Analyzing and detecting problems in Systems of Systems

Mikael Lindvall[1] and Christopher Ackermann[2]
Fraunhofer USA – Center for Experimental Software Engineering, College Park, MD, 20740-3290

William C. Stratton[3] and Deane E. Sibol[4]
Johns Hopkins University Applied Physics Laboratory Space Department Ground Applications Group (APL)
Laurel, MD, 20723-8099

Sally Godfrey[5]
Code 583, Bldg 23, E215
Goddard Space Flight Center (GSFC), Greenbelt, MD 20771

Many software systems are evolving complex system of systems (SoS) for which inter-system communication is mission-critical. Evidence indicates that transmission failures and performance issues are not uncommon occurrences. In a NASA-supported Software Assurance Research Program (SARP) project, we are researching a new approach addressing such problems. In this paper, we are presenting an approach for analyzing inter-system communications with the goal to uncover both transmission errors and performance problems. Our approach consists of a visualization and an evaluation component. While the visualization of the observed communication aims to facilitate understanding, the evaluation component automatically checks the conformance of an observed communication (actual) to a desired one (planned). The actual and the planned are represented as sequence diagrams. The evaluation algorithm checks the conformance of the actual to the planned diagram. We have applied our approach to the communication of aerospace systems and were successful in detecting and resolving even subtle and long existing transmission problems.

I. Introduction

MANY software systems are complex system of systems (SoS) for which inter-system communication is both mission-critical and error-prone. Software failures in the communication between the participating systems in a SoS, e.g. between Flight Software and the Ground System, can cripple system capabilities, cause loss of data, and lead to mission failure. Inter-system communication problems ideally would be detected before deployment, but current state-of-the-art technologies do not easily support their detection.

An analysis of APL's Common Ground System, the system analyzed in this paper, quickly identified 15 trouble reports related to problems with inter-system communication that had adverse mission impacts and for which there were no workarounds. Inter-system communication is conducted through system interfaces and is often the source of problems. One reason for such problems is that the different systems are often developed by different teams with different interpretations of interface specifications. Individual developers may occasionally attempt to build in

[1] Director, Software Architecture Division, Fraunhofer, 4321 Hartwick rd, suite 500, College Park, MD, 20740
[2] Scientist, Software Architecture Division., Fraunhofer, 4321 Hartwick rd, suite 500, College Park, MD, 20740
[3] Senior Professional Staff, Space Dep, JHU/APL, 11100 Johns Hopkins Road, MS 4-118, Laurel, MD 20723
[4] Senior Professional Staff, Space Dep., JHU/APL, 11100 Johns Hopkins Road, MS 4-118, Laurel, MD 20723
[5] Member NASA Software Working Group, Goddard Space Flight Center (GSFC), Greenbelt, MD

[5] Senior Professional Staff, Space Dep., JHU/APL, 11100 Johns Hopkins Road, MS 4-118, Laurel, MD 20723
[5] Member NASA Software Working Group, Goddard Space Flight Center (GSFC), Greenbelt, MD

support to programmatically check that interface specifications are followed, but there is often no systematic way for architects or V&V teams to describe and check these interface specification rules in a consistent manner across a SoS. Previous efforts have analyzed static architectures [1] and have led to the development of a general understanding of dynamic architectures of small software systems [2], but they do not typically address the inter-system communication problems APL has been facing.

In a NASA-supported Software Assurance Research Program (SARP) project called Architecture Analysis of Evolving Complex Systems of Systems, we are researching a new approach that will explicitly address such problems by providing automated support to check system communication across a SoS.

In previous papers, we described the background of this SARP project and the problems we are addressing. We also described aspects of the proposed solution, and some of the results from a study we conducted to determine the feasibility of the proposed solution by manually comparing planned communications with actual ones [5, 6]. In this paper, we describe some of the features of the automated approach and provide diagrams that show examples of the tool output that helped detect defects in the inter-system communications.

III. Technology characteristics

The technology that is developed as part of this project addresses problems related to interfaces and the communication profiles of systems that communicate at runtime. The technology allows the user to define and navigate the expected (a.k.a. planned, specified, desirable, ideal, baselined etc.) communication profile of a system as well as comparing it to the actual communication profile so that s/he may evaluate whether or not the system communication conforms to its specifications.

Interface rules form the basis of the planned communication profile. These interface rules are derived from already existing interface specifications typically found in Interface Control Documents (ICDs). In these specifications, e.g. the order of calls and events, the size and format of data, as well as timing information are specified. We are using sequence diagrams to describe the planned order of messages. Assertions are used describe other aspects of the communication profiles. We collect data from the system during run-time and automatically compare the planned dynamic profile with run-time data in order to detect deviations between the two indicating potential problems.

This functionality is provided as an extension to Fraunhofer's SAVE tool [1], which visualizes and compares the implemented software architecture (actual) with its planned architecture based on static analysis.

We have implemented a prototype for visualizing the observed communications and for checking the conformance of that communication to a set of rules. It automatically annotates a sequence diagram with icons that signal deviations from the planned communication. The example diagrams in this paper were produced using that prototype.

IV. The Common Ground System (CGS)

The system analyzed in this paper is APL's Common Ground System (CGS). All of APL's NASA missions use the CGS for spacecraft I&T and operations. CGS is currently supporting operations for three deep space missions: MESSENGER, STEREO, and New Horizons. Flight software, scientific data processing software, and ground equipment software interface with CGS and depend on its services. CGS [4] consists of 83 different systems (applications). These applications are developed, compiled, and launched independently from each other, and participate in a reusable pipe-and-filter architecture established during run-time. Figure 1 illustrates some of CGS applications and how they are related to each other.

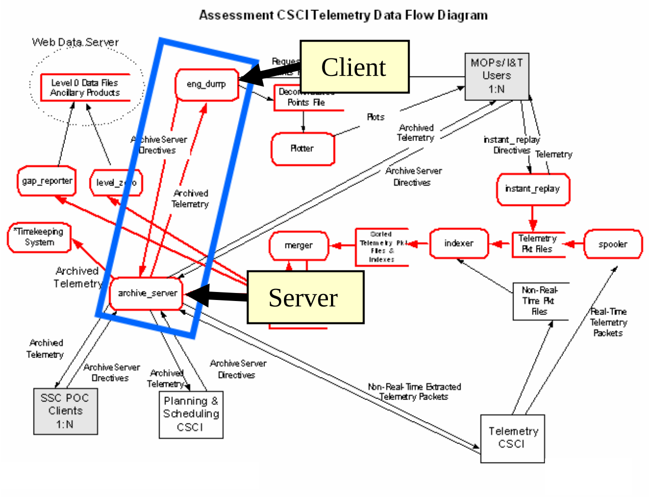


Figure 1 – ArchiveServer and EngDump in Common Ground's pipe and-filter architecture

In this paper, we focus our examination on two applications that are representative of the Common Ground software architecture and which belong to the Assessment sub-system. The Archive Server, which is an application that serves selected telemetry packets from the archive, and the Engineering Dump, which is one of several clients of the archive server that all extract selected telemetry data from the archived packets, converts the raw telemetry data into engineering units for further analysis.

Other teams often develop Archive Server clients and since there is little or no communication between the different teams, interface problems are introduced. APL conducts integration testing to a large extent, but problems still occur in operations because of incorrect use of communication protocols. Communication issues often remain in the system for a long time as they are subtle and difficult to detect.

There are different types of communication rules systems must follow and if violated might lead to transmission problems. We have conducted an analysis of existing ICD's of several protocols and discussed communication problems with CGS architects to identify the kind of rules that need to be addressed by our evaluation approach. As a result of this process we distinguish the follow rule types:

- Message sequencing
- Message content
- Message timing

Systems interact by sending a well-defined sequence of messages. Problems occur when systems do not follow that sequence and send unexpected messages or fail to send a certain message. If one systems sends formulates a request to another system, it expects the data that is sent in response to conform to the criteria specified in the request. Such expectations can be expressed in assertions. Lastly, the messages must adhere to timing rules. This is necessary in order to ensure that the system operates in a time efficient manner and to assert whether the systems adhered to pre-defined time-outs.

V. Implementation

We have implemented a prototype for visualizing captured communication traces and for automatically evaluating the conformance of that communication to sequencing and content rules. We are planning to extend our approach by adding algorithms for timing evaluation in the future. The prototype visualizes an actual communication in a sequence diagram that displays not only the sequence of messages but also information about the message contents and temporal properties. The sequence diagram representing the actual communication can then be compared to another similar diagram representing the planned communication behavior. We have implemented a set of algorithms to check the conformance of the message sequence to the sequencing rules and to evaluate whether the message content adheres to some pre-defined constraints. These constraints are specified as assertions in the planned sequence diagram.

The evaluation components takes as input the planned sequence diagram and the actual sequence with its sequencing and content information. During evaluation, it annotates each message in the diagram with symbols indicating the kind of violation that occurred or if no violation was detected. In particular, the evaluation produces four different annotations:

- ✓ *No error* was detected
- ❗ An *extra message* was detected
- ✖ A *missing message* was detected
- ⚡ A violation of a *content rule* was detected

We believe that producing sequence diagram annotations is more desirable than a list of error reports since (1) the user can quickly identify the kind of error that occurred at a certain point and (2) in which context that error was detected. The context here refers to the point in the protocol, i.e. the communication behavior before and after the error has occurred.

VI. The Study

We studied whether our proposed solution would be feasible for detecting problems stemming from deviations from interface control documents. More specifically, we studied whether it would be possible to compare a planned sequence diagram with actual sequences in order to detect such deviations. We were especially interested in identifying different kinds of communication problems, for example in terms of how much modeling would be necessary to detect them. Software development teams often lacks time and resources and if such modeling requires too much effort, chances are that a new technology that relies on such modeling will never be used.

The study was conducted in the following way.

(1) The APL team produced a sequence diagram that specifies the planned communication between the server and the client. This sequence diagram was based on information provided in the ICD. See Figure 2 and 5.

(2) The APL team captured dynamic data from a correct communication between server and the client. The communication was correct in that sense that it matches the planned sequence diagram, see Figure 3.

(3) The FC-MD team developed a parser based on the ICD and the dynamic data provided by APL. The parser reads the dynamic data and outputs the messages that were sent between the two applications. For each message, the timestamp, the message type, and the message content were extracted.

(4) Once it was determined that the parser worked correctly, the APL team produced a set of three communication sequences that each was not compliant with the planned sequence in one of the following ways: 1. There were missing messages, 2) there were extra messages, or 3) there were messages whose parameter values were inconsistent with the specification. The defects were specified by the APL team as well as the actual and correct system behavior. See Figure 4, 6, and 7.

(5) The FC-MD team imported the sequences into SAVE and used the new SAVE prototype extension that automatically compared the planned sequence diagram to each of the actual sequence diagrams.

(6) Deviations between the planned and the actual sequences were analyzed and reported to the APL team.

(7) The APL team determined whether the detected deviations were true or false.

(8) The APL team and the FC-MD team discussed the feasibility of the proposed approach and potential improvements to make it useable in a “live” situation at APL.

The next subsections will illustrate the approach in four sequence diagrams. First, the specification will be presented, and then the evaluation and analysis results are discussed.

A. Specification: The protocol as an abstract sequence diagram

The clients and the server communicate using a protocol that specifies four different types of messages:

(1) The client defines a set of *filters* that together specifies the type of data that it requests the server to return. Examples of filters are: Type of data, e.g. STP or TP, and Time range, specified by start time and stop time. The filters from the client to the server can be sent in arbitrary order.

(2) When the client has specified, using filters, what data to download from the server, it sends a *BeginPlayBack* command to the server. Once this command has been issued, the client is not expected to send more messages.

(3) When the BeginPlayBack message has been received by the server, the server starts sending *data* messages. Each data message must match the filter specification received earlier. For example, the type of each data message must be as specified and the time stamp of each message must fall within the specified time range.

(4) When there are no more messages, the server sends an *End Of Transmission* (EOT) message to the client signaling that the data transmission is complete; thereafter the communication link is closed.

Since the goal is to develop a modeling and evaluation method that requires limited effort, we started by modeling the protocol using an abstract sequence diagrams without as little detail as possible, see Figure 2. This model models the facts that there might be any number of filters followed by one BeginPlayback message, and that there might be any number of data messages followed by one EOT message. Thus, this abstract model does not provide any information about the type of filters or data messages that we expect to occur.

B. Evaluation based on the high-level sequence diagram

The next step was to evaluate sequences based on the planned sequence diagram. We started by applying it to the first sequence that we received. This sequence is expected to be correct, i.e. we expected it to match the specification without any extra or missing messages. The evaluation result is provided in Figure 3. As one can see in that figure, the final message EOT appears to be missing from the actual sequence. This is surprising because this sequence was supposed to be correct. The APL team analyzed the original ‘nominal case’ data and even re-ran the test example to verify that everything was correct. Still, it was difficult to understand why the EOT is not being sent from the archive server nor how the client knows when to close the socket. A missing EOT can be a significant problem because clients are expected to close the socket to the server once they have received all the requested data, as indicated by the EOT message. If the EOT message is not being used for its intended purpose, clients may be employing some other means of recognizing that all the resulting data has been received. For example, they might check that the time stamp on the data is equal to or exceeds the stop time specified. For a variety of reasons, including timestamp precision, multiple of packets from different paths and/or sources with the same timestamp, etc., this may lead to the client prematurely closing the socket and missing data subsequently to be returned by the server. During extended analysis, the APL team realized that the missing EOT message is actually not missing at all. Because the TCP packets, as reported by snoop (i.e. the network protocol analyzer we used to capture the data), can aggregate data, the EOT message actually does occur at the end of the last TCP packet from the server to the client (i.e., the server produced two write statements for the STP and EOT, but it ended up as one TCP packet). This is perfectly legal, with Nagles algorithm enabled. Our issue in the analysis was that we were treating each TCP packet as a message. This illustrates that our parsing algorithm has to be able to detect aggregated data situations.

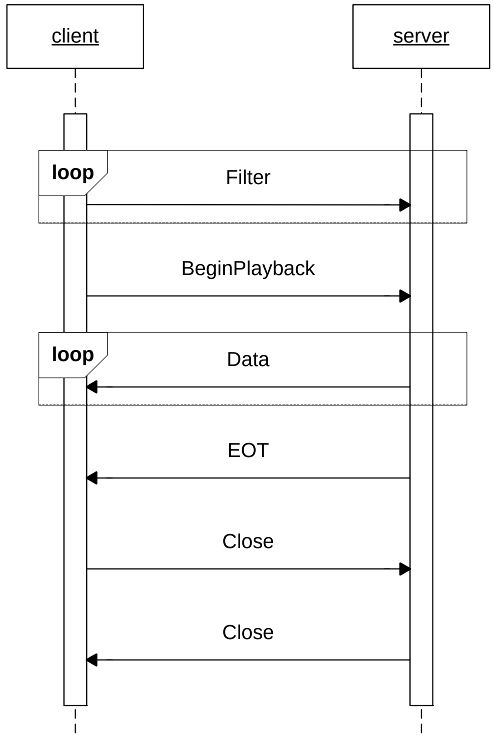


Figure 2 - Abstract sequence diagram

In the next example, it was detected that a filter change/addition was sent from the client to the server after BeginPlayback was sent and while data from server to client was flowing, see Figure 4. This filter message is ignored by the server; in fact, it might not even get read off the socket. This might cause problems because if the client continues to write to the buffer without the server reading from it, the buffer will eventually be full. If the client blocks and is not multi-threaded, it will be stuck and may not ever read the data being sent by the server. There is no timeout on the connection so in such a case, the client might occupy the connection for a long time. In addition, the client might expect the ignored filter to be in effect and thus receives the wrong data.

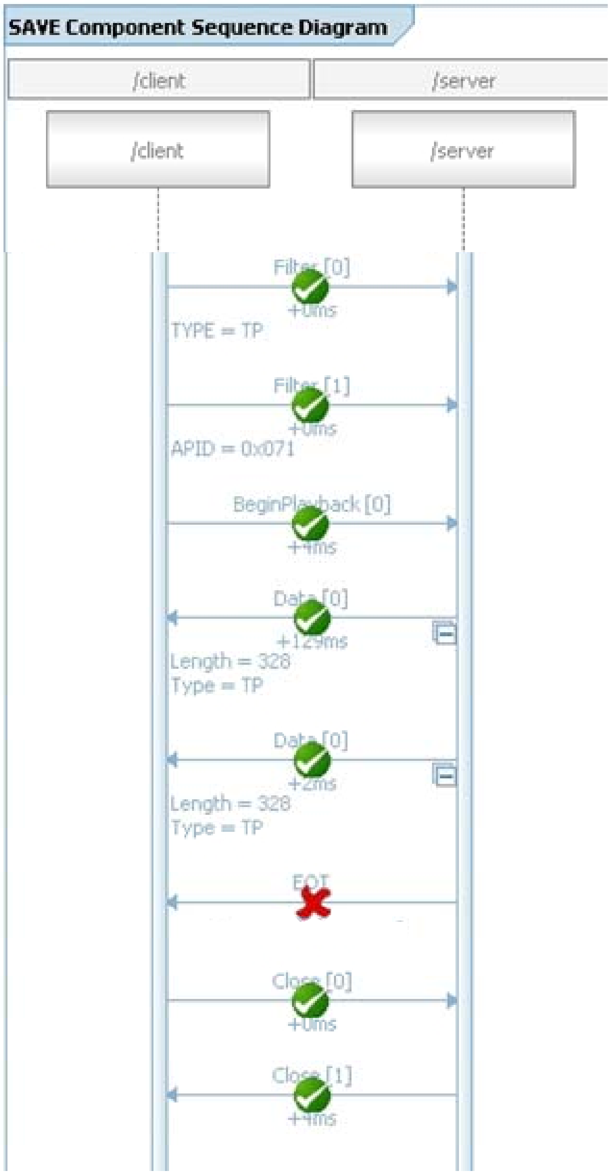


Figure 3 - EOT is undetected

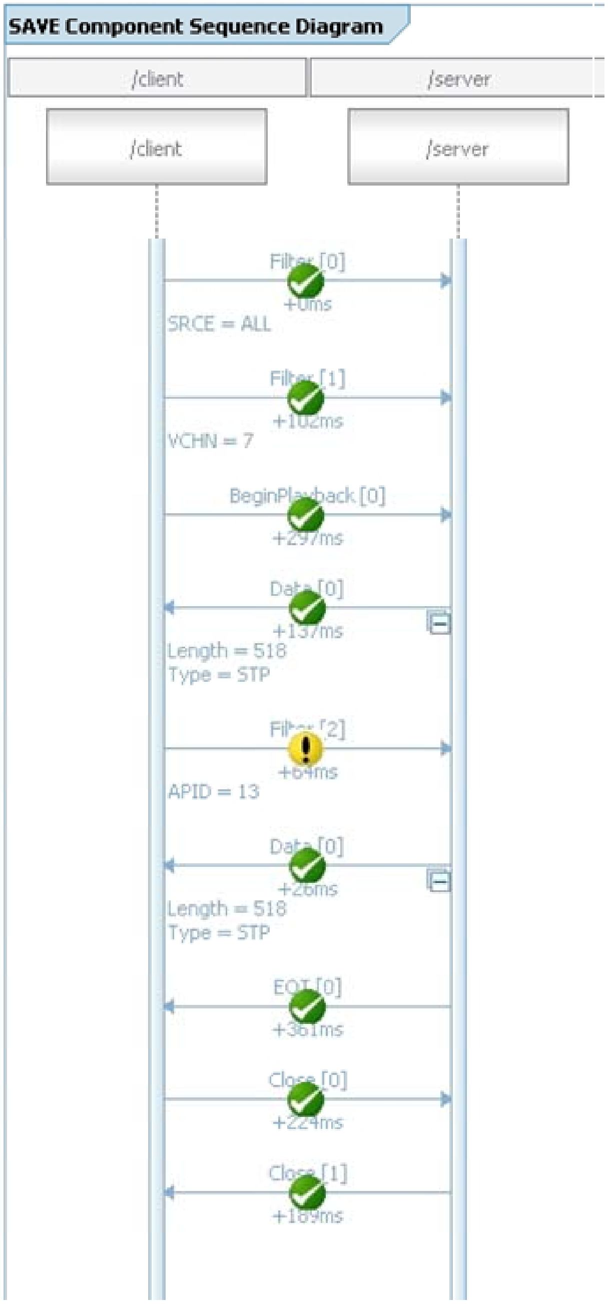


Figure 4. An illegal extra filter is sent after BeginPlayback message has been sent.

C. Specification: The protocol as a detailed sequence diagram

We proceeded by adding more information to the sequence diagram, see Figure 5. Two rules were modeled: 1) The rule that specifies that start time must be less than stop time, as well as 2) The rule that the data type of each of the received data messages must be the same as the specified type. We modeled these rules as assertions and added parameters to the messages. The parameters are used by the assertions and the assertions are evaluated for each message. Since we added more information to the model, we also needed to specify that the order between the different filters is not important. This is denoted by adding a star "*" in front of each filter. In addition, we needed to express the fact that there might be filters of other kinds than the ones we focus on. We express this by adding a general filter with stars as parameters. Since we needed to connect the data messages to the filter messages, we added a parameter Type.

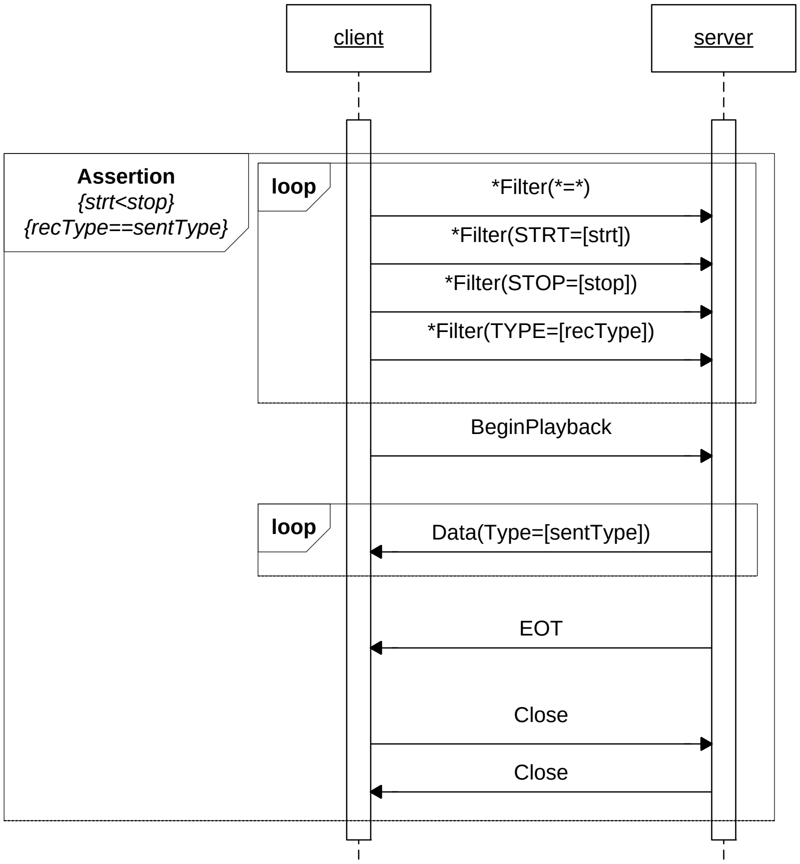
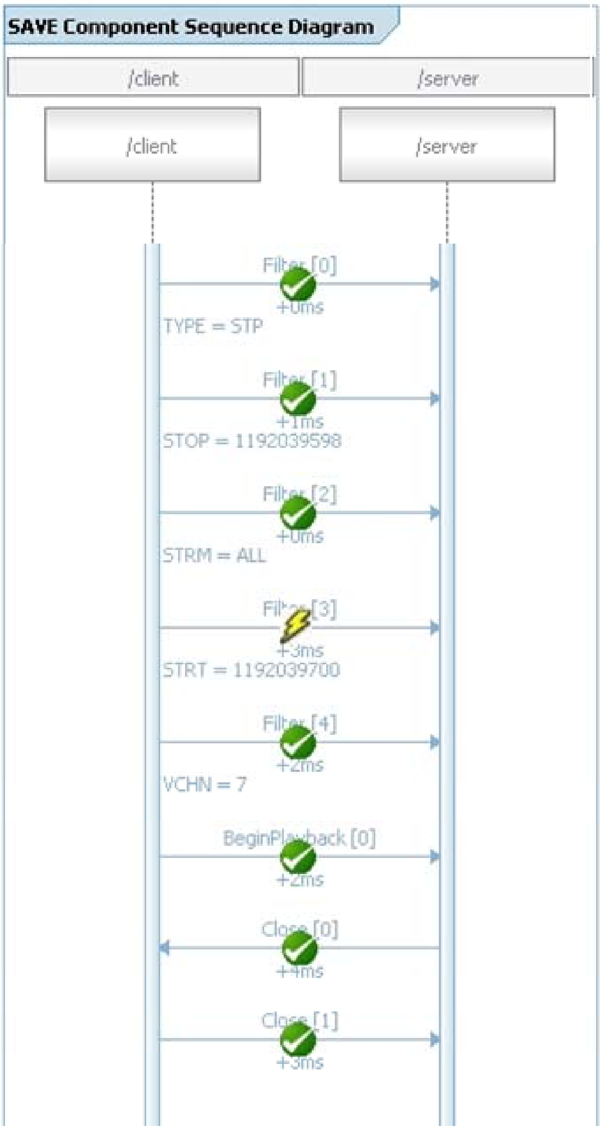


Figure 5 - Detailed planned sequence diagram



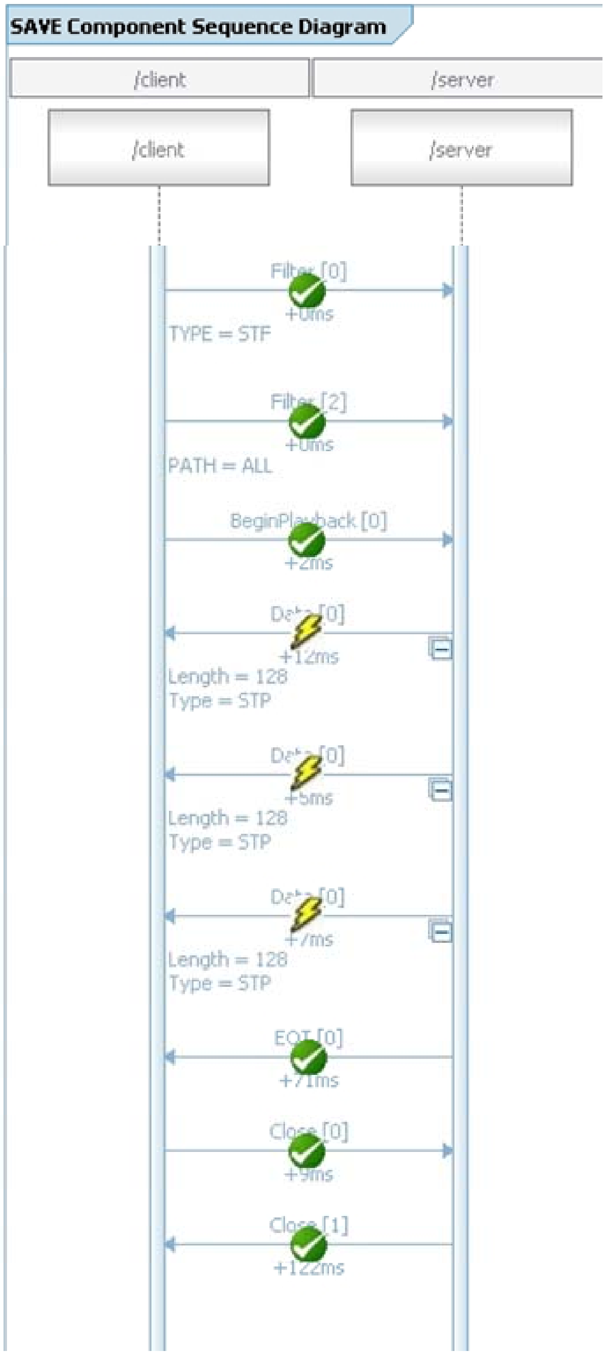


Figure 6 - Start time occurs after Stop time.

Figure 7. "STF" was requested "STP" was received.

D. Evaluation based on the detailed sequence diagram

We evaluated the captured sequences by applying the detailed planned sequence diagram and matching it to the actual sequence. We use a lightning symbol to indicate that the names of the messages are correct but that there is a mismatch between the parameters of the messages. Thus the lightning symbol in

Figure 6 indicates that the time specified in the ‘STRT’ (start time) was after the ‘STOP’ (stop time). Unfortunately, the ICD does not allow for an indication from the server to the client that it has specified an invalid filter. So the close() from the server, without an EOT message, is actually expected. Thus, though not a nominal case, this case illustrates compliance to the protocol defined in the ICD.

In the last example (Figure 10), it was detected that data messages of the wrong type were sent to the client. The root cause is actually that a bad filter is sent from the client to the server: TYPE:STF. STF used to be a supported type, but no longer is. Old legacy clients might still request STF-data, if they have not updated to the new ICD. Since STF is not a valid option, the server returns STPs, which is not expected. In this case, the server simply ignored the invalid filter and used its default which is STP. The correct server behavior should have been to close the socket upon receiving the invalid filter. Dependent on how robustly the client was implemented, the fact the socket is not closed can be a significant problem because the client might assume that the same type of data that was requested will be returned and process the incoming data according to that type. In fact, several clients, including Eng_Dump, are coded in this fashion and do not verify that the type returned and about to be processed is of the type requested. Obviously, processing data of the incorrect type will cause incorrect data to be generated or even client application crashes.

VII. Conclusions

The SARP project develops an approach for dynamic compliance checking and visualizes the results using extended sequence diagrams. The approach has been implemented as an extension to the SAVE tool and was validated in a first pilot study for APL’s CGS.

The results from the study show that problems in the communication between two systems can be detected by using sequence diagrams to model the planned or expected communication and by comparing the planned sequence to the actual sequence. The results also show that there are different kinds of problems and that they can be addressed by modeling the planned sequence using different level of details. Sequencing problems, that is, messages that occur unexpectedly or out of order, can be detected by using high level sequence diagrams without details. Content problems, that is, problems which are related to the content of messages rather than to the order of messages require a more detailed modeling approach. The suggested approach, which is based on assertions in combination with sequence diagrams, seems to be a feasible approach for this problem. High-level modeling may be used without low-level modeling and vice-versa, allowing the user full flexibility over of the amount of time and resources he/she chooses to use the tool to detect such issues. The fact that only simple, standard modeling skills are necessary to become immediately productive with the proposed tool makes the approach appealing.

In the future we are seeking to evaluate the applicability of this approach by applying it to other systems. Depending on the purpose and the constraints in which systems operate, retrieving the actual communication and its evaluation might be different from the systems that were subject to this study. The potential impact might be an extension of our notation for specifying planned communication and the modifications of current evaluation algorithms. Furthermore, new challenges in visualizing the communication arise.

References

[1] Knodel, Jens, Lindvall, Mikael, Muthig, Dirk, and Naab, Matthias Static Evaluation of Software Architectures 2006, 279-294. Conference on Software Maintenance and Reengineering, CSMR 2006.

[2] Claudio Riva, Jordi Vidal Rodriguez, "Combining Static and Dynamic Views for Architecture Reconstruction" Conference on Software Maintenance and Reengineering , 2002 and Hong Y. et al. “DiscoTect: a system for discovering architectures from running systems,” ICSE 2004.

[3] William C. Stratton, Deane E. Sibol, Mikael Lindvall, and Patricia Costa, Technology Infusion of the SAVE Tool into the Common Ground Software Development Process for NASA Missions at JHU/APL, IEEE Aerospace Conference, 2007.

[4] McKerracher, P, Tillman, D, Furrow R M, Herrera, L., Complete Ground Software Re-use: The Common Ground Approach to a Re-usable, Shared Ground System. The Fifth International Symposium on Reducing the Cost of Spacecraft Ground Systems and Operations (RCSGSO). July 8-11, 2003.

[5] Lindvall, M.; Ackermann, C.; Stratton, W.C.; Sibol, D.E.; Ray, A.; Yonkwa, L.; Kresser, J.; Godfrey, S.; Knodel, J.; Using Sequence Diagrams to Detect Communication Problems between Systems; IEEE Aerospace Conference, 2008; 1-8 March 2008 Page(s):1 – 11

[6] Stratton, W.C.; Sibol, D.E.; Lindvall, M.; Costa, P.; Technology Infusion of SAVE into the Ground Software Development Process for NASA Missions at JHU/APL; IEEE Aerospace Conference, 2007; 3-10 March 2007 Page(s):1 – 15

Acknowledgements

The authors acknowledge support from the NASA IV&V Center’s Software Assurance Research Program and, from NSF grant CCF0438933, “Flexible High Quality Design for Software.” The authors would like to thank Dr. Mike Hinchey and Jens Knodel for supporting this work, Ms. Lyly Yonkwa, Dr. Arnab Ray, Mr. Dharmalingam Ganesan, Mr. Patrick Beck,Mr. Jan Kresser, and Mr. Jörg Weinmann for their work on the dynamic analysis, and Ms. Myrna Regardie for proof reading and editing this paper.

Analyzing and detecting problems in Systems of Systems

Chris Ackermann

William C. Stratton (APL), Deane E. Sibol (APL)
Mikael Lindvall, Chris Ackermann, Sally Godfrey (GSFC)

Fraunhofer Center for Experimental Software Engineering Maryland (FC-MD)
Johns Hopkins University Applied Physics Laboratory Space Department Ground Applications Group (APL)
Goddard Space Flight Center (GSFC)

NASA IV&V support through a Software Assurance Research Project (SARP)

Motivation

- Software systems are difficult to understand
- Distributed systems are even more difficult to understand
- Does the system do what it’s supposed to?
- No
  - Trouble reports list many severe transmission failures
- Why?
  - Systems do not behave according to specification
  - Many are related to vague ICDs
- We’re researching how misbehavior can be
  - detected
  - resolved

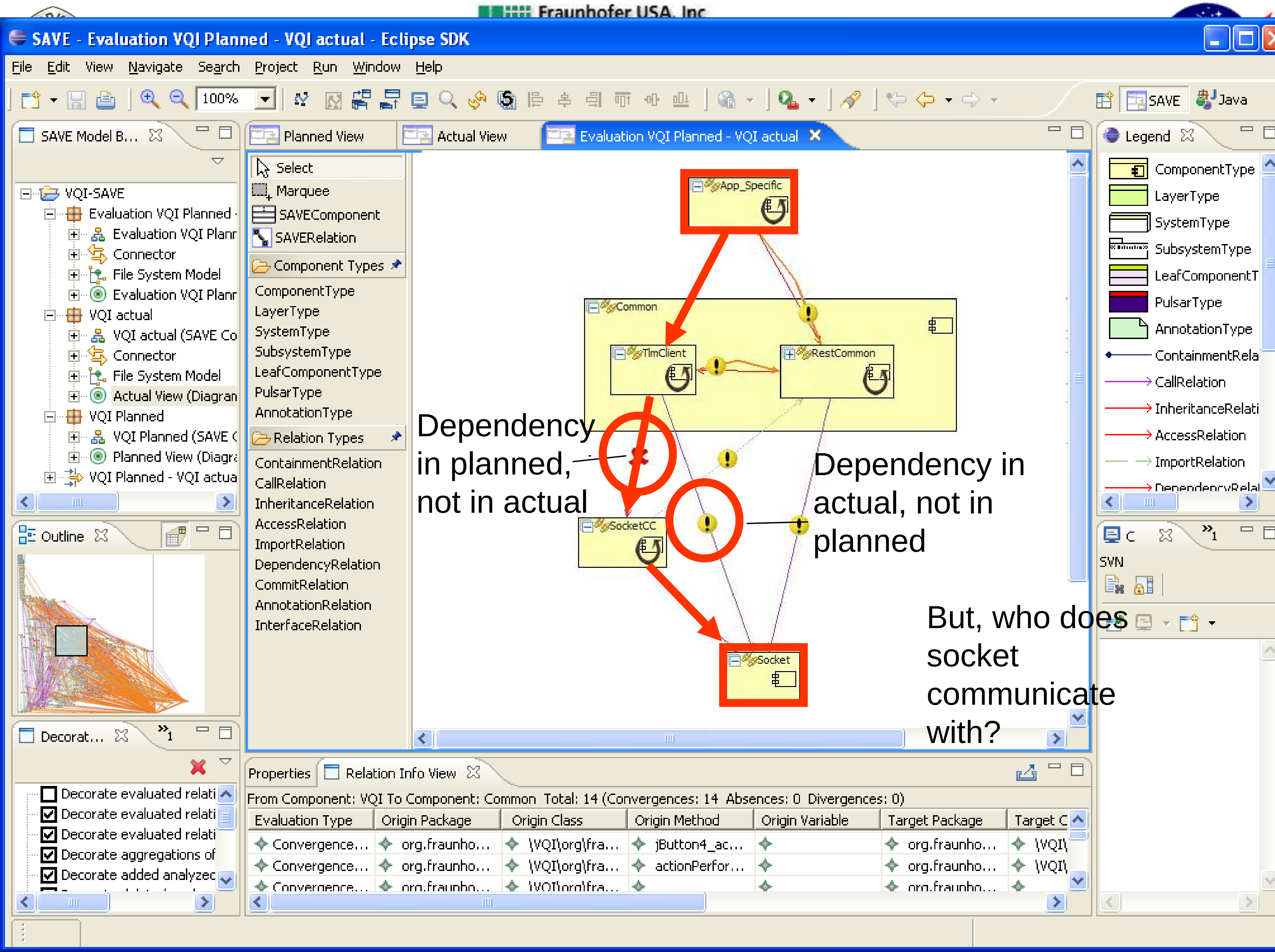
Architecture Compliance Checking

- Does the actual implementation match the planned architecture?
  - Define a *planned* architecture
  - Create an *actual* architecture from source code
  - Identify architectural violations through comparison
- Our Software Architecture Visualization and Evaluation (SAVE) supports static architecture compliance cheching

Our SAVE Experience

- Applied to APL’s Common Ground System (and other systems)
  - NASA Research Infusion project (Aerospace 2007)
- Conclusion
  - The SAVE approach is useful and practical
  - One can quickly model, visualize, analyze, find static architecture violations
  - Good for single software applications
  - But for systems of systems, need to add dynamic information (Dyn-SAVE) as the example will show…

Fraunhofer USA, Inc



☐ Decorate evaluated relati
☑ Decorate evaluated relati
☑ Decorate evaluated relati
☑ Decorate aggregations of
☑ Decorate added analyzec

From Component: VQI To Component: Common Total: 14 (Convergences: 14 Absences: 0 Divergences: 0)

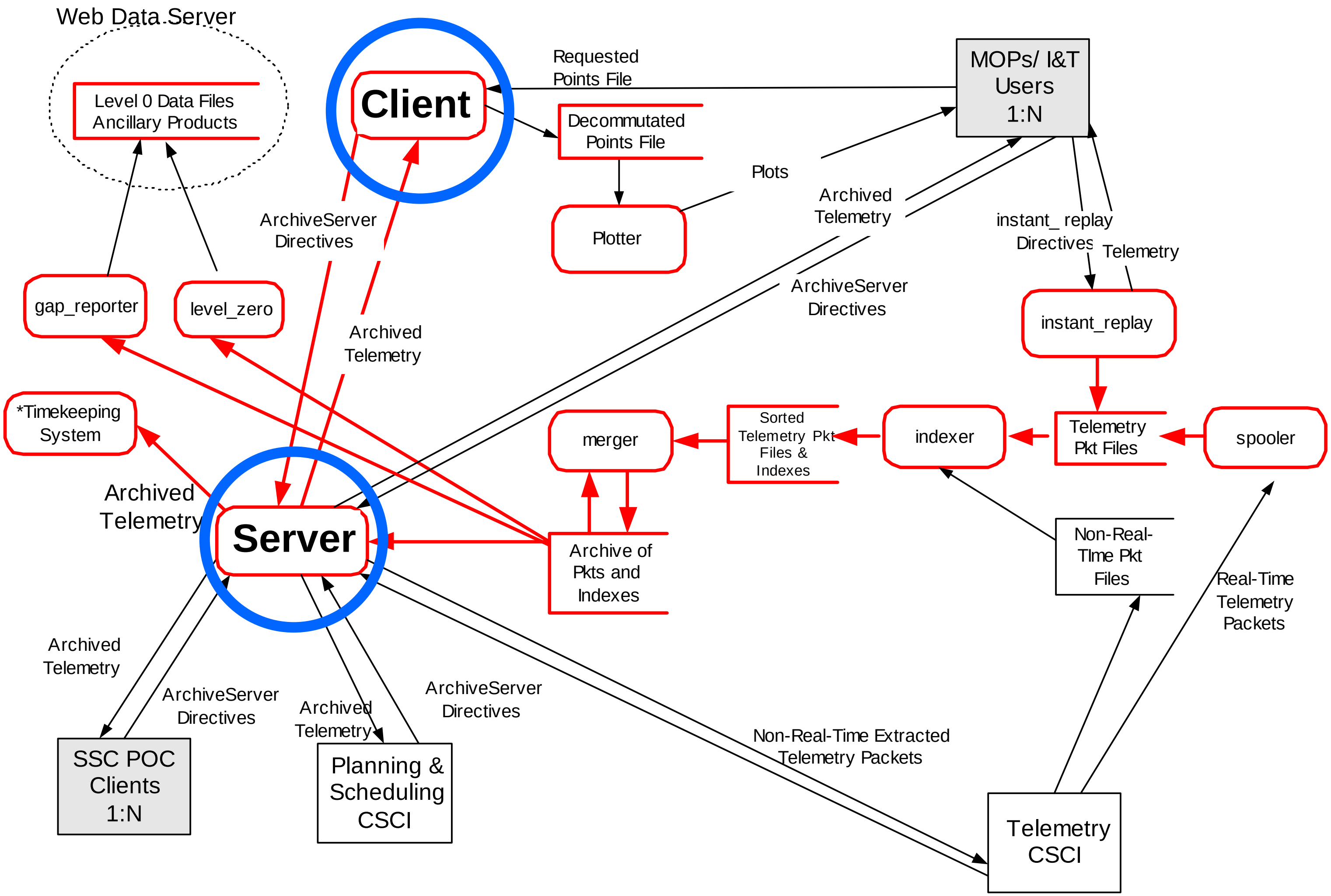
Evaluation Type	Origin Package	Origin Class	Origin Method	Origin Variable	Target Package	Target C
Convergence...	org.fraunho...	\VQI\org\fra...	jButton4_ac...		org.fraunho...	\VQI\
Convergence...	org.fraunho...	\VQI\org\fra...	actionPerfor...		org.fraunho...	\VQI\
Convergence...	org.fraunho...	\VQI\org\fra...			org.fraunho...	

Dependency in planned, not in actual

Dependency in actual, not in planned

But, who does socket communicate with?

The Common Ground System



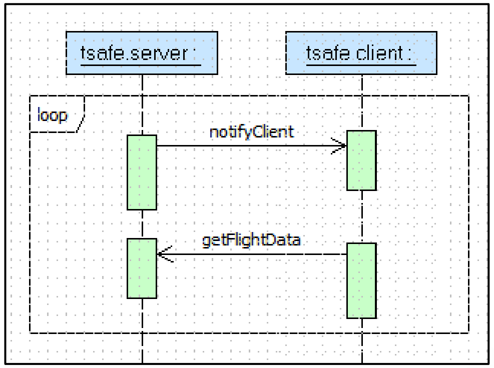
DynSAVE (Vision)

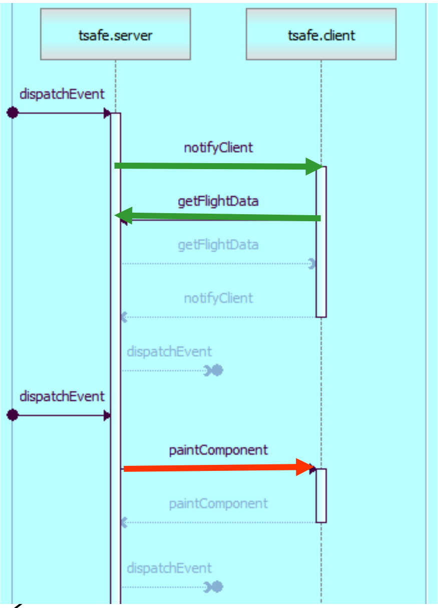
Compare Planned and Actual Behavior

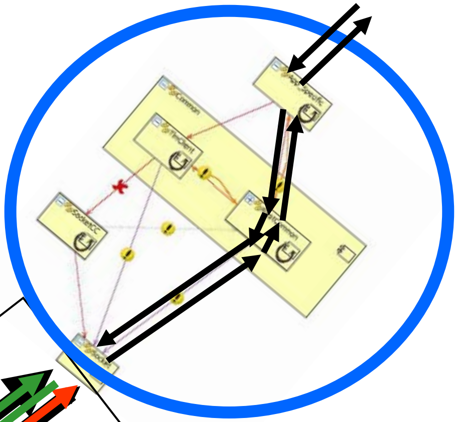
Telemetry **Client**

Form Actual Behavior

Specify Planned Behavior



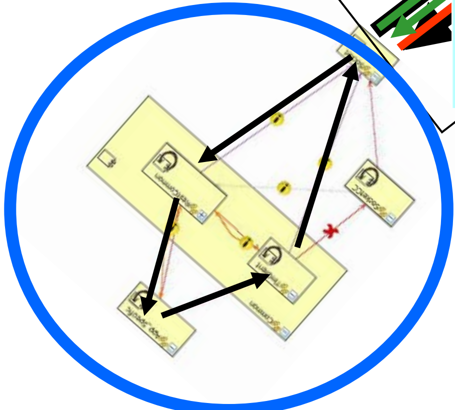




Capture Dynamic Information

Specify Level of Abstraction For analysis

Telemetry **Server**



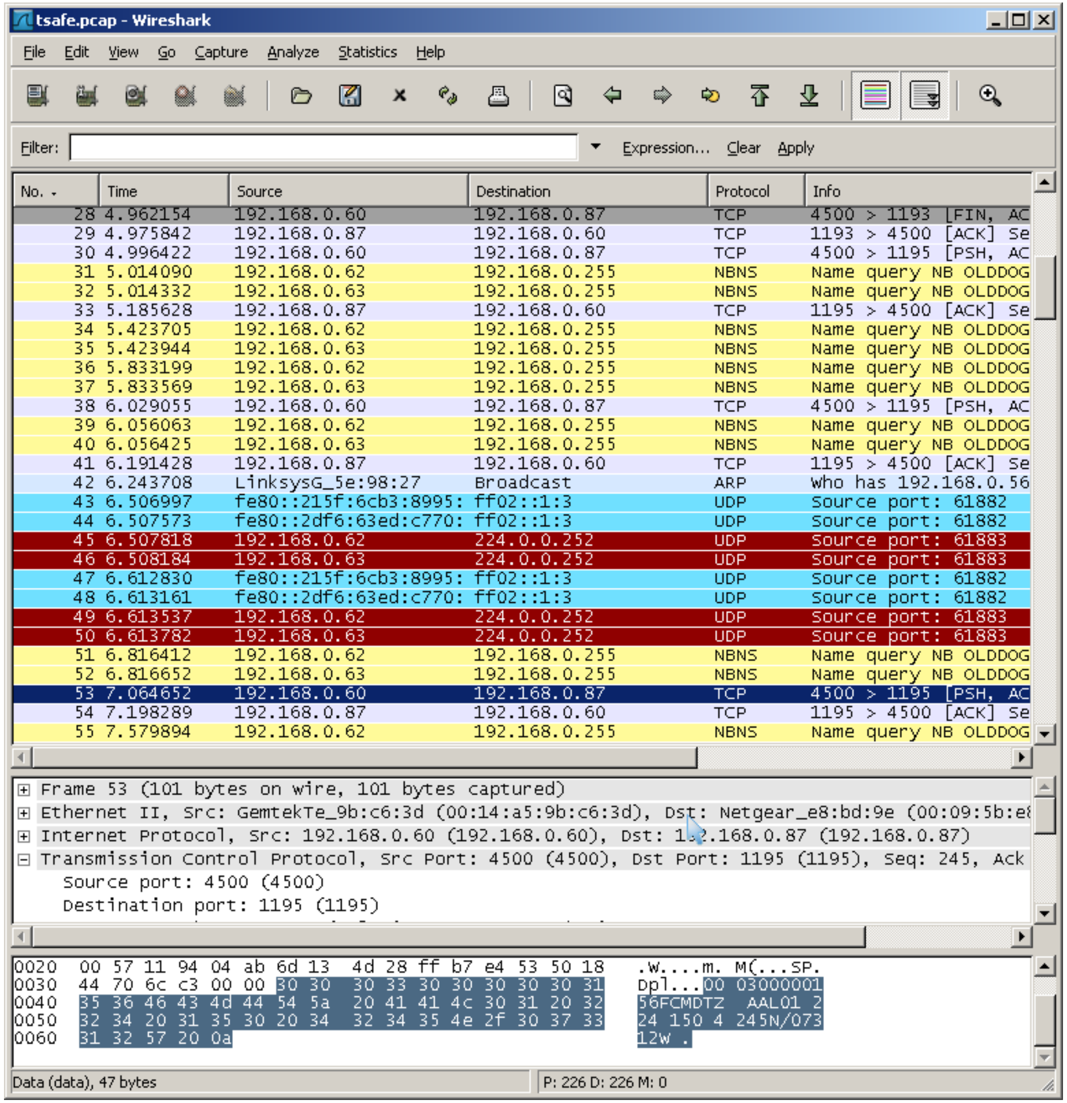
- **Who does socket communicate with?**
- **Is communication according to specification?**

Dynamic Compliance Checking

- Do systems interact as planned?
- Do they adhere to interaction rules?
- Rules specified in ICD’s
  - Sequencing
  - Content
  - Timing

Challenge: Amount and Format of Data

- Amount and format of data being exchanged
  - An interaction consists of a large number of messages
  - The information is often encrypted for efficiency


tsafe.pcap - Wireshark
File Edit View Go Capture Analyze Statistics Help
Filter:
Expression... Clear Apply

```
No. -  Time      Source                     Destination     Protocol  Info
  28 4.962154    192.168.0.60               192.168.0.87    TCP       4500 > 1193 [FIN, AC
  29 4.975842    192.168.0.87               192.168.0.60    TCP       1193 > 4500 [ACK] Se
  30 4.996422    192.168.0.60               192.168.0.87    TCP       4500 > 1195 [PSH, AC
  31 5.014090    192.168.0.62               192.168.0.255   NBNS      Name query NB OLDDOG
  32 5.014332    192.168.0.63               192.168.0.255   NBNS      Name query NB OLDDOG
  33 5.185628    192.168.0.87               192.168.0.60    TCP       1195 > 4500 [ACK] Se
  34 5.423705    192.168.0.62               192.168.0.255   NBNS      Name query NB OLDDOG
  35 5.423944    192.168.0.63               192.168.0.255   NBNS      Name query NB OLDDOG
  36 5.833199    192.168.0.62               192.168.0.255   NBNS      Name query NB OLDDOG
  37 5.833569    192.168.0.63               192.168.0.255   NBNS      Name query NB OLDDOG
  38 6.029055    192.168.0.60               192.168.0.87    TCP       4500 > 1195 [PSH, AC
  39 6.056063    192.168.0.62               192.168.0.255   NBNS      Name query NB OLDDOG
  40 6.056425    192.168.0.63               192.168.0.255   NBNS      Name query NB OLDDOG
  41 6.191428    192.168.0.87               192.168.0.60    TCP       1195 > 4500 [ACK] Se
  42 6.243708    LinksysG_5e:98:27          Broadcast       ARP       Who has 192.168.0.56
  43 6.506997    fe80::215f:6cb3:8995:      ff02::1:3       UDP       Source port: 61882
  44 6.507573    fe80::2df6:63ed:c770:      ff02::1:3       UDP       Source port: 61882
  45 6.507818    192.168.0.62               224.0.0.252     UDP       Source port: 61883
  46 6.508184    192.168.0.63               224.0.0.252     UDP       Source port: 61883
  47 6.612830    fe80::215f:6cb3:8995:      ff02::1:3       UDP       Source port: 61882
  48 6.613161    fe80::2df6:63ed:c770:      ff02::1:3       UDP       Source port: 61882
  49 6.613537    192.168.0.62               224.0.0.252     UDP       Source port: 61883
  50 6.613782    192.168.0.63               224.0.0.252     UDP       Source port: 61883
  51 6.816412    192.168.0.62               192.168.0.255   NBNS      Name query NB OLDDOG
  52 6.816652    192.168.0.63               192.168.0.255   NBNS      Name query NB OLDDOG
  53 7.064652    192.168.0.60               192.168.0.87    TCP       4500 > 1195 [PSH, AC
  54 7.198289    192.168.0.87               192.168.0.60    TCP       1195 > 4500 [ACK] Se
  55 7.579894    192.168.0.62               192.168.0.255   NBNS      Name query NB OLDDOG
```

```
⊞ Frame 53 (101 bytes on wire, 101 bytes captured)
⊞ Ethernet II, Src: GemtekTe_9b:c6:3d (00:14:a5:9b:c6:3d), Dst: Netgear_e8:bd:9e (00:09:5b:e8
⊞ Internet Protocol, Src: 192.168.0.60 (192.168.0.60), Dst: 192.168.0.87 (192.168.0.87)
⊟ Transmission Control Protocol, Src Port: 4500 (4500), Dst Port: 1195 (1195), Seq: 245, Ack
    Source port: 4500 (4500)
    Destination port: 1195 (1195)
```

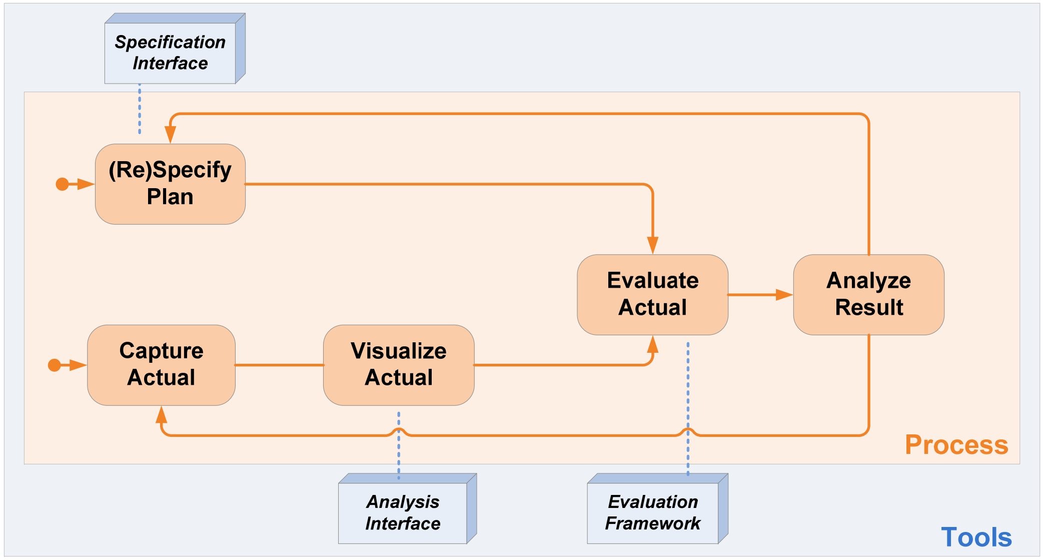
```
0020  00 57 11 94 04 ab 6d 13  4d 28 ff b7 e4 53 50 18   .W....m. M(...SP.
0030  44 70 6c c3 00 00 30 30  30 33 30 30 30 30 30 31   Dpl...00 03000001
0040  35 36 46 43 4d 44 54 5a  20 41 41 4c 30 31 20 32   56FCMDTZ  AAL01 2
0050  32 34 20 31 35 30 20 34  32 34 35 4e 2f 30 37 33   24 150 4 245N/073
0060  31 32 57 20 0a                                     12W .
```

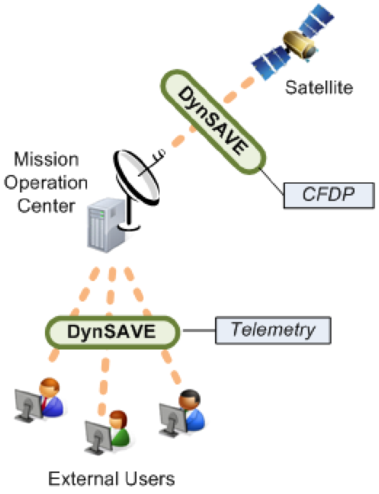
Data (data), 47 bytes | P: 226 D: 226 M: 0

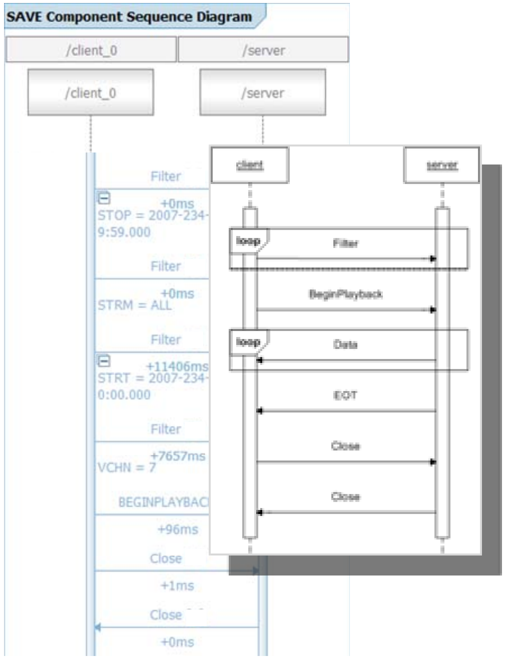
Challenge: Vague ICDs

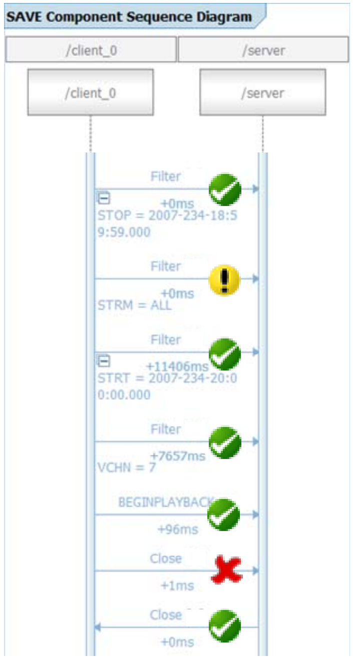
- Interface Control Documents (ICD) specify plan
- But
  - they are missing important information
  - they lack necessary detail
- Why?
  - Difficult to specify complete and consistent ICD
  - Need for tool support

Approach


Specification Interface
(Re)Specify Plan
Evaluate Actual
Analyze Result
Capture Actual
Visualize Actual
Process
Analysis Interface
Evaluation Framework
Tools


Satellite
DynSAVE
CFDP
Mission Operation Center
DynSAVE
Telemetry
External Users


SAVE Component Sequence Diagram
/client_0
/server
/client_0
/server
Filter
+0ms
STOP = 2007-234-
9:59.000
Filter
+0ms
STRM = ALL
Filter
+11406ms
STRT = 2007-234-
0:00.000
Filter
+7657ms
VCHN = 7
BEGINPLAYBAC
+96ms
Close
+1ms
Close
+0ms
client
server
loop
Filter
BeginPlayback
loop
Data
EOT
Close
Close

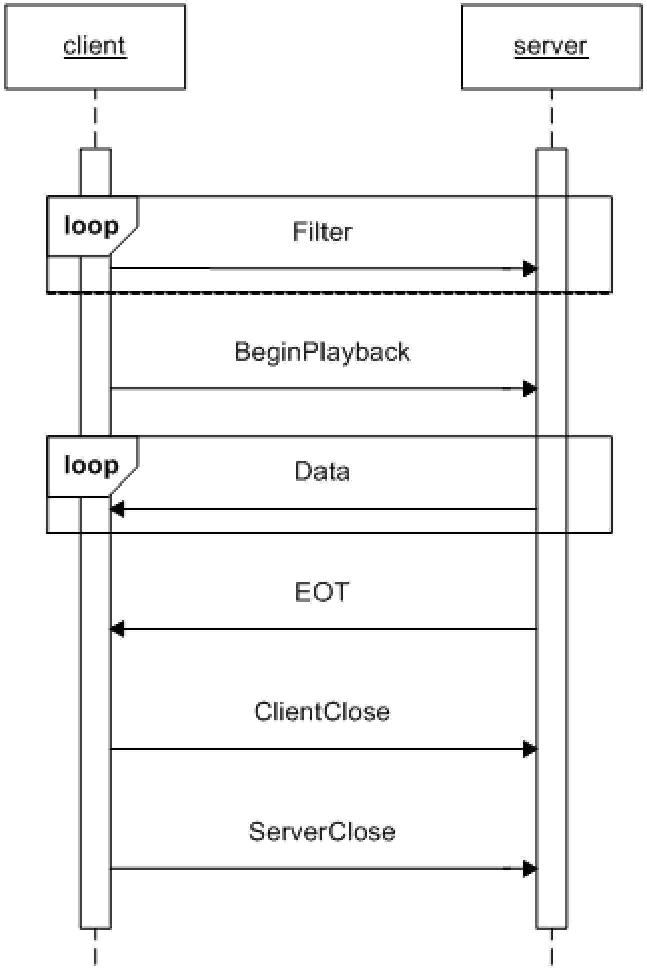

SAVE Component Sequence Diagram
/client_0
/server
/client_0
/server
Filter
+0ms
STOP = 2007-234-18:5
9:59.000
Filter
+0ms
STRM = ALL
Filter
+11406ms
STRT = 2007-234-20:0
0:00.000
Filter
+7657ms
VCHN = 7
BEGINPLAYBACK
+96ms
Close
+1ms
Close
+0ms

Specify the ICD

- Provide intuitive user interface to specify ICD
- Provide evaluation of rules for completeness and consistency
- Allow for iterative specification

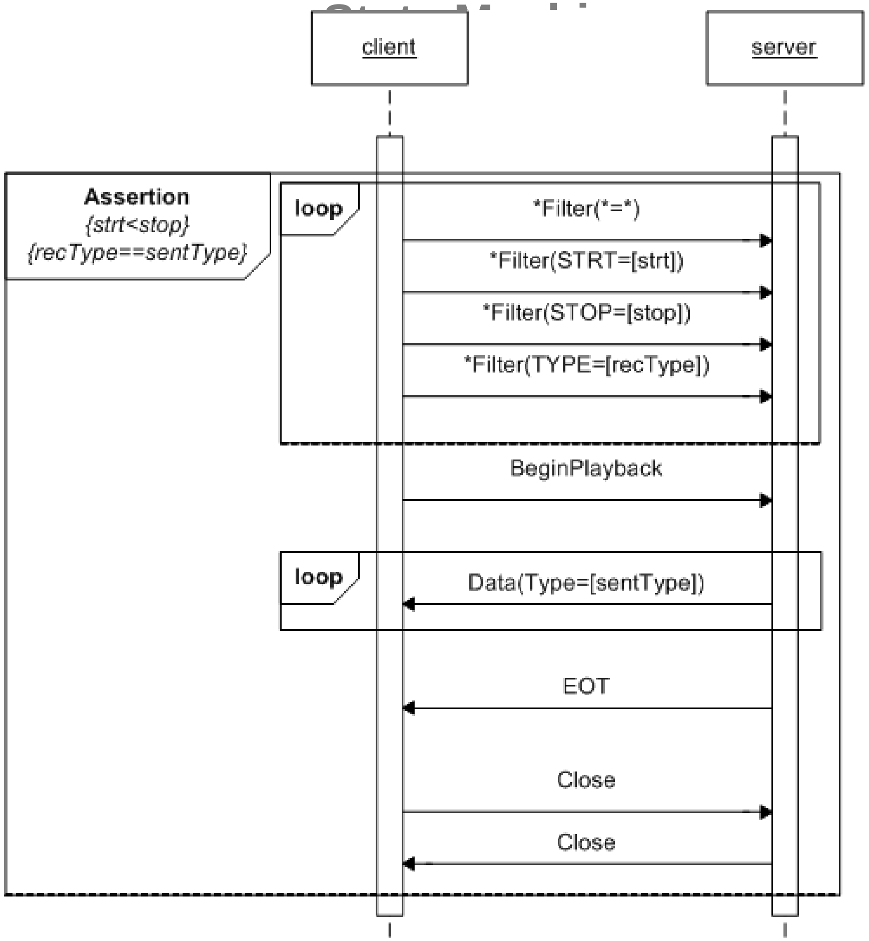
Specifying the Plan

Basic Sequence Diagram



Advanced Sequence Diagram

Could also be specified as a



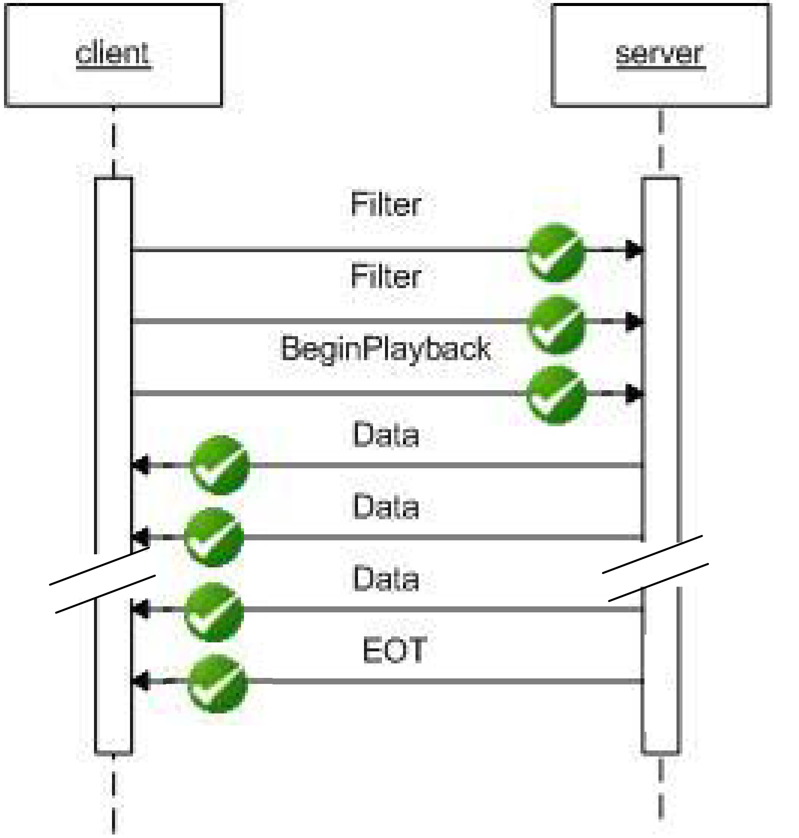
Evaluation Framework

- Sequencing evaluation algorithm
  - Based on DNA sequence analysis
  - Find extra and missing messages
- Content evaluation algorithm
  - Evaluates constraints on message content
  - Example: TYPE==STP
- Timing evaluation algorithm
  - Timing between individual messages
  - Timing over multiple messages
  - Timeouts

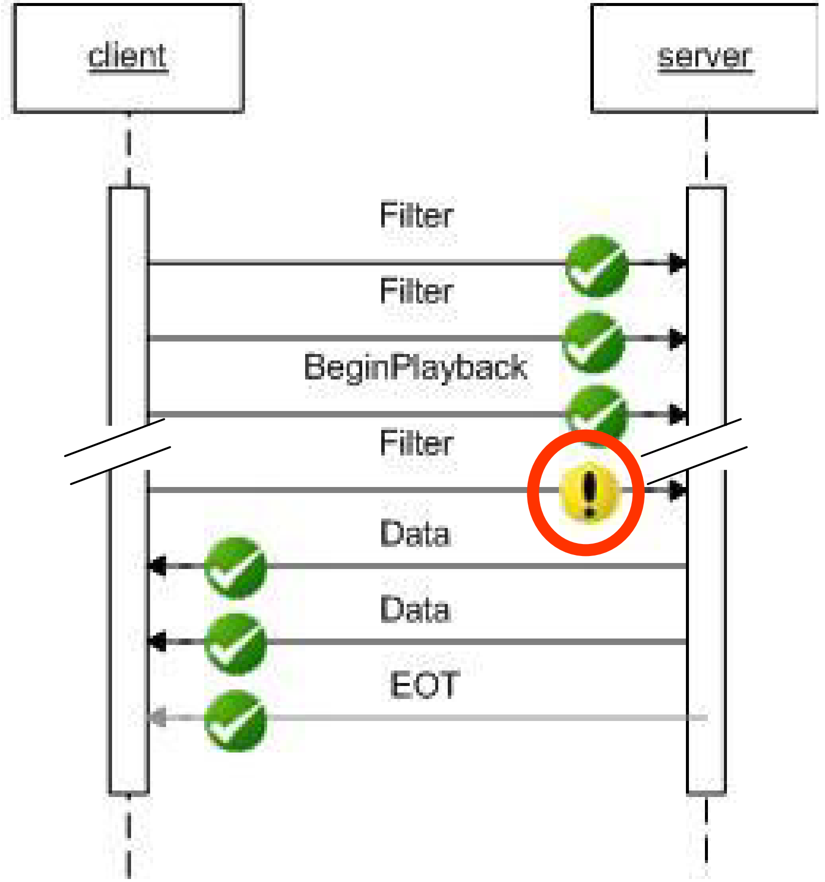
Analysis Framework

- Powerful visualization to detect errors and investigate the problem
- Abstract
  - to highlight potential problems
- Detail
  - to facilitate understanding and knowledge mining

Example 1: Correct Sequence

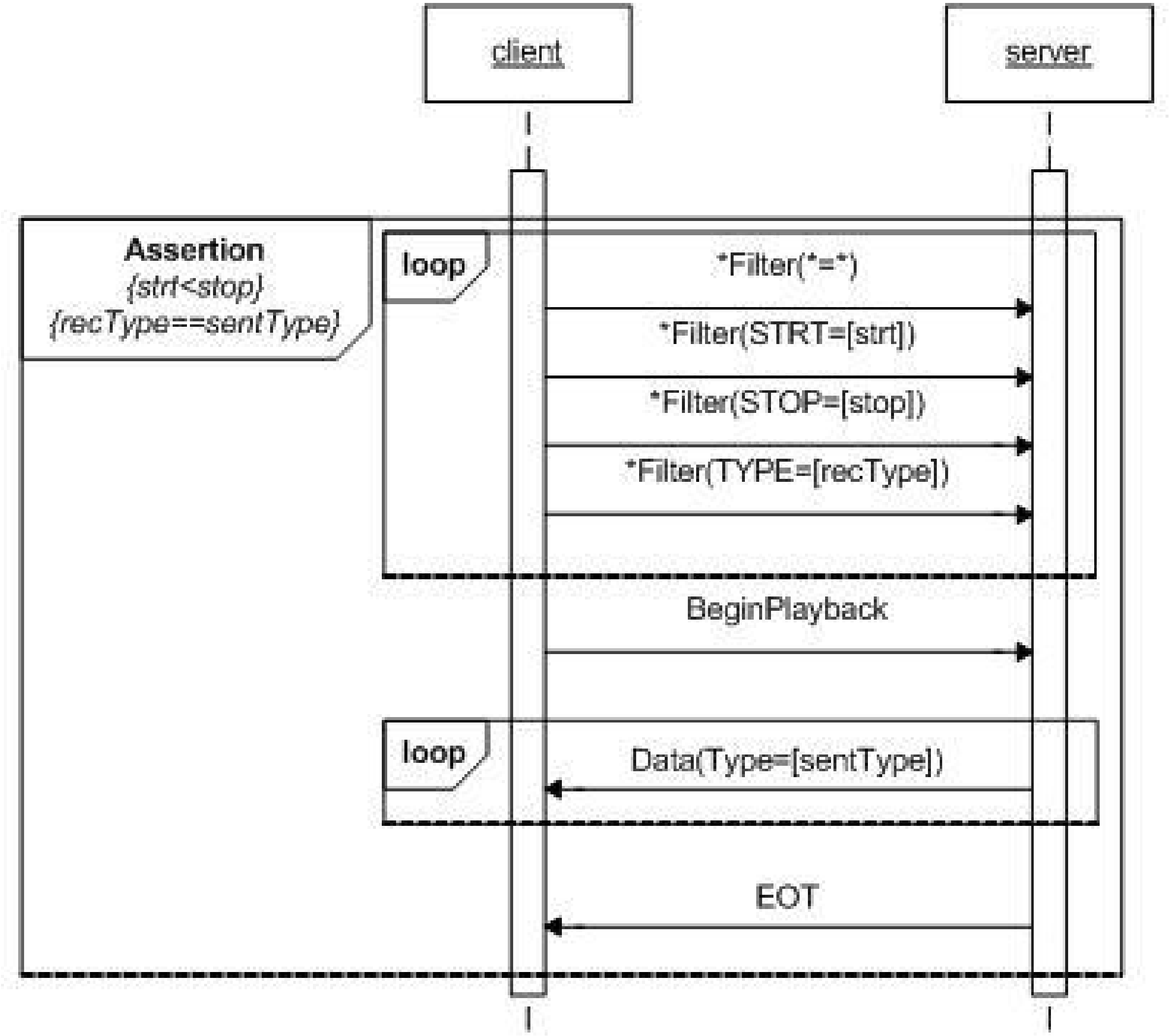


Example 2: Illegal filter



An illegal extra filter is sent after BeginPlayback and Data messages have been sent. The illegal filter is difficult to detect because it is in packet 869.

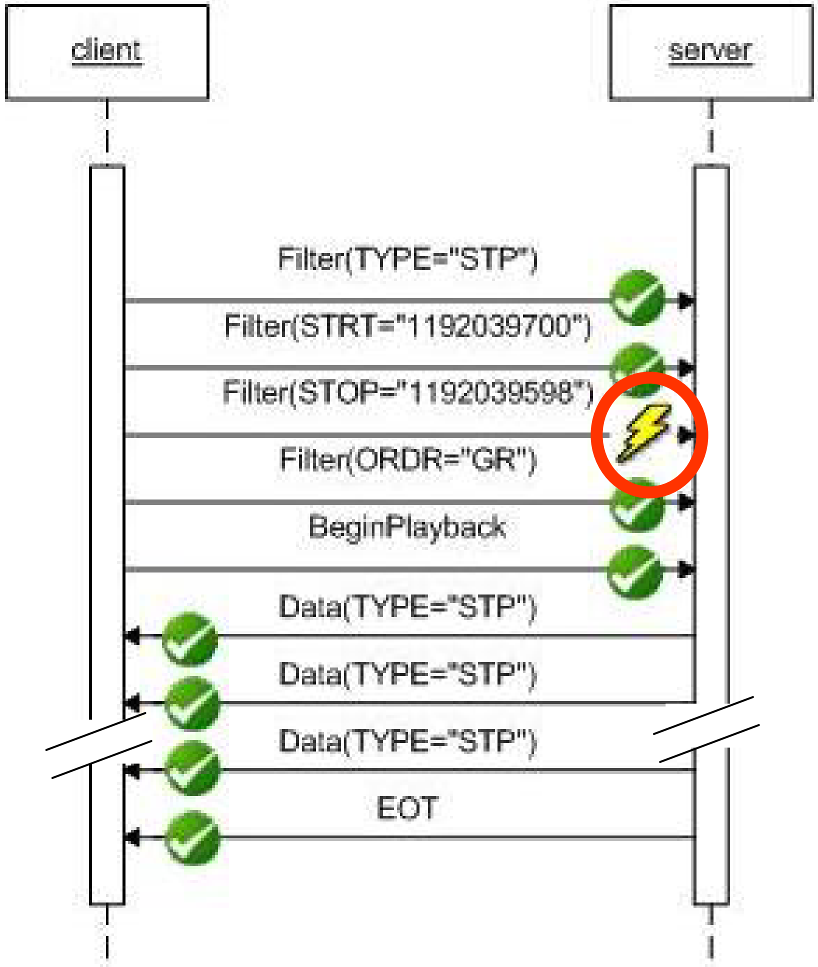
Detailed planned sequence diagram



Rules:

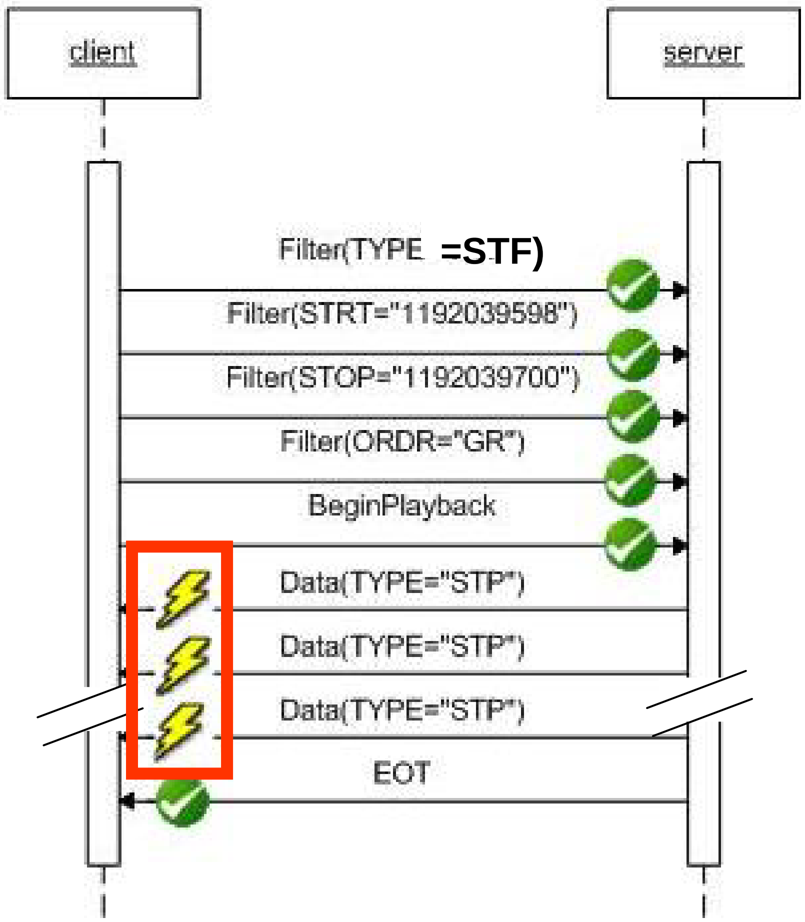
1. Start time must be less that stop time
2. Data type of each of the received data messages must match specification

Example 3: Illegal Time Window specification



Stop time < Start time

Example 4: Illegal Type specification



STF ordered – STP received.

Conclusion

- Summary
  - Prototype detects many of the reported problems
  - Visualization help understand interaction behavior
- Future Work
  - Evaluate applicability to other protocols
  - Make visualization more powerful
  - Combine inter-system interaction with intra-system behavior to form dynamic profile of entire system of systems