechanisms of extensibility and other features.

Solution

Specify content of your messages using tables. A table for each message should contain at least message name, specification of message parameters, specification valid value sets for parameters, and specification of other logical parameter properties (e.g. multiplicity, optionally or conditionally).

Running Example

TeleChess protocol message move is used as an example in Figure 12. TeleChess message move contains chessman’s type, starting point and ending point. Starting point and ending point are valid point in chessboard. Piece is enumerated type and contains all possible chessman types: pawn, rook, knight, bishop, queen and king. Starting point is defined optional, because in some cases a type of chessman is enough to define which chessman moves, e.g. queen or last rook.

Message: Move

parameter	type	presence	multiplicity	description
piece	[pawn, rook, knight, bishop, queen, king]	mandatory	1	Type of chessman.
starting point	[A..H][1..8]	optional	1	Chessman which will move.
ending point	[A..H][1..8]	mandatory	1	Chessman destination.

Fig. 12. Services offered by TeleChess and Mail protocols, and message flows in TeleChess protocol.

Next Pattern

Now you have specified messages and all of their properties excluding their bit presentation. The specification of message bit presentation is considered in ***Message Transfer Syntax*** pattern.

Content of a Message for Machines

A.k.a ***Low-level Abstract Syntax***

Context

You have defined messages and their relationships between entities, but not their content.

Problem

You want to provide a view about messages for machines, or humans who prefers formal presentation of messages.

Forces

- The higher the level of protocol, the more complex the message structure is likely to be.
- Your specification should have enough information so that bit level representation can be derived from your logical description.
- In addition to message information content there may be additional required message properties for messages, such as priority or optionality.
- It might tempting to start message encoding with bits and bytes because they are "concrete" in a sense that when one sees a bit table one can have an impression that one understands what a message contains.
- You have messages, which are complex, and you have selected those message definitions that will be robust and future-proof. This can result complex message definitions where the actual contents are hidden behind mechanisms of extensibility and other features.
- While an informal description aids in one's initial understanding of a messaging system, a precise formalization must exist as a basis in a court of law to determine when exclude devices claiming to be network compliant, when they are not.

Solution

Specify your messages using a formal notification, e.g. ASN.1 [11], without specifying anything about message bit presentation, i.e. separate a concept, e.g. message name, and how it is realized. Your formalism should support for all the needed concepts. If your notation does not explicitly support a concept, then you have to specify how a concept is mapped to the notation.

When specifying the message's logical contents, at least the following must be specified: message name, specification of message parameters, specification of valid value sets for parameters, and specification of other logical parameter properties (e.g. multiplicity, optionally or conditionally).

Consider what other kind of message properties are needed. Consider how message might evolve in the future and how to be prepared for it.

Running Example

TeleChess protocol message move is used as an example. It is specified below using ASN.1 [11]. This specification includes enumeration for chessmen and move protocol data unit (PDU) specification.

```
Piece ::= ENUMERATED {
        pawn, rook, knight, bishop, queen, king
}
MovePDU ::= SEQUENCE {
        piece           Piece,
        fromSquare      Square OPTIONAL,
        toSquare        Square,
}
```

Next Pattern

Now you have specified messages and all of their properties excluding their bit presentation. The specification of message bit presentation is considered in ***Message Transfer Syntax*** pattern.

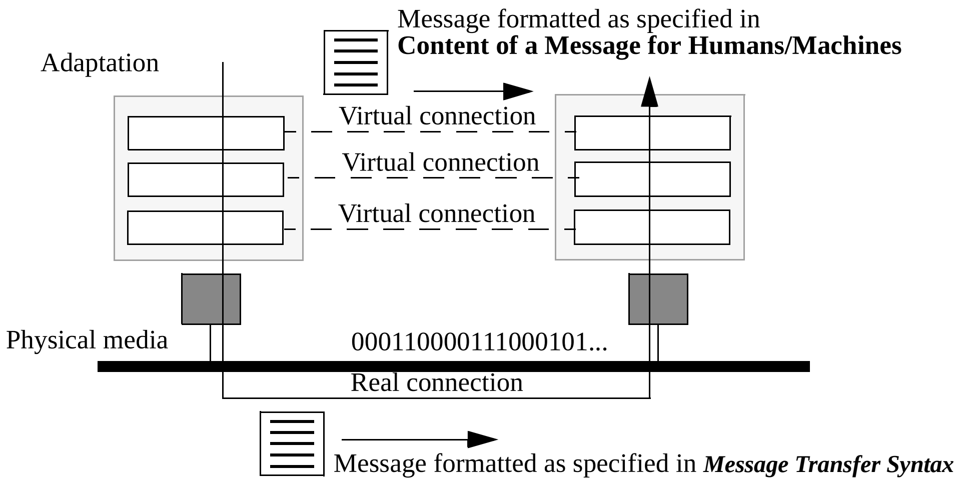

Message formatted as specified in
Content of a Message for Humans/Machines
Adaptation
Virtual connection
Virtual connection
Virtual connection
Physical media
000110000111000101...
Real connection
Message formatted as specified in ***Message Transfer Syntax***

Fig. 13. Flows of messages specified using ***Context of a Message for Humans/Machines*** and ***MessageTransfer Syntax*** patterns

Message Transfer Syntax

A.k.a. What to Send to the Line?

Context

You have specified a message abstract syntax using ***Content of a Message for Humans*** and/or ***Content of a Message for Machines***. This abstract specification defines a logical content of a message that is sent between peer systems.

Problem

What send to the Line?

Forces

A message logical content is specified, but its local bit presentation can vary in different systems depending on many implementation issues like integer presentation in underlying hardware etc.

- A message can be presented in a local system in a string of bits or bytes, but it can also be built as a complicated object tree.
- There is a need for a common transmission format so that a message can be encoded (serialized) into a string of bits or bytes and then the encoded bits or bytes can be sent over a transmission media.
- The available bandwidth might be small resulting need for short peer messages.
- A design trade-off exists between squeezed or robust encoding.
- Peer messages are sent over a transmission media and thus must be encoded (serialized) to strings of bits or bytes. A receiver must be able to determine the type of a message that is contained in a bit string.

Solution

Specify the message bit presentation, and how this bit presentation is mapped to the message logical content and vice versa.

Resulting Context

The separation of logical message contents, Message Abstract Syntax, specified using ***Content of a Message for Humans*** and/or ***Content of a Message for Machines*** and message transmission format, ***Message Transfer Syntax***, makes it possible to refer to information in the message parameters. The complexity of transfer syntax can be encapsulated and hidden from other parts of a specification.

Running example

In TeleChess protocol one of the most important message is move. Messages are sent to line using UTF-8 coding. An example of a move message from TeleChess is:

```
pawn:D4;
```

Next Pattern

Now that you have specified the bit presentation for messages and mapping how this bit presentation is related to the message logical content, the next issue is how to move a bit string to a peer entity using a lower layer service. This is considered in ***Piggy-packing*** pattern.

3.3.1 Supporting Patterns for Communication Message Specification

These patters are used to support ***Message Transfer Syntax*** pattern.

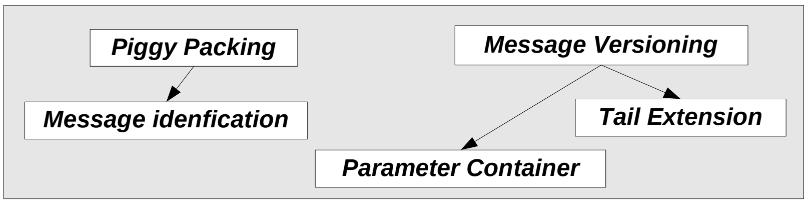


Fig. 14. Support patterns for communication messages specification.

Message Versioning

A.k.a. Prepare for Change

Context

Bugs are found in a protocol specification. New protocol features are introduced. This results in an evolution of the protocol specification.

Problem

Modification of a protocol results in a need to extend or to modify message contents.

Forces

- Backwards-compatibility. (You do not want to update 10.000.000 cellular phones just because a new version of a protocol was just released...)
- Future-proofness. (Of course version 1.0 of a protocol does not have any bugs or need for enhancement...).
- Compactness vs. robustness. Robustness may require auxiliary structure information in messages like encapsulation and identifiers. If bandwidth is scarce then the cost of such extra information and structure may exceed achieved benefits.

Solution

Add version information to every messages. Make clear separation between mechanisms that provide the version information and the actual data carried by the messages. Hide the mechanisms from the view of specifiers that only need to access the information in the messages.

Variations

Obvious and straightforward way is to add message version identification to each message, but if there are many versions of messages the implementation with many version could be complicated. Another common way to versioning is to add to the message a tag to mark new or optional section. Receiver can choose if optional section is considered at all.

Most flexible way is to add to each parameter of message a tag which informs if the parameter in concern is critical or not. Receiver can choose which way message containing unknown parameters is handled. Receiver can drop whole message or only unknown parameters, and make decision based on criticalness of unknown parameters.

To make decision which of these ways to use consider which of the following features are needed:

- addition of new message parameters;
- modification of existing message parameters;
- removal of existing message parameters;
- replacement of existing message parameters;
- need for skipping of unrecognized message parts;
- tree-like version branching, i.e. different protocol versions may evolve in parallel and produce sub-versions.

Running example

To add version number to our running example, message move definition is modified. All messages are sent to line using UTF-8 coding. An example of a move message from TeleChess is:

```
V1:pawn:D4;
```

Next Pattern

To extend messages you can use two different patterns: the ***Parameter Containers*** and the ***Tail Extensions*** patterns.

Tail Extensions

Context

Protocol messages are evolving as shown in ***Message Versioning*** pattern. You don't have lot of bandwidth for messages.

Problem

Protocol messages shall be extended but there is not a lot of bandwidth for message encoding. The extension mechanism should be as light as possible producing a very small encoding.

Solution

Encode all new parameters and extensions to old parameters at the end of a message regardless of their logical position in a message. A receiver that does not know the extensions is designed to ignore the trailing bits.

Variations

One variation of ***Tail Extensions*** is used to replace existing message parameters. In this case you must specify how a device handles old parameters.

- Old parameters are left as garbage. This means that there are two different parameter values for one logical parameter and different protocol versions interpret messages differently.
- Old parameters can be removed. This means that there shall be a presence indication for every parameter even if it is mandatory. If an old parameter is omitted then old protocol implementations do not receive a complete message from their point of view.

Running example

We use ***Tail Extensions*** as follows to add number of move to message. All messages are sent to line using UTF-8 coding. An example of a move message from TeleChess is:

```
V2:pawn:A4:1;
```

Parameter Container

Context

Protocol messages are evolving and they have to extend, as shown in ***Message Versioning*** pattern. In this case, bandwidth is not a problem.

Problem

Protocol messages shall be extended, there are no strict limitations of the use of bandwidth. A receiver shall be able to identify and skip over unknown new parameters.

Solution

Provide a generic container structure, which can be used to encapsulate message parameters. It shall at least contain identification and delimiting for a parameter value. It can also contain different kinds of parameter properties, e.g. if you use a container with ***Message Versioning***, you should add critical or uncritical tags to each container.

Running example

We use ***Parameter Containers*** as follows to add number of move to message. Message format is changed to use XML. One example of move message from TeleChess is:

```
<message>
  <parameter>version<value>3</value></parameter>
  <parameter>move<value>1</value></parameter>
  <parameter>piece<value>pawn</value></parameter>
  <parameter>startPoint<value>D2</value></parameter>
  <parameter>endPoint<value>D4</value></parameter>
</message>
```

Piggy-packing

A.k.a. Data message or Data Transparency

Context

There is a stack of protocol layers. Peer entities communicate with each other using virtual connection. When a message is sent to a peer, it must be sent to a peer entity using lower layer services via real connections.

Problem

You want to keep layers decoupled. The same lower layer could be used with several different kinds of upper layers.

Solution

Provide a data message that is capable to carry encoded upper layer messages. An encoded upper layer message is seen as a payload and it is transparent from the lower layer point of view.

Running example

Example of the usage of ***Piggy-packing*** pattern to send message. In this example TeleChess protocol message is send to other party using email.

```
to: markku.turunen@nokia.com
from: juha.parssinen@vtt.fi
subject: telechess-message

                                                  piggy-packed part
<message>
    <parameter>version<value>3</value></parameter>
    <parameter>move<value>1</value></parameter>
    <parameter>piece<value>pawn</value></parameter>
    <parameter>startPoint<value>D2</value></parameter>
    <parameter>endPoint<value>D4</value></parameter>
</message>
```

Next Pattern

Now you have an encoded message and can send it to the peer entity, the question to consider is how the peer entity identifies the nested upper layer message. This is considered in the ***Message Identification*** pattern.

Message Identification

Context

A lower layer data message carries encoded messages as a byte string.

Problem

An upper layer has to identify what kind of message it has been received.

Forces

- New messages might be introduced in the future.
- How unique shall the identification of a protocol message be? Shall all the messages within a protocol be distinct, or is it enough that only messages that go in one direction are distinct, or are there several protocols that must co-operate in one network node and all the messages shall be distinct.

Solution

Provide a message identification field (or fields) that determine the kind of a message. Such identification can be external to protocol messages or it can be included in the messages themselves.

Variations

In the case of external identification the lower layer data message contains an identifier field in addition to the payload field. Furthermore, a table, which maps identifiers to messages, shall be specified.

In the case of internal identification, the lower layer data message contains just the payload field. It is the responsibility of upper layer messages to identify themselves. Usually there is an auxiliary identifier field in a message. The field is used as a selector over a possible set of messages.

If messages need to be globally distinct then a known mechanism shall be used for the identifier fields. If the need for uniqueness is smaller, then a lighter means can be used (e.g. a closed range of integer values).

Running example

Example the usage of Message Identification pattern to send message. In this example TeleChess protocol message is send to other party using email. This example uses both internal and external identifications. External identification is used to identify that this message is meant to TeleChess (subject: telechess-message) and internal identification is used to identify message type inside TeleChess protocol itself (move).

```
to: markku.turunen@nokia.com
from: juha.parssinen@vtt.fi
subject: telechess-message

<message>
    <id>move</id>
    <parameter>version<value>3</value></parameter>
    <parameter>move<value>1</value></parameter>
    <parameter>piece<value>pawn</value></parameter>
    <parameter>startPoint<value>D2</value></parameter>
    <parameter>endPoint<value>D4</value></parameter>
</message>
```

4 Acknowledgements.

During the years many people have supported us and facilitated our work with protocol patterns with comments, ideas, encouragements, and resources. Especially we like to thank our shepherds who are helped us in this quest: Rick Dewar (EuroPLoP2002), Norm Kerth (EuroPLoP2001), and Michael Stall (PLoP2000).

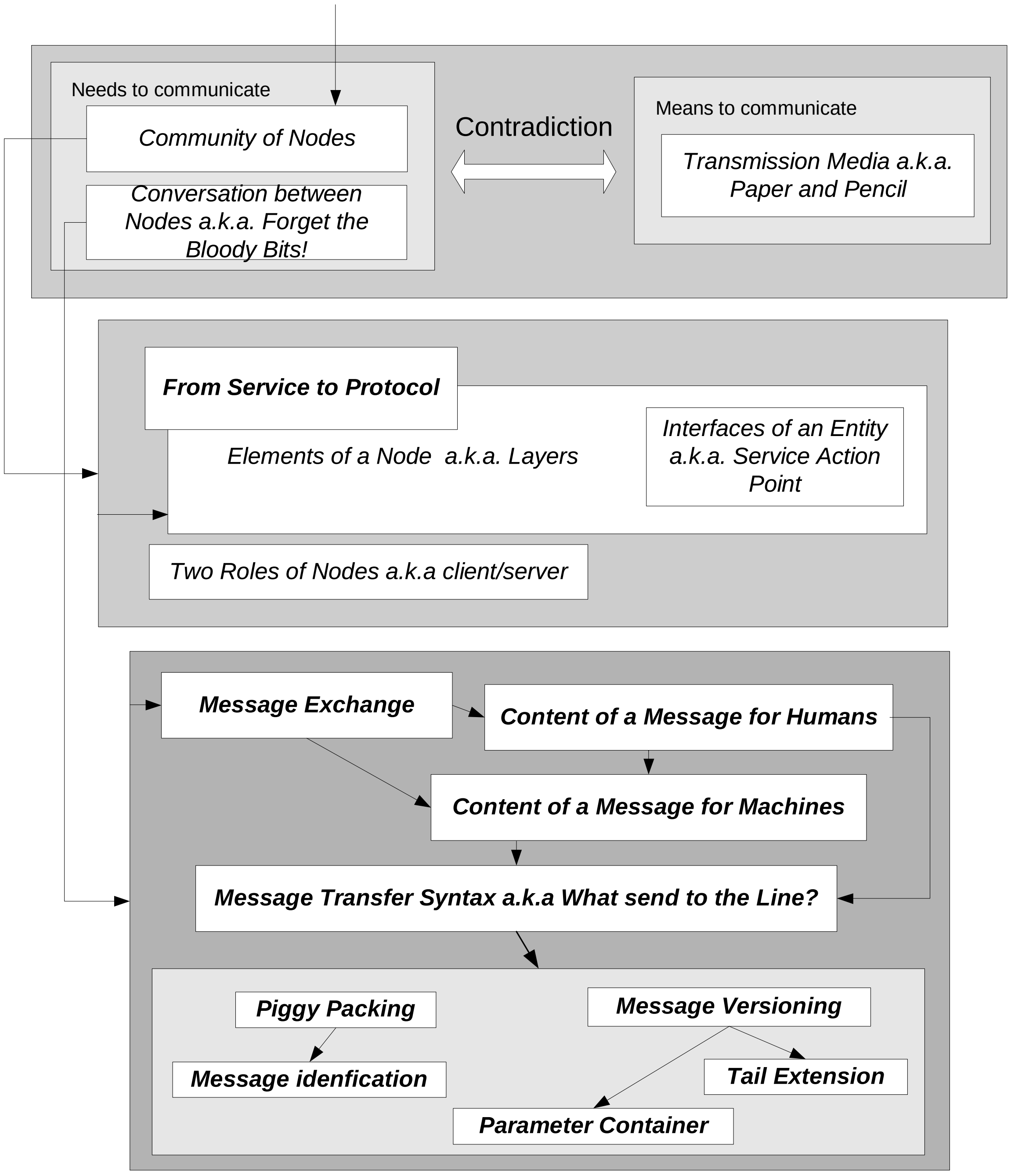
5 References

[1] J. Pärssinen, N. von Knorring, J. Heinonen, M. Turunen, *UML for Protocol Engineering - Extensions and Experience*, Tools Europe 2000, 2000.
[2] J. Pärssinen, M. Turunen, *Patterns for Protocol System Architecture*, a pattern workshop paper presented at PLoP2000, August 13-16, 2000, Allerton Park, Monticello, Illinois, USA.
[3] J. Pärssinen, M. Turunen, *Pattern Language for Architecture of Protocol Systems*, a pattern workshop paper presented at EuroPLoP2001, August 13-16, 2000, 4 - 8 July 2001, Irsee, Germany.
[4] K. Wolf, C. Liu, *New Clients with Old Servers: A Pattern Language for Client/server Frameworks*, Pattern Languages of Program Design 1, pp. 51-64, Addison-Wesley Longman, 1995.
[5] B. Rubel, *Patterns for Generating a Layered Architecture*, Pattern Languages of Program Design 1, pp. 119-128, Addison-Wesley Longman, 1995.
[6] M. T. Rose, *The Open Book, A Practical Perspective on OSI*, Prentice-Hall, 1990.
[7] ITU-T, *Information Technology - Open Systems Interconnection - Basic Reference Model: The Basic Model, Recommendation X.200*, ITU, 1994.
[8] E. Gamma, R. Helm, R. Johnson, J. Vlissides, *Design Patterns - Elements of Reusable Object-Oriented Software*, Addison-Wesley, 1994.
[9] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, M. Stal, *Pattern-Oriented Software Architecture: A System of Pattern*s, Wiley, 1996.
[10]S. Yacoub, H. Ammar, *Finite State Machine Patterns*, Pattern Languages of Program Design 4, pp. 413-440, Addison-Wesley Longman, 2000.
[11]ITU-T, *Abstract Syntax Notation One, X.680-X.693*, ITU, 2002.

Juha Pärssinen can be reached at the VTT Information Technology, P.O.Box 1203, FIN-02044 VTT, Finland; juha.parssinen@vtt.fi

Markku Turunen can be reached at the Nokia Research Center, P.O.Box 407, FIN-00045 NOKIA GROUP; markku.turunen@nokia.com

Appendix A. Pattern Language Road Map



Needs to communicate
Community of Nodes
Conversation between Nodes a.k.a. Forget the Bloody Bits!
Contradiction
Means to communicate
Transmission Media a.k.a. Paper and Pencil
From Service to Protocol
Elements of a Node a.k.a. Layers
Interfaces of an Entity a.k.a. Service Action Point
Two Roles of Nodes a.k.a client/server
Message Exchange
Content of a Message for Humans
Content of a Message for Machines
Message Transfer Syntax a.k.a What send to the Line?
Piggy Packing
Message Versioning
Message idenfication
Tail Extension
Parameter Container

Fig. 15. Pattern Language road map