Refining Interactions in a Distributed System

Neelam Soundarajan

Computer and Information Science
The Ohio State University
2015 Neil Avenue
Columbus, OH 43210
USA

e-mail: neelam@cis.ohio-state.edu

Refining Interactions in a Distributed System

Abstract:

We identify a type of refinement called *interaction refinement* that is useful in designing distributed systems. In a step of interaction refinement, a system designer refines a 'high-level' interaction into a sequence of 'low-level' interactions. In doing this, the designer is not refining the *internal structure* of the processes; rather he[1] is implementing a high-level interaction between two (or more) processes in terms of a sequence of low-level communications between them.

We motivate the idea of interaction refinement with simple examples and develop a preliminary formalism that can be used to establish correctness of systems developed using interaction refinement. We apply the formalism to validate some communication protocols.

Keywords: Interaction refinement, Communication traces, Specification and verification, Communication protocols.

1 Introduction and Motivation

Two types of refinement that have been studied extensively are *procedural refinement* (*PR*) and *structural refinement* (*SR*).[2] In a step of *PR* we refine the task of achieving a given goal starting from a specified initial condition, into subtasks for achieving subgoals, such that the given initial condition implies the initial condition of the first subtask, the subgoal achieved by the first subtask implies the initial condition required by the second subtask, etc., and the subgoal of the last subtask implies the original required goal. The process of refinement is repeated with each subtask until the final subtasks can be directly implemented using the primitives of the given programming language. *PR* is a powerful tool and is used extensively, especially in designing sequential programs. Formal methods developed by Hoare, Dijkstra, Gries, and others, allow us to reason precisely about such programs, and the reasoning closely models the refinement steps used in developing the program.

While *PR* is concerned with refining the procedure for achieving the given task, *structural refinement* (*SR*) allows us to refine the *structure* of the system being designed. Thus, for instance, when designing a process P that will exhibit a certain communication behavior with external agents, we may decide to structure P into a pair of communicating processes P_1 and P_2 that will communicate with each other and with the external agents in such a way that when the internal communications are abstracted away, the resulting communication behavior matches the behavior specified for P. Methods such as those of [1, 10, 11] formalize this type of refinement.

[1]Following standard practice, we use 'he', 'his', etc. as abbreviations for 'he or she', 'his or her' etc.

[2]Procedural refinement is often also called *stepwise refinement*; we prefer the term procedural refinement to distinguish it from the other types of refinement which are also stepwise.

Our goal is to explore another type of refinement which we call *interaction refinement* (*IR*). Consider an example: Suppose we are designing a system that includes two processes A and B where A is expected to send a sequence of files to B. For instance, these processes might be part of a network of workstations, A being a text editor on one of the workstations in the network, and B a file server that will save the files that A sends to it. An experienced designer might decide to have A send *differential* information, i.e., information on how each file differs from the previous one it sent, rather than complete files. In those cases where A is indeed something like an editor, this would result in an enormous reduction in network traffic; and since B could easily combine this differential information and the file it currently has to obtain the new file, it serves essentially the same purpose as the original specification. But according to most current formalisms, this would not be a correct refinement since the refined communications between A and B are not complete files as was originally required; yet most designers would agree that this is a correct, and even a good refinement. In this paper we discuss the importance of *IR* in the design of distributed systems, and present a preliminary formalism for validating the correctness of such refinements.[3]

Consider another example: Suppose we were designing a system consisting of a 'client object' *CO* and a 'server object' *SO*. The high-level specification of the system might say that the purpose of the first interaction between these two objects is to identify the client to the server. We may decide to map this into a series of interactions in which *CO* provides its 'user-id' and a 'password'; if *CO* provides an invalid password, *SO* might allow it several chances to get the password right; etc. We may even introduce a new object, an authentication object *AO*, to mediate this interaction, and encapsulate the authentication of the password in *AO*. Indeed it might be better to have *CO* send the password directly to *AO* rather than via *SO*, since the protocol on how many attempts the client is allowed to get the password correct etc., can then be decided by *AO* which is probably where it belongs. In any case, what we have done is to refine the high-level interaction *identify-client* between *CO* and *SO* into a series of interactions involving *CO*, *SO*, and *AO*. Again this refinement would not be considered correct by most formalisms since the interactions between *CO* and *SO* have been modified.

There is an important difference between *IR* on the one hand, and *PR* and *SR* on the other. In both *PR* and *SR* what we refine is *internal* to the object or process being designed; the *external* behavior is not being modified, indeed that is almost the definition of the correctness of a step of *PR* or *SR*. In *IR* this is no longer true. What we refined in the first example was not internal to the processes A or B; rather we modified the externally visible interactions between these two processes. The reason such a modification seems correct is that it seems consistent with the intent behind the original specification. Similarly in the second example, we did not consider the internal structure of either *CO* or *SO* but instead introduced new interactions between these objects, and even a new object *AO* and interactions between *CO*, *SO*, and this new object, in order to achieve the purpose behind the original specification.

A number of authors have considered ideas that seem to be related to the notion of interaction refinement. Brinskma et al [2] consider what they call *interface refinement* as a method of refining

[3]Many of the ideas presented in this paper can also be applied in the setting of (sequential) Object-Oriented programs since objects, being autonomous entities, are similar to processes in a distributed system. Indeed in this paper we often use the terms 'object' and 'process' interchangeably.

the interface, i.e., the alphabet of interactions, between two processes. Our interaction refinement seems more general and has to do with refining the entire sequence of interactions that a process or a group of processes go through; indeed, as in the second example above, we even allow new objects to be introduced to mediate the interactions between the original objects. Gerth et al [6] also use the term 'interface refinement' but they seem to be interested in a somewhat different problem. They ask 'under what conditions is it legitimate to replace synchronous communications with asynchronous ones?', and we do not consider such refinements. But both [2] and [6] have one important thing in common with our work: the refinements they consider also manifest themselves in the *external* behavior of the object(s) in question, rather than being refinements of the internals. Another general approach (see [9], for instance) is to show that a refined system, in some sense, is a 'simulation' of the original system. We will return to the relation of our work to that of others later in the paper.

The main contributions of the paper are the following:

- Present the idea of interaction refinement, and argue that it is useful in the design of distributed systems.
- Present a preliminary formalization of interaction refinement in the form of a proof rule, that allows us to verify the correctness of a particular refinement.
- Apply the formalism to a simple example, the *Alternating Bit Protocol (ABP)*.

In the next section we present our approach to specifying processes, a fairly standard one using invariants over traces, and then proceed to define precisely what we mean by interaction refinement. Next we propose a rule for establishing the correctness of a step of *IR*. In section 3 we apply our approach to a series of examples, a standard set of communication protocols, the final one being the *ABP*; each protocol is shown to be obtained by an *IR* step from the previous one. In the final section we summarize our work and compare it with related work.

2 Specifying and Verifying Interaction Refinements

We will use a *trace-based*[4] approach to specify processes. With each process P we associate a trace τ that will record all interactions that P engages in. An *interaction* is often, but not always, a *communication* in which one process sends a specific value to another. In general, low level interactions tend to be communications, while higher level interactions tend not to be. Thus in the example of the client process *CO* and server process *SO* of the last section, 'identify-client' is a high level interaction but not a communication, whereas the low level interaction in which *CO* sends a password to *SO* (or to *AO*) is a communication.[5] We use an element of the form (c, v) in

[4]We could use other models such as *failures* [7] that would provide more information than traces can; but our goal is to explore the idea of interaction refinement in a simple setting, so we use traces.

[5]Actually, sending a password may be a complex interaction involving a number of communications. This just goes to show that interaction refinement, like other types of refinement, is in general a multi-step activity.

a trace to record a communication in which the value v was sent on the channel c.[6] We record an interaction that is not a communication as an element that specifies the name of the interaction, the identities of the processes involved, and any parameter values. Note that a communication involves exactly two processes, a general interaction may involve two or more processes.

Consider a system S. A specification of S will be an invariant[7] I on its trace which would impose restrictions on the interactions that take place between the component processes of S and with external processes, and the relative orders of these interactions. Suppose we obtain S' from S following a step of interaction refinement, and suppose we have a specification I' for S'. The question we are interested in is, What relation must hold between I and I' in order for us to claim that this refinement step was correct?

Note that I and I' deal with traces that contain different kinds of elements. I deals with traces of what we have called high-level interactions, whereas I' deals with traces of low-level interactions that *implement* those high-level interactions.[8] Let τ denote the trace that I refers to and τ' the trace that I' refers to. Both τ and τ' record all interactions between the various component processes (of S and S' respectively) and with external processes. The difference is that the view recorded in τ is a high-level view whereas that in τ' is a low(er)-level view.

Let Σ be the set of elements corresponding to the high-level interactions, i.e., that can appear in τ, and Σ' the set of elements corresponding to the low-level interactions. A simple situation would be for each element of Σ to be implemented by a *sequence* of elements of Σ'. This can be represented by a mapping of the form:

$$\rho : \Sigma \Rightarrow \Sigma'^*$$

where Σ'^* is the set of all (finite) sequences over Σ'.

If $\rho(i) = \langle i_1, \ldots, i_n \rangle$, that means that the high level interaction i is being implemented as the sequence of low-level interactions $i_1, \ldots, i_n$. Note that there is no assumption that only processes involved in i may be participants in $i_1, \ldots, i_n$.

Once ρ is given, we can see what relation must hold between I and I': We need to ensure that if a given τ' satisfies I', then the corresponding τ will satisfy I:

$$\forall \tau'.\ I' \Rightarrow [\exists \tau.(\tau' = \rho(\tau) \wedge I)] \tag{1}$$

where $\rho(\tau)$ is obtained by applying ρ to each element of τ and then appending together all the resulting sequences. If there is no difference between the high-level and low-level interactions, ρ reduces to the identity map, and (1) to the usual implication relation.

But the function ρ is too restrictive. It requires that *every* instance of an element of Σ must be mapped to the same sequence of Σ' elements. This would prevent refinements that exploit the past history of interactions. In the editor/file server example, a high-level interaction in which A sends

[6] If communication were asynchronous, as in the examples in section 3, we will record the send and receive as two separate events.

[7] For simplicity we are considering only *safety* properties, thus invariants are sufficient. Our ideas will however apply to other kinds of properties than safety, and to other kinds of specifications than invariants.

[8] Existing formalisms just require I' to imply I, and this would be the correct requirement if S and S' go through the same communications. We are trying to generalize this so the interactions in S' can be different from those in S.

a file to B is mapped into a low-level communication in which A sends differential information to B, which clearly depends on what it has sent previously. The solution is to have a mapping function not over the individual interactions but over traces:

$$\rho : \Delta \Rightarrow \Delta'$$

where Δ is the set of all possible high-level traces (the set Σ^*), and Δ' the set of all possible low-level traces. Towards the end of section 3 we will propose a further generalization of ρ.

3 Examples

We will apply our approach to a series of communication protocols, each obtained by interaction refinement from the previous one. Our goal is not to provide formal proofs of correctness of these protocols, but rather to show how *IR* plays an important role in the design of such systems and how our approach can be used to justify the individual steps of interaction refinement. The protocols are taken from chapter 4 of Holzmann's book [8] on the subject.

Holzmann starts with a simple protocol, the *No-Flow-Control* protocol, where the sender S sends a sequence of messages, which we call *data* messages to distinguish them from other messages to be introduced shortly, and the receiver R accepts them. Since the processes are asynchronous, this can lead to buffer overflow. So the next refinement[9] Holzmann introduces is the *Ping-Pong* protocol. Here S sends a data message, and waits for an *acknowledgment* from R before sending the next message. R, similarly, receives a data message, sends the acknowledgment message, and waits for the next message. Note the interaction refinement: the original protocol only talked about data messages from S to R; in the refinement we have introduced *new* messages, in the form of acknowledgments from R to S.[10]

While the *Ping-Pong* protocol solves the buffer overflow problem, it does not account for the possibility of messages being lost in the channels. The problem cannot be fixed by introducing timeouts and having S re-send the message if it doesn't receive an acknowledgment after a period of time because then there would be a problem of matching the acknowledgment and the message being acknowledged. This leads to the next refinement, the famous *Alternating Bit Protocol*. Here we refine the data messages to include a 1-bit sequence number, so that alternating data messages have the sequence numbers 0, 1, 0, $\ldots$. Also, the acknowledgments carry the sequence number of the message being acknowledged. The sender S sends a data message, including the sequence number and waits for an acknowledgment with the same sequence number. If within a fixed period, no acknowledgment is received, or only acknowledgments with the 'wrong' sequence number are received, S re-sends the message (including the sequence number). If the expected acknowledgment is received, S can be sure that the data message has been received by the receiving process R, and can proceed to the send the next data message. R similarly waits for a data message with an

[9]Holzmann's first refinement is actually the *X-on/X-off* protocol; we will skip this protocol.

[10]If these new messages had been *internal* to S or R, this would be an instance of structural refinement, but since the messages are exchanged *between* the two processes, this is an interaction refinement step. In the *X-on/X-off* protocol that we skipped, not only are new messages between S and R introduced, but also each of S and R is split into a pair of parallel processes; the former is an instance of *IR*, the latter an instance of *SR*.

appropriate sequence number, 0 or 1. If it receives a message with the incorrect sequence number (1 when 0 was expected, or vice-versa), it sends an acknowledgment with the sequence number it received, and continues to wait; this corresponds to the case when the previous acknowledgment it sent was not received by S. If R receives a message with the expected sequence number, it sends an acknowledgment with this sequence number, proceeds to deal with the received message, and then waits for the next data message. Note that this interaction refinement step has resulted in two distinct changes; first, data messages have been modified to include acknowledgment numbers; second, acknowledgment messages have been modified to include a number. [11]

We will start with the invariant for the *No-Flow-Control (NFC)* protocol. Next we will write down an invariant for the *Ping-Pong (PP)* protocol and show, using the approach of section 2, that it is a correct refinement of the *NFC* invariant. Finally, we will consider the invariant for *ABP* and show that it is a correct refinement of *PP*. Under the usual definitions of correctness, such refinements would not be allowed; the specification of *NFC* has no mention of acknowledgment messages, so standard approaches would not let us treat the invariant of *PP* as a refinement of that specification. But from a designer's point of view, one could argue that *PP* is indeed a refinement of *NFC* and that *ABP* is a refinement of *PP*.

Consider *NFC*. All we need say is that S sends out a sequence of values, and R receives some of these values. Let a be the output channel of S and b the input channel of R. The trace τ of $S \parallel R$ will be made up of elements of the form (a, v) representing a value v being sent out on channel a (by S) and of the form (b, v) representing the value v being received on channel b (by R). The invariant I_N of *NFC* should assert that the values received on b were sent out on a. We cannot require $\tau_{/b}$, obtained from τ by retaining only the values communicated on elements of the form (b, v), to be a prefix of $\tau_{/a}$, since *NFC* is supposed to allow message losses. Let $s \prec\!\prec t$ denote that if some zero or more elements of t are omitted, then the resulting sequence is equal to s. The invariant for *NFC* is:

$$I_N \equiv [\tau^N_{/b} \prec\!\prec \tau^N_{/a}]$$

τ^N is the trace of *NFC*, $\tau^N_{/b}$ the sequence obtained from τ^N by retaining only the values communicated on b. Note that I_N allows elements sent on a to be lost, but not duplicated or modified.

Next consider *PP*. The elements in its trace τ^P occur in a very precise order: first a value is sent (by S) on a, then it is received (by R) on b, then an acknowledgment is sent (by R) on a new channel, let us call it α, then the acknowledgment is received (by S) on another new channel, call it β; and then the whole pattern repeats. The invariant I_P will thus consist of four clauses, one of which, for instance, will be of the form ($|\tau^P|$ is the number of elements in τ^P; $\tau^P[k]$ is the k^{th} element of τ^P):

$$\forall k \leq |\tau^P|.[(k \texttt{ mod } 4) = 2] \Rightarrow [\mathit{chan}(\tau[k]) = b \wedge \mathit{val}(\tau[k]) = \mathit{val}(\tau[k-1])]$$

where *chan* returns the identity of the channel corresponding to a given element of the trace; and *val* returns the value communicated in the element.

[11]Holzmann considers some other refinements. One of these, the *sliding window protocol (SWP)*, allows S to send several messages while waiting for the acknowledgments of earlier messages. We will consider *SWP* towards the end of the section.

This states that the appropriate elements of τ^P correspond to (R) receiving on the channel b the value just sent (by S) on the channel a. We omit the remaining clauses since they are similar.

To show that *PP* is a correct refinement of *NFC*, we must first define a mapping ρ between the traces of *NFC* and *PP*: $\rho(\tau^P)$ is obtained by omitting all acknowledgment elements. It is easy to verify that if τ^P satisfies *PP*'s invariant, then $\rho(\tau^P)$ satisfies *NFC*'s invariant.

Now consider *ABP*. As in the case of *PP*, we have four channels, corresponding to sending and receiving the value, and sending and receiving the acknowledgment. But both the values as well as the acknowledgments have a one bit sequence number, alternating between 0 and 1. The invariant is somewhat complex since it should allow for data values as well as acknowledgments to arrive in a far more arbitrary order than in the case of *PP*. Essentially, it should assert that: values are sent on a with alternating bits as sequence numbers; that this element may be received (on b) more than once, but neither the data nor the sequence number is corrupted; that when R sends an acknowledgment on α, it must follow the receipt of a data item on b, and that the sequence number in the acknowledgment is the same as that in the item received on b; when a new value is sent by S on a, the acknowledgment with the appropriate sequence number must have been received; etc. We leave the the precise form of the invariant to the interested reader.

Next we have to define the ρ between *ABP* and *PP*. This is reasonably straightforward. All we have to do, given a trace τ^A of *ABP*, is to project out the extra data value messages that are sent (by S) following receipt of an acknowledgment with the wrong sequence number, and the extra acknowledgments sent following the receipt of a data value message with the wrong sequence number, and then remove the sequence numbers from each of the messages. If we had written down the invariant of *ABP* we would then be able to complete the proof of the fact that *ABP* is a legitimate refinement of *PP* by showing that the corresponding sequence would satisfy I_P.

Next let us briefly consider *SWP*, the sliding window protocol. The idea here is to allow the sender to be several steps ahead of the receiver; in other words, even if the acknowledgment of a given message has not yet been received, the sender can send further messages (which are within the 'window'). As acknowledgments for various messages are received, the receiver checks off those messages and 'slides the window forward'. This would require bigger sequence numbers than one bit; indeed we can consider the *ABP* as a sliding window protocol with a window of size 1. It would seem reasonable to consider *SWP* as a refinement of *ABP* but there is a problem: Consider a trace of *SWP* in which the sender has sent, say, five data values, and has only received acknowledgment for the first one (presumably the window size was four or more). No corresponding trace of *ABP* would satisfy *ABP*'s invariant because a new data value cannot be sent in that protocol before acknowledgment for the previous one had been received. But note that ultimately the acknowledgments of the previous messages must be received in *SWP* also. This suggests a generalization of our formalism: If the system S' is a refinement of S, then for every trace τ' that satisfies the invariant of S', there must be a trace τ'' such that:

τ' is a prefix of τ'';

τ'' satisfies the invariant of S'; and

the trace τ of S that ρ maps to τ'' satisfies the invariant of S.

4 Discussion

We have identified *interaction refinement* as an important tool in the design of distributed systems and provided a preliminary formalism that can be used to establish the correctness of systems designed using *IR*. It is worth repeating that when using *IR* what we are doing is to decide to implement a desired 'high-level interaction' (or a sequence of such interactions) by a particular sequence of 'low-level interactions'. We are *not* designing the internal *structure* of any process, as we would be if we were using one of the other refinement procedures. The key question we addressed was how to redefine the notion of correctness to allow for modifications in the interactions.

Another approach would be to accept *IR* as a design tool but not try to formalize it,i.e., we could simply throw away the original specifications once the refined specifications were arrived at. Our motivation was the belief that there is a reason why a designer performs a particular refinement, and why he thinks it is faithful to the intention behind the original specification; hence it is important to record such decisions, and record also the justification behind the decision in the form of the ρ function and the proof that the two specifications are related as required by the formalism.

Several authors have considered ideas similar to *IR*. We have already mentioned the work of [2] and [6]. Broy [3] considers many different types of refinement (including what we have called procedural and structural refinements). One of these is called interaction refinement; but Broy's notion seems closer to the interface refinement of [2] rather than the general notion we are considering. Thus it is not clear whether, using Broy's definition, the *ABP* would be a valid refinement of *PP*. The simulation approach (see for instance, [9]) seems more general, and it is possible that *IR* steps can be validated using that approach. One important distinction with our approach is that while ours is axiomatic, the simulation approach, as the name suggests, is more operational. Another important difference is that that approach does not distinguish between different types of refinement, whereas our focus is on interaction refinements.

Interestingly, Creveuil and Roman [4] consider, without using a formalism that explicitly allows for *IR* steps, the correctness of a message router whose design involves several interaction refinement steps; specifically, these steps introduce additional communications in the system. [4] relies, for its correctness arguments of these steps, on the fact that since such communications are not explicitly *forbidden* by the high level specification, they are allowed! While this is an interesting approach, it will not work if a particular interaction refinement *altered* any of the high-level interactions, as in some of our examples.

We should mention also the work of Francez and Forman [5] on *Interacting Processes*. They introduce the idea of a *team* – a template for a particular type of multi-party interaction. Whenever a set of processes wishes to engage in this type of interaction, they all 'enroll' in the particular team. Is a team an interaction refinement? We believe not. Rather, a team is a particular implementation of the various communications needed to achieve the multi-party interaction; it is certainly a refinement, but seems to be more an instance of structural refinement, rather than interaction refinement.

5 References

1. K Apt, N Francez, W deRoever, A proof system for CSP, ACM TOPLAS, 1980.

2. E Brinskma, B Jonsson, F Orava, Refining interfaces in communicating systems, *TAPSOFT '91*, *LNCS 494*, 1991.

3. M Broy, (Inter)Action refinement: the easy way, at `http://www4.informatik.tu-muenchen.de/paper_db/papers/broy_easy_way.html`

4. C Creveuil, GC Roman, Formal specification and design of a message router, ACM TOSEM, vol. 3, pp. 271-308, 1994.

5. N Francez, I Forman, Interacting processes, Addison-Wesley.

6. R Gerth, R Kuiper, J Segers, Interface refinement in reactive systems, *CONCUR 92*, *LNCS 630*, 1992.

7. CAR Hoare, Communicating Sequential Processes, Prentice-Hall, 1985.

8. G Holzmann, Design and validation of computer protocols, Prentice-Hall, 1991.

9. NA Lynch, Distributed algorithms, Morgan-Kaufman, 1996.

10. J Misra, KM Chandy, Proofs of networks of processes, IEEE Trans. on Software Eng., 1981.

11. S Owicki, D Gries, Verifying properties of parallel processes, CACM, 1976.