Evaluating the Benefits of Orpheus/BSPL for Iterative and Incremental Development in MAS

International Workshop on Asynchronous Programming Models - special edition @ ETAPS 2026

Matteo Baldoni[1]

Torino, Italy, April 16th, 2026

Thanks to Cristina Baroglio, Amit K. Chopra, Samuel H. Christie V, Elisa Marengo, Roberto Micalizio, Munindar P. Singh, Stefano Tedeschi

[1]Università di Torino, Torino, Italy

Multiagent system = agents + interaction protocol

Power of AI agents: their flexibility

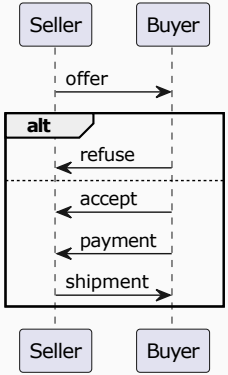
An *interaction protocol* models the communication constraints between agents in a multiagent system

Engineering multiagent system based on protocols offers key benefits

- *Decentralized MAS*; without relying on a distinguished locus of state or control
- *Clear implementation*, separation between the coordination aspects and business logic of an agent
- *Loose coupling*, changes in one agent's implementation do not affect the implementation of others
- *Reducing agent complexity*, avoiding programming errors

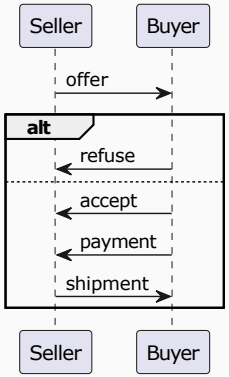
An example: The EBusiness Protocol

The Seller may submit an offer; Buyer may either accept or refuse. After accepting, the Buyer may proceed with payment, and the Seller ship the item.




```
1 EBusiness_1 {
2   Seller -> Buyer : offer [ID, item, price]
3   Buyer -> Seller : accept [ID, item, price,
     decision]
4   Buyer -> Seller : refuse [ID, item, price,
     decision, status]
5   Buyer -> Seller : payment [ID, price, decision
     , amountC]
6   Seller -> Buyer : shipment [ID, item, price,
     decision, status]
7 }
```

An example: The EBusiness Protocol

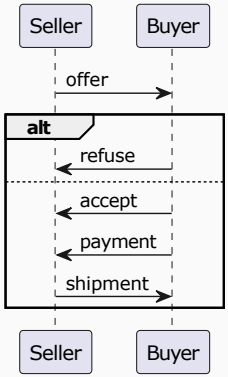


Seller:

```
1  +!offer
2    :  step(start) & on_offer(Id, Item, Price)
3    <- ...
4       .send(Buyer, tell, offer(Id, Item, Price));
5       +step(Id, offer).
6
7  +refuse(Id, Item, Price)[source(Buyer)]
8    :  step(Id, offer)
9    <- ...
10      -step(Id, offer);
11      +step(Id, refuse).
12
13 +accept(Id, Item, Price, Decision)[source(Buyer)]
14   :  step(Id, offer)
15   <- ...
16      -step(Id, offer);
17      +step(Id, accept).
18 ...
```

Source code of examples: *https://gitlab.di.unito.it/baldoni/argonauts*

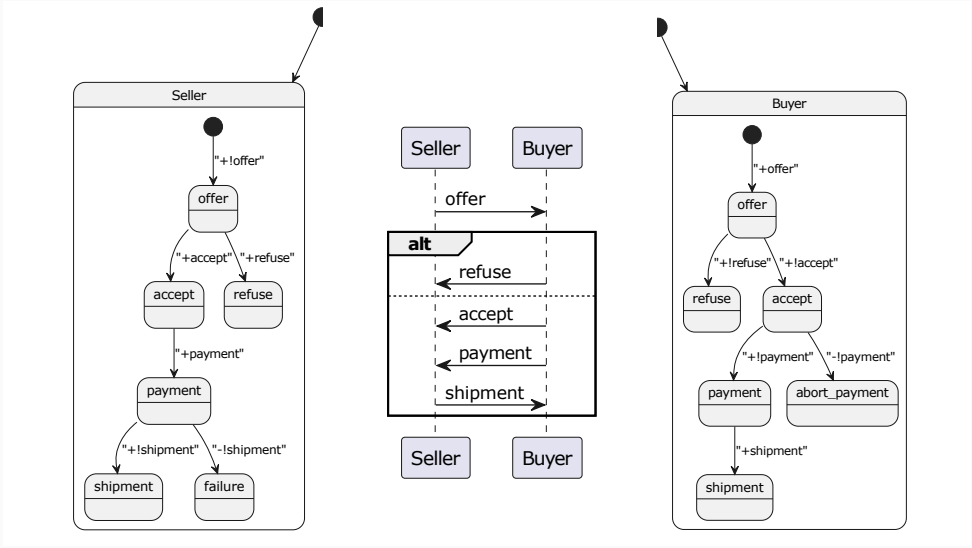
An example: The EBusiness Protocol



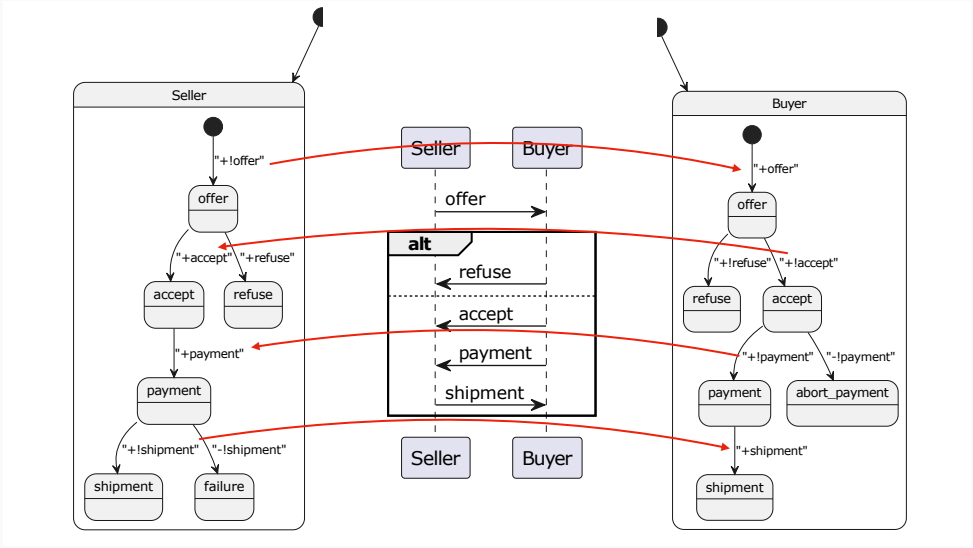
Seller:

```
18 ...
19 +payment(Id, Price, Decision, AmountC)[source(Buyer)]
20   : step(Id, accept)
21   <- ...
22      -step(Id, accept); +step(Id, payment);
23      !shipment(Id, Price, Decision)[receiver(Buyer)].
24
25 +!shipment(Id, Price, Decision)[receiver(Buyer)]
26   : step(Id, payment) &
27      accept(Id, Item, Price, Decision)[source(Buyer)] &
28      in_stock(Item)
29   <- ...
30      .send(Buyer, tell, shipment(Id, Item, Price, Decision, Status));
31      -step(Id, payment); +step(Id, shipment).
32
33 +!shipment(Id, Price, Decision)[receiver(Buyer)]
34   : step(Id, payment) &
35      accept(Id, Item, Price, Decision)[source(Buyer)] &
36      not in_stock(Item)
37   <- ...
38      -step(Id, payment); +step(Id, failure).
```

An example: The EBusiness Protocol

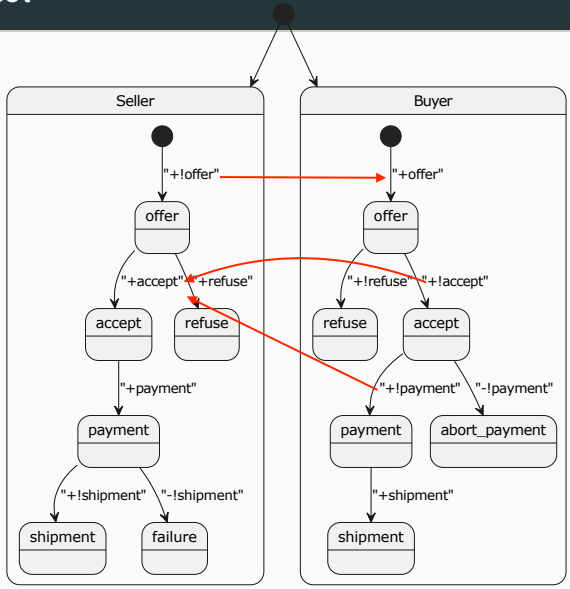


An example: The EBusiness Protocol



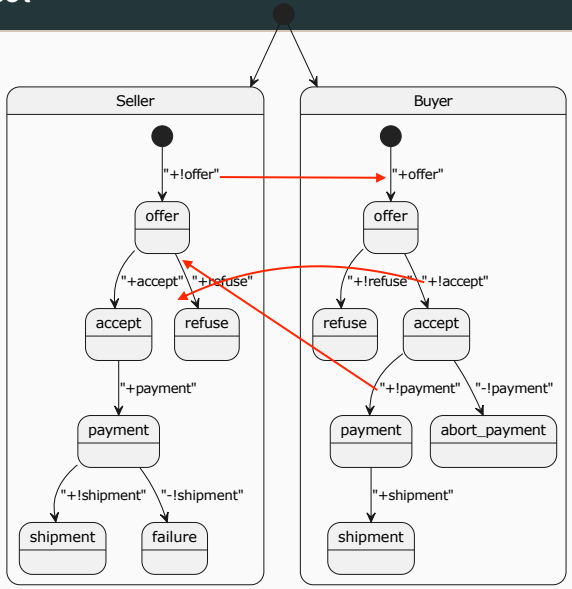
An example: The EBusiness Protocol

- What happen if the Buyer sends messages faster than Seller can process them?



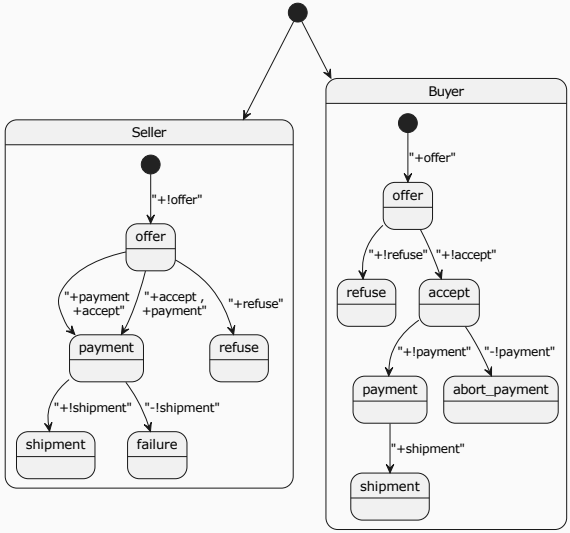
An example: The EBusiness Protocol

- What happen if the Buyer sends messages faster than Seller can process them?
- The Seller's automata does not correctly recognize the current situation, and the Seller gets stuck!
- In other words: the two agents are not truly autonomous...



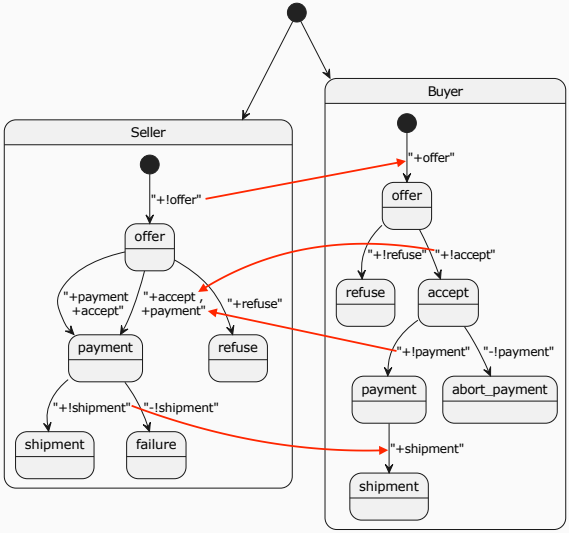
An example: The EBusiness Protocol

```
12 ...
13 +accept(Id, Item, Price, Decision)[source(
   Buyer)]
14   : step(Id, offer) &
15     payment(Id, Price, Decision, AmountC,
         Choice)[source(Buyer)]
16   <- ...
17     -step(Id, offer); +step(Id, payment);
18     !shipment(Id, Price, Decision)[
         receiver(Buyer)].
19
20 +payment(Id, Price, Decision, AmountC)[
   source(Buyer)]
21   : step(Id, offer) &
22     accept(Id, Item, Price, Decision)[
         source(Buyer)]
23   <- ...
24     -step(Id, offer); +step(Id, payment);
25     !shipment(Id, Price, Decision)[
         receiver(Buyer)].
26 ...
```



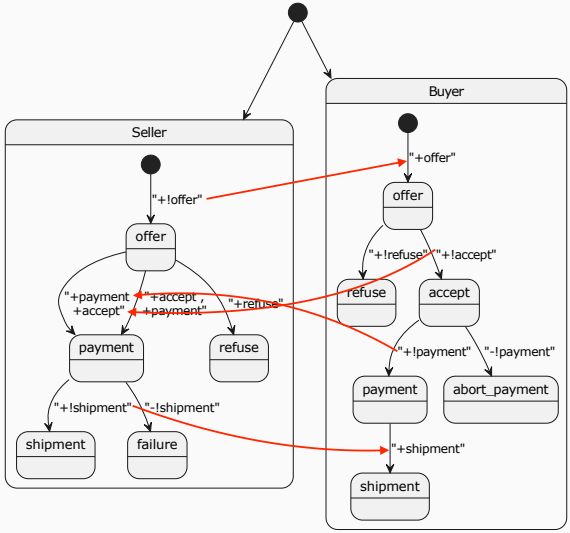
An example: The EBusiness Protocol

```
12 ...
13 +accept(Id, Item, Price, Decision)[source(
   Buyer)]
14   : step(Id, offer) &
15     payment(Id, Price, Decision, AmountC,
         Choice)[source(Buyer)]
16   <- ...
17     -step(Id, offer); +step(Id, payment);
18     !shipment(Id, Price, Decision)[
         receiver(Buyer)].
19
20 +payment(Id, Price, Decision, AmountC)[
   source(Buyer)]
21   : step(Id, offer) &
22     accept(Id, Item, Price, Decision)[
         source(Buyer)]
23   <- ...
24     -step(Id, offer); +step(Id, payment);
25     !shipment(Id, Price, Decision)[
         receiver(Buyer)].
26 ...
```



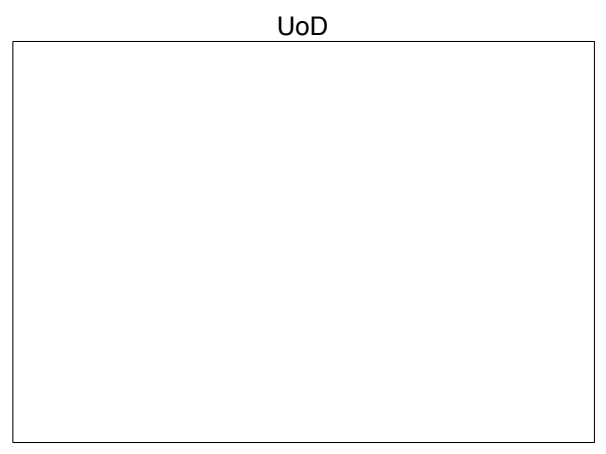
An example: The EBusiness Protocol

```
12 ...
13 +accept(Id, Item, Price, Decision)[source(
   Buyer)]
14   : step(Id, offer) &
15     payment(Id, Price, Decision, AmountC,
         Choice)[source(Buyer)]
16   <- ...
17     -step(Id, offer); +step(Id, payment);
18     !shipment(Id, Price, Decision)[
         receiver(Buyer)].
19
20 +payment(Id, Price, Decision, AmountC)[
   source(Buyer)]
21   : step(Id, offer) &
22     accept(Id, Item, Price, Decision)[
         source(Buyer)]
23   <- ...
24     -step(Id, offer); +step(Id, payment);
25     !shipment(Id, Price, Decision)[
         receiver(Buyer)].
26 ...
```



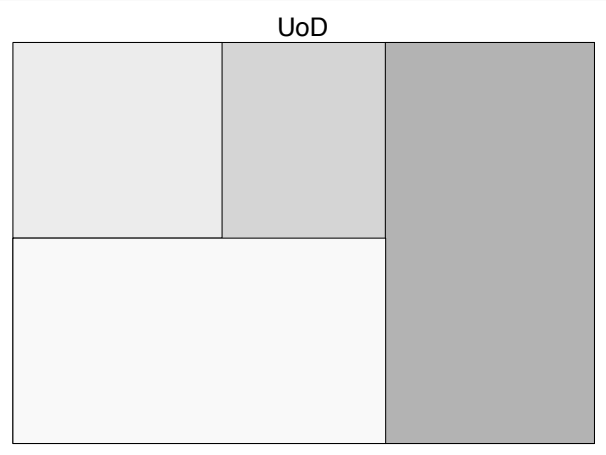
Agent Development Processes

Like in software development processes... divide et impera. Incremental approach.



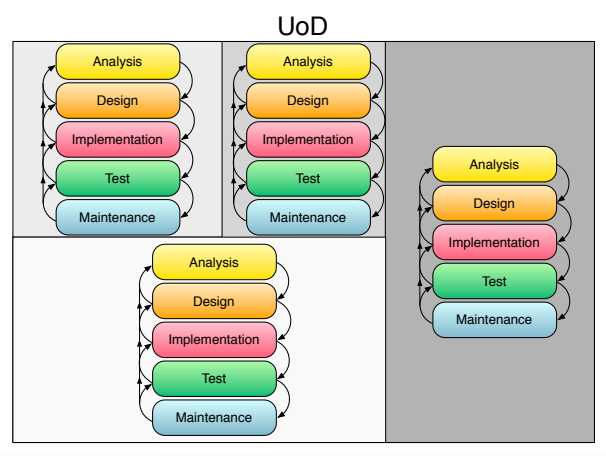
Agent Development Processes

Like in software development processes... divide et impera. Incremental approach.



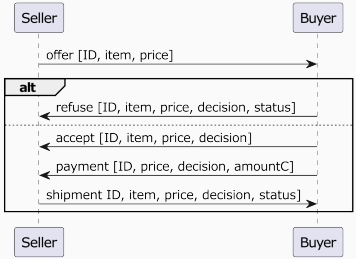
Agent Development Processes

Like in software development processes... divide et impera. Incremental approach.



EBusiness, iteration 1 – “Basic Transaction”

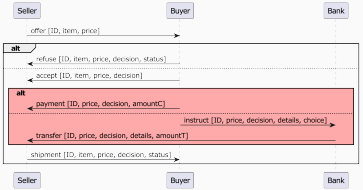
The Seller may submit an offer; Buyer may either accept or refuse. After accepting, the Buyer may proceed with payment, and the Seller ship the item.




```
1 EBusiness_1 {
2   Seller -> Buyer : offer [ID, item, price]
3   Buyer -> Seller : accept [ID, item, price, decision]
4   Buyer -> Seller : refuse [ID, item, price, decision, status]
5   Buyer -> Seller : payment [ID, price, decision, amountC]
6   Seller -> Buyer : shipment [ID, item, price, decision, status]
7 }
```

EBusiness, iteration 2 – “Introduction of an intermediary”

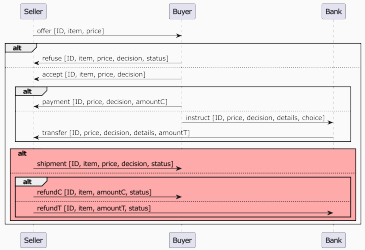
The Buyer may choose to pay the Seller directly or instruct the Bank to make the transfer.




```
1  EBusiness_2 {
2    Seller -> Buyer : offer [ID, item, price]
3    Buyer -> Seller : accept [ID, item, price, decision]
4    Buyer -> Seller : refuse [ID, item, price, decision, status]
5    Buyer -> Seller : payment [ID, price, decision, amountC]
6
7    Buyer -> Bank : instruct [ID, price, decision, details, choice
       ]
8    Bank -> Seller : transfer [ID, price, decision, details,
       amountT]
9
10   Seller -> Buyer : shipment [ID, item, price, decision, status]
11 }
```

EBusiness, iteration 3 – “Refund mechanism”

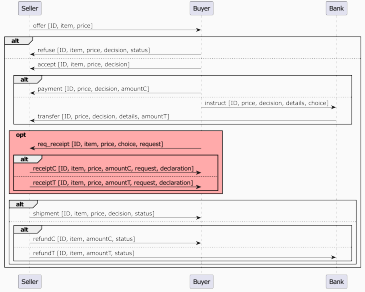
If for some reason the Seller is unable to ship the item, the Seller may ussue a refund to the Buyer (or to the Bank).




```
1  EBusiness_3 {
2    Seller -> Buyer : offer [ID, item, price]
3    Buyer -> Seller : accept [ID, item, price, decision]
4    Buyer -> Seller : refuse [ID, item, price, decision, status]
5    Buyer -> Seller : payment [ID, price, decision, amountC]
6
7    Buyer -> Bank : instruct [ID, price, decision, details, choice
       ]
8    Bank -> Seller : transfer [ID, price, decision, details,
       amountT]
9
10   Seller -> Buyer : shipment [ID, item, price, decision, status]
11
12   Seller -> Buyer : refundC [ID, item, amountC, status]
13   Seller -> Bank : refundT [ID, item, amountT, status]
14 }
```

EBusiness, iteration 4 – “Proof of receipt”

After making the payment, the Buyer may request a declaration of receipt from the Seller.

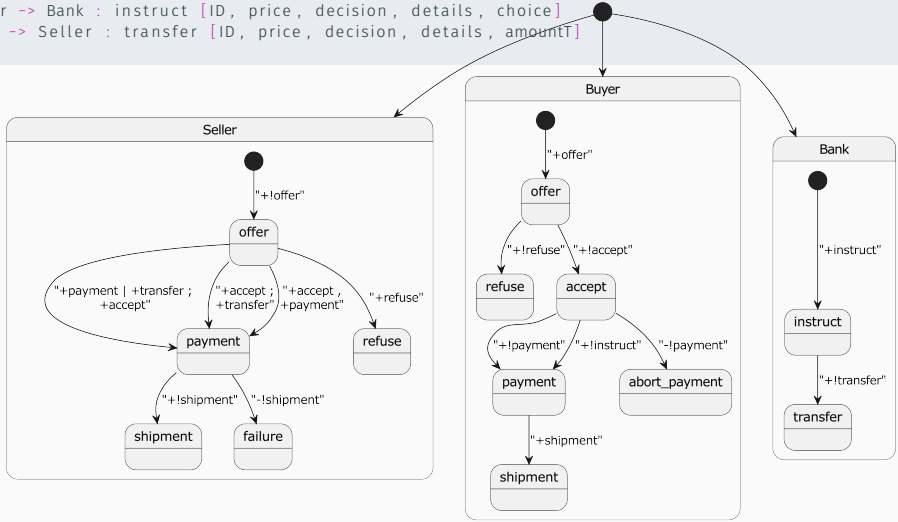



```
1  EBusiness_4 {
2    Seller -> Buyer : offer [ID, item, price]
3    Buyer -> Seller : accept [ID, item, price, decision]
4    Buyer -> Seller : refuse [ID, item, price, decision, status]
5    Buyer -> Seller : payment [ID, price, decision, amountC]
6
7    Buyer -> Bank : instruct [ID, price, decision, details, choice
       ]
8    Bank -> Seller : transfer [ID, price, decision, details,
       amountT]
9
10   Buyer -> Seller : req_receipt [ID, item, price, choice,
       request]
11   Seller -> Buyer : receiptC [ID, item, price, amountC, request,
       declaration]
12   Seller -> Buyer : receiptT [ID, item, price, amountT, request,
       declaration]
13
14   Seller -> Buyer : shipment [ID, item, price, decision, status]
15
16   Seller -> Buyer : refundC [ID, item, amountC, status]
17   Seller -> Bank : refundT [ID, item, amountT, status]
18 }
```

Iterative and Incremental Development Process

Iteration 1 ⟶ Iteration 2

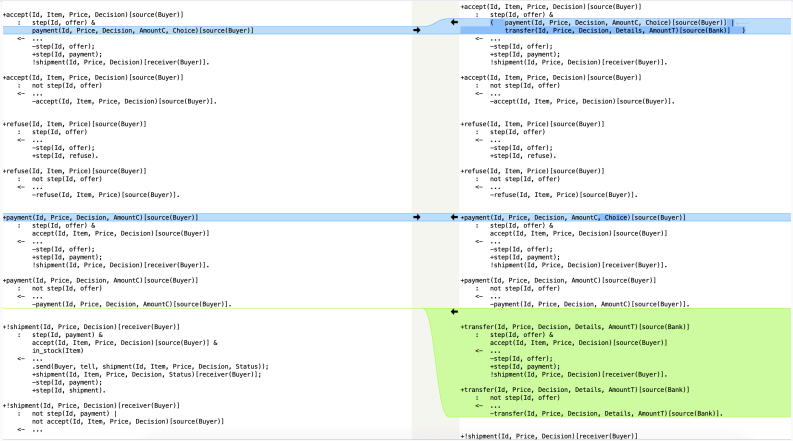
```
...
Buyer -> Bank : instruct [ID, price, decision, details, choice]
Bank -> Seller : transfer [ID, price, decision, details, amount]
...
```



Iterative and Incremental Development Process

Iteration 1 ⟶ Iteration 2

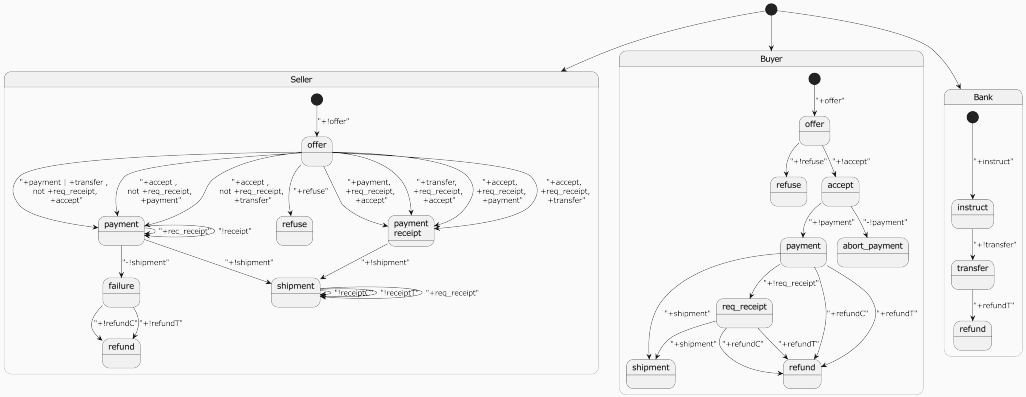
```
1  ...
2  Buyer -> Bank : instruct [ID, price, decision, details, choice]
3  Bank -> Seller : transfer [ID, price, decision, details, amountT]
4  ...
```



Iterative and Incremental Development Process

Iteration 3 ⟶ Iteration 4

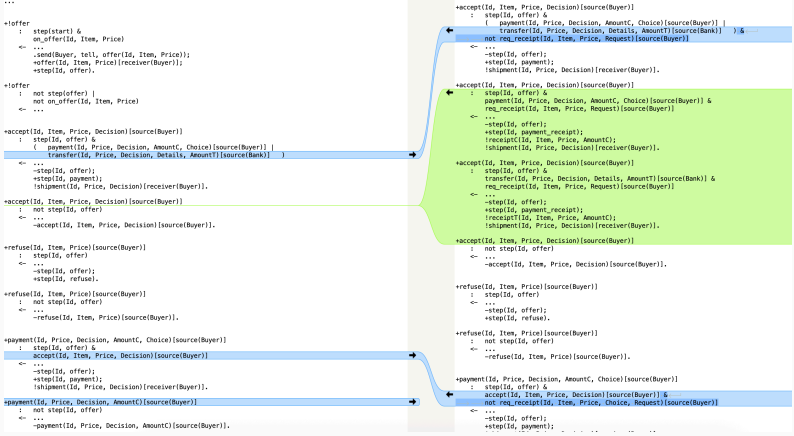
```
...
Buyer -> Seller : req_receipt [ID, item, price, choice, request]
Seller -> Buyer : receiptC [ID, item, price, amountC, request, declaration]
Seller -> Buyer : receiptT [ID, item, price, amountT, request, declaration]
...
```



Iterative and Incremental Development Process

Iteration 3 ⟶ Iteration 4

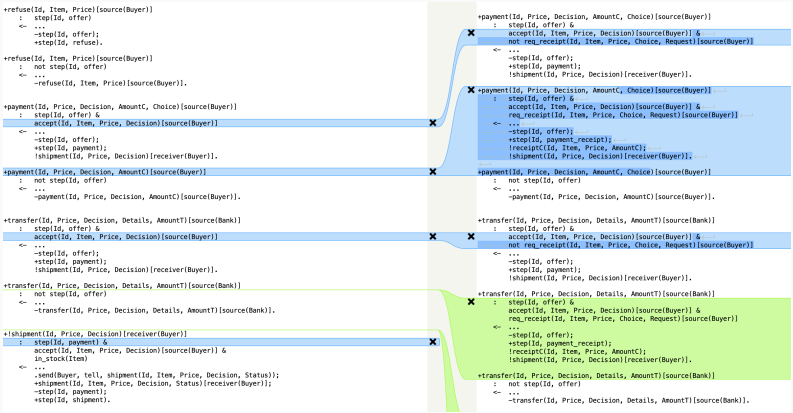
```
1  ...
2  Buyer -> Seller : req_receipt [ID, item, price, choice, request]
3  Seller -> Buyer : receiptC [ID, item, price, amountC, request, declaration]
4  Seller -> Buyer : receiptT [ID, item, price, amountT, request, declaration]
5  ...
```



Iterative and Incremental Development Process

Iteration 3 ⟶ Iteration 4

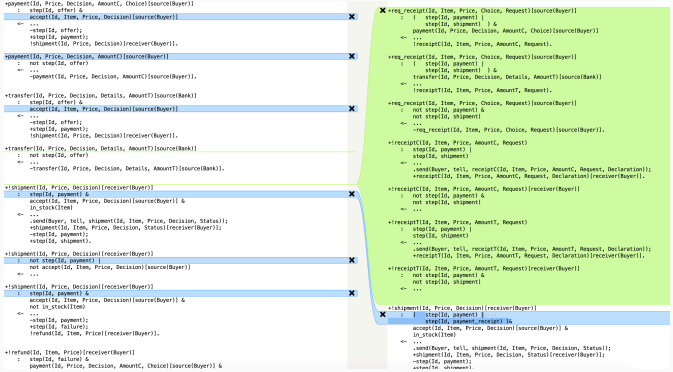
```
1  ...
2  Buyer -> Seller : req_receipt [ID, item, price, choice, request]
3  Seller -> Buyer : receiptC [ID, item, price, amountC, request, declaration]
4  Seller -> Buyer : receiptT [ID, item, price, amountT, request, declaration]
5  ...
```



Iterative and Incremental Development Process

Iteration 3 ⟶ Iteration 4

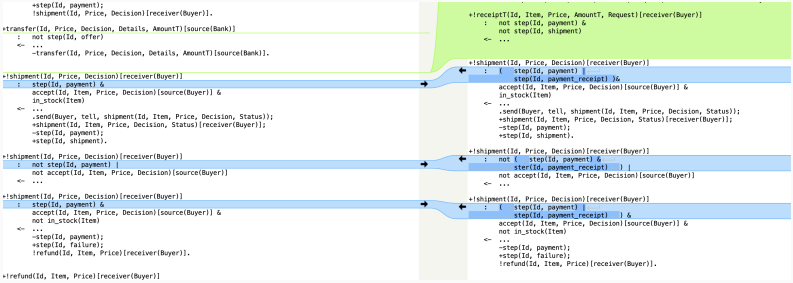
```
1  ...
2  Buyer -> Seller : req_receipt [ID, item, price, choice, request]
3  Seller -> Buyer : receiptC [ID, item, price, amountC, request, declaration]
4  Seller -> Buyer : receiptT [ID, item, price, amountT, request, declaration]
5  ...
```



Iterative and Incremental Development Process

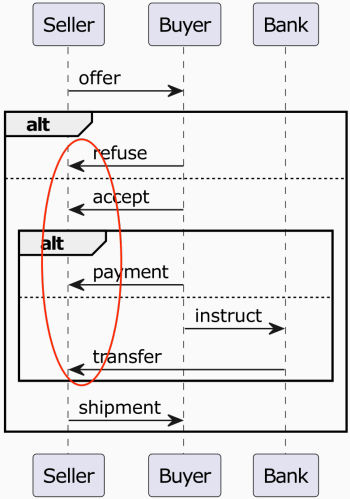
Iteration 3 ⟶ Iteration 4

```
1  ...
2  Buyer -> Seller : req_receipt [ID, item, price, choice, request]
3  Seller -> Buyer : receiptC [ID, item, price, amountC, request, declaration]
4  Seller -> Buyer : receiptT [ID, item, price, amountT, request, declaration]
5  ...
```



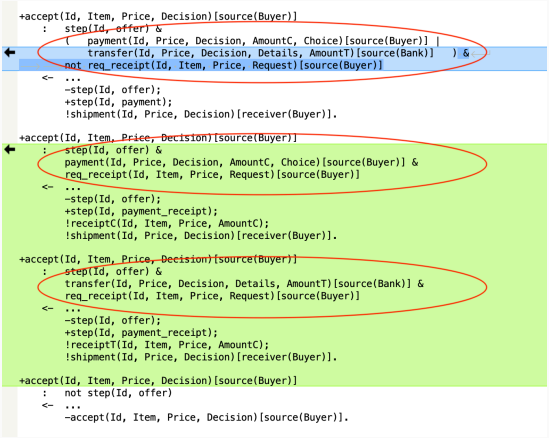
Shortcomings: Dealing with asynchronicity

- A new iteration imposes modifications for the sender of a message, but it imposes even more subtle modifications for the receiver due to the asynchronicity and multiplicity of multi-agent interactions.



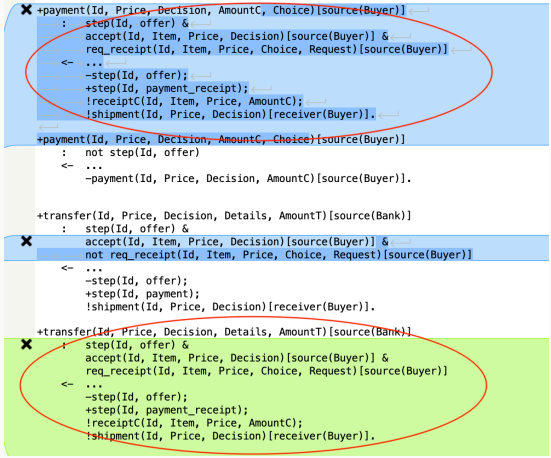
Shortcomings: Dealing with asynchronicity

- The receiver must manage not only the message itself, but also the way in which the message is received, in relation to the other incoming messages, regardless of the order in which the sender(s) has(have) sent them
- A new iteration introduces significant changes into the previously tested code, and the impact is potentially global



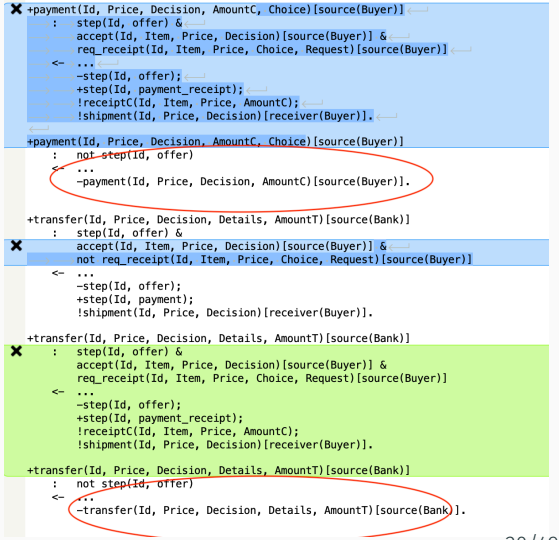
Shortcomings: Dealing with asynchronicity

- The receiver must manage not only the message itself, but also the way in which the message is received, in relation to the other incoming messages, regardless of the order in which the sender(s) has(have) sent them
- A new iteration introduces significant changes into the previously tested code, and the impact is potentially global



Shortcomings: Dealing with asynchronicity

- Even worse, the developer must also handle messages and goals that arrive in the wrong order or as duplicates
- Duplicated messages, or messages referring to the same interaction session, should not contribute to the agent's knowledge base and should be discarded
- Goals involving message sending should be fulfilled only in the appropriate context



Shortcomings: Dealing with asynchronicity

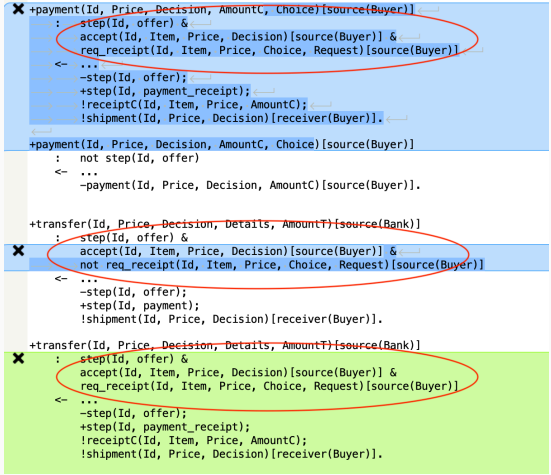
- Even worse, the developer must also handle messages and goals that arrive in the wrong order or as duplicates
- Duplicated messages, or messages referring to the same interaction session, should not contribute to the agent's knowledge base and should be discarded
- Goals involving message sending should be fulfilled only in the appropriate context

```
+!receiptC(Id, Item, Price, AmountC, Request)[receiver(Buyer)]
    :   not step(Id, payment) &
        not step(Id, shipment)
    <-  ...

+!receiptT(Id, Item, Price, AmountT, Request)
    :   step(Id, payment) |
        step(Id, shipment)
    <-  ...
        .send(Buyer, tell, receiptT(Id, Item, Price, AmountT, Request, Declaration));
        +receiptT(Id, Item, Price, AmountT, Request, Declaration)[receiver(Buyer)].

+!receiptT(Id, Item, Price, AmountT, Request)[receiver(Buyer)]
    :   not step(Id, payment) &
        not step(Id, shipment)
    <-  ...
```

```
+!shipment(Id, Price, Decision)[receiver(Buyer)]
    :   ( step(Id, payment) |
          step(Id, payment_receipt) )&
        accept(Id, Item, Price, Decision)[source(Buyer)] &
        in_stock(Item)
    <-  ...
        .send(Buyer, tell, shipment(Id, Item, Price, Decision, Status));
        +shipment(Id, Item, Price, Decision, Status)[receiver(Buyer)];
        -step(Id, payment);
        +step(Id, shipment).

+!shipment(Id, Price, Decision)[receiver(Buyer)]
    :   not ( step(Id, payment) &
              ster(Id, payment_receipt) ) |
        not accept(Id, Item, Price, Decision)[source(Buyer)]
    <-  ...
```

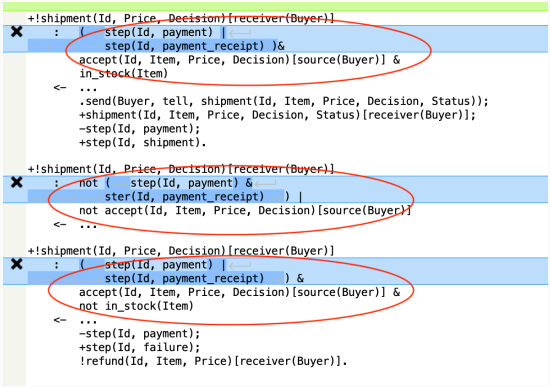
Shortcomings: Dealing with asynchronicity

- As the asynchronicity increase, the developer must rely more heavily on previously received messages to determine the correct context, which in turn makes the code even more difficult to maintain in subsequent iterations



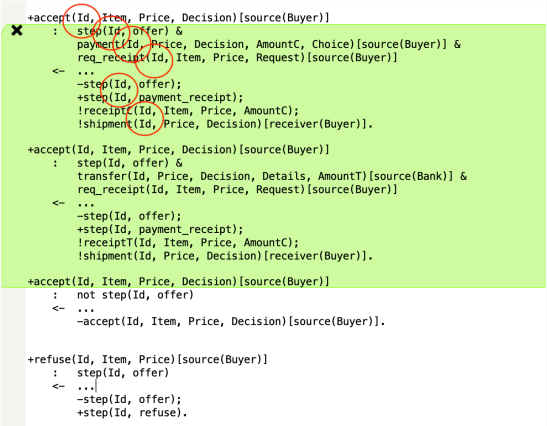
Shortcomings: Coupling between history and control flow

- In order to send a message, the agent must extract information from messages that have already arrived, so this process depends on the actual messages received, which again makes the code even more difficult to maintain in subsequent iteration



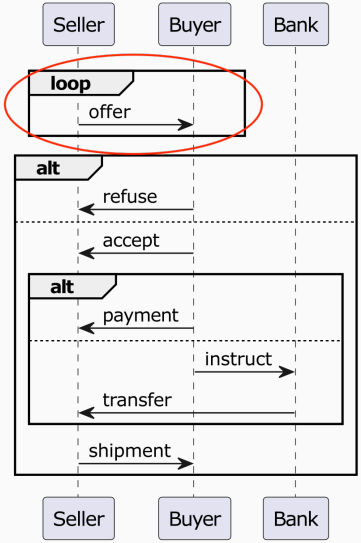
Shortcomings: Dealing with concurrent instances

- To deal with multiple instances of the protocol, the common approach is to use a session identifier associated with all exchanged messages



Shortcomings: Dealing with concurrent instances

- What happens if multiple messages from the same sender are allowed?
- What happens if the identical messages from different senders are allowed?
- Increasing sophisticated custom identifiers are introduced, which again makes the code even more difficult to maintain in subsequent iterations



Shortcomings of current approaches to agent communication

As a result:

- Asynchronous message exchanges and multi-party solutions are more difficult to design and to program and they are more prone to errors
- Incompatibilities between agents due to the message schemas being blended into business logic
- Semantic errors due to a lack of a formal model
- Inflexibility due to the programmer having to maintain the protocol state via a state machine

Shortcomings of current approaches to agent communication

In 2012, Michael Winikoff [Winikoff, 2012] highlighted two shortcomings about AOPLs

AOPLs supported little more than *primitives* for sending and receiving messages

- GOTOs and labels: Winikoff saw the use of such primitives as transferring control between agents and drew an unflattering analogy with the use of *gotos* in programming

Interaction protocols (typically in AUML) were *message-centric* and *over-constrained* the interaction between agents

- Less flexibility and robustness Interaction protocols (AUML) do not leave the agents room to be autonomous or to exploit their flexibility and robustness when interacting with other agents

BSPL and Commitments

- Adoption of **BSPL** [Singh, 2011] (Blindingly Simple Protocol Language): A fully declarative and fully asynchronous model for communication, with information at the center, supporting causal relationships, integrity, and compositionality
- Adoption of **Commitments** [Singh, 1999, Singh, 2013]: social semantics for the messages, clear responsibility distributions of tasks

An example: The EBusiness Protocol

```
EBusiness {
  role Buyer, Seller
  parameter out ID key, out item, out price, out status

  Seller -> Buyer : offer [out ID key, out item, out price]

  Buyer -> Seller : accept [in ID key, in item, in price, out decision]
  Buyer -> Seller : refuse [in ID key, in item, in price, out decision, out status]

  Buyer -> Seller : payment [in ID key, in price, in decision, out amountC]

  Seller -> Buyer : shipment [in ID key, in item, in price, in decision, out status]
}
```

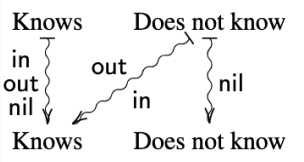
BSPL information protocols [Singh, 2011]

- Roles
- Message schemas
  - A name
  - A sender role
  - A receiver role
  - One or more parameters
- A message may be received at *any time* in any relative order with respect to other messages
- The emission of a message *depends upon* what information the agent has

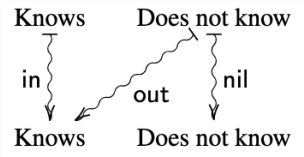
BSPL information protocols [Singh, 2011]

- A *message instance* is a tuple of bindings for the parameters of that schema that are adorned either ⌜in⌝ or ⌜out⌝
- The ⌜key⌝ parameters of a schema form a composite key and uniquify its instances

Receiver:



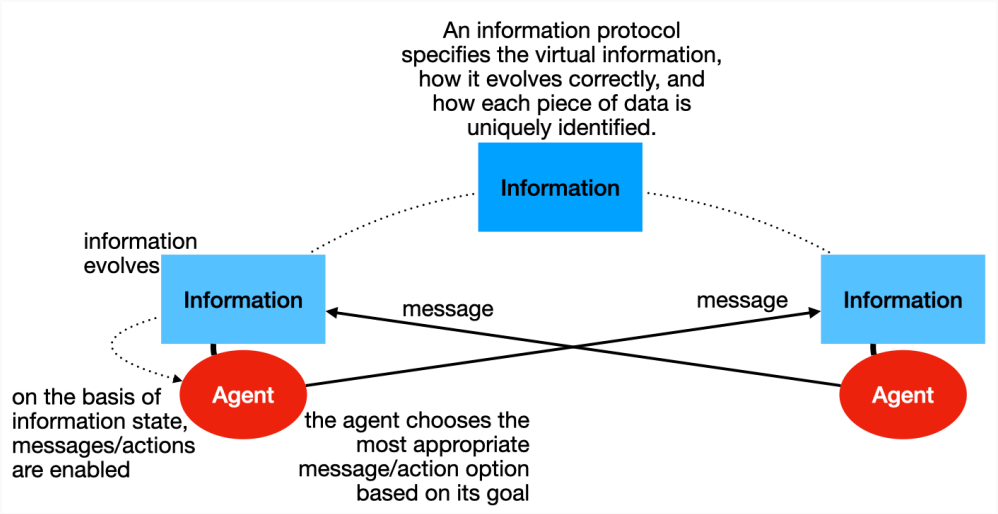
Sender:



BSPL information protocols [Singh, 2011]

- No two message instances with the same bindings for overlapping ⌜key⌝ parameters may have distinct bindings for common non-key parameters
- No two message instances may have overlapping key parameter bindings as well as a binding of the same ⌜out⌝ parameter
- The key parameters of a protocol provides a basis for the uniqueness of its enactments

BSPL information protocols [Singh, 2011]



Orpheus [Baldoni et al., 2025]

Orpheus unites two aspects of autonomy

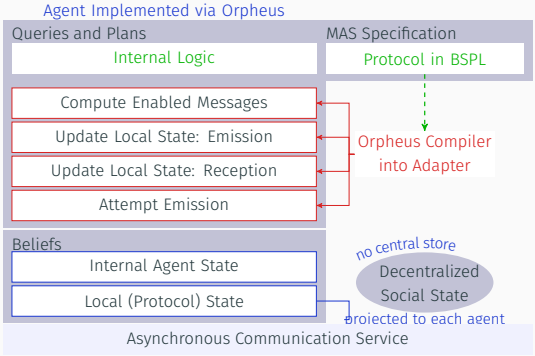
- **Cognitive autonomy**, via Jason [Vieira et al., 2007]
- **Social autonomy**, via information protocols, in particular, *Blindingly Simple Protocol Language* (BSPL) [Singh, 2011]

Orpheus overcomes shortcomings of message-centric interaction protocols

- Offering to *programmers/developers* AOPLs that include **higher level abstractions** that **hide** low-level messaging concerns (as recommended in [Winikoff, 2012])
- its centrepiece is the generation of Jason adapter that supports an agent programming (API) that enables engineering loosely coupled, flexible, and decentralised MAS

Designing Agents with Orpheus

- Orpheus focuses not on reactions to incoming messages
- Orpheus focuses on computing **messages enabled** to be sent given the protocol semantics and the **information available** to the agent
- Orpheus abstracts out reasoning about the protocol into automatic generated code (through the *Orpheus Tool*)



Designing Agents with Orpheus

An incoming message is added to the local state if it is consistent with the local state

- I.e., if no other binding is already known for any its parameters (relative to the key)

Outgoing messages

- An enabled instance is a partial instance in that:
  - its IN parameters are bound because their bindings are known, and
  - its OUT parameters are not bounded because they are not known
- An attempt is successful if the completed messages are mutually consistent in their bindings; the sent messages are added to the local state

Orpheus Programming Model: Enabled-Based Programming Model

Orpheus supports a novel programming model based on message enablement, in which the developer specifies plans for emitting enabled messages

To achieve some goal, the agent

- **queries** if there are enabled instances corresponding to the message it wants to send,
- **completes** them by producing bindings for their OUT parameters, and
- **attempts** to send them in one shot

Orpheus Programming Model: Enabled-Based Programming Model

Orpheus supports a novel programming model based on message enablement, in which the developer specifies plans for emitting enabled messages

To achieve some goal, the agent

Repeat:

- observes the world and updates its internal model
- builds options
- selects selected option
- completes selected option
- performs selected option
- apllies effetcts

Plan Pattern and Orpheus Primitives

Listing 1: Identify a goal and which messages to send for it

```
+!g
  :  enabled(m1) &...& enabled(mq)
  <- !complete(m1,...,mq);
     !attempt(m1,...,mq).

+!attempt(m1,...,mq)
  :  consistent(m1,...,mq)
  <- for (.member(m[receiver(R)], [m1,...,mq])) {
        .send(R, tell, m);
        +sent(m)
     }.

enabled(m(...)) :- ... //BSPL semantics

consistent(m1...mq) :-...//BSPL semantics

+sent(m) <- ... // BSPL semantics

+m : consistent(m, local) <- ... // BSPL semantics
```

An example: The EBusiness Protocol — Iteration 1

```
1  EBusiness {
2    role Buyer, Seller
3    parameter out ID key, out item, out price, out status
4
5    Seller -> Buyer : offer [out ID key, out item, out price]
6
7    Buyer -> Seller : accept [in ID key, in item, in price, out decision]
8    Buyer -> Seller : refuse [in ID key, in item, in price, out decision, out status]
9
10   Buyer -> Seller : payment [in ID key, in price, in decision, out amountC]
11
12   Seller -> Buyer : shipment [in ID key, in item, in price, in decision, out status]
13 }
```

An example: The EBusiness Protocol — Iteration 2

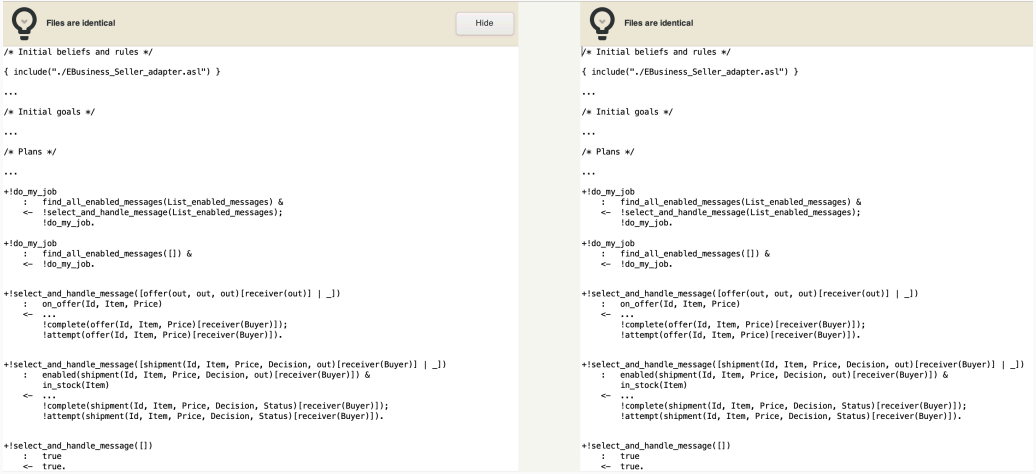
```
EBusiness {
  role Buyer, Seller, Bank
  parameter out ID key, out item, out price, out status

  Seller -> Buyer : offer [out ID key, out item, out price]
  Buyer -> Seller : accept [in ID key, in item, in price, out decision]
  Buyer -> Seller : refuse [in ID key, in item, in price, out decision, out status]

  Buyer -> Seller : payment [in ID key, in price, in decision, out amountC, out choice]
  Buyer -> Bank : instruct [in ID key, in price, in decision, out details, out choice]
  Bank -> Seller : transfer [in ID key, in price, in decision, in details, out amountT]

  Seller -> Buyer : shipment [in ID key, in item, in price, in decision, out status]

}
```

Buyer: Iteration 1 ⟶ Iteration 2

```
/* Initial goals */

...

/* Plans */

...

+!do_my_job
    :   find_all_enabled_messages(List_enabled_messages) &
    <-  !select_and_handle_message(List_enabled_messages);
        !do_my_job.

+!do_my_job
    :   find_all_enabled_messages([]) &
    <-  !do_my_job.


+!select_and_handle_message([accept(Id, Item, Price, out)[receiver(Seller)] | _])
    :   enabled(accept(Id, Item, Price, out)[receiver(Seller)]) &
        wish(Item) &
        acceptable_price(Item, Price)
    <-  ...
        !complete(accept(Id, Item, Price, Decision)[receiver(Seller)]);
        !attempt(accept(Id, Item, Price, Decision)[receiver(Seller)]).


+!select_and_handle_message([refuse(Id, Item, Price, out, out)[receiver(Seller)] | _])
    :   enabled(refuse(Id, Item, Price, out, out)[receiver(Seller)]) &
        (   not wish(Item) |
            not acceptable_price(Item, Price)   )
    <-  ...
        !complete(refuse(Id, Item, Price, Decision, Status)[receiver(Seller)]);
        !attempt(refuse(Id, Item, Price, Decision, Status)[receiver(Seller)]).


+!select_and_handle_message([payment(Id, Price, Decision, out)[receiver(Seller)] | _])
    :   enabled(payment(Id, Price, Decision, out)[receiver(Seller)]) &
        cash_available(Price)
    <-  ...
        !complete(payment(Id, Price, Decision, AmountC)[receiver(Seller)]);
        !attempt(payment(Id, Price, Decision, AmountC)[receiver(Seller)]).


+!select_and_handle_message([])
    :   true
    <-  true.
```

```
+!do_my_job
    :   find_all_enabled_messages(List_enabled_messages) &
    <-  !select_and_handle_message(List_enabled_messages);
        !do_my_job.

+!do_my_job
    :   find_all_enabled_messages([]) &
    <-  !do_my_job.


+!select_and_handle_message([accept(Id, Item, Price, out)[receiver(Seller)] | _])
    :   enabled(accept(Id, Item, Price, out)[receiver(Seller)]) &
        wish(Item) &
        acceptable_price(Item, Price)
    <-  ...
        !complete(accept(Id, Item, Price, Decision)[receiver(Seller)]);
        !attempt(accept(Id, Item, Price, Decision)[receiver(Seller)]).


+!select_and_handle_message([refuse(Id, Item, Price, out, out)[receiver(Seller)] | _])
    :   enabled(refuse(Id, Item, Price, out, out)[receiver(Seller)]) &
        (   not wish(Item) |
            not acceptable_price(Item, Price)   )
    <-  ...
        !complete(refuse(Id, Item, Price, Decision, Status)[receiver(Seller)]);
        !attempt(refuse(Id, Item, Price, Decision, Status)[receiver(Seller)]).


+!select_and_handle_message([payment(Id, Price, Decision, out, out)[receiver(Seller)] | _])
    :   enabled(payment(Id, Price, Decision, out, out)[receiver(Seller)]) &
        cash_available(Price)
    <-  ...
        !complete(payment(Id, Price, Decision, AmountC, Choice)[receiver(Seller)]);
        !attempt(payment(Id, Price, Decision, AmountC, Choice)[receiver(Seller)]).


+!select_and_handle_message([instruct(Id, Price, Decision, out, out)[receiver(out)] | _])
    :   enabled(instruct(Id, Price, Decision, out, out)[receiver(out)]) &
        funds_available_account(Price)
    <-  ...
        !complete(instruct(Id, Price, Decision, Details, Choice)[receiver(Bank)]);
        !attempt(instruct(Id, Price, Decision, Details, Choice)[receiver(Bank)]).


+!select_and_handle_message([])
    :   true
    <-  true.
```

Seller: Iteration 1 ⟶ Iteration 2



Files are identical

Hide

```
/* Initial beliefs and rules */

{ include("./EBusiness_Seller_adapter.asl") }

...

/* Initial goals */

...

/* Plans */

...

+!do_my_job
    :   find_all_enabled_messages(List_enabled_messages) &
    <-  !select_and_handle_message(List_enabled_messages);
        !do_my_job.

+!do_my_job
    :   find_all_enabled_messages([]) &
    <-  !do_my_job.


+!select_and_handle_message([offer(out, out, out)[receiver(out)] | _])
    :   on_offer(Id, Item, Price)
    <-  ...
        !complete(offer(Id, Item, Price)[receiver(Buyer)]);
        !attempt(offer(Id, Item, Price)[receiver(Buyer)]).


+!select_and_handle_message([shipment(Id, Item, Price, Decision, out)[receiver(Buyer)] | _])
    :   enabled(shipment(Id, Item, Price, Decision, out)[receiver(Buyer)]) &
        in_stock(Item)
    <-  ...
        !complete(shipment(Id, Item, Price, Decision, Status)[receiver(Buyer)]);
        !attempt(shipment(Id, Item, Price, Decision, Status)[receiver(Buyer)]).


+!select_and_handle_message([])
    :   true
    <-  true.
```

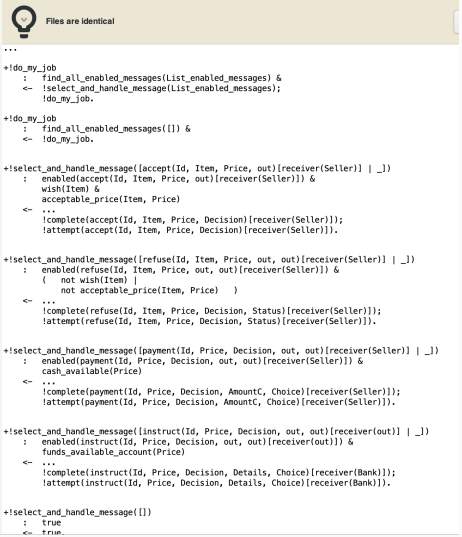
Files are identical

```
/* Initial beliefs and rules */

{ include("./EBusiness_Seller_adapter.asl") }

...

/* Initial goals */

...

/* Plans */

...

+!do_my_job
    :   find_all_enabled_messages(List_enabled_messages) &
    <-  !select_and_handle_message(List_enabled_messages);
        !do_my_job.

+!do_my_job
    :   find_all_enabled_messages([]) &
    <-  !do_my_job.


+!select_and_handle_message([offer(out, out, out)[receiver(out)] | _])
    :   on_offer(Id, Item, Price)
    <-  ...
        !complete(offer(Id, Item, Price)[receiver(Buyer)]);
        !attempt(offer(Id, Item, Price)[receiver(Buyer)]).


+!select_and_handle_message([shipment(Id, Item, Price, Decision, out)[receiver(Buyer)] | _])
    :   enabled(shipment(Id, Item, Price, Decision, out)[receiver(Buyer)]) &
        in_stock(Item)
    <-  ...
        !complete(shipment(Id, Item, Price, Decision, Status)[receiver(Buyer)]);
        !attempt(shipment(Id, Item, Price, Decision, Status)[receiver(Buyer)]).


+!select_and_handle_message([])
    :   true
    <-  true.
```

An example: The EBusiness Protocol — Iteration 3

```
EBusiness {
  role Buyer, Seller, Bank
  parameter out ID key, out item, out price, out status

  Seller -> Buyer : offer [out ID key, out item, out price]
  Buyer -> Seller : accept [in ID key, in item, in price, out decision]
  Buyer -> Seller : refuse [in ID key, in item, in price, out decision, out status]

  Buyer -> Seller : payment [in ID key, in price, in decision, out amountC, out choice]
  Buyer -> Bank : instruct [in ID key, in price, in decision, out details, out choice]
  Bank -> Seller : transfer [in ID key, in price, in decision, in details, out amountT]

  Seller -> Buyer : shipment [in ID key, in item, in price, in decision, out status]

  Seller -> Buyer : refundC [in ID key, in item, in amountC, out status]
  Seller -> Bank : refundT [in ID key, in item, in amountT, out status]
}
```

Buyer: Iteration 2 ⟶ Iteration 3



Seller: Iteration 2 $\longrightarrow$ Iteration 3

```
...

/* Initial goals */

...

/* Plans */

...

+!do_my_job
    :   find_all_enabled_messages(List_enabled_messages) &
    <-  !select_and_handle_message(List_enabled_messages);
        !do_my_job.

+!do_my_job
    :   find_all_enabled_messages([]) &
    <-  !do_my_job.


+!select_and_handle_message([offer(out, out, out)[receiver(out)] | _])
    :   on_offer(Id, Item, Price)
    <-  ...
        !complete(offer(Id, Item, Price)[receiver(Buyer)]);
        !attempt(offer(Id, Item, Price)[receiver(Buyer)]).


+!select_and_handle_message([shipment(Id, Item, Price, Decision, out)[receiver(Buyer)] | _])
    :   enabled(shipment(Id, Item, Price, Decision, out)[receiver(Buyer)]) &
        in_stock(Item)
    <-  ...
        !complete(shipment(Id, Item, Price, Decision, Status)[receiver(Buyer)]);
        !attempt(shipment(Id, Item, Price, Decision, Status)[receiver(Buyer)]).


+!select_and_handle_message([])
    :   true
    <-  true.
```

```
+!do_my_job
    :   find_all_enabled_messages(List_enabled_messages) &
    <-  !select_and_handle_message(List_enabled_messages);
        !do_my_job.

+!do_my_job
    :   find_all_enabled_messages([]) &
    <-  !do_my_job.


+!select_and_handle_message([offer(out, out, out)[receiver(out)] | _])
    :   on_offer(Id, Item, Price)
    <-  ...
        !complete(offer(Id, Item, Price)[receiver(Buyer)]);
        !attempt(offer(Id, Item, Price)[receiver(Buyer)]).


+!select_and_handle_message([shipment(Id, Item, Price, Decision, out)[receiver(Buyer)] | _])
    :   enabled(shipment(Id, Item, Price, Decision, out)[receiver(Buyer)]) &
        in_stock(Item)
    <-  ...
        !complete(shipment(Id, Item, Price, Decision, Status)[receiver(Buyer)]);
        !attempt(shipment(Id, Item, Price, Decision, Status)[receiver(Buyer)]).


\+!select_and_handle_message([refundC(Id, Item, AmountC, out)[receiver(Buyer)] | _])
    :   enabled(refundC(Id, Item, AmountC, out)[receiver(Buyer)]) &
        not in_stock(Item)
    <-  ...
        !complete(refundC(Id, Item, AmountC, Status)[receiver(Buyer)]);
        !attempt(refundC(Id, Item, AmountC, Status)[receiver(Buyer)]).


+!select_and_handle_message([refundT(Id, Item, AmountT, out)[receiver(Bank)] | _])
    :   enabled(refundT(Id, Item, AmountT, out)[receiver(Bank)]) &
        not in_stock(Item)
    <-  ...
        !complete(refundT(Id, Item, AmountT, Status)[receiver(Bank)]);
        !attempt(refundT(Id, Item, AmountT, Status)[receiver(Bank)]).


+!select_and_handle_message([])
    :   true
    <-  true.
```

An example: The EBusiness Protocol — Iteration 4

```
EBusiness {

  role Buyer, Seller, Bank
  parameter out ID key, out item, out price, out status

  Seller -> Buyer : offer [out ID key, out item, out price]

  Buyer -> Seller : accept [in ID key, in item, in price, out decision]
  Buyer -> Seller : refuse [in ID key, in item, in price, out decision, out status]

  Buyer -> Seller : payment [in ID key, in price, in decision, out amountC, out choice]
  Buyer -> Bank : instruct [in ID key, in price, in decision, out details, out choice]
  Bank -> Seller : transfer [in ID key, in price, in decision, in details, out amountT]

  Buyer -> Seller : req_receipt [in ID key, in item, in price, in choice, out request]
  Seller -> Buyer : receiptC [in ID key, in item, in price, in amountC, in request, out declaration]
  Seller -> Buyer : receiptT [in ID key, in item, in price, in amountT, in request, out declaration]

  Seller -> Buyer : shipment [in ID key, in item, in price, in decision, out status]

  Seller -> Buyer : refundC [in ID key, in item, in amountC, out status]
  Seller -> Bank : refundT [in ID key, in item, in amountT, out status]

}
```

Buyer: Iteration 3 ⟶ Iteration 4

```
+!do_my_job
    :   find_all_enabled_messages(List_enabled_messages) &
    <-  !select_and_handle_message(List_enabled_messages);
        !do_my_job.

+!do_my_job
    :   find_all_enabled_messages([]) &
    <-  !do_my_job.


+!select_and_handle_message([accept(Id, Item, Price, out)[receiver(Seller)] | _])
    :   enabled(accept(Id, Item, Price, out)[receiver(Seller)]) &
        wish(Item) &
        acceptable_price(Item, Price)
    <-  ...
        !complete(accept(Id, Item, Price, Decision)[receiver(Seller)]);
        !attempt(accept(Id, Item, Price, Decision)[receiver(Seller)]).


+!select_and_handle_message([refuse(Id, Item, Price, out, out)[receiver(Seller)] | _])
    :   enabled(refuse(Id, Item, Price, out, out)[receiver(Seller)]) &
        (   not wish(Item) |
            not acceptable_price(Item, Price)   )
    <-  ...
        !complete(refuse(Id, Item, Price, Decision, Status)[receiver(Seller)]);
        !attempt(refuse(Id, Item, Price, Decision, Status)[receiver(Seller)]).


+!select_and_handle_message([payment(Id, Price, Decision, out, out)[receiver(Seller)] | _])
    :   enabled(payment(Id, Price, Decision, out, out)[receiver(Seller)]) &
        cash_available(Price)
    <-  ...
        !complete(payment(Id, Price, Decision, AmountC, Choice)[receiver(Seller)]);
        !attempt(payment(Id, Price, Decision, AmountC, Choice)[receiver(Seller)]).


+!select_and_handle_message([instruct(Id, Price, Decision, out, out)[receiver(out)] | _])
    :   enabled(instruct(Id, Price, Decision, out, out)[receiver(out)]) &
        funds_available_account(Price)
    <-  ...
        !complete(instruct(Id, Price, Decision, Details, Choice)[receiver(Bank)]);
        !attempt(instruct(Id, Price, Decision, Details, Choice)[receiver(Bank)]).


+!select_and_handle_message([])
    :   true
    <-  true.
```

```
        !do_my_job.

+!do_my_job
    :   find_all_enabled_messages([]) &
    <-  !do_my_job.


+!select_and_handle_message([accept(Id, Item, Price, out)[receiver(Seller)] | _])
    :   enabled(accept(Id, Item, Price, out)[receiver(Seller)]) &
        wish(Item) &
        acceptable_price(Item, Price)
    <-  ...
        !complete(accept(Id, Item, Price, Decision)[receiver(Seller)]);
        !attempt(accept(Id, Item, Price, Decision)[receiver(Seller)]).


+!select_and_handle_message([refuse(Id, Item, Price, out, out)[receiver(Seller)] | _])
    :   enabled(refuse(Id, Item, Price, out, out)[receiver(Seller)]) &
        (   not wish(Item) |
            not acceptable_price(Item, Price)   )
    <-  ...
        !complete(refuse(Id, Item, Price, Decision, Status)[receiver(Seller)]);
        !attempt(refuse(Id, Item, Price, Decision, Status)[receiver(Seller)]).


+!select_and_handle_message([payment(Id, Price, Decision, out, out)[receiver(Seller)] | _])
    :   enabled(payment(Id, Price, Decision, out, out)[receiver(Seller)]) &
        cash_available(Price)
    <-  ...
        !complete(payment(Id, Price, Decision, AmountC, Choice)[receiver(Seller)]);
        !attempt(payment(Id, Price, Decision, AmountC, Choice)[receiver(Seller)]).


+!select_and_handle_message([instruct(Id, Price, Decision, out, out)[receiver(out)] | _])
    :   enabled(instruct(Id, Price, Decision, out, out)[receiver(out)]) &
        funds_available_account(Price)
    <-  ...
        !complete(instruct(Id, Price, Decision, Details, Choice)[receiver(Bank)]);
        !attempt(instruct(Id, Price, Decision, Details, Choice)[receiver(Bank)]).

✖ +!select_and_handle_message([req_receipt(Id, Item, Price, Choice, out)[receiver(Seller)] | _])
    :   enabled(req_receipt(Id, Item, Price, Choice, out)[receiver(Seller)]) &
        wish(receipt(Id, Item, Price))
    <-  ...
        !complete(req_receipt(Id, Item, Price, Choice, Request)[receiver(Seller)]);
        !attempt(req_receipt(Id, Item, Price, Choice, Request)[receiver(Seller)]).


+!select_and_handle_message([])
    :   true
    <-  true.
```

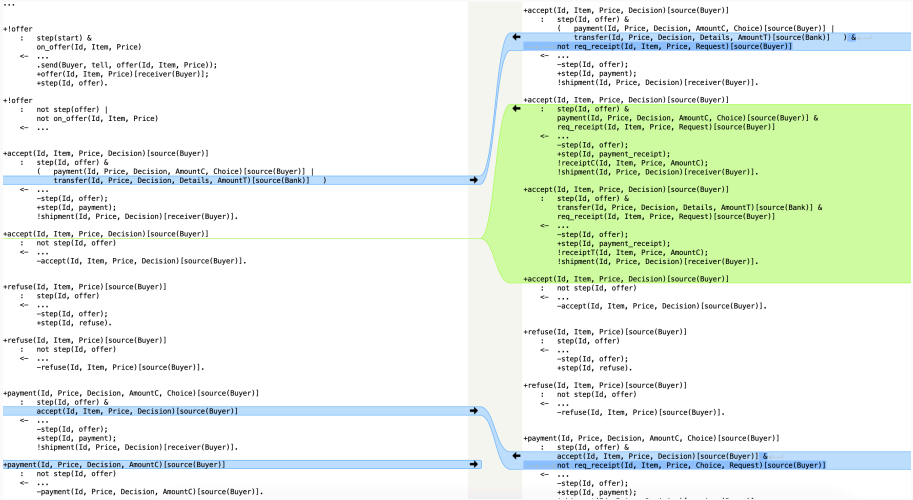
Seller: Iteration 3 ⟶ Iteration 4

```
...

/* Plans */

...

+!do_my_job
    :   find_all_enabled_messages(List_enabled_messages) &
    <-  !select_and_handle_message(List_enabled_messages);
        !do_my_job.

+!do_my_job
    :   find_all_enabled_messages([]) &
    <-  !do_my_job.


+!select_and_handle_message([offer(out, out, out)[receiver(out)] | _])
    :   on_offer(Id, Item, Price)
    <-  ...
        !complete(offer(Id, Item, Price)[receiver(Buyer)]);
        !attempt(offer(Id, Item, Price)[receiver(Buyer)]).


+!select_and_handle_message([shipment(Id, Item, Price, Decision, out)[receiver(Buyer)] | _])
    :   enabled(shipment([Id, Item, Price, Decision, out)[receiver(Buyer)]) &
        in_stock(Item)
    <-  ...
        !complete(shipment(Id, Item, Price, Decision, Status)[receiver(Buyer)]);
        !attempt(shipment(Id, Item, Price, Decision, Status)[receiver(Buyer)]).


+!select_and_handle_message([refundC(Id, Item, AmountC, out)[receiver(Buyer)] | _])
    :   enabled(refundC(Id, Item, AmountC, out)[receiver(Buyer)]) &
        not in_stock(Item)
    <-  ...
        !complete(refundC(Id, Item, AmountC, Status)[receiver(Buyer)]);
        !attempt(refundC(Id, Item, AmountC, Status)[receiver(Buyer)]).


+!select_and_handle_message([refundT(Id, Item, AmountT, out)[receiver(Bank)] | _])
    :   enabled(refundT(Id, Item, AmountT, out)[receiver(Bank)]) &
        not in_stock(Item)
    <-  ...
        !complete(refundT(Id, Item, AmountT, Status)[receiver(Bank)]);
        !attempt(refundT(Id, Item, AmountT, Status)[receiver(Bank)]).


+!select_and_handle_message([])
    :   true
    <-  true.
```

```
+!oo_my_job
    :   find_all_enabled_messages([]) &
    <-  !do_my_job.


+!select_and_handle_message([offer(out, out, out)[receiver(out)] | _])
    :   on_offer(Id, Item, Price)
    <-  ...
        !complete(offer(Id, Item, Price)[receiver(Buyer)]);
        !attempt(offer(Id, Item, Price)[receiver(Buyer)]).

✖ +!select_and_handle_message([receiptC(Id, Item, Price, AmountC, Request, out)[receiver(Buyer)] | _])
    :   enabled(receiptC(Id, Item, Price, AmountC, Request, out)[receiver(Buyer)])
    <-  ...
        !complete(receiptC(Id, Item, Price, AmountC, Request, Declaration)[receiver(Buyer)]);
        !attempt(receiptC(Id, Item, Price, AmountC, Request, Declaration)[receiver(Buyer)]).


+!select_and_handle_message([receiptT(Id, Item, Price, AmountT, Request, out)[receiver(Buyer)] | _])
    :   enabled(receiptT(Id, Item, Price, AmountT, Request, out)[receiver(Buyer)])
    <-  ...
        !complete(receiptT(Id, Item, Price, AmountT, Request, Declaration)[receiver(Buyer)]);
        !attempt(receiptT(Id, Item, Price, AmountT, Request, Declaration)[receiver(Buyer)]).


+!select_and_handle_message([shipment(Id, Item, Price, Decision, out)[receiver(Buyer)] | _])
    :   enabled(shipment(Id, Item, Price, Decision, out)[receiver(Buyer)]) &
        in_stock(Item)
    <-  ...
        !complete(shipment(Id, Item, Price, Decision, Status)[receiver(Buyer)]);
        !attempt(shipment(Id, Item, Price, Decision, Status)[receiver(Buyer)]).


+!select_and_handle_message([refundC(Id, Item, AmountC, out)[receiver(Buyer)] | _])
    :   enabled(refundC(Id, Item, AmountC, out)[receiver(Buyer)]) &
        not in_stock(Item)
    <-  ...
        !complete(refundC(Id, Item, AmountC, Status)[receiver(Buyer)]);
        !attempt(refundC(Id, Item, AmountC, Status)[receiver(Buyer)]).


+!select_and_handle_message([refundT(Id, Item, AmountT, out)[receiver(Bank)] | _])
    :   enabled(refundT(Id, Item, AmountT, out)[receiver(Bank)]) &
        not in_stock(Item)
    <-  ...
        !complete(refundT(Id, Item, AmountT, Status)[receiver(Bank)]);
        !attempt(refundT(Id, Item, AmountT, Status)[receiver(Bank)]).


+!select_and_handle_message([])
    :   true
```

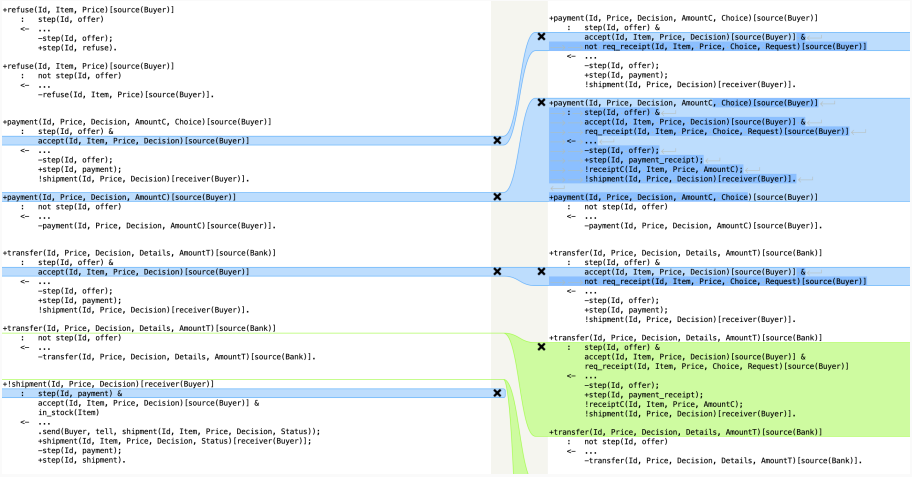
Seller: Iteration 3 ⟶ Iteration 4

In Jason...



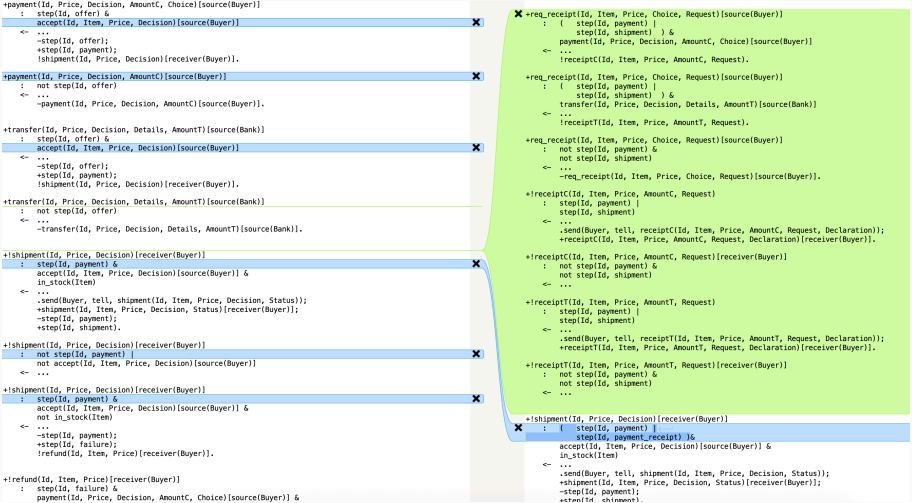
Seller: Iteration 3 $\longrightarrow$ Iteration 4

In Jason...



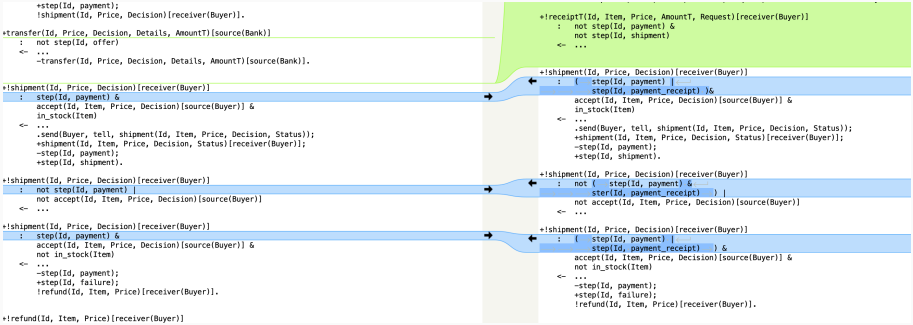
Seller: Iteration 3 ⟶ Iteration 4

In Jason...



Seller: Iteration 3 ⟶ Iteration 4

In Jason...



Comparison: new code lines and modified code lines

Agent	Change	Jason				Orpheus			
		1	$1 \rightarrow 2$	$2 \rightarrow 3$	$3 \rightarrow 4$	1	$1 \rightarrow 2$	$2 \rightarrow 3$	$3 \rightarrow 4$
Buyer	Lines	169	191	203	252	155	185	185	212
	New lines		22	12	49		30	0	25
	Mod. lines		8	0	6		8	0	0
Seller	Lines	144	160	208	311	106	106	154	186
	New lines		15	48	103		0	48	32
	Mod. lines		4	1	8		0	0	0
Bank	Lines		52	64	64		58	58	58
	New lines			12	0			0	0
	Mod. lines			0	0			0	0

Comparison: new code lines and modified code lines

Agent	Change	Jason				Orpheus			
		1	$1 \rightarrow 2$	$2 \rightarrow 3$	$3 \rightarrow 4$	1	$1 \rightarrow 2$	$2 \rightarrow 3$	$3 \rightarrow 4$
Buyer	Lines	169	191	203	252	155	185	185	212
	New lines		22	12	49		30	0	25
	Mod. lines		8	0	6		8	0	0
Seller	Lines	144	160	208	311	106	106	154	186
	New lines		15	48	103		0	48	32
	Mod. lines		4	1	8		0	0	0
Bank	Lines		52	64	64		58	58	58
	New lines			12	0			0	0
	Mod. lines			0	0			0	0

Orpheus, $1 \rightarrow 2$: no new lines and modifications for the Seller.

Comparison: new code lines and modified code lines

Agent	Change	Jason				Orpheus			
		1	$1 \rightarrow 2$	$2 \rightarrow 3$	$3 \rightarrow 4$	1	$1 \rightarrow 2$	$2 \rightarrow 3$	$3 \rightarrow 4$
Buyer	Lines	169	191	203	252	155	185	185	212
	New lines		22	12	49		30	0	25
	Mod. lines		8	0	6		8	0	0
Seller	Lines	144	160	208	311	106	106	154	186
	New lines		15	48	103		0	48	32
	Mod. lines		4	1	8		0	0	0
Bank	Lines		52	64	64		58	58	58
	New lines			12	0			0	0
	Mod. lines			0	0			0	0

Orpheus, $2 \rightarrow 3$: no new lines or modifications for the Buyer.

Comparison: new code lines and modified code lines

Agent	Change	Jason				Orpheus			
		1	$1 \rightarrow 2$	$2 \rightarrow 3$	$3 \rightarrow 4$	1	$1 \rightarrow 2$	$2 \rightarrow 3$	$3 \rightarrow 4$
Buyer	Lines	169	191	203	252	155	185	185	212
	New lines		22	12	49		30	0	25
	Mod. lines		8	0	6		8	0	0
Seller	Lines	144	160	208	311	106	106	154	186
	New lines		15	48	103		0	48	32
	Mod. lines		4	1	8		0	0	0
Bank	Lines		52	64	64		58	58	58
	New lines			12	0			0	0
	Mod. lines			0	0			0	0

Orpheus, $1 \rightarrow 2 \rightarrow 3$: no new lines or modifications for the Bank.

Comparison: new code lines and modified code lines

Agent	Change	Jason				Orpheus			
		1	$1 \rightarrow 2$	$2 \rightarrow 3$	$3 \rightarrow 4$	1	$1 \rightarrow 2$	$2 \rightarrow 3$	$3 \rightarrow 4$
Buyer	Lines	169	191	203	252	155	185	185	212
	New lines		22	12	49		30	0	25
	Mod. lines		8	0	6		8	0	0
Seller	Lines	144	160	208	311	106	106	154	186
	New lines		15	48	103		0	48	32
	Mod. lines		4	1	8		0	0	0
Bank	Lines		52	64	64		58	58	58
	New lines			12	0			0	0
	Mod. lines			0	0			0	0

Orpheus: fewer modified lines (and fewer new lines).

Comparison: new plans and modified plans

Agent	Change	Jason				Orpheus			
		1	$1 \to 2$	$2 \to 3$	$3 \to 4$	1	$1 \to 2$	$2 \to 3$	$3 \to 4$
Buyer	Plans	19	21	23	30	18	22	22	26
	New plans		2	2	7		4	0	4
	Mod. plans		3	0	6		4	0	0
Seller	Plans	18	20	27	40	14	14	22	28
	New plans		2	7	13		0	8	6
	Mod. plans		2	1	7		0	0	0
Bank	Plans		7	9	9		8	8	8
	New plans			2	0			0	0
	Mod. plans			0	0			0	0

Comparison: new plans and modified plans

Agent	Change	Jason				Orpheus			
		1	$1 \to 2$	$2 \to 3$	$3 \to 4$	1	$1 \to 2$	$2 \to 3$	$3 \to 4$
Buyer	Plans	19	21	23	30	18	22	22	26
	New plans		2	2	7		4	0	4
	Mod. plans		3	0	6		4	0	0
Seller	Plans	18	20	27	40	14	14	22	28
	New plans		2	7	13		0	8	6
	Mod. plans		2	1	7		0	0	0
Bank	Plans		7	9	9		8	8	8
	New plans			2	0			0	0
	Mod. plans			0	0			0	0

Orpheus, $1 \to 2$: no new or modified plans for the Seller.

Comparison: new plans and modified plans

Agent	Change	Jason				Orpheus			
		1	$1 \to 2$	$2 \to 3$	$3 \to 4$	1	$1 \to 2$	$2 \to 3$	$3 \to 4$
Buyer	Plans	19	21	23	30	18	22	22	26
	New plans		2	2	7		4	0	4
	Mod. plans		3	0	6		4	0	0
Seller	Plans	18	20	27	40	14	14	22	28
	New plans		2	7	13		0	8	6
	Mod. plans		2	1	7		0	0	0
Bank	Plans		7	9	9		8	8	8
	New plans			2	0			0	0
	Mod. plans			0	0			0	0

Orpheus, $2 \to 3$: no new ore modified plans for the Buyer.

Comparison: new plans and modified plans

Agent	Change	Jason				Orpheus			
		1	$1 \to 2$	$2 \to 3$	$3 \to 4$	1	$1 \to 2$	$2 \to 3$	$3 \to 4$
Buyer	Plans	19	21	23	30	18	22	22	26
	New plans		2	2	7		4	0	4
	Mod. plans		3	0	6		4	0	0
Seller	Plans	18	20	27	40	14	14	22	28
	New plans		2	7	13		0	8	6
	Mod. plans		2	1	7		0	0	0
Bank	Plans		7	9	9		8	8	8
	New plans			2	0			0	0
	Mod. plans			0	0			0	0

Orpheus, $1 \to 2 \to 3$: no new or modified plans the Bank.

Comparison: new plans and modified plans

Agent	Change	Jason				Orpheus			
		1	$1 \to 2$	$2 \to 3$	$3 \to 4$	1	$1 \to 2$	$2 \to 3$	$3 \to 4$
Buyer	Plans	19	21	23	30	18	22	22	26
	New plans		2	2	7		4	0	4
	Mod. plans		3	0	6		4	0	0
Seller	Plans	18	20	27	40	14	14	22	28
	New plans		2	7	13		0	8	6
	Mod. plans		2	1	7		0	0	0
Bank	Plans		7	9	9		8	8	8
	New plans			2	0			0	0
	Mod. plans			0	0			0	0

Orpheus: fewer modified plans (and fewer new plans).

Comparison: Jason vs Orpheus

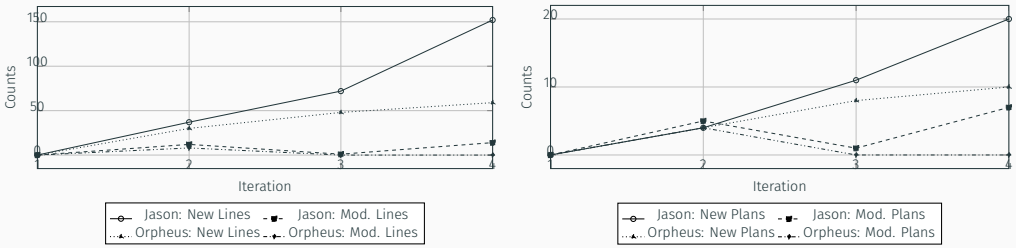


Figure 1: Left: plot of new lines and modified lines in Jason and Orpheus along the four iterations; Right: plot of new plans and modified plans in Jason and Orpheus along the four iterations

The latest versioin of Orpheus (and Azorus)

https://gitlab.di.unito.it/baldoni/argonauts

Conclusions

- Azorus [Chopra et al., 2025], semantics for the messages
- Development of middleware that integrates BSPL adapter functionality and makes their use fully transparent.
  We are working on an extended version of JADE that replaces traditional message queues with BSPL adapters
- Methodology for developing agents that exploit BSPL protocols
- Methodology for developing BSPL protocols.
  We are developing an approach based on describing information evolution rather than the protocol itself, since writing a correct BSPL protocol is not always easy
- Agentic AI: Inspired by Agora meta-protocol [Marro et al., 2024], when the same kind of interaction become recurrent we can standardize the communication by the adoption of protocols

Baldoni, M., Christie V, S. H., Singh, M. P., and Chopra, A. K. (2025).
Orpheus: Engineering multiagent systems via communicating agents.
In *Proceedings of the 39th AAAI Conference on Artificial Intelligence*, Philadelphia. AAAI.

Chopra, A. K., Baldoni, M., Christie V, S. H., and Singh, M. P. (2025).
Azorus: Commitments over protocols for BDI agents.
In *Proceedings of the 24th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, Detroit. IFAAMAS.

Marro, S., Malfa, E. L., Wright, J., Li, G., Shadbolt, N., Wooldridge, M., and Torr, P. (2024).
A scalable communication protocol for networks of large language models.

Singh, M. P. (1999).

An ontology for commitments in multiagent systems: Toward a unification of normative concepts.
Artificial Intelligence and Law, 7:97–113.


Singh, M. P. (2011).
Information-driven interaction-oriented programming: BSPL, the Blindingly Simple Protocol Language.
In *Proceedings of the 10th International Conference on Autonomous Agents and MultiAgent Systems (AAMAS)*, pages 491–498, Taipei. IFAAMAS.


Singh, M. P. (2013).
Norms as a basis for governing sociotechnical systems.
ACM Transactions on Intelligent Systems and Technology (TIST), 5(1):21:1–21:23.


Vieira, R., Moreira, Á. F., Wooldridge, M. J., and Bordini, R. H. (2007).

On the formal semantics of speech-act based communication in an agent-oriented programming language.
Journal of Artificial Intelligence Research (JAIR), 29:221–267.

Winikoff, M. (2012).
Challenges and directions for engineering multi-agent systems.
CoRR, abs/1209.1428.