Engineering Service Engagements via Commitments

Pankaj Telang, Anup Kalia, Munindar Singh

Abstract

A service engagement describes how two or more independent parties interact with each other. Traditional approaches specify these interactions as message sequence charts (MSCs), hiding underlying business relationships, and consequently complicating modification.

We present Comma, a commitment-based approach that produces a business model drawn from an extensible pattern library, and which yields flexible MSCs.

An empirical study shows that models produced via Comma yield superior flexibility (measured objectively), comprehensibility by others (measured subjectively), and take less time (measured objectively) and effort (measured subjectively) to produce.

Keywords: Commitments; Business service modeling; Choreography

Introduction

A service engagement involves two or more autonomous parties interacting as they pursue their business relationships. Existing approaches, however, disregard business relationships, emphasizing data and control flows among the participants. Consequently, they produce rigid operational models that mandate strictly ordered message exchanges. Introducing choice complicates such models.

We describe Comma, an approach that captures the commitments of each party to another to help produce an operational model that is correct at the business level, i.e., no party violates its commitments [1]. Comma goes beyond previous commitment-based approaches, e.g., [2], through its library of reusable business patterns.

We empirically compare Comma with a traditional UML-based modeling approach for service engagements. We find that models resulting from Comma are more flexible and comprehensible and take less time and effort to produce.

Scenario

Our method can accommodate complex scenarios such as Oracle's Quote-To-Cash process (http://www.oracle.com/us/industries/045546.pdf) [1]. For simplicity, however, we consider a purchase scenario involving a buyer (Buyer), a vendor (Paragon), the vendor's bank (SellerBank), a credit-card issuer (BuyerBank), a credit-card processor (Authorize), and two shippers (FedUp and UpFed).

Buyer selects goods from Paragon's website. Paragon displays the total charge. Buyer provides his credit-card information, which Paragon sends to Authorize. Authorize contacts BuyerBank and relays BuyerBank's approve-or-deny decision to Paragon. Paragon accepts the order if and only if BuyerBank approves the transaction. Buyer may retry a rejected order up to five times.

Upon success, BuyerBank transfers the authorized amount from Buyer's account to Paragon's account with SellerBank. Paragon pays Authorize a fee for each transaction. Paragon asks FedUp or UpFed to ship the goods, and pays their charges.

If the goods are damaged in transit, Buyer ships them back to Paragon. Paragon pays the shipper and requests Authorize to reverse the transaction. Authorize requests BuyerBank to credit Buyer's account.

Traditional Approach and Solution

By *traditional*, we mean an approach in which *message sequence charts (MSCs)* (UML sequence diagrams) are directly developed to capture a given scenario as elicited from stakeholders. Each interaction step from the scenario maps to one or more messages. (Some traditional approaches [3] use multiple UML diagram types [5] but we adopt one that doesn't.)

Let's review MSCs, simplified for our needs, as Figure 1 shows. A (vertical) lifeline represents a participant's view. A directed horizontal line represents a message. Time flows downward so the messages are naturally sequenced. UML provides operators to modularize interactions into fragments. OPT declares its single enclosed fragment optional. ALT declares its multiple enclosed fragments alternatives to each other. LOOP specifies bounded iterations of its single fragment. Each fragment may execute only when its stated *guard* expression (true if omitted) evaluates to true. The choice is nondeterministic when more than one guard in an ALT is true.

[Figure 1 about here.]

Figure 1(a) shows how Buyer orders goods from Paragon. Paragon requests credit-card details from Buyer, who provides them to Paragon. Figure 1(b)'s guard means Buyer has provided credit-card details to Paragon. Paragon requests Authorize to authorize the transaction on Buyer's account. Authorize, in turn, requests BuyerBank for authorization. BuyerBank either approves or denies the transaction.

Critique

MSCs directly capture a given scenario and are intuitive for stakeholders and developers. Traditionally produced MSCs, however, suffer from significant shortcomings. Often, they are over-specified. A particular sequence of steps may only be representative, yet the traditional approach codifies it. Moreover, MSCs promote operational details and hide business relationships: In case of an exception or opportunity, the participants lack a principled basis to change their interaction.

Commitments

In contrast, Comma emphasizes business relationships expressed using commitments. A *commitment* C(DEBTOR, CREDITOR, antecedent, consequent) means that the DEBTOR commits to the CREDITOR to bring about the consequent if the antecedent holds [1]. The debtor may create a commitment making it *active*. An *active* commitment is *conditional* if its antecedent is false, and *detached* if its antecedent is true. A *conditional* commitment *expires* if its antecedent timeout occurs, and a *detached* commitment is *violated* if its consequent timeout occurs. A commitment is *satisfied* when its consequent becomes true regardless of its antecedent.

For example, $C_0 = \mathsf{C}(\text{BUYER}, \text{SELLER}, \mathsf{goods}, \mathsf{pay})$ is a commitment from BUYER to SELLER to paying if the SELLER provides the goods. BUYER creates this commitment making it *conditional* by sending a purchase order to SELLER. If SELLER provides the goods, this commitment *detaches*, and BUYER becomes unconditionally committed to paying SELLER. If BUYER now fails to pay, the commitment is *violated*. Regardless of SELLER providing the goods, C_0 satisfies when BUYER pays SELLER. We may specify a reciprocal commitment, $\mathsf{C}(\text{SELLER}, \text{BUYER}, \mathsf{pay}, \mathsf{goods})$, which is *violated* if SELLER fails to provide goods after BUYER pays.

Since it is autonomous, a debtor may violate a commitment. As appropriate, one would include additional commitments in the model, e.g., a commitment from BUYER to pay a penalty for violating C_0.

[Figure 2 about here.]

We develop MSCs to operationalize commitments. Such MSCs model all possible executions that satisfy the given commitments. Figure 2 shows an MSC for C_0. BUYER creates C_0 by sending a purchase order. The ALT block specifies three alternate message fragments that satisfy the commitment: (a) goods followed by pay, (b) pay followed by goods, and (c) pay only. For each Comma pattern, we develop a set of MSCs that operationalize that pattern.

Comma

Table 1 summarizes the Comma methodology [4]. Comma involves a library of business patterns that capture common business relationships [1] along with a mapping of each pattern to the most general MSC that operationalizes it.

[Table 1 about here.]

Guided by Comma's library of business patterns, Step 1 identifies discrete business interactions (subscenarios) from a service engagement scenario. Steps 2 and 3 identify roles and business tasks from each subscenario. Step 4 assembles a business model from the business patterns of each subscenario. Step 5 creates an operational model by introducing Comma-specified MSCs for each pattern.

[Table 2 about here.]

We apply the Comma methodology to the purchase scenario. Table 2 shows the subscenarios that we extract per Step 1. For each subscenario, the table shows the roles and tasks per Steps 2 and 3, and the Comma pattern that we identify per Step 4.

[Figure 3 about here.]

[Figure 4 about here.]

Figure 3(a) shows a commercial transaction between Paragon and Buyer [1]. Here, C_1 and C_2, exemplify reciprocal commitments---interchanged antecedent and consequent. Specifically, Paragon commits to shipping goods for payment and Buyer commits to paying for goods shipped. Figure 3(b) shows how Paragon outsources the shipping of goods to a shipper. Here, C_2 is Paragon's commitment to Buyer to ship the goods; C_7 is Paragon's commitment to the shipper to pay if the shipper creates C_5; C_6 and C_7 are reciprocal commitments, and C_5 is the shipper's commitment to Buyer to unconditionally (antecedent is true) ship the goods.

Figure 4 assembles the instantiated patterns for all the subscenarios into a comprehensive business model.

The following instantiate <u>outsourcing</u>:

C_2, C_5, C_6, C_7: as above.

C_9, C_{10}, C_{11}, C_{12}: Paragon outsources picking up goods returned by the buyer to the shipper.

The following instantiate <u>commercial transaction</u>:

C_1, C_2: as above.

C_3, C_4: Paragon and Authorize to exchange payment for authorizing (either approving or denying, $\mathsf{CCApproveA} \vee \mathsf{CCDenyA}$) a credit card.

C_8, C_9: Buyer and Paragon to provide a refund for returning the goods.

And, the following instantiate <u>unilateral commitment</u>:

C_{13}: BuyerBank commits to Authorize to approve or deny credit cards.

C_{14}: SellerBank commits to pay BuyerBank.

[Figure 5 about here.]

Figure 5 shows selected MSCs produced via Comma. Note that *offerPrice*, orderGoods, *reqCCVerification*, *reqPayment*, and *reqCCVerificationB*, respectively, signify the creation of C_1, C_2, C_3, C_4, and C_{13}. Importantly, these MSCs arise *modularly* from Figure 4's business patterns.

Figures 5(a-b) follow the commercial transaction of Figure 3(a). Here, Buyer orders goods from Paragon. Part(a) shows two ALT fragments. Either Buyer sends *orderGoods* to Paragon, creating C_1 and, subsequently, Paragon sends *offerPrice* to Buyer creating C_2. Or, Paragon creates C_2 followed by Buyer creating C_1. Part(b) shows Paragon requesting credit-card details, and the buyer providing the details. To reduce clutter, we omit message annotations, e.g., *orderGoods* means the creation of C_1, as Figure 15(a) in the appendix shows.

In Figures 5(c-d), Paragon requests Authorize and, in turn, Authorize requests BuyerBank, to authorize the transaction. Part(c) shows two ALT fragments: (1) Paragon sends *reqCCVerification* to Authorize, creating C_3; subsequently, Authorize sends *reqPayment* to Paragon, creating C_4. (2) Authorize sends *reqPayment* to Paragon, creating C_4, followed by Paragon sending *reqCCVerification*, and creating C_3. Part(d) shows Authorize sending *reqCCVerificationB* followed by BuyerBank sending either *CCApproved* or *CCDenied*. Note that *reqCCVerification* makes proposition reqCCVerification true and detaches C_{13}. And, *CCApproved* or *CCDenied* make verifyCCB true, satisfying commitment C_{13}. Further, *CCApproved* makes pay true, satisfying C_1 and detaching C_2.

Comma employs Boolean expressions over message names as guards, thereby capturing desired interleavings. Although Comma accommodates multiparty MSCs, because of the guards, two-party MSCs are adequate (as in our examples) for capturing multiparty interactions.

The foregoing shows that Comma yields superior MSCs than Traditional. First, we know when a commitment is satisfied or violated. Thus commitments provide a standard for correctness with respect to a participant: we can verify whether an MSC violates a commitment and that commitment's creditor should find that MSC unacceptable. Second, Comma yields logically distinct MSCs based on the patterns. These MSCs compose, thereby improving flexibility during enactment.

Evaluation: Empirical Study

Our empirical study sought to evaluate if developers could obtain Comma's benefits. Our study addressed the following threats to validity [7].

Differences in expertise. We employed a within-subject design: each subject applied both approaches so the differences have no effect.

Learning across subjects. We required subjects to work independently to assess Comma's effectiveness on individuals.

Instrumentation or tools. All subjects used IBM Rational Software Architect 8.0.2 for developing their MSCs; for the Comma business model, the subjects used an Eclipse plugin (http://research.csc.ncsu.edu/mas/code/Protos).

External validity or whether the results would apply in practice. Our subjects were 39 computer science graduate students: seven were currently employed professionals (mean experience: five years), 15 were previously employed and returned to school (mean experience: two years), and 17 had no industry experience. We discerned no effect of experience on the reported metrics (see appendix).

Dependent Variables

We measure the following dependent variables for each approach, and compare them via statistical significance tests.

Time (in minutes) taken to create a model. Summed over each subject's reports.

Difficulty a subject perceives in modeling: an integer 1-5, interpreted as *extremely easy*, *easy*, *neutral*, *difficult*, and *extremely difficult*. Average over each subject's reports, weighted by time spent in each report.

Flexibility: the number of executions a model permits. Greater flexibility in general leads to more choices for a participant. We employ two measures of flexibility.

- **MSC count:** indicates modularity and generally greater interleavings of messages from multiple MSCs.
- **Count of ALT, OPT, and PAR fragments:** indicates more numerous possible executions.

Scenario coverage: *high* (covers the entire scenario), *medium*, *low*, and *very low*.

Scenario precision: *high* (no unnecessary aspects), *medium*, *low*, and *very low*.

Comprehensibility: *high* (easy for a human to comprehend), *medium*, *low*, and *very low*.

Results

Each subject submitted a worklog three times a week, reporting time spent and perceived difficulty. We computed flexibility programmatically and subjectively judged coverage, precision, and comprehensibility from the final solution.

Flexibility

[Figure 6 about here.]

Figure 6 shows boxplots for our flexibility measures. The median MSC count is higher for Comma (five) than for Traditional (one), and the median sum of ALT, OPT, and PAR fragments is higher for Comma (eight) than for Traditional (five). We attribute the higher flexibility of Comma to its foundation in commitments.

Quality

Figure 7 shows that Comma yields superior subjective quality results---scenario coverage, precision, and comprehensibility---than Traditional. (The appendix presents the objective quality results.)

[Figure 7 about here.]

Time and Difficulty

Figure 8 shows boxplots of the time and difficulty reported by subjects. Traditional yields higher median time (240 versus 140) and difficulty (3 versus 2.1) than Comma. We attribute Comma's improved efficiency and ease to its reusable patterns.

[Figure 8 about here.]

Discussion

Our results support our claims of Comma's superiority over Traditional for modeling service engagements. We attribute Comma's higher coverage and comprehensibility to its systematic nature and reusable patterns. Statistical hypothesis testing (see appendix) shows that, at the 5% significance level, Comma performed better than Traditional with respect to difficulty, number of fragments, and number of MSCs, but not with respect to time taken.

For logistical reasons, we had subjects first apply Traditional, then Comma. Since Traditional and Comma models have no common elements, we assume the particular order is irrelevant. A study comparing Comma with RosettaNet (in *both* permutations) corroborates this assumption [4]. Our subjects were graduate students in Singh's class. We mitigated potential biases (subjects' and ours) by reviewing surveys after posting grades, as we informed subjects prior to the study. A threat to validity is that the target population of Comma users would be business analysts, not computer scientists: we would conjecture that analysts would find Comma even more superior to Traditional.

Our evaluation employed a scenario of moderate complexity so our subjects could complete the exercises within a few weeks. For more complex scenarios, we conjecture Comma to gain over Traditional.

A *choreography* specifies message exchanges among participants in operational terms from a global perspective [6]. Thus, Comma and Traditional are approaches to produce choreographies.

Other high-level modeling approaches lack the combination of rigor and flexibility of commitments. For example, value transfer [8] produces a semiformal, centralized model, and without support for flexible operationalization.

We hope to compare Comma with more elaborate methodologies such as UMM 2.0 [9].

References

[1] Pankaj Telang, Munindar Singh. Specifying and verifying cross-organizational business models. *IEEE Trans. Services Computing*, 5(3):1-14, 2012.

[2] Michael Winikoff. Designing commitment-based agent interactions. *Proc. IEEE/WIC/ACM International Conf. Intelligent Agent Technology*, 363-370, 2006.

[3] Bernhard Bauer, James Odell. UML 2.0 and agents. *Engineering Apps. Artificial Intelligence*, 18(2):141-157, 2005.

[4] Pankaj Telang, Munindar Singh. Comma: A commitment-based business modeling methodology and its empirical evaluation. *Proc. International Conf. Auton. Agents MultiAgent Syst.*, 1073-1080, 2012.

[5] Object Management Group. *UML 2.0 Superstructure Specification*, 2004.

[6] Munindar Singh, Michael Huhns. *Service-Oriented Computing*. Wiley, 2005.

[7] Claes Wohlin, Per Runeson, Martin Host, Magnus Ohlsson, Bjorn Regnell, Anders Wesslen. *Basics of Software Engineering Experimentation*. Kluwer, 2001.

[8] Roel Wieringa, Jaap Gordijn. Value-oriented design of service coordination processes. *Proc. ACM Symp. Applied Computing*, 1320-1327, 2005.

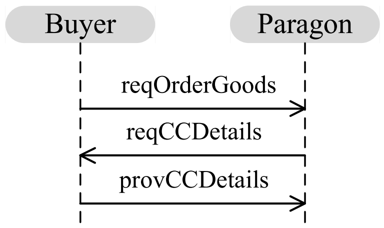
[9] UN/CEFACT. *Unified Modeling Method, 2013. http://umm-dev.org/umm-specification/*

Pankaj Telang is an IT Architect with Cisco Systems. He earned a Ph.D. from NC State University. Telang's research interests include business processes and software engineering.

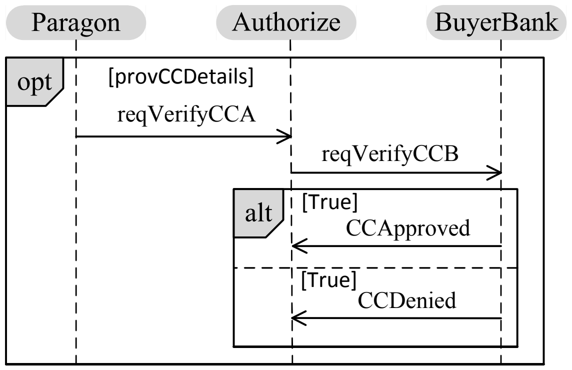
Anup Kalia is a Ph.D. student at NC State University. Kalia's research interests include ad hoc business processes and multiagent systems.

Munindar Singh is a Professor at NC State University. His research interests include multiagent systems and service-oriented computing. Singh is an IEEE Fellow.

Figures



(a) Place order.



(b) Authorize transaction.

Figure 1: Representative MSCs developed following the traditional method. The appendix includes the entire set.

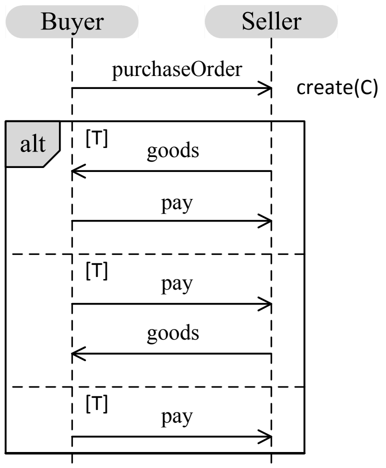
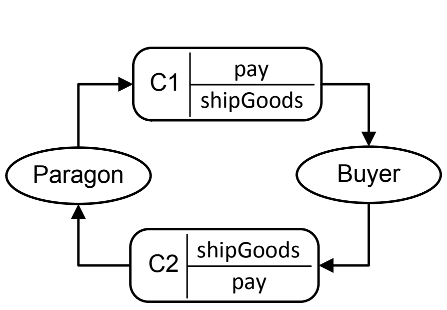
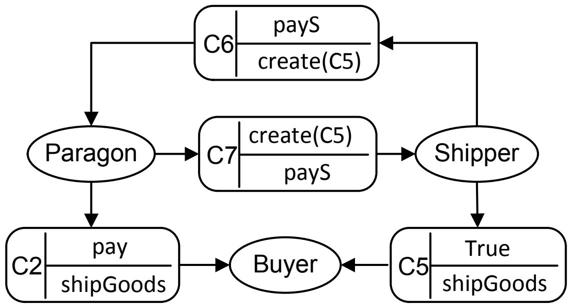


Figure 2: The MSC that operationalizes a commitment from BUYER to SELLER.



(a) Commercial transaction.



(b) Outsourcing.

C_1	C(PARAGON, BUYER, pay, shipGoods)
C_2	C(BUYER, PARAGON, shipGoods, pay)
C_5	C(SHIPPER, BUYER, $\top$, shipGoods)
C_6	C(SHIPPER, PARAGON, payS, create(C_5))
C_7	C(PARAGON, SHIPPER, create(C_5), payS)

Figure 3: Representative business patterns. An oval is a role and a rounded rectangle is a commitment, showing its name (left), antecedent (top), and consequent (bottom). Each commitment has a directed edge to its creditor and from its debtor.

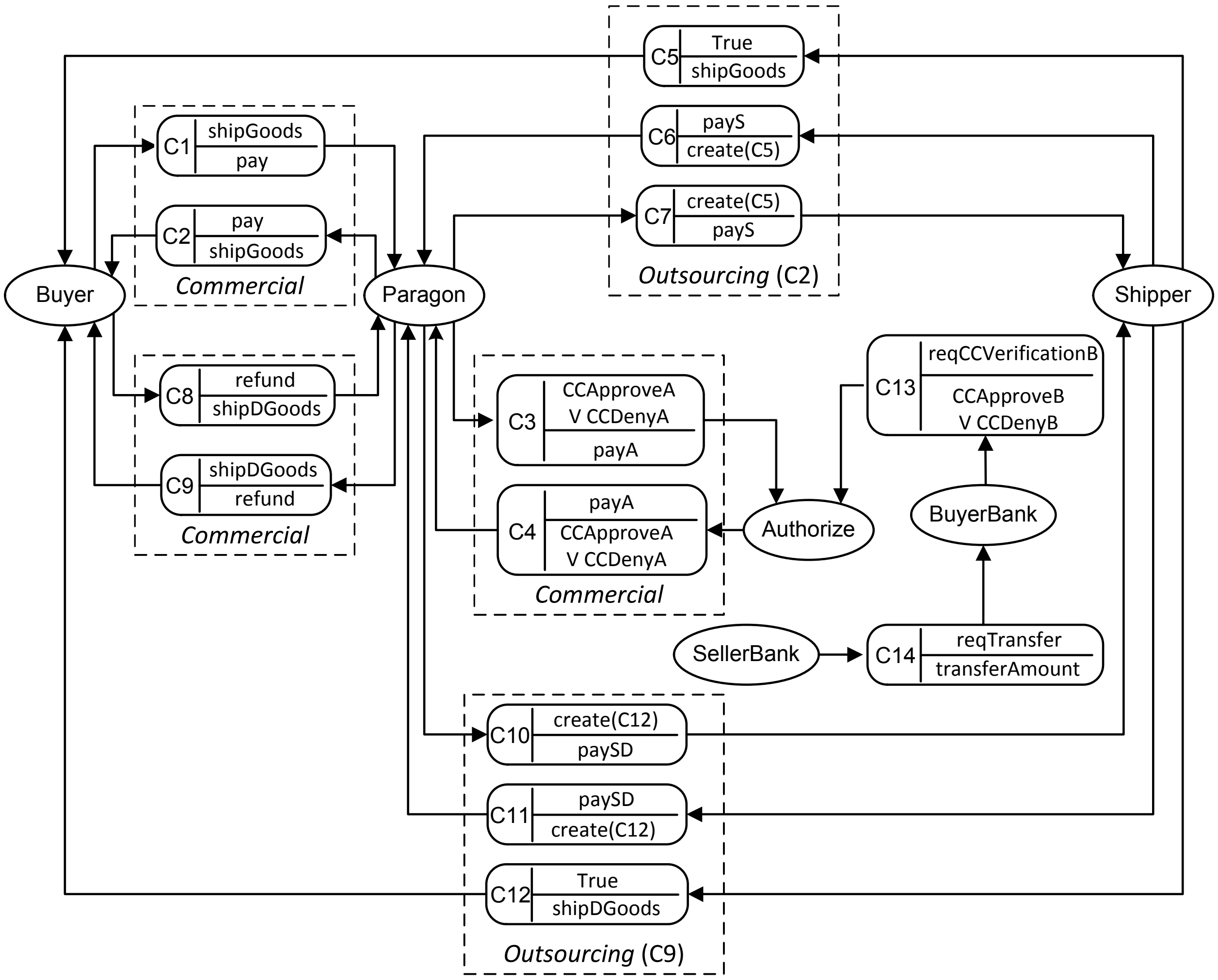
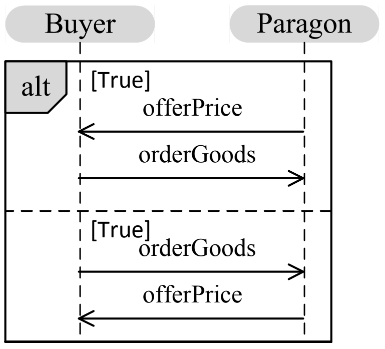
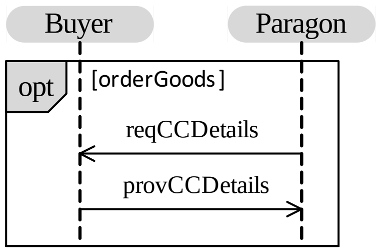


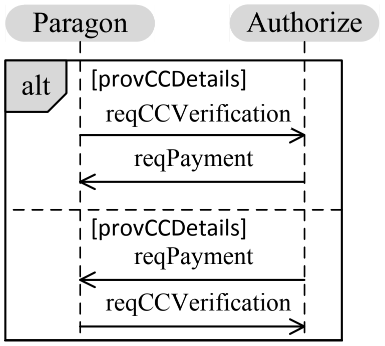
Figure 4: The resulting business model.



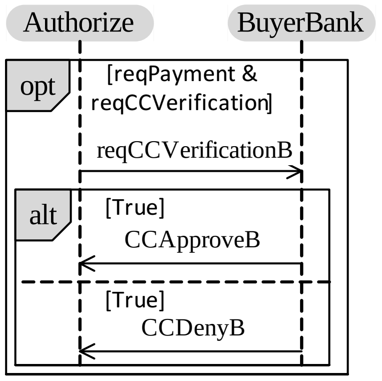
(a) Place order.



(b) Provide credit card.



(c) Authorize verifies credit card.



(d) BuyerBank verifies credit card.

Figure 5: Representative MSCs produced via Comma. The appendix includes the entire set.

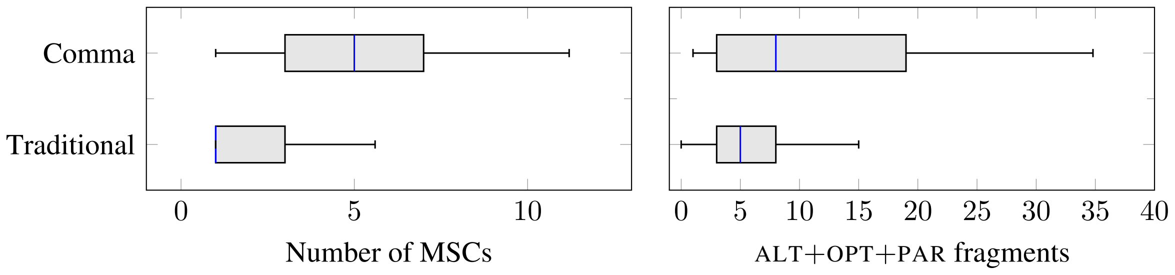


Figure 6: Flexibility, as judged based on objective criteria.

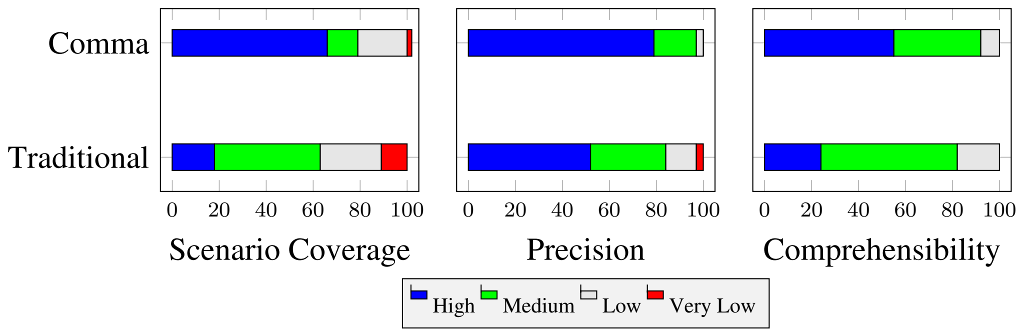


Figure 7: Distributions of quality, as judged by experts.

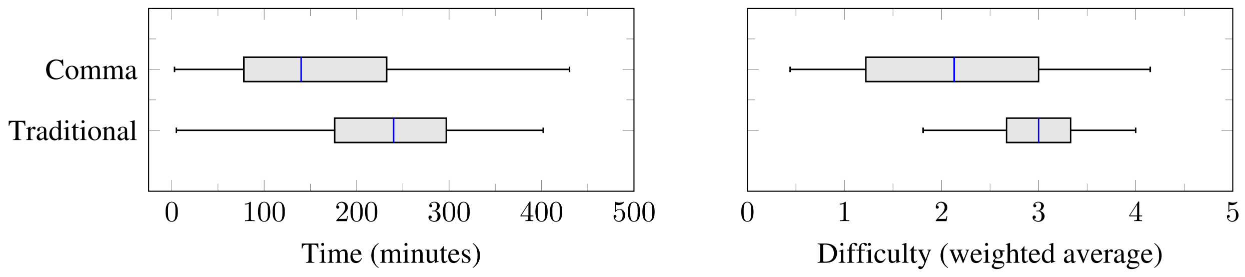


Figure 8: Time taken and average perceived difficulty (weighted by time spent), both reported by subjects.

Tables

Table 1: Comma summarized.

Step	Description	Input	Output
1	Extract subscenarios	Business scenario	Subscenarios
2	Identify roles	Subscenario	Roles
3	Identify business tasks	Subscenario	Tasks
4	Introduce a Comma pattern for each subscenario	Comma pattern, subscenario, roles, tasks	Business model
5	Introduce MSCs	Comma pattern MSCs, subscenario, roles, tasks	Operational model

Table 2: Comma applied to our scenario.

Subscenario	Roles	Tasks	Pattern
Purchase goods	Buyer, Paragon	pay, shipGoods	Commercial
Return goods	Buyer, Paragon	refund, shipDGoods	Commercial
Authorize: Verify credit card	Paragon, Authorize	payA, CCApproveA, CCDenyA	Commercial
Outsource shipping	Buyer, Paragon, Shipper	payS, shipGoods	Outsourcing
Outsource return goods shipping	Buyer, Paragon, Shipper	paySD, shipDGoods	Outsourcing
Transfer amount	SellerBank, BuyerBank	reqTransfer, transferAmount	Unilateral
BuyerBank: Verify credit card	BuyerBank, Authorize	reqCCVerificationB, CCApproveB, CCDenyB	Unilateral

Online Appendix

Hypotheses

We now present the claims regarding Comma's effectiveness as a set of alternative hypotheses.

$\mathbf{H}_{Time}^{\mu_c<\mu_u}$ The mean time to develop a Comma model μ_c is less than the mean time to develop a Traditional model μ_u. This hypothesis claims that Comma is more efficient than Traditional.

$\mathbf{H}_{Difficulty}^{\mu_c<\mu_u}$ The mean difficulty that the subjects encounter in Comma modeling μ_c is lower than what they encounter in Traditional modeling μ_u.

$\mathbf{H}_{Alt-Opt-Par}^{\mu_c>\mu_u}$ The mean sum of alternate, option, and parallel fragments in Comma MSCs μ_c is higher than in Traditional MSCs μ_u. This hypothesis claims that Comma yields more flexible MSCs as compared to the Traditional approach.

$\mathbf{H}_{MSC-Count}^{\mu_c>\mu_u}$ The mean number of MSCs in a Comma model μ_c is higher than that in a Traditional model μ_u. This hypothesis claims that Comma produces more flexible MSCs as compared to Traditional.

A null hypothesis corresponding to an alternative hypothesis claims that there is no significant difference between the two means that the alternative hypothesis is testing. For example, corresponding to the alternative hypothesis $\mathrm{H}_{Time}^{\mu_c<\mu_u}$, the null hypothesis $\mathrm{H}_{Time}^{\mu_c=\mu_u}$ claims that there is no significant difference between the mean time for Comma modeling μ_c and the mean time for Traditional modeling μ_u.

[Table 3 about here.]

Table 3 shows the results of Student's t-test for the above hypotheses. The p-value for $\mathrm{H}_{Time}^{\mu_c=\mu_u}$ is 0.196, which is greater than 0.05. We therefore accept the null hypothesis. This indicates that the Comma modeling time is not significantly lower than Traditional modeling time. We reject the remaining null hypotheses since their p-values are lower than 0.05, and conclude that the alternate hypothesis hold. That is, Comma is less difficult than Traditional, and Comma is more flexible (higher alt-opt-par, higher MSC count) than Traditional.

Complete Set of Traditionally Generated MSCs

This section presents all of the MSCs we developed using Traditional. In Figure 9(a), a buyer sends an order to Paragon. Paragon requests payment in the form of credit-card details from the buyer. The buyer sends credit-card details to Paragon.

[Figure 9 about here.]

In Figure 9(b), after the buyer provides credit-card details, Paragon requests Authorize to verify and authorize the credit-card transaction. Authorize requests BuyerBank to verify and authorize the credit-card transaction. BuyerBank either approves or denies the authorization request.

In Figure 10(c), if BuyerBank approves the credit-card transaction, Authorize informs Paragon of the approval. Paragon informs BuyerBank of the order acceptance. Paragon pays transaction fees to Authorize.

[Figure 10 about here.]

In Figure 10(d), after informing BuyerBank of order acceptance, Paragon requests a shipper to deliver the order to Buyer.

In Figure 11(e), once the Buyer's credit-card details are rejected, the buyer can reorder goods by providing another credit card. The buyer can attempt placing an order up to five times.

[Figure 11 about here.]

In Figure 12(f), after a shipper receives the request from Paragon for delivering the order, the shipper brings about the delivery of the goods to the buyer.

In Figure 12(g), after Paragon accepts the buyer's order, BuyerBank transfers funds equal to the order amount to SellerBank. SellerBank notifies Paragon of the receipt of the funds.

[Figure 12 about here.]

In Figure 13h, Paragon pays the shipper after requesting the shipper to deliver the goods. In Figure 13i, if BuyerBank denies the credit-card authorization request, then Authorize informs Paragon of the denial. Paragon informs the buyer of the denial.

[Figure 13 about here.]

In Figure 14(j), after the goods are delivered, the buyer notifies Paragon of the damage to the goods. Paragon requests the buyer to return the damaged goods. The buyer requests a shipper to ship the damaged goods back to Paragon, and the shipper accepts the request. The shipper returns the damaged goods to Paragon, and Paragon pays the shipping fees to the shipper. Paragon notifies the buyer of the refund amount.

[Figure 14 about here.]

In Figure 14(k), after notifying the buyer of the refund amount, Paragon requests Authorize to credit the refund amount on the buyer's credit card. Authorize requests BuyerBank to credit the refund amount. BuyerBank informs Authorize of crediting the requested amount, and Authorize informs Paragon of crediting the requested amount.

Complete Set of MSCs Generated via Comma

Since the guard is true, this MSC may execute unconditionally. In response, Paragon sends pricing information to the buyer. Alternately, Paragon sends an offer to a buyer, and in response the buyer orders goods from Paragon. By sending an order to Paragon, the buyer commits to paying Paragon if Paragon ships the goods (C_1). Conversely, by sending an offer to a buyer, Paragon commits to the buyer to shipping the goods if the buyer pays (C_2). In Figure 5(b), Paragon requests Authorize to verify the credit card, and then Authorize requests Paragon to pay. Alternately, Authorize first requests Paragon to pay, and then Paragon requests Authorize to verify the credit card. By requesting the credit-card verification, Paragon commits to Authorize to paying if Authorize verifies the credit card (C_3). Conversely, by requesting payment from Paragon, Authorize commits to verifying the credit card if Paragon pays (C_4). In Figure 5(c), Authorize requests BuyerBank for credit-card verification. In response, BuyerBank verifies the credit card and either approves or denies the transaction. Note that this MSC executes only after Authorize requests payment from Paragon.

[Figure 15 about here.]

[Figure 16 about here.]

[Figure 17 about here.]

[Figure 18 about here.]

[Figure 19 about here.]

[Figure 20 about here.]

[Figure 21 about here.]

[Figure 22 about here.]

Outsourcing Pattern

[Figure 23 about here.]

Figure 23 shows the *outsourcing pattern* [1]. An outsourcer delegates a task to a subcontractor. Here, C_1 is the original commitment from the outsourcer to a client to execute a task if the client pays the outsourcer (payOut). C_2 is the outsourced commitment from the contractor to the client to execute the same task. The antecedent of C_2 is true ($\top$), which means that it is unconditional. C_3 and C_4 are the commitments in which the outsourcer and the contractor commit to pay (payCon) and to create C_2, respectively.

[Figure 24 about here.]

Figure 24 shows the MSCs for the outsourcing pattern. In Figure 24(a), the outsourcer sends m_1 to the client, which creates commitment C_1. The client sends payOut to the outsourcer upon receiving m_1, which detaches C_1 since it is C_1's antecedent. In Figure 24(b), after receiving m_1, the outsourcer sends m_2 to the contractor, and after receiving m_2 the contractor sends m_3 to the outsourcer. Alternatively, the contractor first sends m_3 to the outsourcer, and after receiving m_3, the outsourcer sends m_2 to the contractor. m_2 creates C_3 and m_3 creates C_4. In Figure 24(c), after m_2 and m_3 are exchanged, the outsourcer sends payCon to the contractor and the contractor sends m_4 to the outsourcer in either order. Now payCon satisfies C_3 and detaches C_4; and, m_4 creates C_2 and satisfies C_4. In Figure 24(d), after m_4 is exchanged, the contractor sends task (message) to the client. This satisfies C_1 and C_2 since task is their consequent. As part of creating a model, a modeler substitutes the message labels m_i with domain-specific terms.

Objective Quality

Guards in MSCs relate them to each other. For example, a message in one MSC may appear as a guard on an OPT fragment in another: the fragment may not execute until the specified message is transmitted. Omitting necessary guards leads to undesired executions: thus (wrongly) missing guards is an effective measure of MSC quality.

[Figure 25 about here.]

Figure 25(left) shows that more subjects missed more guards with Traditional than with Comma. Figure 25(right) shows that Traditional fares worse on incorrect MSCs than Comma.

Effect of Experience

We found no tangible effect of industry experience on the performance of the subjects with respect to the dependent variables of this study. Figure 26, 27, and 28 show the scatter plot and a fitted linear regression line of time, difficulty, and quality with respect to experience.

[Figure 26 about here.]

Intuitively, experienced subjects should take lesser modeling time. However, Figure 26(left) shows that the modeling time increases with experience for Traditional. But Pearson's correlation coefficient in this case is 0.132, which indicates very low correlation. We conjecture that this result is due to confounding factors.

Figure 26(right) shows that the modeling time decreases as the experience increases for Comma. Pearson's correlation coefficient in this case is -0.129, which indicates weak correlation.

[Figure 27 about here.]

Figure 27 shows that the difficulty encountered reduces as the experience increases for both Traditional and Comma. Difficulty is weakly correlated with experience: we obtain correlation coefficients of -0.436 and -0.305 for Traditional and Comma, respectively.

The values shown in Figure 28 are arithmetic means of the values for the three quality measures.

[Figure 28 about here.]

Figure 28 shows that the quality improves with experience for both Traditional and Comma. The correlation coefficients for quality with respect to experience are 0.076 and 0.061 for Traditional and Comma, respectively, indicating weak correlation.

Figures

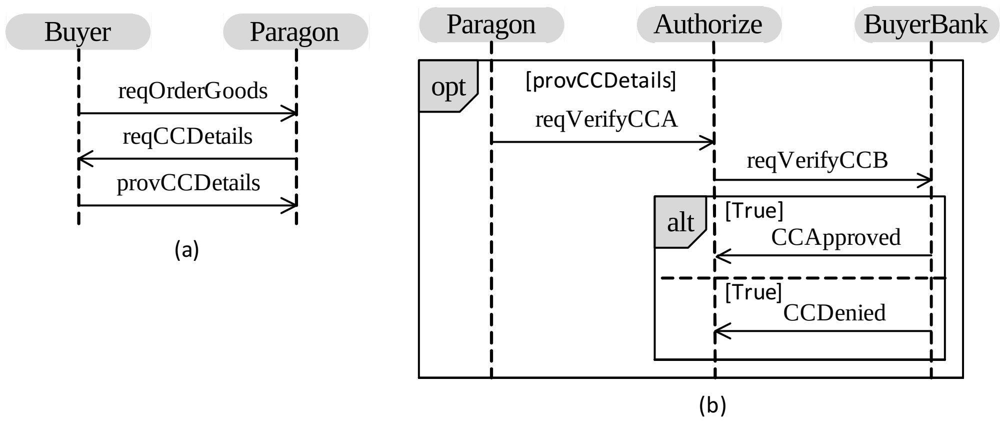


Figure 9: MSCs representing verification of credit-card (CC) details.

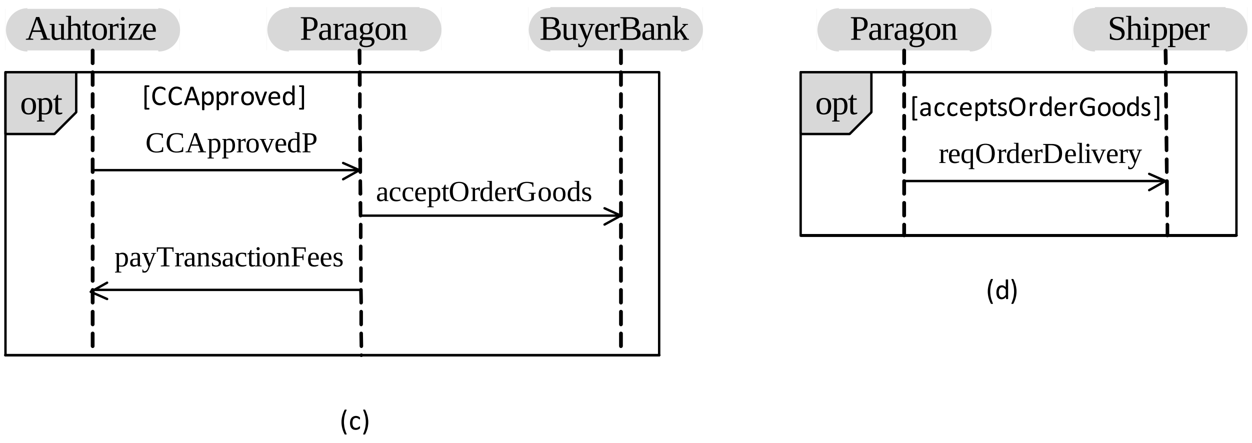
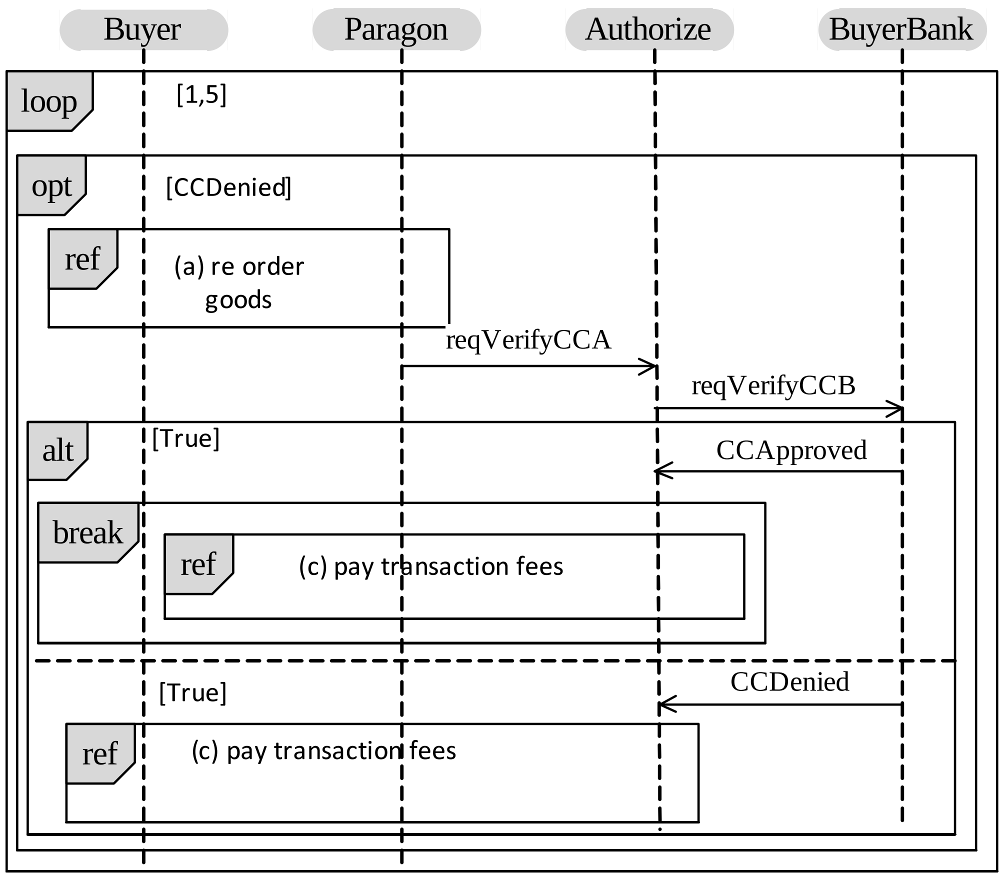
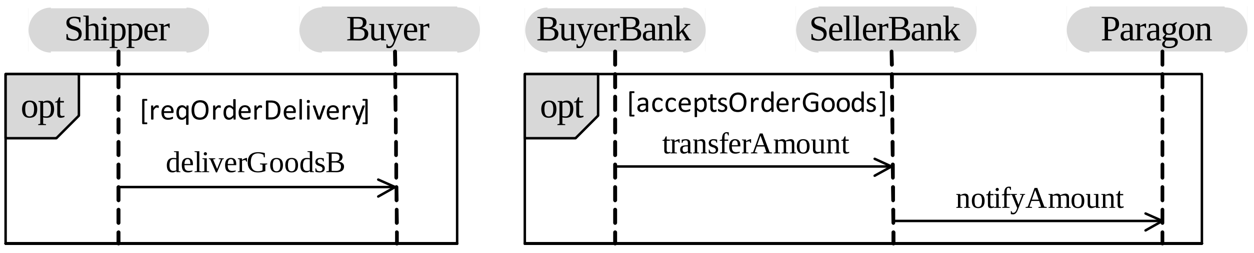


Figure 10: MSCs representing ordering of goods.



(e)

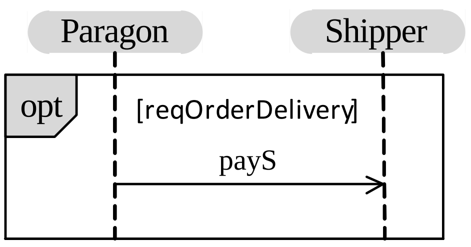
Figure 11: MSCs representing reordering of goods from Buyer.



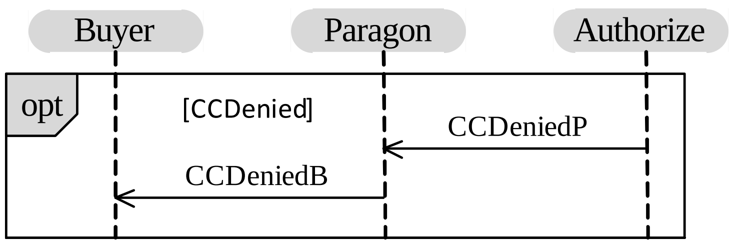
(f)

(g)

Figure 12: MSCs representing delivery of goods and payment for goods.

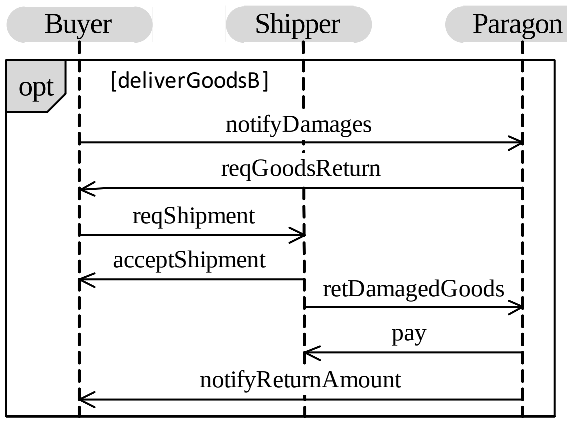


(h)

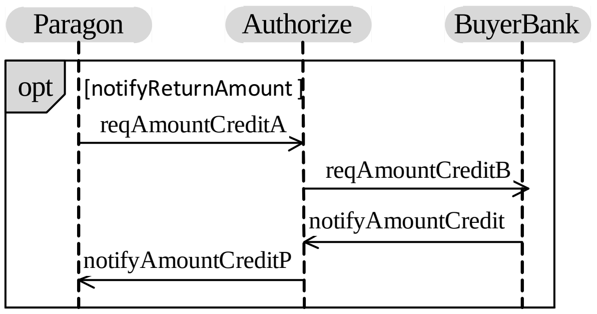


(i)

Figure 13: MSCs representing payment for shipment and denial of credit-card details.

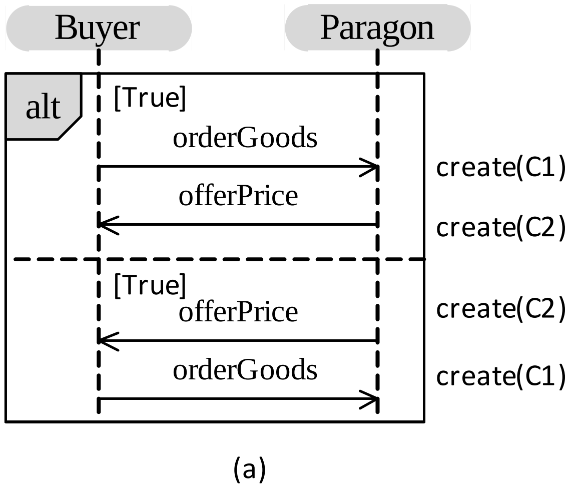


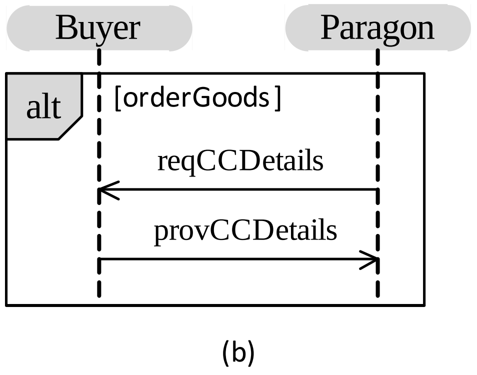
(j)



(k)

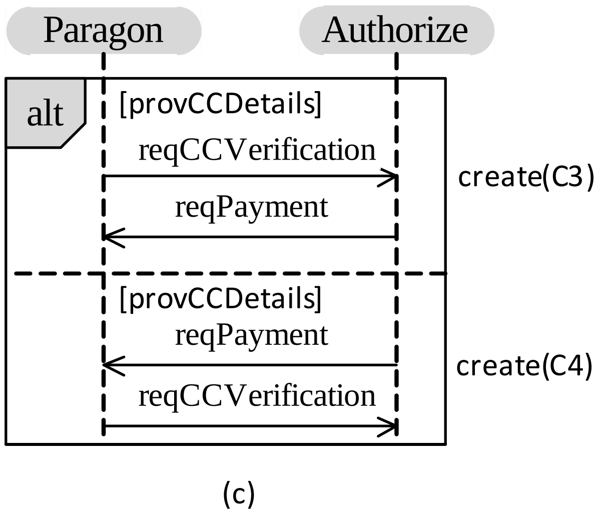
Figure 14: MSCs representing return of damaged goods and refund.

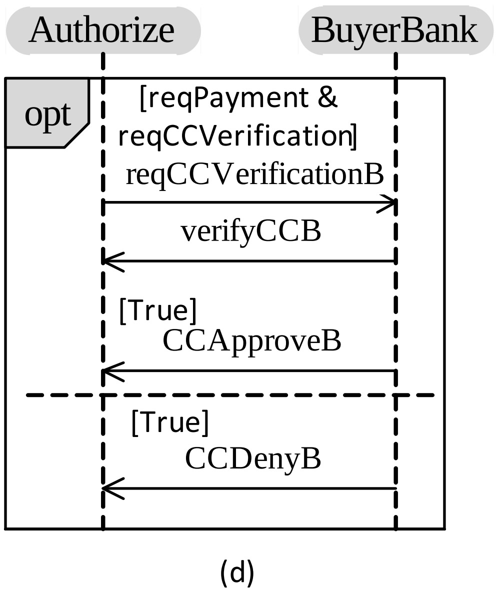




orderGoods $\rightarrow$ $\mathsf{create}(\mathsf{C}_1)$
offerPrice $\rightarrow$ $\mathsf{create}(\mathsf{C}_2)$

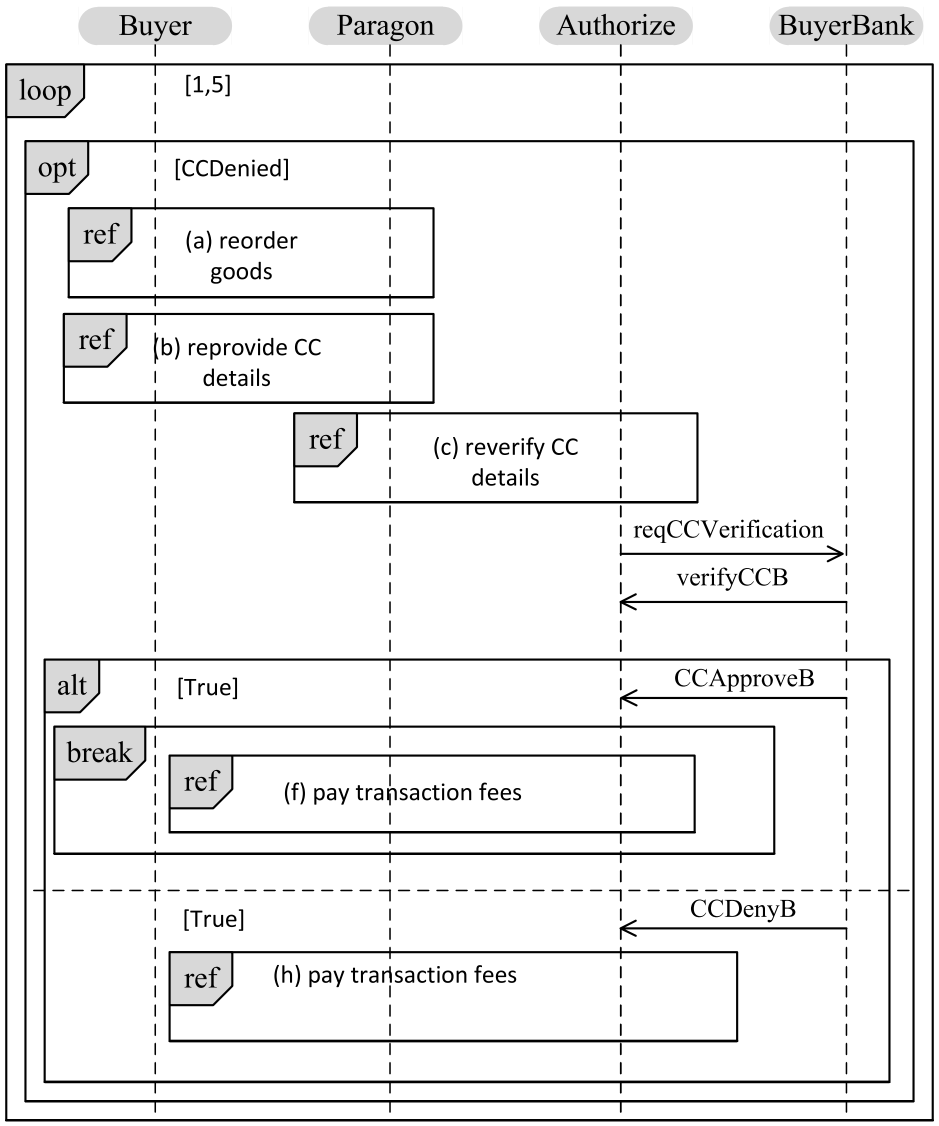
Figure 15: MSCs representing (a) order placement, and (b) request of credit-card (CC) details.





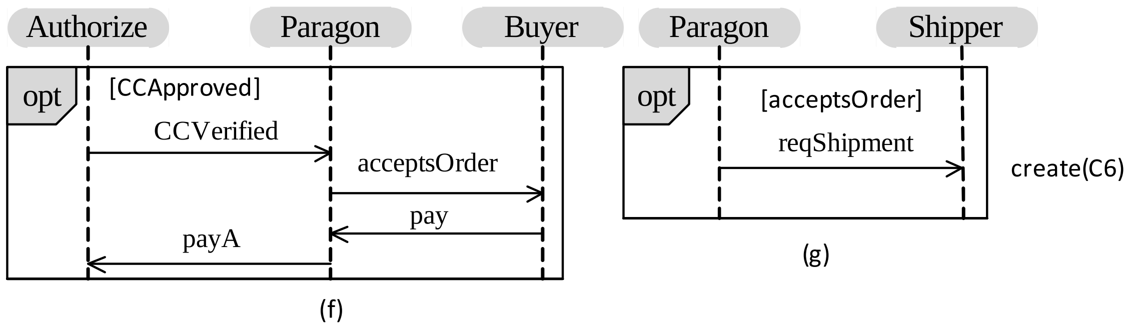
reqCCVerification $\rightarrow$ $\mathsf{create}(\mathsf{C}_3)$
reqPayment $\rightarrow$ $\mathsf{create}(\mathsf{C}_4)$
reqCCVerificationB $\rightarrow$ $\mathsf{create}(\mathsf{C}_{13})$

Figure 16: MSCs representing verification of credit-card (CC) details.



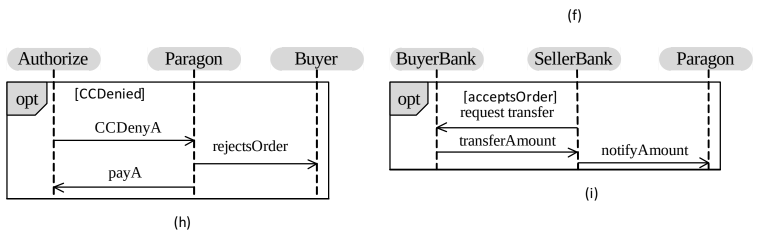
(e)

Figure 17: MSCs representing reverifying credit-card details.



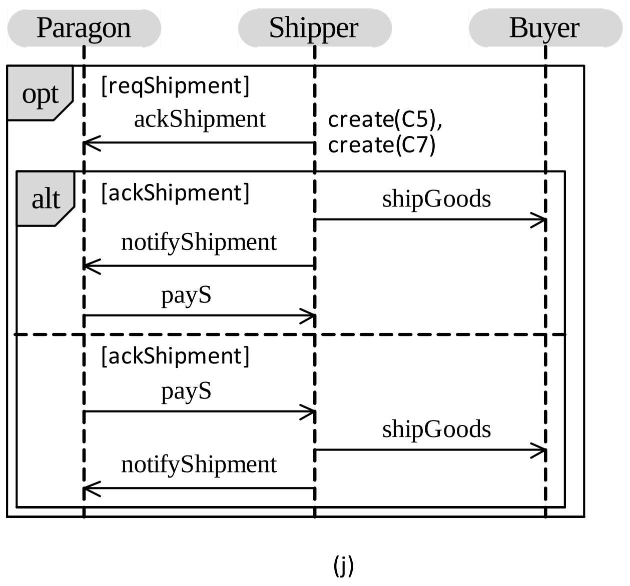
request shipment $\rightarrow$ $\mathsf{create}(\mathsf{C}_6)$

Figure 18: MSCs representing ordering of goods.



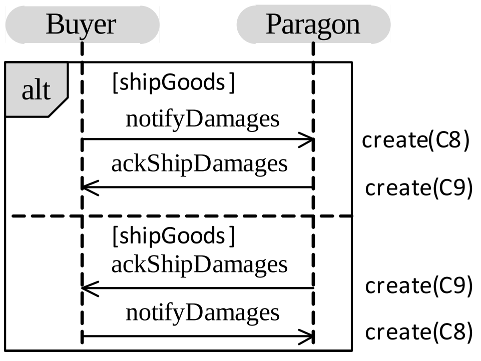
request transfer $\rightarrow$ $\mathsf{create}(\mathsf{C}_{14})$

Figure 19: MSCs representing the denial of credit-card details and payment for goods.

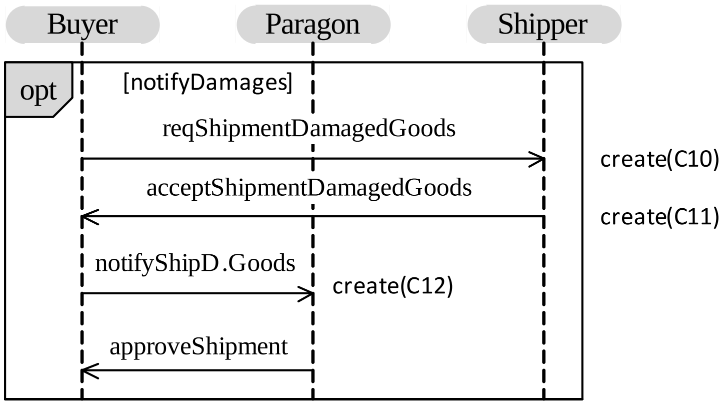


acknowledge shipment $\rightarrow$ $\mathsf{create}(\mathsf{C}_5) \wedge \mathsf{create}(\mathsf{C}_7)$

Figure 20: MSCs representing the shipment of goods.



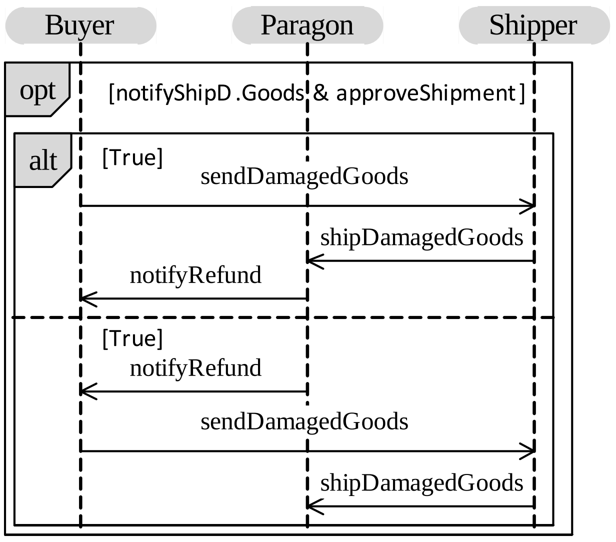
(k)



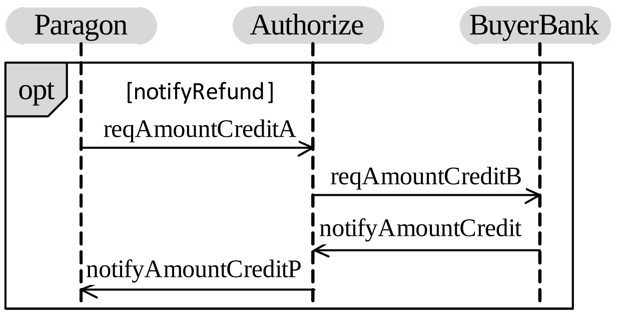
(l)

notify damages $\rightarrow$ create(C_8)
acknowledge ship damages $\rightarrow$ create(C_9)
request shipment of damage goods $\rightarrow$ create(C_{10})
notify shipment of damage goods $\rightarrow$ create(C_{12})

Figure 21: MSCs representing the acknowledgment of damaged goods.

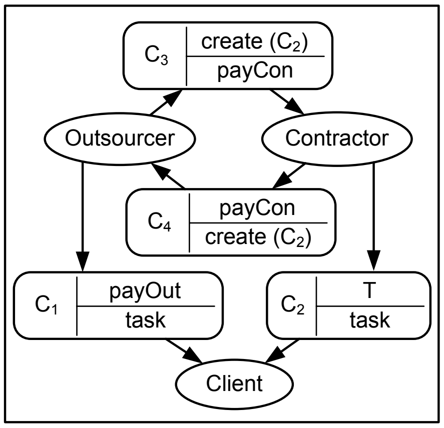


(m)



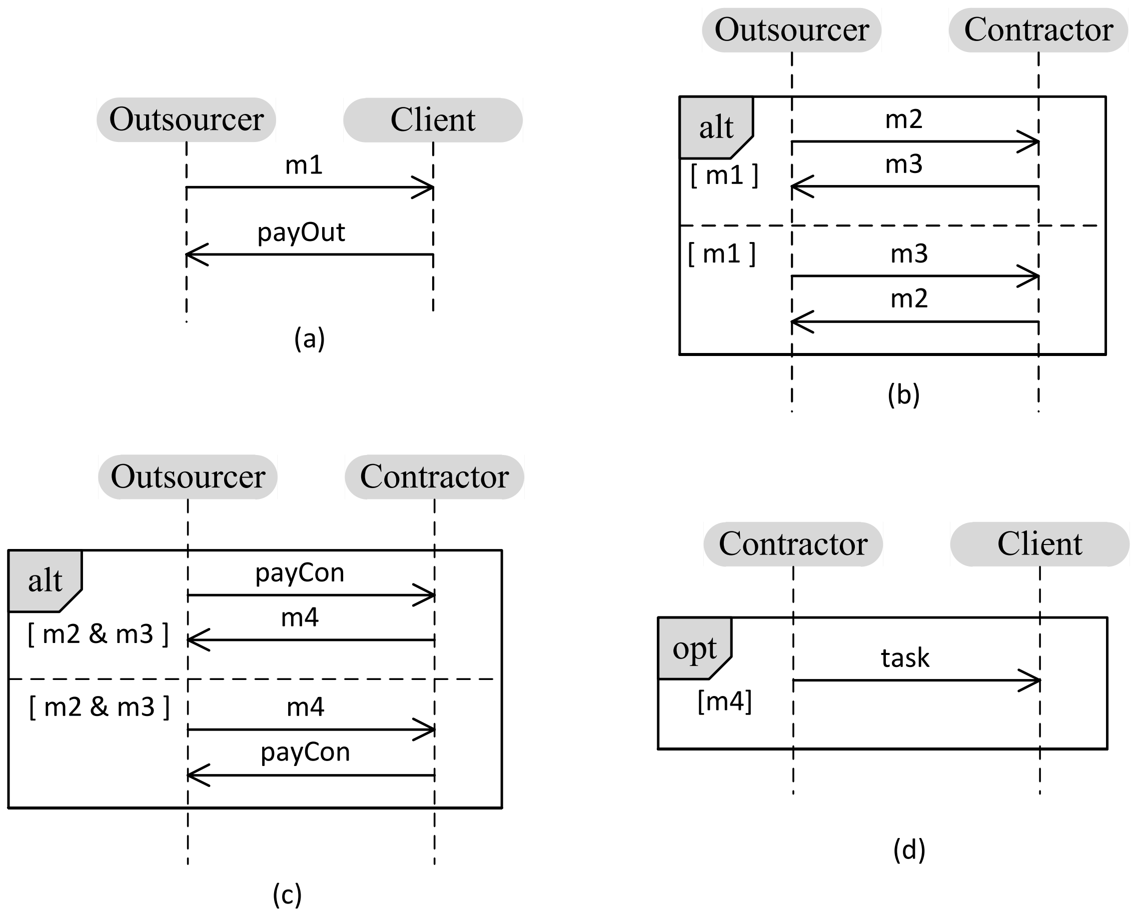
(n)

Figure 22: MSCs representing the shipment of damaged goods and refund.



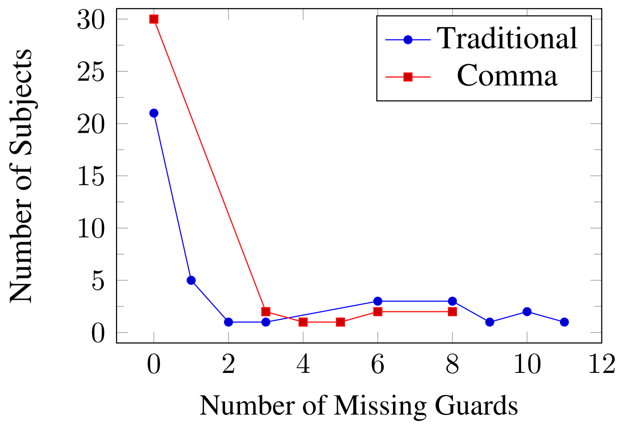
C_1	C(OUTSOURCER, CLIENT, payOut, task)
C_2	C(CONTRACTOR, CLIENT, $\top$, task)
C_3	C(OUTSOURCER, CONTRACTOR, create(C2), payCon)
C_4	C(CONTRACTOR, OUTSOURCER, payCon, create(C2))

Figure 23: The outsourcing business pattern.



m1 $\rightarrow$ create(C_1)
m2 $\rightarrow$ create(C_3)
m3 $\rightarrow$ create(C_4)
m4 $\rightarrow$ create(C_2)

Figure 24: MSCs for outsourcing.



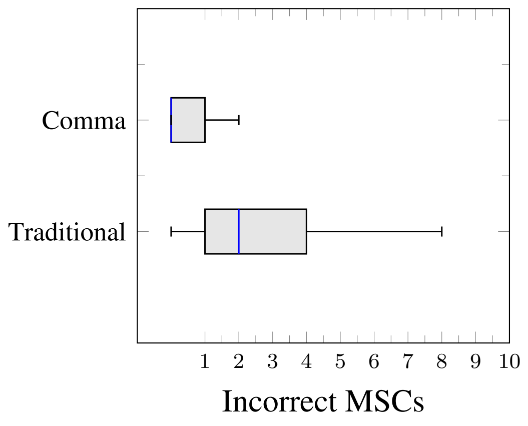


Figure 25: Objective measures of quality.

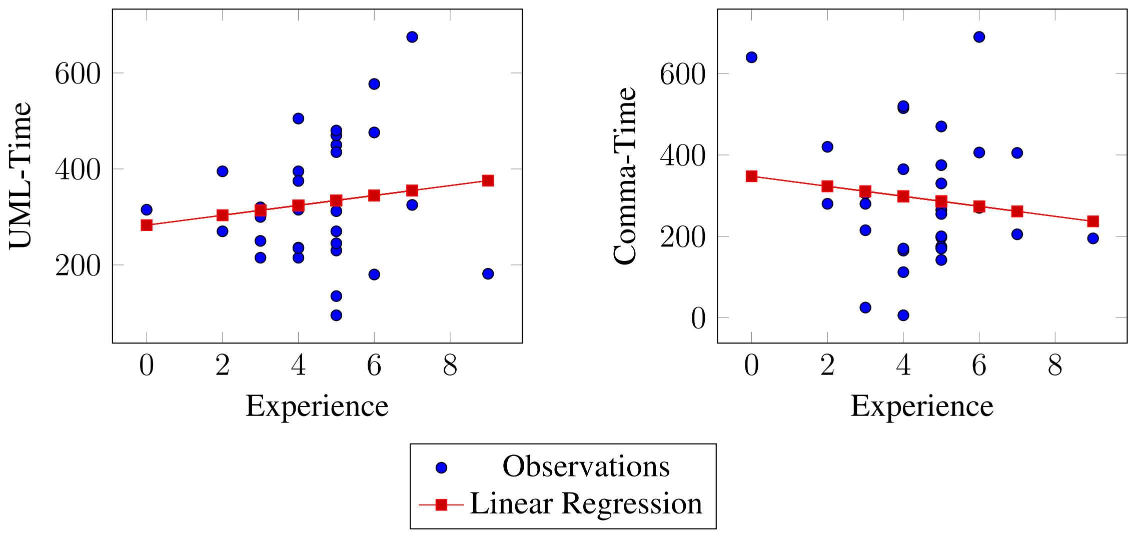


Figure 26: Correlation between experience and time.

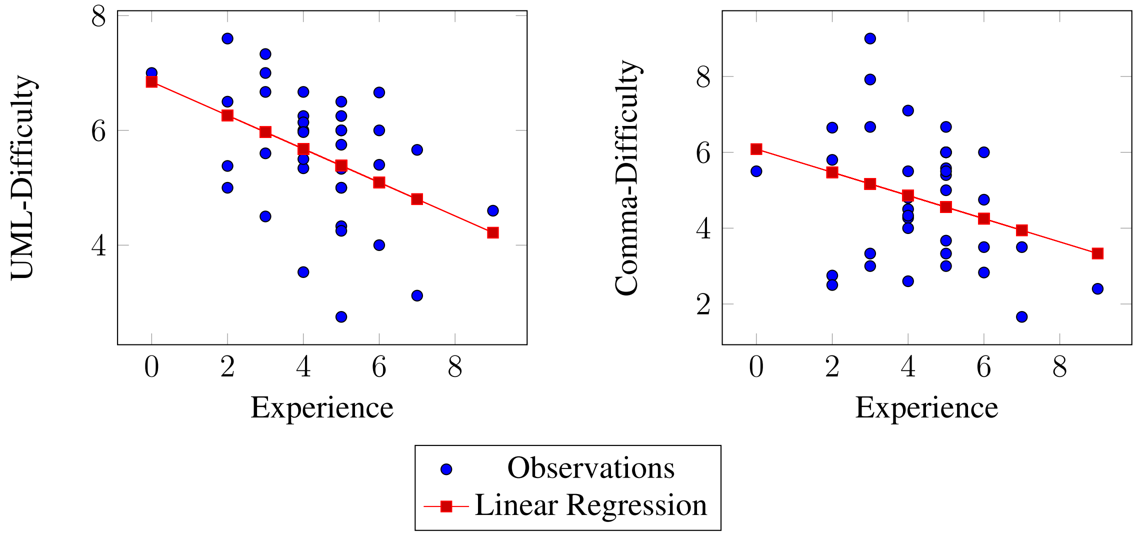


Figure 27: Correlation between experience and difficulty.

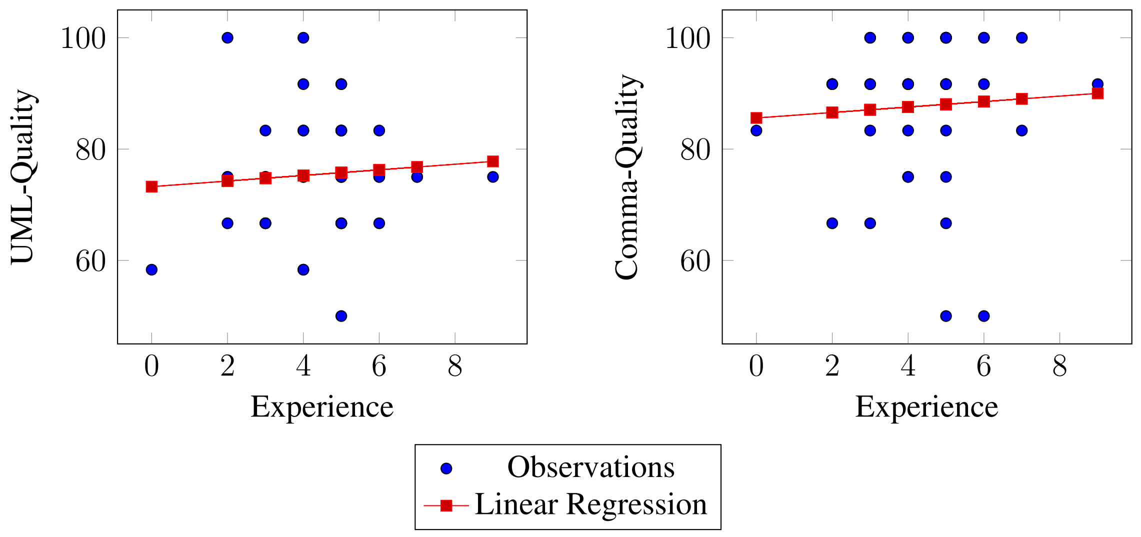


Figure 28: Correlation between experience and quality.

Tables

Table 3: Hypothesis testing.

ID	**Comma Mean (μ_c)**	**Traditional Mean (μ_u)**	**Null Hypothesis $[\mu_c = \mu_u]$ p-value**	**Accepted at p-value of 5%?**
$\text{H}^{\mu_c<\mu_u}_{Time}$	226.89	162.94	0.0012	$\times$
$\text{H}^{\mu_c<\mu_u}_{Difficulty}$	2.94	2.20	0.0004	$\times$
$\text{H}^{\mu_c>\mu_u}_{Alt-Opt-Par}$	12.57	6.03	0.000	$\times$
$\text{H}^{\mu_c>\mu_u}_{MSC-Count}$	10.05	7.19	0.000	$\times$