A Domain Specific Language for Uncovering Abstract Protocols and Testing Message Scenarios

Tim Reichert[1], Dominikus Herzberg[2]

[1]School of Computing, Engineering & Inf. Sciences, Northumbria University
Newcastle upon Tyne, NE2 1XE, United Kingdom
tim.reichert@unn.ac.uk

[2]Department of Software Engineering, Heilbronn University
74081 Heilbronn, Germany
herzberg@hs-heilbronn.de

Abstract: We present CFR, a language for the specification of protocols and communication scenarios and show how this language can be used for systematic testing of distributed systems. Our language is based on three basic concepts, namely channels, filters and rules. Using our current implementation, we can fully generate sophisticated analyzers from CFR specifications.

1 Introduction

Many of the errors in today's distributed systems occur in the context of complex communication scenarios involving multiple distributed components. Such errors are intricate and cannot be found by component testing alone. System testing complex scenarios is, however, a difficult task, mainly because it is hard to specify correct behavior at the right level of abstraction and in a way that can be used for automated testing.

Protocol analyzers can be used for monitoring and testing system communication on different layers of a protocol stack. However, the parts of the communication design that are not specified as part of a protocol are invisible during analysis. Instead, only the effect a design has on protocol layers can be traced. This has severe consequences, as test cases have to be specified at an inappropriate level of abstraction, which makes them overly complex and likely sources of errors. We provide an example of this in Section 3.

We developed a language called CFR (Channel Filter Rule) that allows comprehensive specification of communication behavior and the full generation of specialized protocol analyzers that can monitor and test this communication behavior. The main purpose of the CFR approach is to uncover and document hidden design intentions in protocol use and to enable specification of communication behavior at appropriate levels of abstraction.

2 The CFR Language

In this section, we introduce CFR, a domain specific language for the specification of both protocols and message scenarios. CFR models can be interpreted in two ways: As a specification of a protocol or as a specification of a protocol analyzer. In our current implementation, we use them to generate analyzers that can monitor and automatically test even complex scenarios.

A protocol layer can be described by relating its communication patterns to the communication patterns on the next lower layer. CFR models do this in a machine-processable form by combining the following basic language constructs:

- A *channel* is a medium that transports messages. The messages obey a protocol.
- A *filter* is a passive and stateless unit that redirects messages from an incoming channel to one of two outgoing channels. Depending on a given pattern, the filter determines which message is to be delivered to which outgoing channel. Since both the input and output of a filter are messages, the output of a filter can be used as the input of another filter. Filters might be put in parallel or in sequence. Filters are passive elements as they do not modify or create messages.
- A *rule* is an active and stateful message processing unit that consumes messages from one or more incoming channels and returns messages to one or more outgoing channels. In opposition to filters, a rule can modify incoming messages or create new messages. To do that, a rule contains defined logic in form of a state machine or a set of state machines with a selector function. Similar to filters, the output of one rule might be the output of another rule.

We represent the inputs and outputs of rules by pins with connectors between them. The message flow is from output to input pins. Channels and rules have an arbitrary number of input and output pins, while filters have exactly one input and two output pins. We use regular expressions to define the selection criteria in filters. State machines are used to define the memory of rules. We provide an example of a CFR model in Section 4.

3 Uncovering Abstract Protocols

A concrete protocol is a specification of messages and a set of message sequences. An abstract protocol constitutes its own communication architecture based on a concrete protocol. The communication architecture may be layered. At application level, however, strict layering is often impractical and broken by tunneling. An example is due. HTTP is a stateless and connectionless concrete protocol. It consists solely of request and reply messages and only a client can initiate communication via a request message. An application using HTTP may compensate for HTTP limitations and define its own interaction scheme. For instance, it might be necessary for a web browser based application to receive

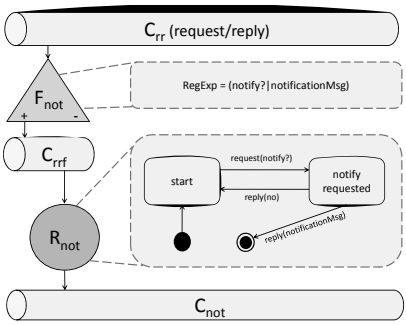

Figure 1: CFR model for the notification example in visual notation

notifications of important events from a web server. As the server cannot contact the client directly, an application programmer needs to design an abstract protocol to implement event notifications based on HTTP.

One way to realize the "illusion" of a server-based notification service is to let the client periodically send request messages to the server, asking whether there is a notification available. Upon receiving a request, the server replies by either sending a negative indication or it delivers the notification. This is exactly how the now popular AJAX web technology works. Other parts of the application might use the notification service via the abstract protocol – but they are not forced to do so. Application designers working with such a notification service certainly think in terms of notifications and probably do not even know about requests and replies. A conventional protocol analyzer on the other hand would only analyze the HTTP protocol, not knowing anything about the abstract protocol and its communication architecture. With CFR, we can specify abstract protocols and generate analyzers that understand these protocols.

Figure 1 shows a CFR model that defines the notification protocol just described. In our visual notation, Channels are tubes, filters are triangles and rules are circles. Channel C_{rr} transports all request/reply messages. The Filter F_{not} ensures that channel C_{rrf} transports only those requests and replies related to the notification service. The rule R_{not} uses a state machine to transform a request followed by a positive reply into a notification message and puts it on channel C_{not}.

4 Testing Scenarios with CFR Analyzers

Communication behavior in a distributed system can be described by a set of scenarios. A scenario for the aforementioned example might, for instance, be an update occurring on the server of which the client application needs to be notified. On the communication level, this involves at least two messages: An update message to the server followed by a notification message to the client. An effective method for testing the correctness of such a scenario is to stimulate the server by sending an update message and to check if the

expected notification message is sent. Using a conventional protocol analyzer, the stimulation and testing could in principle be done. However, the specification of the scenario would be at an inappropriate level of abstraction, namely at the request/reply layer.

If we view scenarios as sequences of messages, we can use CFR models to specify them as specialized high-level abstract protocols. By following a layered approach [HB05], we ensure that these specifications are always at the appropriate level of abstraction. We can specify even complex scenarios involving different protocols and subnetworks by combining all relevant messages onto a single abstract protocol layer and by then using this protocol layer for the definition of the scenario layer.

We require scenarios to be uniquely identifiable via their message sequences. Ambiguities have to be resolved at the specification level. This ensures that the step from merely monitoring a system to testing it is feasible. Based on a set of CFR models, we can use our analyzers to identify unspecified - and thus potentially faulty - communication behavior. In addition to that, we can use the analyzers to systematically stimulate components in order to trigger scenarios that are then verified. Under real conditions, a large number of different concurrent scenarios might be monitored simultaneously. We can refine our analyzers by assigning time intervals to the transitions of state charts within rules. This defines how long the monitor waits for the continuation of a scenario. If the expected future is not confirmed within a given time frame, tracing of the scenario is canceled and an error is reported.

5 Conclusions and Future Work

Our work is related to the large body of work done in forward engineering and analyzing protocols. Using a formalism based on channels, filters and rules, augmented with state charts and regular expressions, we are able to specify, monitor and test even very complex scenarios in distributed systems. Our layered approach ensures that specification can always be performed at the right level of abstraction. We have developed both a textual and visual notation for our language and extensively use code generation techniques to generate analyzers for different technology platforms. Currently, we are using our approach for testing AJAX based web applications and automotive systems [MH07] in order to gain more real-world experience. In future work, we will apply machine learning techniques for the automated extraction of CFR models from communication data and we are working on giving CFR a stronger formal basis by mapping CFR models to finite state processes.

References

[HB05] Dominikus Herzberg and Manfred Broy. Modeling layered distributed communication systems. *Formal Aspects of Computing, 28(4):751-763*, May 2005.

[MH07] Ansgar Meroth and Dominikus Herzberg. An Open Approach to Protocol Analysis and Simulation for Automotive Applications. In *Embedded World Conference*, 2007.