Distributed Commerce Transactions

Steven P. Ketchpel and **Hector Garcia-Molina**
Stanford University
Computer Science Department
Stanford, CA 94305
{ketchpel, hector}@cs.stanford.edu

Abstract

In situations where self-interested agents are interacting in an environment of distrust, commercial exchanges may be blocked due to a lack of trust. We propose a fully distributed algorithm that each agent may run to provide guarantees about the outcomes of such exchanges. The algorithm is shown in operation on two examples, one feasible and one not.

Introduction

In a multi-agent system with many information sources covering a vast range of topics, an agent with a query or information request might enlist the help of a number of brokers and sources in answering the query. These other agents may add value by combining results, or having better knowledge about or access to relevant sources. Information brokers may obtain documents from other sources and re-sell them to customers, who may in turn be brokering the documents to someone else. In exchange for provided services, these agents expect payment.

The environment is one of distrust, so that a customer will not give payment before being certain of getting the document. Similarly, a source will not provide a document before being certain of getting paid. Trusted intermediaries (such as a shopping mall, Internet service provider, electronic bank) alleviate the problem, providing a secure way to handle a transaction. The trusted intermediary receives the document from the source and money from the customer, then performs the exchange. If one party does not provide its promised piece, the exchange is canceled, and the goods are returned to their original owner.

If all of the participants could find a single trusted intermediary, the interaction would be straightforward. But these are not realistic assumptions. Moreover, the parties may have no previous history of interaction, may not exist in the same jurisdiction or even know each other's identities. Therefore, a single trusted intermediary is unlikely. More reasonable, we believe, is a series of pair-wise interactions, which taken in combination, result in the query being answered.

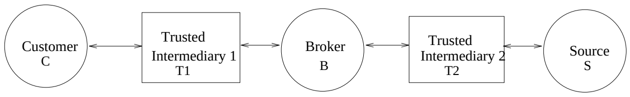

Figure 1: The parties involved in Example 1.

For example, a simple interaction between five agents is shown in Figure 1. In this case, the requesting agent C (which we call the *customer*) has an information request. C knows B, a *broker* or middleman that can locate relevant sources for such requests. The customer does not know the ultimate *source* S that provides the document, but as long as the broker B knows where to find some source, the request can be fulfilled. The exchanges between C and B and between B and S use *trusted intermediaries* T1 and T2 respectively to guarantee the outcome of the pair-wise exchange. Brokers, customers, and sources are collectively known as *principals*. Brokers and sources are *providers* of information.

In this example, the broker faces a potential risk: B may purchase the document from S, then find that C is no longer interested, and will not buy the document. In order to remove this risk, customer C can give the money for the document to trusted intermediary T1, enabling broker B to purchase the desired document from source S, confident that the customer will not be able to back out of the purchase. In order to protect itself from an untrusted source, broker B completes the pair-wise exchange with the source through trusted intermediary T2. The trusted intermediary T1 guarantees that when the broker B provides the document, the money that the customer has entrusted to it will be paid to B, with C receiving the document at the same time.

The risk of a broker's being stuck with an unwanted document is one source of dependencies among the pair-wise transactions. A second is the customer's desire to buy a conjunction of desired documents, agreeing to pay for each of the documents (also called *conjuncts)* only if *all* of the documents can be obtained. These are combined with the risks intrinsic to a pair-wise exchange to obtain the full set of risks:

1. When the agent acts as a customer, it never spends money without being guaranteed of receiving the promised document in return.
2. When the agent acts as a provider, it never sends a document to a customer without being guaranteed of receiving payment.
3. When the agent acts as a customer with a conjunctive request, it will never pay for one document unless it is able to obtain all of the conjuncts. (This risk is known as "buying half a conjunction.")
4. When the agent acts as a broker, it will never purchase a document unless it is guaranteed that a customer will re-purchase the document.

The actions that each agent may execute are limited. An agent may *send a document* to another agent, which concludes with the recipient agent knowing the content of the document. An agent may *send money* to another agent, resulting in the sender's balance being decremented by the amount, and the recipient's balance being credited by the amount. A customer may *request a document* from a provider, expecting it to be sent to a trusted intermediary; similarly, a provider may *request payment* be sent to the trusted intermediary from a customer. Trusted intermediaries can *notify* a principal that the exchange is lacking only the piece that must be provided by that principal. For instance, in a simple exchange, the trusted intermediary would receive the money from the customer then notify the provider that when the provider sends the goods, the exchange may be completed.

The goal is to find a way for the agents to use the permissible actions in order to move resources from the sources to the customer. A partial order of actions which is undertaken by the agents is called an *execution sequence* or just a *sequence*. However, the distrust between agents results in added constraints on these execution sequences. Intuitively, we want to prevent agents from being in a position where they might be cheated by a malicious agent. The solution to the problem must be *safe* for each agent, where a sequence is safe if it guards against all four of the risks enumerated above, even if other principals deviate from the expected sequence.

The process of finding a safe sequence is complicated by the fact that no single agent has a view of all the pair-wise exchanges and constraints. There is a tension between one agent's local view of the transaction and the global view of all of the pair-wise transactions that lead to the query's fulfillment. From the microscopic view, an agent will expect satisfaction or payment if it fulfills its part of the agreement, irrespective of what others do. But the macroscopic view is more complicated due to the interrelated dependencies, the sum of which no single agent is aware of. Yet, in many cases, a careful ordering of the component pairwise transactions can produce a safe sequence. Not every problem instance has a safe sequence, in which case we say the instance is *infeasible*.

More formally, a *distributed commerce transaction* instance (Ketchpel & Garcia-Molina 1996a)is described by:

- The agents and the resources (both documents and money) that they control.
- The connectivity of the agents, including the "first resort" agent for each request, and the trusted intermediary to be used between each pair of principals.
- The decomposition of a query into smaller pieces. It may be necessary to divide a query into sub-queries in order to match available resources that will successfully answer the query. Different agents may choose different decompositions.

A solution to that instance is a safe execution sequence. The new contribution of this paper is the description of a distributed algorithm for finding solutions to distributed commerce transactions.

Algorithm Overview

The algorithm for finding safe sequences that we describe here is fully distributed. Each agent is able to plan and carry out its actions based on its own state and the requests it receives from neighboring agents. The approach we describe is sound, so that if the algorithm runs to completion, the proposed execution sequence is safe. The algorithm described here is also complete. That is, if there is some safe sequence of the permitted actions that results in a completed exchange for a particular scenario, the algorithm will also find it or an equivalent one. The proofs of soundness and completeness may be found in (Ketchpel & Garcia-Molina 1996b).

Each agent operates as an autonomous processor, with its own knowledge limited to the documents for which it serves as a source, and the areas of expertise for agents with which it can communicate directly. These data, along with the customer's information request, form the inputs to the algorithm. The output is the sequence of steps which is executed by the agents in the system. This sequence will fulfill (if possible) the customer's request, while still ensuring that no agent risks entering an unsafe state. If the algorithm is unable to find a safe sequence, then none exists.

When one agent has an information request, it is sent to appropriate neighboring agents who either fulfill the request or re-distribute it to other sources not directly available to the original customer. Several pair-wise exchanges may then be required to move the information back to the ultimate customer. The status of each of these pair-wise exchanges is kept locally by the agents involved, and is updated as goods and money flow back and forth. The agents are "event-driven", reacting to events that are incoming messages describing customer requests, notifications from trusted intermediaries, or the delivery of documents or money. These

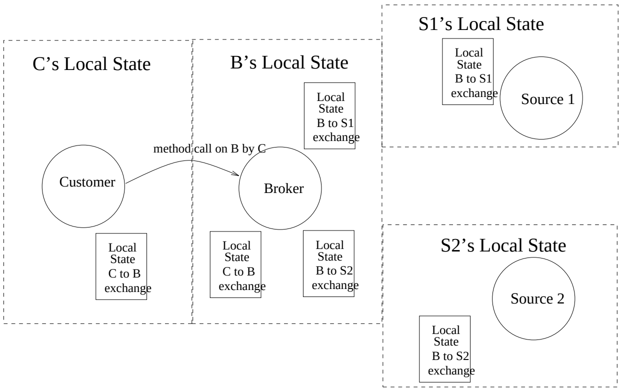

Figure 2: A conceptual view of the operation of the algorithm.

messages are represented as method invocations by the sender upon the recipient agent.

Figure 2 shows an example of a distributed commerce transaction where a customer contacts a broker who decomposes the request into two sub-queries that are sent to two different sources. This figure is overlaid with a representation of the different process spaces, showing the computational boundaries between the agents and which pieces of the state of the global exchange are available to each agent.

A simple description of these agents is as event processing loops. Their operation follows a simple cycle:

1. Wait for a message event (request, notify, delivery) to arrive.
2. Update the local state concerning the pair-wise exchange with the message sender, based on the content of the message.
3. Based on the new local state, select the next action.

The state update of Step 2 is deterministic and always well-defined. It records whether a document has been requested, sent, or received, and whether the corresponding payment has been requested, sent, or received. In Step 3, the recipient agent considers the new state of this pair-wise exchange, in combination with the state of other pair-wise exchanges that might be part of the same conjunction as the newly updated exchange. As a result of this consideration, the agent decides which actions to take next. There are four cases that must be separated:

1. *The agent has all of the pieces to fulfill the information request.* In this case, the agent is either done (if it is the ultimate customer) or can send the answer on to the requester.

   In the example of Figure 2, if the broker had already received the document from Source 2, then when it receives the document from Source 1, B has all of the necessary components to fulfill the request from C. Since B is not the ultimate customer (C is), B will send them on to C.

2. *The agent has promises from trusted intermediaries that they have the documents that will fulfill the request.* In this case, the agent can safely send money to the trusted intermediaries to obtain the documents, reducing the problem to the previous case.

   In the example of Figure 2, the broker will request the documents from Sources 1 and 2. The sources have nothing to lose by sending them to the trusted intermediaries that would facilitate the exchange between themselves and the broker. Therefore, when the trusted intermediaries (not shown in this figure) receive the requested goods, they send promises to the broker that when the broker sends money, it can be guaranteed to receive the desired documents in exchange. When both promises arrive at the broker, this case occurs.

3. *The agent and trusted intermediaries are still missing one piece for the request.* In this case, the agent guarantees payment to the source of the last piece, if its customer has done likewise. This may expedite the acquisition of the missing piece.

   In the example of Figure 2, if the broker has made requests of the two sources, but one has decided not to comply immediately, then this case will arise. The broker will receive notification from one of the trusted intermediaries saying that the first document is available. However, without the second document, the first is of no value. Therefore, the broker undertakes more effort to acquire the second document. By sending money to the trusted intermediary, the broker demonstrates its good faith without putting itself at any risk.

4. *The agents and trusted intermediaries are missing two or more pieces for the request, or the customer has not guaranteed payment.* In this case, the agent is reduced to merely asking sources to provide the necessary pieces without guarantee of payment. The sources may be willing to do so if they are not required to spend any resources. If they have to spend money to acquire these documents from other sources, however, they will not without a binding promise of payment.

   In the example of Figure 2, the broker is not willing to spend money for the documents from the two sources unless it is sure that it will obtain both. Therefore, rather than sending money, it first makes a request, asking one or both of the sources to forward the desired document to the shared trusted intermediary in order to move the exchange along.

Assumptions

We make several reasonable assumptions that simplify the process of solving distributed commerce transactions. First, we assume the presence of techniques such as watermarking and delivery receipts that ensure customers will not distribute illegal copies of the document

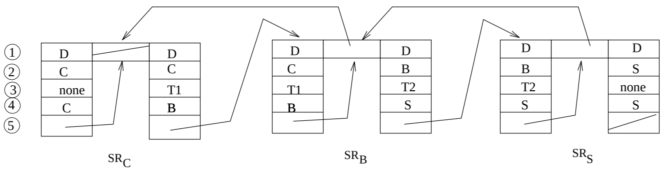

Figure 3: The partial contents of the SetRecords for Example 1. In each record, field 1 is the document requested; 2 is the requester; 3 is the trusted intermediary; 4 is the provider; and 5 is a linking pointer.

and sources will not send bogus goods. Second, we assume that the agents use a message delivery protocol which prevents lost messages. Finally, we make two assumptions which are particular to our algorithm. One, any pair of principals that wishes to make a pair-wise exchange has access to a shared trusted intermediary. And, two, the price of any single document is negligible compared to the resources of an agent. Therefore, a broker can always pay "out-of-pocket" to acquire a document without relying on its customer's payment. Relaxing these two assumptions is grounds for future work.

Algorithm Details

Since the inner workings of the algorithm are somewhat hard to follow in the abstract, we show the basics on the example from Figure 1. Customer C orders a document D from broker B, who in turn acquires it from source S. The consumer and broker make their exchange through trusted intermediary T1, while the broker and the source use T2 for their exchange.

The remainder of this section describes the each of the steps in the safe sequence. Figure 3 shows the data structures used and the relationship among the records constructed in this example. Each of the arch shapes is a *SetRecord*, with the *clientTaskRecord* appearing on the left, the *parent pointer* in the middle, and the *sourceTaskRecord(s)* on the right.

1. C's setting up the request at C. In order to initiate the exchange, the consumer creates a *TaskRecord* that will store the details of the exchange between two principals C and B. The first components of the TaskRecord (also abbreviated TR) are the agents and the document involved in the exchange, in this case C is obtaining D from B through T1. Since C knows about the areas of expertise of its immediate neighbors, C is able to select B as the most relevant to handle a request for D. C further knows that T1 is the trusted intermediary that it should use for transactions with B. This TaskRecord will also be updated as the exchange continues, reflecting the current status of the transaction.

Two status variables are used in each TR: one to record the status of the document, the other, the payment. For instance, the document's status might indicate that C has received D from T1.

However, this TaskRecord does not record all of the information that C has about the exchange. C wants the document D, and has set aside funds for it. In order to represent this commitment, we essentially model a separate exchange between two personas of C, one that wants the document, and the one that has the money. A new TR is created, with C playing the role of both client and source, with no trusted intermediary in the middle. By making each agent both a customer and a source, the algorithm is able to treat all of the agents in a uniform matter. In this special case, the banker persona has sent the money while the other persona has requested the document.

C is initiating the exchange with B in order to satisfy the exchange with its own banker persona. Therefore, there is a linkage between the two TaskRecords. The document received by C from B in the first TaskRecord will be sent to C's banker persona in the second. We show this linkage in a new data structure called a *SetRecord*, abbreviated SR. There is one SR per agent. The SR is composed of a "clientTR" for the TaskRecord between the agent of the SR and that agent's client (the banker persona in this case), and one or more "SourceTRs" between the agent of the SR and the providers (B, in this case). The agent of the SR obtains all of the documents listed in the SourceTR's, combines them, and sends them to the agent in the ClientTR.

The combination of source and client TR's within a SetRecord shows linkages within a single agent. The SR also contains a "Parent" pointer showing the linkages between agents. For example, when broker B acquires D from S, it does so ultimately because C requeseted it. That dependency would be recorded in the SetRecord as a pointer from B's SetRecord to C's SetRecord, its parent. Subsequently, when B receives the document from S, B knows to send it on to C. Each TR also maintains a pointer to allow the traversal of these SetRecord chains. A clientTR has a pointer to its containing SR. The SourceTRs' however, point to the SRs that are constructed by the source providing the document.

2. C's communicating the request and sending C's money to T1. Having constructed this SR, C starts the search for a safe execution sequence that will satisfy. Since C does not have D, C plans a way to obtain it, filling the source agent into the TR. Then, based on the current status of the transaction, C selects one of the four clauses described in the Overview.

In this case the third clause is selected, since one document, D, has not been acquired or requested. Therefore, C takes the next step to obtain the missing document. Since C's "banker persona" has allocated the money for the purchase of D, C is able to send its money to T1, the trusted intermediary, along with the SR which identifies the circumstances of the exchange. T1 stores locally the receipt of payment and saves the payment for future delivery to B, when B provides D.

3. T1's notifying B of the request, and the presence of C's money.

T1 notifies B that B will receive money as soon as D is provided. Since this request is a new one to B, B creates a new SetRecord, setting the contents of its clientTR to the same as the value of the SourceTR from C, reflecting our intuition that the "out-box" of the source is the "in-box" of the client.

In order to complete B's SetRecord, the Parent field must be filled in. Since B undertook this acquisition to fulfill C's request, C's SetRecord is the parent of B's. The last step in creating the SR is to break C's request into documents that B will be able to obtain. In this example, since B knows that S can directly provide D, this step is trivial, and no decomposition is required.

4. B's communicating the request and sending B's money to T2. As C did before it, B now invokes the algorithm on its new SetRecord. As before, the third clause is activated. Because C has guaranteed its intention to purchase by giving money to T1, B can safely give its own money to T2. B is confident that when S provides the desired document, B will be able to get its money back by giving the document to T1. B sends its payment to T2. T2, as T1 did before it, checks to see if this payment completes an exchange. Finding that it does not, T2 calls on S to continue the transaction.

5. T2's notifying S of the request and the presence of B's money. This wakeup call causes S to create a new SR, which we call SR_S, with B's SR as its parent, the incoming TR as its clientTR, and a new SourceTR showing that document D is being requested. But S quickly discovers that it has the document in question. Therefore, the finalized SourceTR sets up an "exchange" between two personas of S.

6. S's sending the document to T2. When S tries to determine what to do next, the first clause (showing that all documents have been received) is activated. Since S is not the final customer, the documents must be sent to their customer via trusted intermediary T2, that B shares with S.

7. T2's sending the document to B, B's money to S.

Since T2 already has the payment for this exchange from B, it will complete the exchange, sending the document to the broker B and payment to source S. S deposits the payment, updates the status of the TR, and checks to see if there is anything further to do for this transaction. But none of the clauses is activated, so processing at S ends. T2 accomplishes the second half of the exchange by sending the document to B.

8. B's sending the document to T1. When B gets the document, it checks the four clauses to determine the next step. B finds that the first clause is applicable, because all of the documents (in this case, just D) have been received. Since B is not the ultimate customer, it sends the document to trusted intermediary T1.

9. T1's sending the document to C, the money to B.

T1 sees a situation similar to the one T2 faced for the previous exchange, and continues it in the same way—B is sent the payment, and C, the document. Both actions result in status updates and evaluations of what step to take next. B finds that none of the clauses is warranted, so B is done with this transaction. C discovers that the first clause is activated, but since C is the ultimate customer, rather than sending the document on, the transaction is successfully completed. No further processing occurs at any site, and the example is concluded.

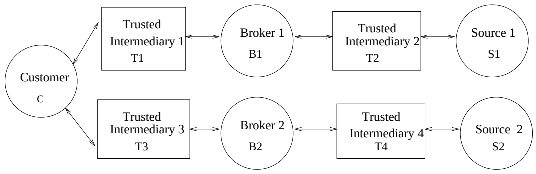

Figure 4: The parties involved in Example 2.

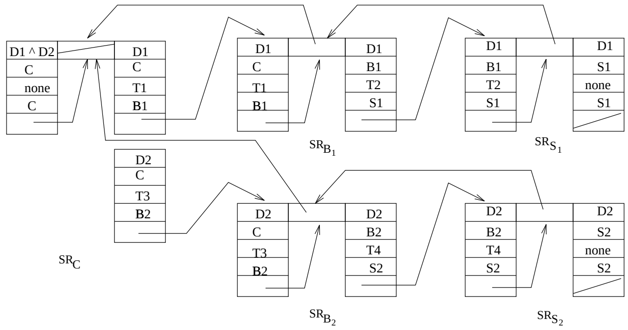

Figure 5: Contents of the SetRecords for Example 2.

Conjunctive Example

In the second example (Fig. 4), customer C desires two documents (D1 and D2) and interacts with brokers B1 and B2 in order to get them from S1 and S2. T1 is the trusted intermediary between C and B1, T2 is for C to B2, T3 is B1 to S1, and T4 is B2 to S2. The SetRecords that the algorithm generates for this exchange is shown in Figure 5. This example is infeasible, so the algorithm will not find a safe execution ordering.

The initial SetRecord for C is similar to that of the first example in that it has a clientTR showing a transfer between two personas of the same agent. We do see a difference however, in that the target information request may only be fulfilled by a conjunction of two components, D1 and D2. Since these documents are coming from different sources, there are two different TaskRecords in the SourceTRs set.

When C evaluates the algorithm on this SetRecord, only the fourth clause is activated. Since there are two documents that are not yet at the trusted intermediary, C is unable to advance payment. C faces the risk of "buying half a conjunction"—for example B1 providing document D1 while B2 gives up, and returns the payment. In that case, C has spent half of its money but not obtained the desired set of documents. So, instead, C performs the riskless action of requesting both documents from the respective brokers.

The processing performed by B1 and B2 is exactly symmetric, and the operations may be performed in parallel, or interleaved in some way unknown to C. Since there are no interactions, we will assume that B1 processes its request first. The first step that B1 undertakes is to create a new SetRecord, using C's TR as the client TaskRecord. Since C has already performed a decomposition requesting only D1 from B1, no further decomposition is performed.

B evaluates the algorithm on its SR. Although there is a single SourceTR which has not yet been received, the third clause is not applicable, because the broker has not received payment from C. Therefore, control falls through to the fourth clause, where B1 requests D1 from S1. In response to the request, S1 creates a new SetRecord.

Evaluating this SR, S1 recognizes that it has the desired document D1. Therefore, the TR is filled out showing a completed exchange from S1 to S1, using no trusted intermediary. Given that S1 now has all of the desired documents listed in the SR, the first clause is applicable. S1 sends D1 to T3, the trusted intermediary it shares with B1. T3 checks for whether this new arrival completes an exchange, but finds instead that no money has yet been sent for this document, so T3 notifies the client B1 that the document has been received, and the exchange can be completed as soon as the money is received.

When B1 learns this, the second clause is activated, since all of the desired documents in the SourceTRs set (just D1) have already been sent to the trusted intermediary. B1 checks whether C has guaranteed payment for this document by sending money to their trusted intermediary T1, but C has not, since facing the risk of buying half a conjunction it could not commit payment for either piece. Without funds committed from C, B1 is unable in turn to extend money to S1, in case C decides to retract its offer. Therefore, B1 cannot send money to T3, and this branch of the transaction blocks until payment is received from C.

On the other branch of the transaction, going from C to B2 to S2, a completely symmetric sequence of steps is occurring, yielding the same result: D2 is transferred to T4, but in spite of notification of this fact, B2 cannot transfer funds to pay for it. As in the other branch, completion of the exchange is blocked, and the global exchange cannot be completed.

Conclusion

In distributed network environments, what seems like a single sale to a customer may in fact be fulfilled by many sources and brokers contributing to a final information product. The customer wishes the whole sale will take place in an atomic fashion, so that if one part of the acquisition fails, none of the acquisitions succeed, and no money is spent. However, the sources have a different point of view. They are not concerned with the eventual disposition of documents that they sell, but expect to be compensated if they sell a document to a broker, even if that broker fails to generate a final sale to the customer. Therefore, to give the customer the desired semantics, but still provide protection to the brokers, the sequence of pairwise sales that lead to the full transfer must be carefully ordered.

We demonstrated a fully distributed algorithm that produces this ordering. We showed its operation on two examples, one where it found a safe execution sequence, and one where a safe sequence did not exist.

Building contractors, wedding consultants, information brokers, and travel agents are all examples of professionals who facilitate transactions between multiple parties. Yet in the on-line world today, the mechanisms for structuring transactions to include these intermediaries are not readily available. Moreover, in the widely distributed on-line world, there is no guarantee that all of the parties to a transaction will be in the same jurisdiction or trust a universal intermediary. Our framework obviates the need for a universally trusted intermediary, instead enabling transactions where only pair-wise trusted intermediaries exist. The formal properties of this framework provide a high degree of assurance that all of the parties in the transaction will profit from the interaction and will be protected from unscrupulous participants.

In (Ketchpel & Garcia-Molina 1996b; 1997) we have shown the soundness and completeness of this algorithm, as well as extenstions which permit direct trust between agents (not every transaction must use a trusted intermediary) and deadlines. We plan to extend this framework to explicitly model the uncertainty of source availability and delivery times using a decision theoretic framework. There may be cases when an agent will undertake a risky action (such as forwarding payment before being sure of receiving all the goods, or paying for goods which are not guaranteed to arrive before the deadline) when potential rewards outweigh the costs of a poor outcome, factoring in the relative likelihood of the outcomes.

References

Ketchpel, S., and Garcia-Molina, H. 1996a. Making trust explicit in distributed commerce transactions. In *Proceedings of the 16th International Conference on Distributed Computing Systems*, 270–281.

Ketchpel, S., and Garcia-Molina, H. 1996b. A sound and complete distriubted algorithm for distributed commerce transactions. Technical Report SIDL-WP-1996-0040, Stanford University, .

Ketchpel, S., and Garcia-Molina, H. 1997. Adding time to distributed commerce transactions. In , ed., *Submitted to IJCAI-97*. : .