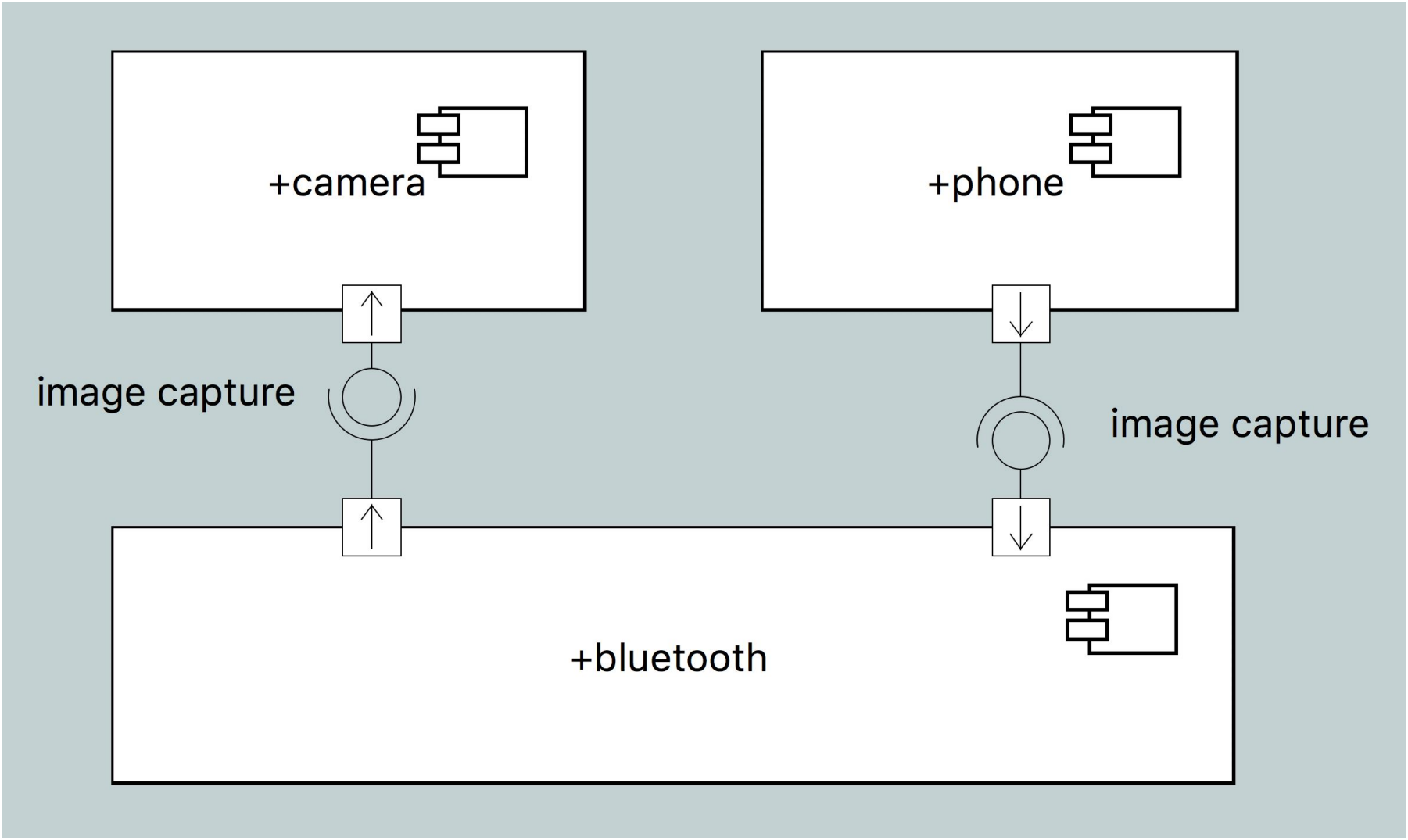
Channel Modeling

using xtUML to build abstract interfaces

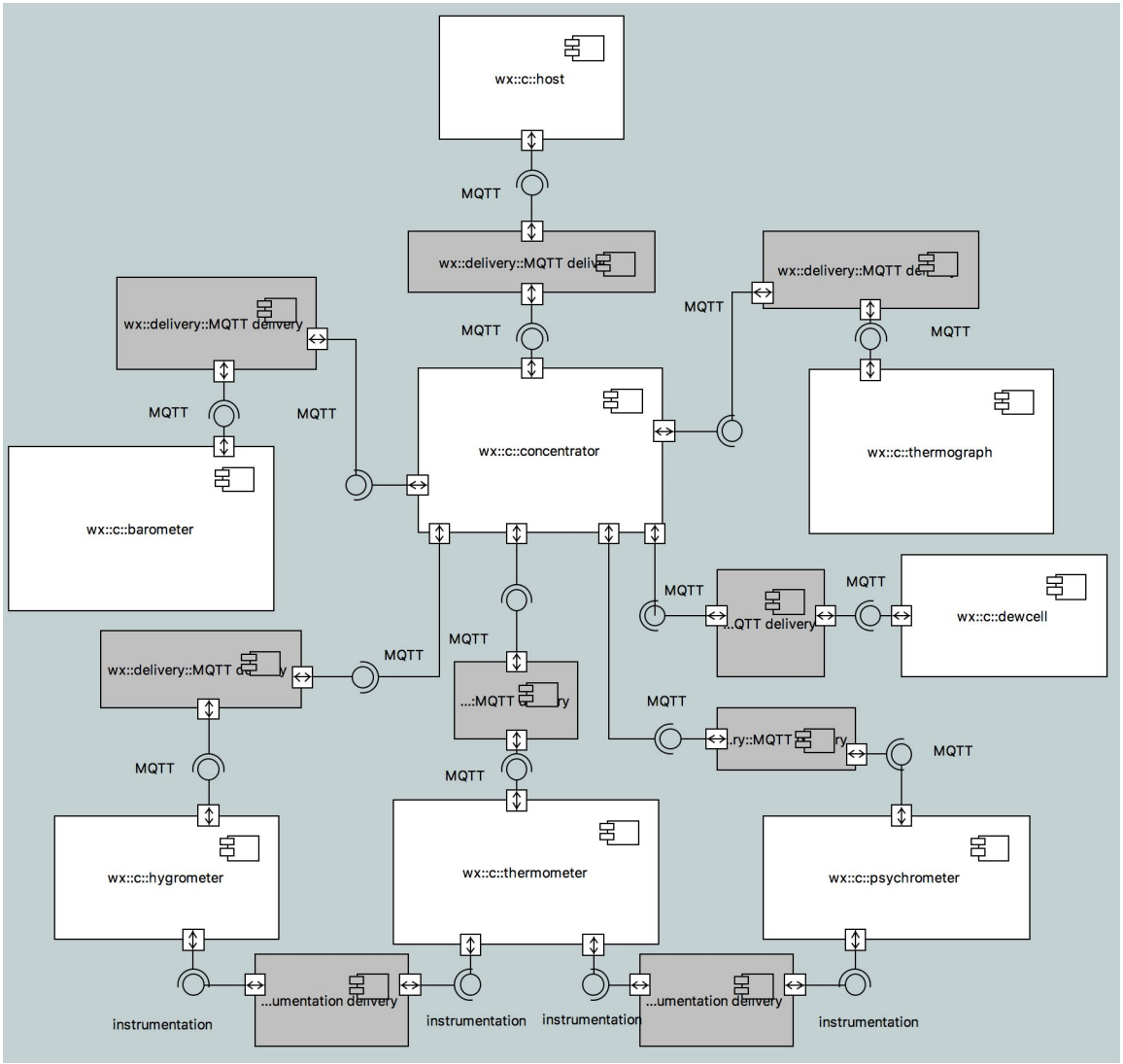


Levi Starrett
October 2017

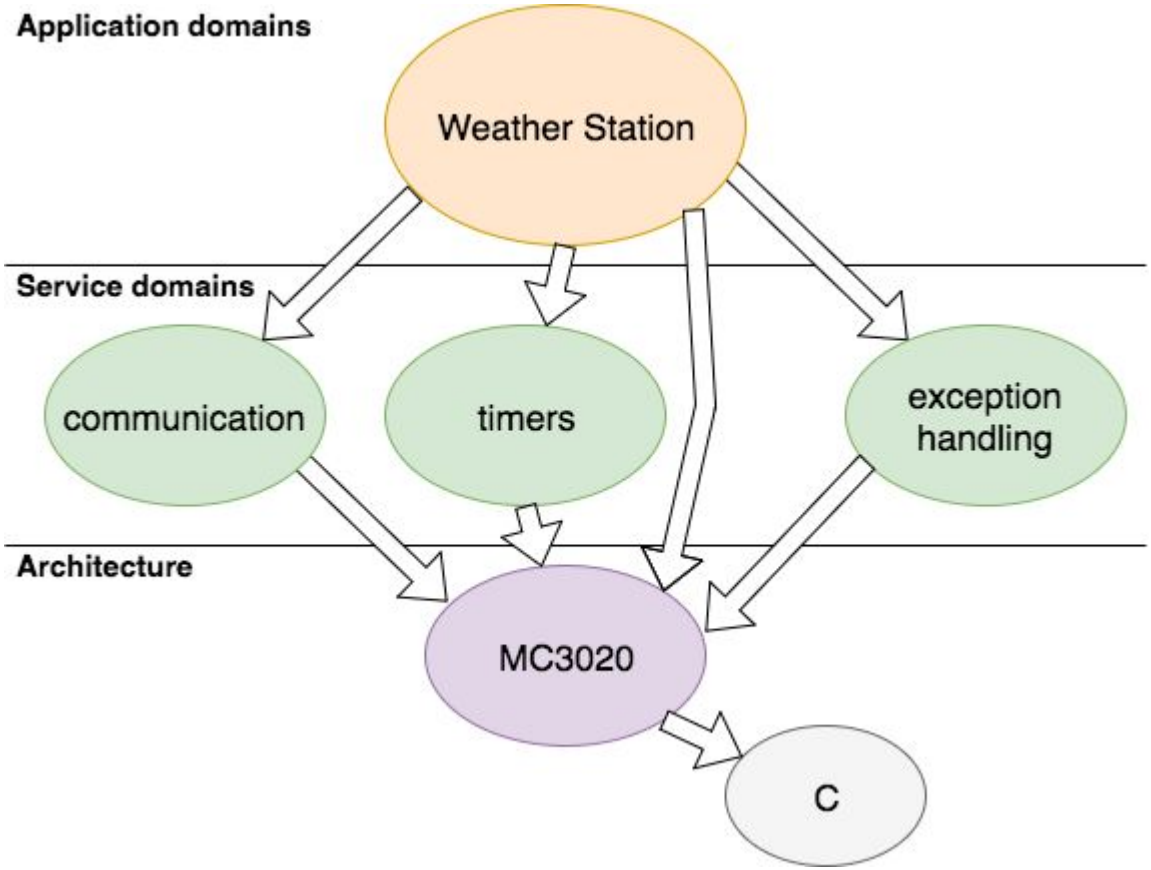
Does this look familiar?



How about this?



Layers of Abstraction

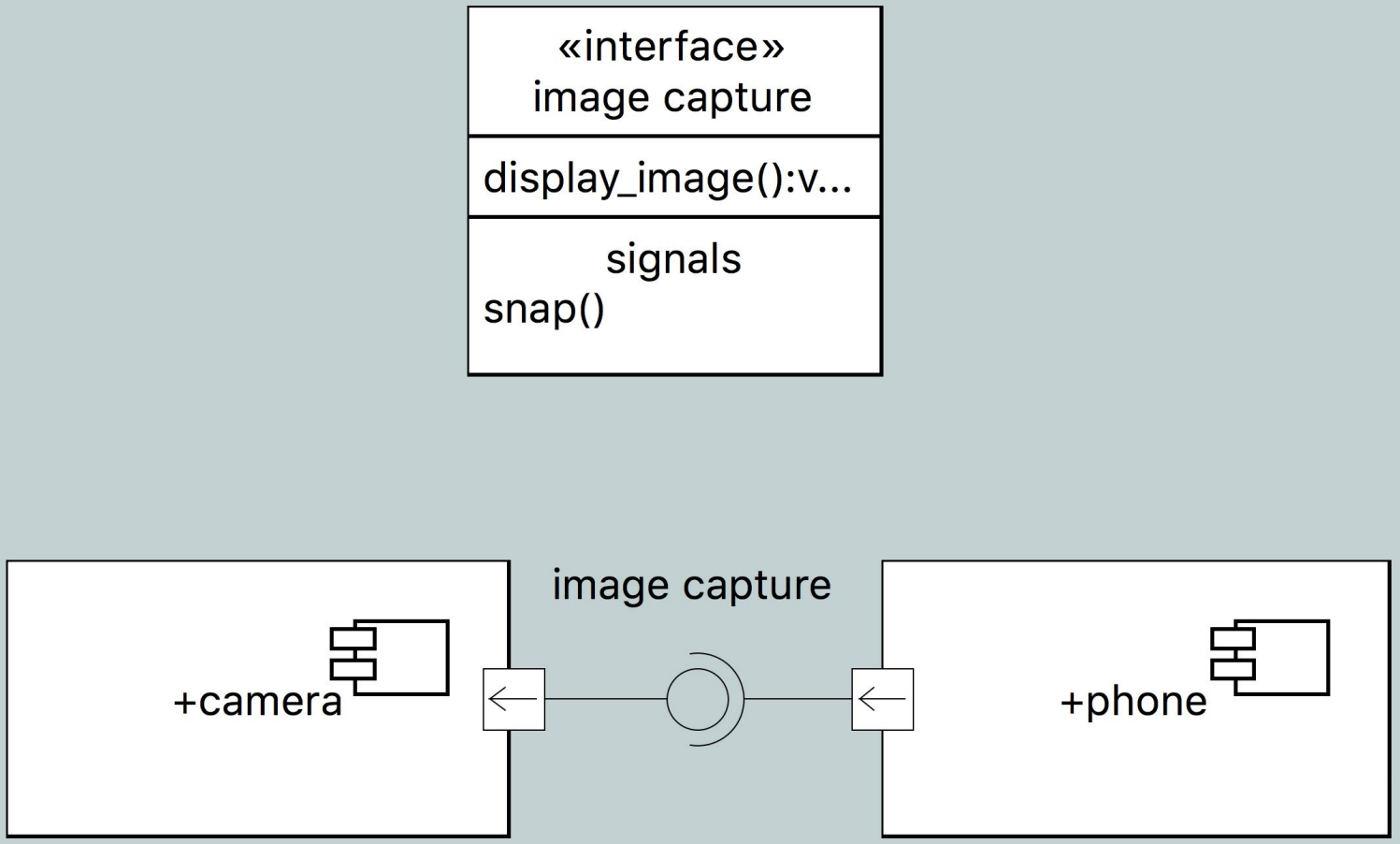


- Application level domains
- Service domains that provide low level services for application domains
- Model compilers define the architectural implementation
- How to model communication apart from application?

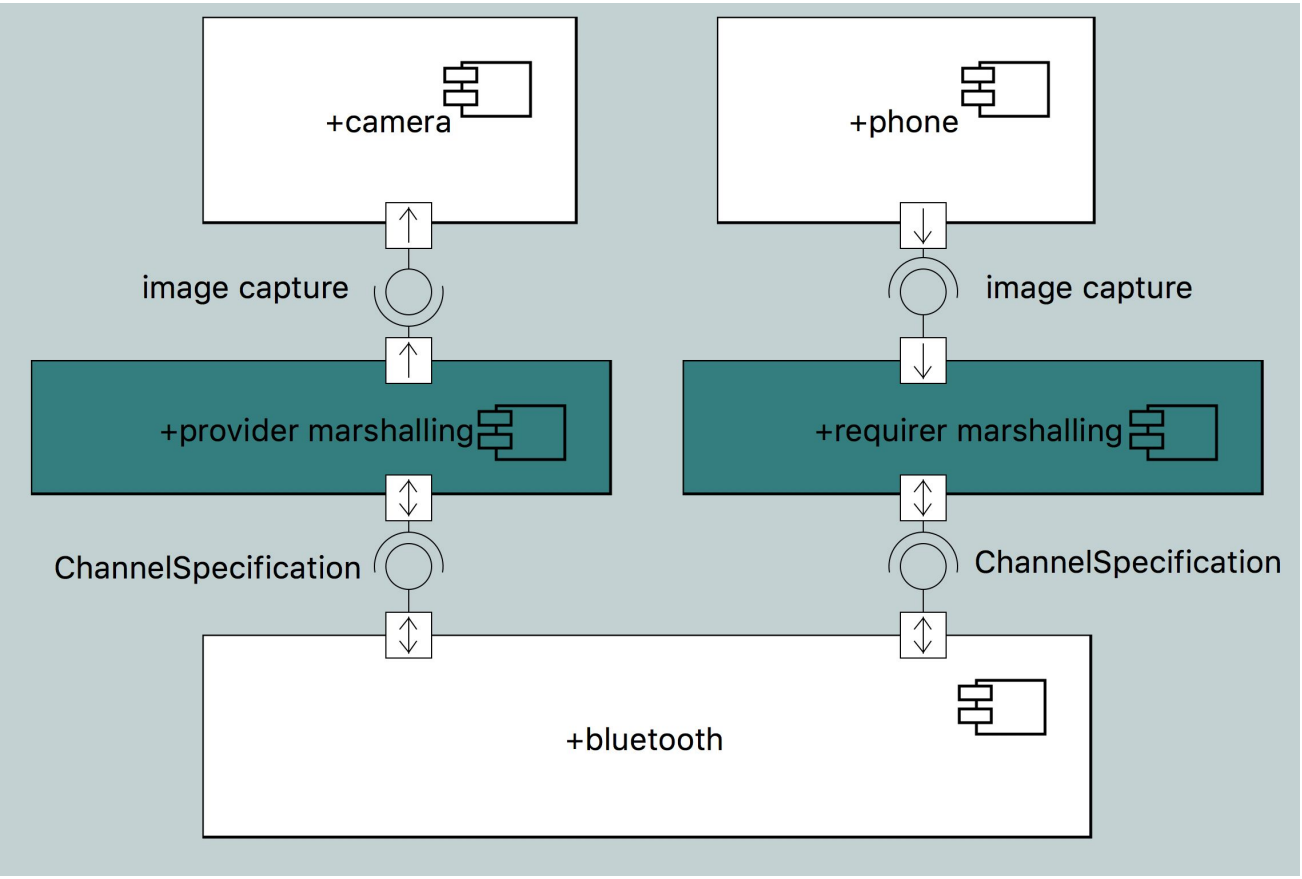
Benefits of abstract communication

- ***Portability***: Models of communication channels can be reused for any number of applications *and* architectures
  - Generalization of interfaces allows communication models to be reused for any interface
  - Communication models can be generated for any architecture
- ***Abstraction***: Communication channels can be modeled at a high level
  - Benefits of modeling in xtUML apply to modeling communication channels
- ***Flexibility***: Internal behavior of communication components is not restricted
  - Any delivery mechanism can be modeled in this way

Application (analyst perspective)

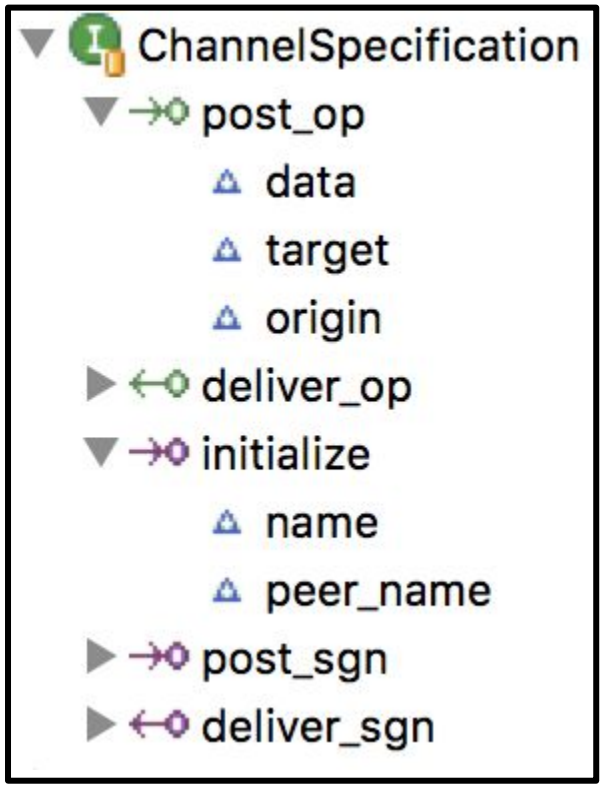


Application (logical representation)



- Marshalling components generated by model compiler
- Ensure arbitrary interfaces are represented to communication domain in a consistent format

"ChannelSpecification" interface



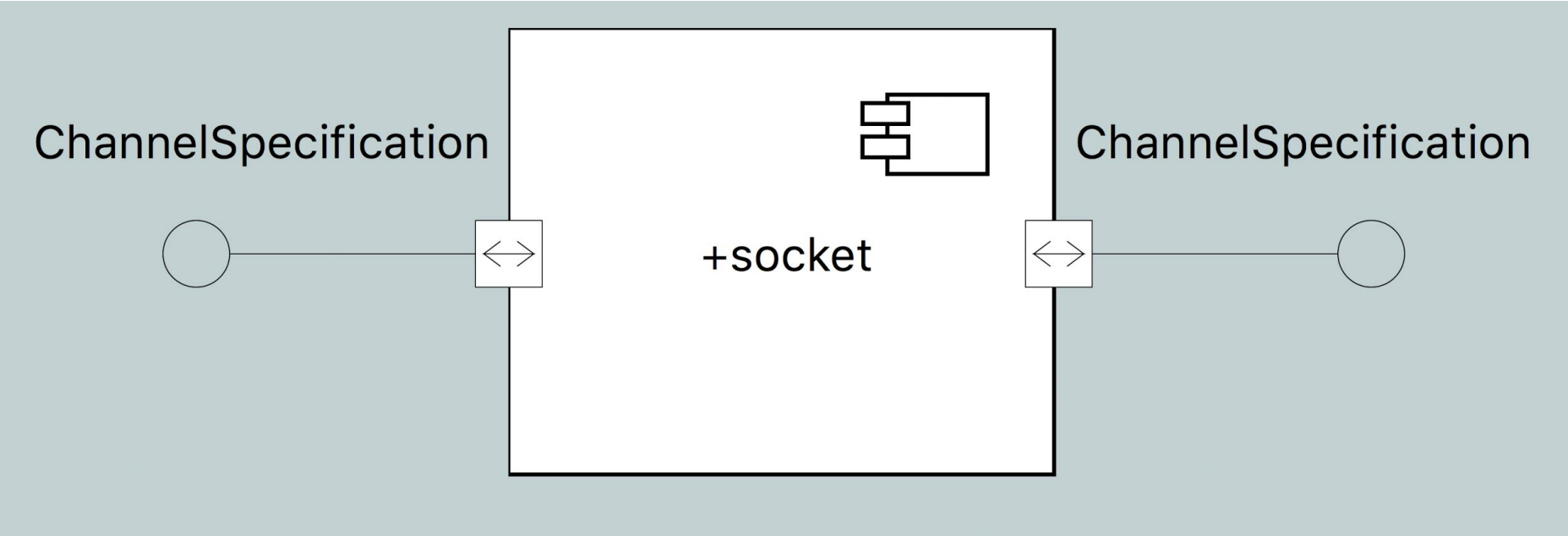
- ***post_op***: to-provider synchronous implementation of message passing
- ***deliver_op***: from-provider synchronous message to requiring component
- ***initialize***: to-provider asynchronous signal to set up channel
- ***post_sgn***: to-provider asynchronous implementation of message passing
- ***deliver_sgn***: from-provider asynchronous signal to requiring component

Data marshalling

- How is data structured
  - C struct?
  - Character arrays?
  - Other?
- Yes!
  - Marshalling is a model compiler concern

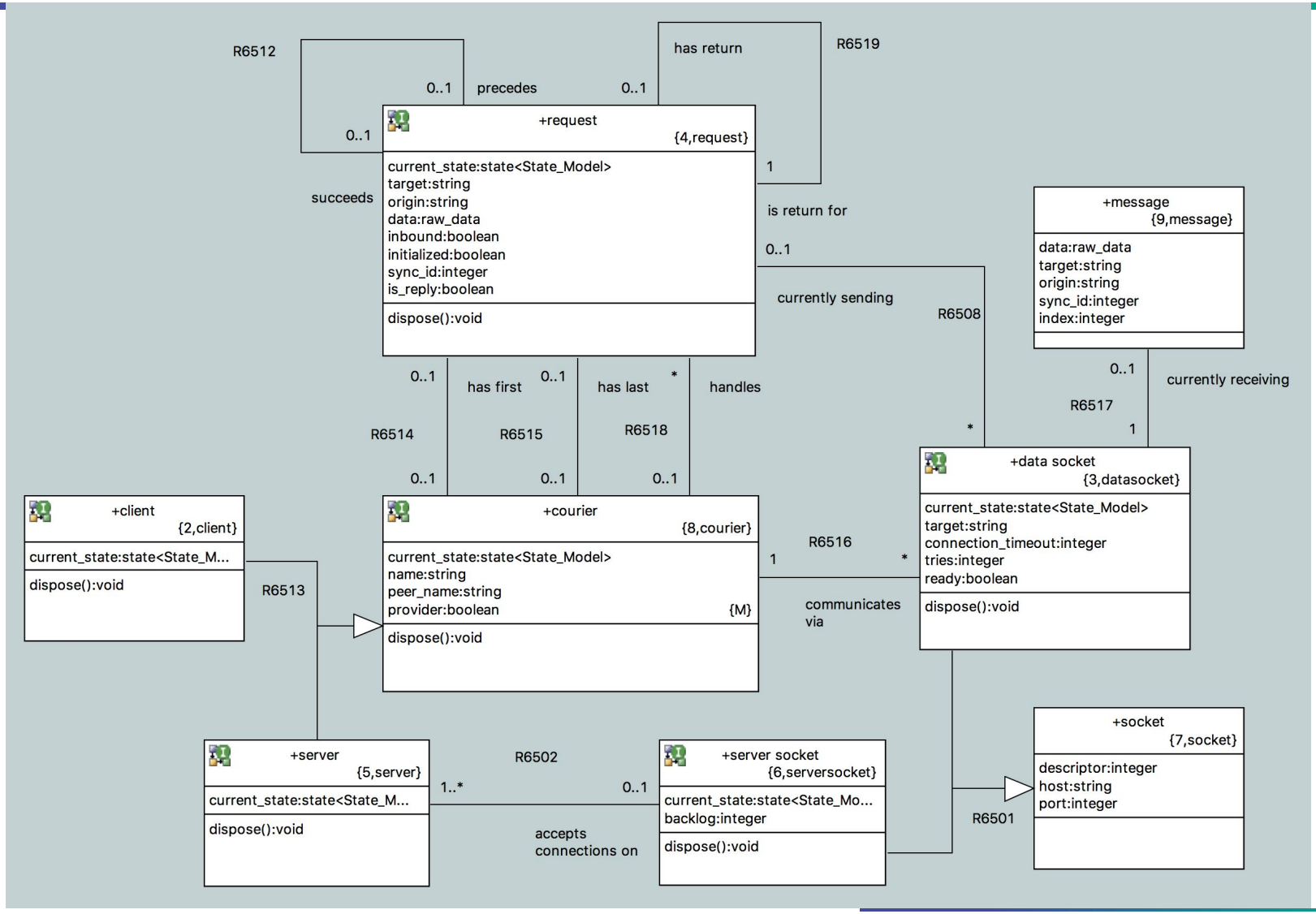


Sockets



- Bonus: must mark project to use at least 2 threads
  - Why?

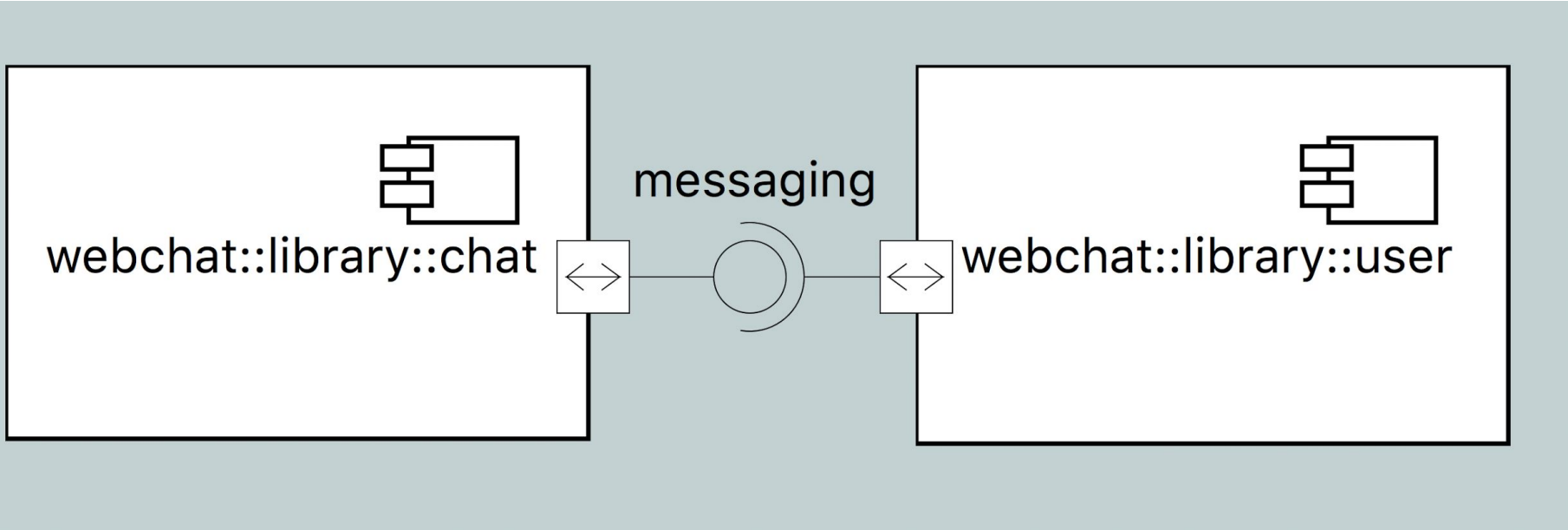
Socket class diagram



Mapping interfaces to socket classes

xtUML element	socket model element	comments
system model	`socket` component	There is exactly one instance of the socket component for a system
component	server socket	Each component that has at least one provided port has exactly one server socket which listens for connections on a user-specified port
required port	client, data socket	Each required port maps to a client and a corresponding data socket. The user specifies an address and port for each requirement. The address and port are the address and port of the target component
provided port	server, data socket(s)	Each provided port maps to a server and 1 to many data sockets. Because provisions can satisfy multiple requirers each data socket maps to a single requirer.
message invocation	requests	Each message invocation maps to at least 2 request. One request is created on the sending side and is passed to the data socket for transport; another is created by the data socket on the receiving side and is passed to the client/server for delivery. When the message is synchronous, there are 4 requests (2 extra for the return value)

webchat



- "chat" provides the messaging service for 0 to many "user"s across an interface implemented by internet sockets

Marking webchat

- Satisfactions marked with name of implementing component
  - Satisfactions identified by label

```
.//
.//============================================================================
.// Mark Channel Implementations
.//
.// This mark is used to specify a modelled implementation of a channel to apply to
.// specific satisfisfaction(s). The component containing the implementation must
.// have two provided ports "INTERFACE_PROVIDER" and "INTERFACE_REQUIRER". Both
.// must implement the "ChannelSpecification" or congruent interface.
.//
.// MarkChannelImplementation ( component_name, satisfaction_label )
.//
.// Where the input parameter(s) are:
.//   component_name = the name of the component that contains the implementation
.//   satisfaction_label = the label of the satisfaction to apply the implementation
.//                        or "*" to mark all satisfactions
.//
.// Example:
.//   .invoke MarkChannelImplementation ( "Socket", "UI::UI::UI -o)- Tracking::UI::UI" )
.//============================================================================
.invoke MarkChannelImplementation ( "socket", "chat::MSG::messaging -o)- user::MSG::messaging" )
.//
```

Another service domain: Security

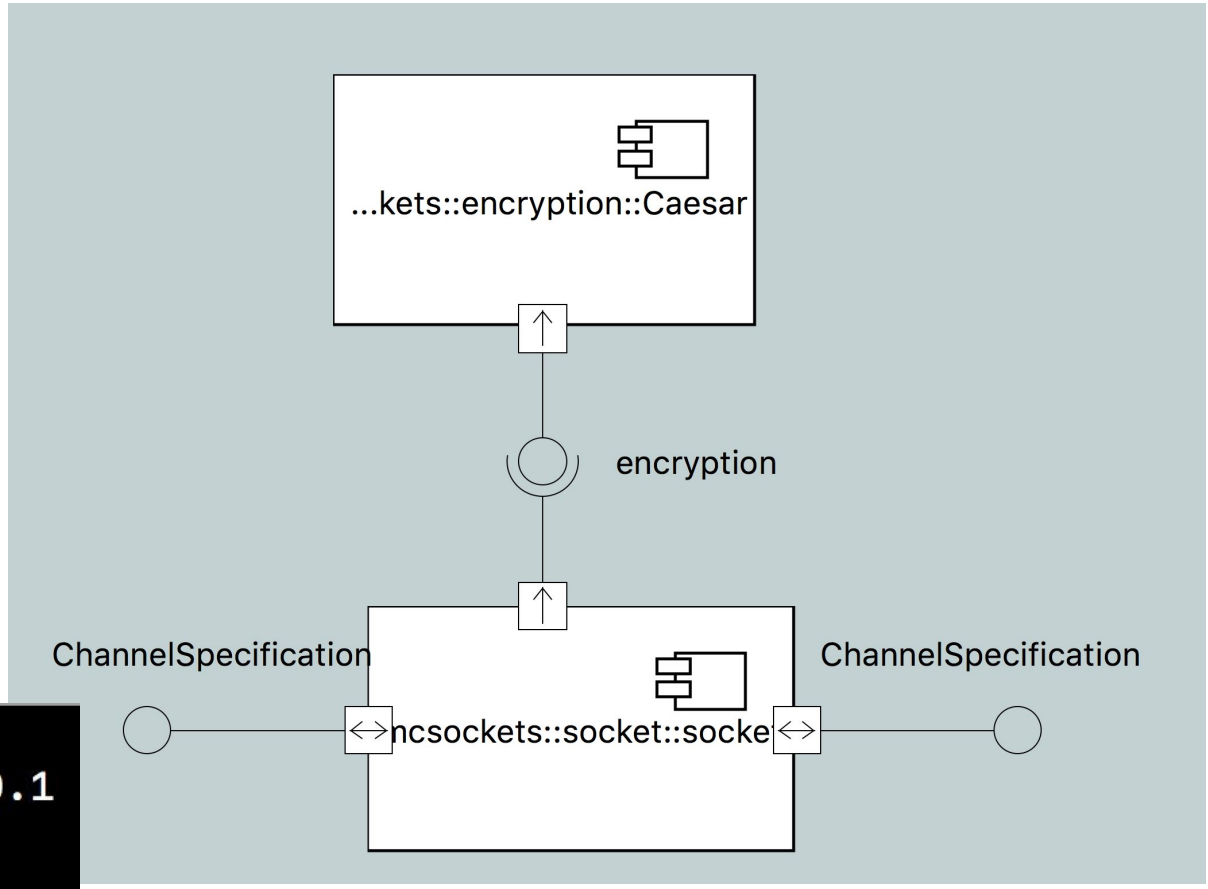
- Authentication
- Encryption
- Discovery

- Is security part of communication?
  - What do you think?



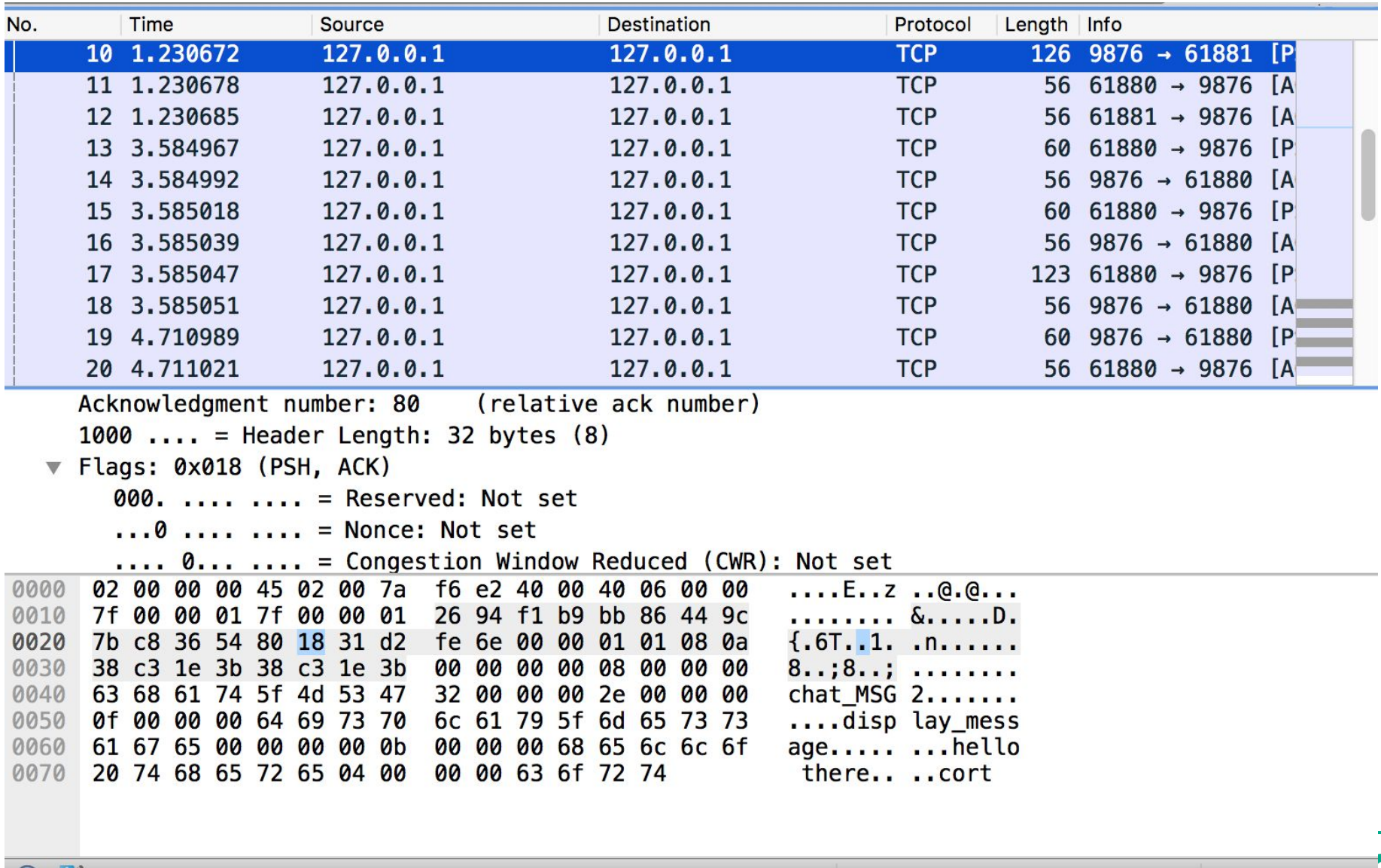
Encrypting webchat

- Encryption as a service domain of socket
- Simple Caesar cipher (rot13)
- Encryption enabled through config file




```
1 # Client parameters
2 server_host, 127.0.0.1
3 server_port, 9876
4 timeout, 10
5 encryption, false
```

Sniffing the wire — plaintext



No.	Time	Source	Destination	Protocol	Length	Info
10	1.230672	127.0.0.1	127.0.0.1	TCP	126	9876 → 61881 [P
11	1.230678	127.0.0.1	127.0.0.1	TCP	56	61880 → 9876 [A
12	1.230685	127.0.0.1	127.0.0.1	TCP	56	61881 → 9876 [A
13	3.584967	127.0.0.1	127.0.0.1	TCP	60	61880 → 9876 [P
14	3.584992	127.0.0.1	127.0.0.1	TCP	56	9876 → 61880 [A
15	3.585018	127.0.0.1	127.0.0.1	TCP	60	61880 → 9876 [P
16	3.585039	127.0.0.1	127.0.0.1	TCP	56	9876 → 61880 [A
17	3.585047	127.0.0.1	127.0.0.1	TCP	123	61880 → 9876 [P
18	3.585051	127.0.0.1	127.0.0.1	TCP	56	9876 → 61880 [A
19	4.710989	127.0.0.1	127.0.0.1	TCP	60	9876 → 61880 [P
20	4.711021	127.0.0.1	127.0.0.1	TCP	56	61880 → 9876 [A

```
Acknowledgment number: 80    (relative ack number)
1000 .... = Header Length: 32 bytes (8)
▼ Flags: 0x018 (PSH, ACK)
   000. .... .... = Reserved: Not set
   ...0 .... .... = Nonce: Not set
   .... 0... .... = Congestion Window Reduced (CWR): Not set
```

```
0000  02 00 00 00 45 02 00 7a  f6 e2 40 00 40 06 00 00   ....E..z ..@.@...
0010  7f 00 00 01 7f 00 00 01  26 94 f1 b9 bb 86 44 9c   ........ &.....D.
0020  7b c8 36 54 80 18 31 d2  fe 6e 00 00 01 01 08 0a   {.6T..1. .n......
0030  38 c3 1e 3b 38 c3 1e 3b  00 00 00 00 08 00 00 00   8..;8..; ........
0040  63 68 61 74 5f 4d 53 47  32 00 00 00 2e 00 00 00   chat_MSG 2.......
0050  0f 00 00 00 64 69 73 70  6c 61 79 5f 6d 65 73 73   ....disp lay_mess
0060  61 67 65 00 00 00 00 0b  00 00 00 68 65 6c 6c 6f   age..... ...hello
0070  20 74 68 65 72 65 04 00  00 00 63 6f 72 74          there.. ..cort
```

Sniffing the wire — ciphertext

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	127.0.0.1	127.0.0.1	TCP	60	61878 → 9876 [P
2	0.000048	127.0.0.1	127.0.0.1	TCP	56	9876 → 61878 [A
3	0.000071	127.0.0.1	127.0.0.1	TCP	84	61878 → 9876 [P
4	0.000078	127.0.0.1	127.0.0.1	TCP	56	9876 → 61878 [A
5	0.000086	127.0.0.1	127.0.0.1	TCP	103	61878 → 9876 [P
6	0.000093	127.0.0.1	127.0.0.1	TCP	56	9876 → 61878 [A
7	1.126079	127.0.0.1	127.0.0.1	TCP	60	9876 → 61877 [P
8	1.126127	127.0.0.1	127.0.0.1	TCP	56	61877 → 9876 [A
9	1.126147	127.0.0.1	127.0.0.1	TCP	60	9876 → 61878 [P
10	1.126149	127.0.0.1	127.0.0.1	TCP	126	9876 → 61877 [P
11	1.126163	127.0.0.1	127.0.0.1	TCP	56	61878 → 9876 [A

```
.... .0.. .... = ECN-Echo: Not set
.... ..0. .... = Urgent: Not set
.... ...1 .... = Acknowledgment: Set
.... .... 1... = Push: Set
.... .... .0.. = Reset: Not set
.... .... ..0. = Syn: Not set
```

```
0000  02 00 00 00 45 00 00 7a  7a 5d 40 00 40 06 00 00   ....E..z z]@.@...
0010  7f 00 00 01 7f 00 00 01  26 94 f1 b5 34 ef d6 59   ........ &...4..Y
0020  b6 db 4d 11 80 18 31 d5  fe 6e 00 00 01 01 08 0a   ..M...1. .n......
0030  38 c0 e7 09 38 c0 e7 09  00 00 00 00 08 00 00 00   8...8... ........
0040  63 68 61 74 5f 4d 53 47  32 00 00 00 2e 00 00 00   chat_MSG 2.......
0050  0f 00 00 00 71 76 66 63  79 6e 6c 5f 7a 72 66 66   ....qvfc ynl_zrff
0060  6e 74 72 00 00 00 00 0b  00 00 00 75 72 79 79 62   ntr..... ...uryyb
0070  20 67 75 72 65 72 04 00  00 00 70 62 65 67          gurer.. ..pbeg
```



onefact.net