DOI: 10.15514/ISPRAS-2026-38(5)-7

OCSD: Exploration of SOA-Based Systems Using Object-Centric Sequence Diagrams

Jinyu Zhou, ORCID: 0009-0005-0827-2910 <tschzhou_3@edu.hse.ru>
Sergey A. Shershakov, ORCID: 0000-0001-8173-5970 <sshershakov@hse.ru>

Faculty of Computer Science, HSE University,
20, Myasnitskaya st., Moscow, 101000, Russia.

Abstract. A traditional approach to Process Mining, when working with flat event logs, primarily focuses on the control-flow aspect, whereas the object-centric approach enables the analytical perspectives of highly distributed and interdependent event data produced by Service-Oriented Architecture (SOA) systems. Specifically, Object-Centric Process Mining (OCPM) addresses this issue by preserving the relationships of multiple interacting objects in a more faithful representation of the system behavior. In this work, we present a comparative study of traditional and object-centric process mining approaches applied to SOA systems and show how the modeling of multiple interacting objects facilitates the analysis of service interactions. To support the transition from object-centric event data to software engineering practices, we develop a new tool that automatically visualizes object-centric event logs (reconstructed from flattened event logs) as Object-Centric Sequence Diagrams (OCSDs). Our method improves over the OCEL visualizations by providing (i) temporal interaction details and (ii) contextually consistent tracking. A real-world case study shows that our tool reveals unknown patterns hidden in service interactions, such as redundant object accesses and temporal bottlenecks, which support the diagnosis and optimization of distributed services.

Keywords: object-centric process mining; sequence diagrams; SOA systems.

For citation: Zhou J., Shershakov S.A. OCSD: Exploration of SOA-Based Systems Using Object-Centric Sequence Diagrams. Trudy ISP RAN/Proc. ISP RAS, vol. 38, issue 5, 2026, pp. 103-118. DOI: 10.15514/ISPRAS-2026-38(5)-6

Acknowledgements. This article was prepared within the framework of the HSE University Basic Research Programme.

OCSD: Исследование СОА-систем с помощью объектно-центрических диаграмм последовательности

Ц. Чжоу, ORCID: 0009-0005-0827-2910 <tschzhou_3@edu.hse.ru>
С.А. Шершаков, ORCID: 0000-0001-8173-5970 <sshershakov@hse.ru>

Факультет компьютерных наук НИУ «Высшая школа экономики»,
Россия, 101000, Москва, ул. Мясницкая, 20.

Аннотация. Традиционный подход к процессному анализу (Process Mining) при работе с плоскими журналами событий в первую очередь фокусируется на аспекте потока управления, тогда как объектно-центрический подход открывает аналитические перспективы для сильно распределенных и взаимозависимых данных о событиях, порождаемых системами, основанными на сервис-ориентированной архитектуре (СОА). В частности, объектно-центрический процессный анализ (Object-Centric Process Mining, OCPM) решает эту проблему, сохраняя связи между множеством взаимодействующих объектов в более точном представлении поведения системы. В данной работе мы представляем сравнительное исследование традиционного и объектно-центрического подходов к процессному анализу, применяемых к СОА-системам, и показываем, как моделирование множества взаимодействующих объектов облегчает анализ сервисных взаимодействий. Для поддержки перехода от объектно-центрических данных о событиях к практикам разработки программного обеспечения мы разрабатываем новый инструмент, который автоматически визуализирует объектно-центрические журналы событий (восстановленные из плоских журналов) в виде объектно-центрических диаграмм последовательности (Object-Centric Sequence Diagrams, OCSD). Наш метод дополняет существующие визуализации в формате OCEL, предоставляя (i) детальную информацию о временных взаимодействиях и (ii) контекстное отслеживание. Реальное практическое исследование показывает, что наш инструмент выявляет закономерности, скрытые в сервисных взаимодействиях, такие как избыточные обращения к объектам и временные узкие места, что способствует диагностике и оптимизации распределенных сервисов.

Ключевые слова: объектно-центрический анализ процессов; диаграммы последовательности; сервис-ориентированная архитектура

Для цитирования: Чжоу Ц., Шершаков С.А. OCSD: Исследование СОА-систем с помощью объектно-центрических диаграмм последовательности. Труды ИСП РАН, 2026, том 38, вып. 5, стр. 103–118 (на английском языке). DOI: 10.15514/ISPRAS-2026-38(5)-7.

Благодарности. Исследование выполнено в рамках Программы фундаментальных исследований НИУ ВШЭ.

1. Introduction

Service-Oriented Architecture (SOA)-based systems are widely used in many modern digital services, including online payments, e-commerce platforms, and banking infrastructures [1]. These systems generate vast amounts of distributed and interdependent data as information flows through interconnected service domains. An accurate analysis of these data will facilitate the resolution of bottlenecks, enhance operational efficiency and reduce overhead costs.

Analysis, in turn, can be conducted using several different approaches. One such method is statistical analysis, which employs classical data mining techniques and enables searches based on absolute values for tasks such as clustering and boundary violation detection. However, processes supported by SOA-based systems require approaches that take into account the dynamic nature of the process data. A well-established alternative lies in the domain of process mining [2], whose methods allow analysts to establish a process model and explore its properties through various perspectives. However, while classical process mining primarily focuses on the control-flow perspective and often relies on flat, case-based event logs, it faces limitations when applied to complex SOA-based systems. These limitations are mainly associated with overlapping and interacting entities. These interaction services forward a large amount of information, which are a serialized form of objects at

the level of the system's information model. The correct analysis of this data could considerably enrich the system models, but for this purpose it would be necessary to go beyond the boundaries of the approaches in classical process mining.

The development of the ideas of classical process mining became *Object-Centric Process Mining* (*OCPM*) [3], the methods of which maintain the relationships between multiple interacting objects [4-5], rather than focusing on individual cases. This enables the detection of process anomalies such as repeated access to specific data objects, bottlenecks in interaction sequences, or loops of inefficient service invocation.

To demonstrate the limitations of the *classical process mining* as a running example we will consider a specific use case of a large European touristic Computer Reservation System (CSR) with distributed SOA-based systems, which was previously explicitly analyzed on violations of architectural principles and patterns in the paper [6]. This article proposed solutions using classical process mining techniques. However, these approaches did not consider the data contained within the objects exchanged between services, which constituted valuable information in the form of *payload* in the original log.

This paper aims to bridge existing gaps in system analysis by incorporating methods recently proposed in the field of Object-Centric Process Mining. We introduce a novel visualization approach based on UML Sequence Diagrams, the construction of which incorporates the life cycle of processes in the perspective of service-object interaction. We will refer to the latter as the *Object-centric Sequence Diagram* (*OCSD*). The proposed method has been implemented as a software tool that generates a script compatible with the D2 text-to-diagram platform to produce the target sequence diagram, from which we obtained new knowledge that can convert into values.

The rest of the paper is organized as follows. In Section 2 we discuss some related works. Section 3 introduces the proposed methodology with theoretical propositions and data preprocessing procedure for conducting the experiments. In Section 4 we explicitly identify objects in the business processes and suggest solutions for associated issues. Section 5 formalizes the Object-Centric Sequence Diagrams semantics, while in Section 6 we discuss the tool implementation details. And, finally, Section 7 presents the results of our research and concludes with future directions for expanding this work.

2. Related Work

In this section we will review the contributions related to the observed topic and highlight similarities and differences of the approaches under study. They will be grouped according to different types of process modeling approaches.

2.1 Imperative Object-Centric Models

A first line of work focuses on extending classical imperative models to handle multiple interacting objects. OCPNs in [7] and [8] extend Petri nets with object tokens, enabling concurrency and synchronization across object types. Similarly, OC-DFGs [4] generalize directly-follows graphs by distinguishing event paths by object classes, preventing misleading aggregations. However, these models, in their current form, are often considered to have over-approximations of actual process behavior, since they do not capture all contextual details. A key limitation is that current object-centric approaches typically represent only object identifiers, without additional attributes. For instance, if it refers to an order, its value remains unknown. This absence of object properties prevents more precise modeling.

2.2 Constraint-Oriented Object-Centric Models

Another stream emphasizes declarative representations. Declarative approaches capture inter-object rules rather than prescribing full execution flows. Object-Centric Behavioral Constraints

(OCBC) [9-10] integrate data and control-flow perspectives and support many-to-many relations, but they remain largely abstract, focusing on constraint sets instead of concrete execution. Similarly, Object-Centric Constraint Graphs (OCCGs) [11] enable runtime monitoring but currently cover only limited constraint types and lack support for temporal ordering or timed conditions, which are crucial in practice. Ongoing research also highlights open issues such as subtyping, implicit behavioral patterns, and consistency checking across data and process constraints.

2.3 Hybrid UML and Visualization-Oriented Methods

An early contribution in this direction is described in [12], where authors generated hybrid UML models (class, activity, and sequence diagrams) from flat SOA event logs. Their work can be seen as a precursor to object-centric visualization, demonstrating how event data can be transformed into interpretable diagrams and paving the way for tools like the sequence diagram visualizer proposed in this thesis. However, the suggested approach is not suitable for presenting object-centric event logs which demonstrate a broader view of the observed processes.

2.4 Contribution of this paper

Our research addresses these gaps by reconstructing object attributes from enriched XML-based logs and incorporating them into UML sequence diagrams (OCSD). Unlike existing approaches that represent objects only through abstract identifiers, our method reveals their specific properties and values, making the diagrams both better visualization abilities and closer to real-world execution. This allows to represent not necessarily only structural dependencies, but also the actual values of objects (e.g., cost, customer name, booking time), thereby capturing how objects evolve during interactions. The suggested approach assists in detecting long-life-span objects and provides temporal detail about the corner cases. This more detailed investigation allows practitioners to detect potential anomalies in the system and eliminate them effectively.

3. Logs structure and preprocessing of the event logs

The description and preparation of event logs is discussed in this section due to the direct influence of their structure on the performance of the proposed method.

3.1 Logs Structure

We will start our exploration by analyzing the SOA-based system described in our running example, the structure of which is described in [6]. In our case, the CRM system of the running example contains communication between the services and the processes orchestrating them. In the event logs each entry of the corresponding event represents a record of the operation executed by a process or a service. The events are represented as records in *TracingEvent* table shown in Fig. 1.

Events related to a single business process receive unique invocation identificator (*InvID*). Both processes (*PR*) and services (*SV*) handle a request message (*RQ*) and respond with either a response (*RS*) or an exception (*SE*) as a returning message. Besides *InvID* each event is described by the following attributes: *Interface_ID* and *OperationName*, which in concatenation or as discrete values may serve as an action with respect to the classical process mining perspective; *EventTimestamp* that is normally used to order the events in the log trace; *ActionType* (*RQ, RS, SE*) and other attributes. Finally, we would especially consider the attribute payload individually, which represents a "container" of data transferred by the invoked operation as an argument. The structure of these data depends on the specific operation and can vary in size of embedded information. Classical process mining methods generally overlook content from the payload, while its extraction provides valuable information about the objects transmitted between the processes and the services, and serves as a foundation to perform the analysis using OCPM methods. This study extends the approach described in [6] by considering objects extracted from the payload attribute.

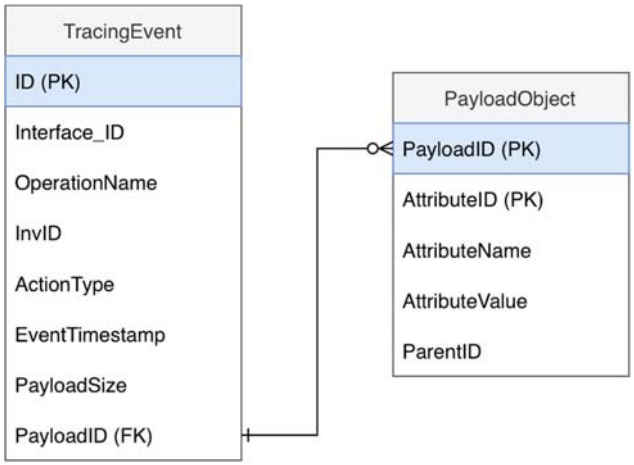


Fig. 1. Scheme of a transformed flat event log with extracted objects represented as a relational database.

3.2 Preprocessing of the event logs

In the mentioned example, the analysis was performed on a 24-hour event log, presented as a collection of XML files amounting to 10 GB in total.

The object extraction process includes:

- Converting event log from its XML representation to a corresponding relational representation [13].
- Parsing payload XML node into separated attribute instances.
- Defining objects and their types based on its internal structure.
- Detecting identical objects and their lifecycles.

In order to represent object attributes in hierarchical form which is closer to its XML representation, we design dataset structure depicted in Fig. 1. Such representation method supports discovering of event-object interaction patterns.

Moreover, it enables tracing object lifecycles and analyzing inter-object communication over time by providing necessary SQL queries which take a part of analysis scheme. The proposed representation is realized as a database table $PayloadObject$ which consists of the following five attributes:

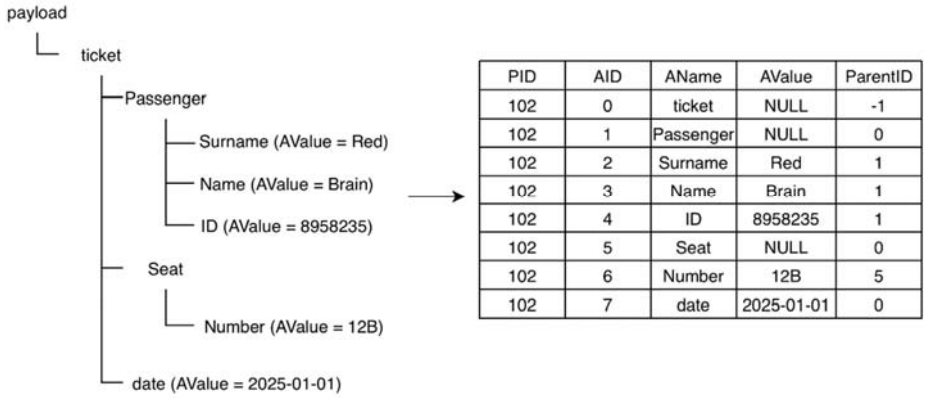
- $PayloadID$ – Identifier for a single payload section instance. It has usually one-to-one relation with a single event represented as a record in TracingEvent table.
- $AttributeID$ – Sequential identifier of an attribute within the payload.
- $AttributeName$ – Name of the attribute extracted from the XML representation.
- $AttributeValue$ – Specific value assigned to the attribute.
- $ParentID$ – Identifier of the parent attribute, enabling a hierarchical structure.

Special interpretation rules (shown in Fig. 2) are introduced for handling hierarchical data:

- $PayloadID$ (PID) is assigned to be the $Event\ ID$ which all objects inside this payload section belong to.

- $AttributeID$ (AID) is an increment and unique within a single payload.
- If $AttributeValue$ (AValue) is $NULL$, the attribute contains nested data (i.e., it is
- considered as a container for inherited attributes).
- If an attribute represents some root node of a payload tree, then $Parent\ ID$ (Parent) is set to the special value -1.

Both $PayloadID$ and $AttributeID$ constitute the globally unique attribute identifier. According to this scheme, an object is defined as the top-level (root) attribute within a single payload section, and hence its $AttributeValue$ is always $NULL$. An example of the transformation from an XML payload structure to its corresponding relational representation is presented in Fig. 2. In this instance, the payload is assigned a $PayloadID$ of 106 and comprises 8 attributes, where the key attribute constitutes an object of the type «ticket», with its nested attributes representing the values for this specific object instance. It should be noted that the $Passenger$ and $Seat$ attributes are also structured containers, each possessing their own nested attributes. Conversely, attributes such as $Surname, Name, Number$, and $date$ are scalar elements, defined respectively by the string, enum, and date/timestamp data types.



PID	AID	AName	AValue	ParentID
102	0	ticket	NULL	-1
102	1	Passenger	NULL	0
102	2	Surname	Red	1
102	3	Name	Brain	1
102	4	ID	8958235	1
102	5	Seat	NULL	0
102	6	Number	12B	5
102	7	date	2025-01-01	0

Fig. 2. Rules implementation example for PayloadID 106.

4. Objects in OCEL

Objects in the analyzed event log do not possess a unique global identifier, which is necessary to trace an object instance's complete life path across potential modifications to its value. This absence of consistent identifiers represents a common data quality issue in real-world event logs, often associated with both relational and object-based challenges [14]. However, it is still necessary to identify possible matching objects in different events. For example, we may assume that Payload Objects under the Event 2 and Event 4, having the same payload size, are actually the same object (Table 1).

Before defining object, it is essential to introduce the definition of multiset.

Definition 1 (Multiset) *Let X be a set. A multiset over X is a pair (X, f), where $f : X \to N$ is a function that assigns to each element $x \in X$ a natural number $f(x)$, called the multiplicity of x. The multiset may also be denoted by $[x_1{}^{f(x_1)}, x_2{}^{f(x_2)}, \dots, x_n{}^{f(x_n)}]$, where $f(x_i)$ represents the number of occurrences of x_i.*

In the following, we consider O to be such a *multiset* of object instances extracted from event logs.

Table 1. Partial Database Records Extracted from Initial Event Logs with Assumed Identical Objects.

ID	Interface_ID	OperationName	InvID	ActionType	PayloadSize
0	1	logon	1286522252	RQ	537
1	1	logon	1286522252	RS	819
2	2	**searchReservation**	53230551	RQ	**171**
3	2	searchReservation	53230551	RS	1154
4	2	**searchReservation**	53230551	RQ	**171**
5	2	invoiceDelivering	53230551	RQ	1452

Therefore, now, an object can be defined as,

Definition 2 (Object) *Let O be a multiset of object instances extracted from event logs. Each object $o \in O$ is defined as a tuple:*

$$o = (id, parent, name, value, attributes)$$

where:

- $id \in N$ *is a unique identifier of the object instance,*
- $parent \in N \cup \{\perp\}$ *is the identifier of the parent object, or* $\perp$ *if none exists,*
- $name \in \Lambda^*$ *is the symbolic name of the object,*
- $value \in N \cup \Lambda * \cup B \cup \{NULL\}$ *represents the current value of the object (numeric, string, boolean, or NULL),*
- $attributes \subseteq O$ *is a finite set of sub-objects (attributes) that structurally belong to* o.

We will refer to the object tuple elements as o.id, o.parent, o.name and etc. From the Definition 2 it is evident that the objects are considered as recursive structures with possible deviations. Therefore, before we can determine the equivalence between the object instances, we first distinguish between primitive and structural objects, since this categorization affects how equivalence is interpreted. By definition,

Definition 3 (Objects by Structure) *Let O be the set of all objects.*

An object $o \in O$ is called:

- ***primitive*** *if o.attributes= $\emptyset$,*

$$O_{prim} = \{o \in O | o.attributes = \emptyset \wedge o.value \in R \cup \Lambda^* \cup B\}$$

- ***structural***

$$O_{struct} = \{o \in O | o.attributes \neq \emptyset\}.$$

From Definition 3 we observe that primitive objects represent atomic values (numbers, strings, booleans) without internal structure, whereas structural objects consist of collections of sub-objects or attributes. For the *primitive* objects we will consider only their $AName$ and $AValue$. For example, object date from Fig. 2. Its $o.AName = date, o.Avalue = 2025-01-01$. For structural objects, we will observe in addition its components. The example for this class of objects will be Passenger. Its $o.AName = Passenger, o.Avalue = NULL, o.attributes = \{Surname, Name, ID\}$. Our analysis centers on the latter class of objects.

Structural objects can be identified through a simple SQL query performed on the database considered previously that selects all attributes with $NULL$ values and groups them by name:


```
WITH hypo AS (
        SELECT * FROM PayloadObject
        WHERE AValue = 'NULL'
)
SELECT AName, COUNT(*) FROM hypo
GROUP BY AName;
```

AName	Count
AcSuppBatchParameterMsg	16
AcSuppForeignId	7590
AcSuppSpiRq	18
AcSuppSpiRs	18
AcSuppTicketingInfoReferenceRq	16
...	...
voucherPrintDate	140
vouchers	24
zipCode	75

Fig. 3. SQL query for extracting NULL values and the resulting frequency table.

Now it is possible to define equivalent objects, considering their structural properties.

Definition 4 (Objects Equivalence)

$$\forall o_1, o_2 \in O: \begin{bmatrix} o_1.AName = o_2.AName, \\ o_1.AValue = o_2.AValue, \\ \forall a_1 \in o_1.atributes, \forall a_2 \in o_2.atributes: \\ a_1.AName = a_2.AName, \\ a_1.AValue = a_2.AValue \end{bmatrix} \Leftrightarrow [o_1 = o_2]$$

4.1 Defining Identical Objects

Having introduced the notion of object equivalence, we now describe how such relationships are represented within an *Object-Centric Event Log (OCEL).*

Definition 5 (Object-Centric Event Log) *Let E denote a finite set of events, A to be a finite set of activities manifested in the events of a log, T a set of timestamps, O a finite set of object instances, and Θ a finite set of object types. Each object $o \in O$ has a type $type(o) \in \Theta$. Then, an Object-Centric Event Log (OCEL) is a tuple:*

$$L = (E, O, \Theta, act, ts, obj)$$

where:

- $act: E \to A$ *assigns an activity to each event, since the activity involves the concatenation of InterfaceID and OperationName.*
- $ts: E \to T$ *assigns a timestamp commonly used to order event appearances in a log.*
- $obj: E \to 2^O$ *maps each event to the set of objects it refers to.*

Let us consider our running example. The specific representation of this Definition 5 depends on the analytical choice when constructing the process perspective. As an illustration, we refer to a set of activities from *OperationName: {logon, searchReservation, invoiceDelivering, copyDocument, cancelReservation, etc.}*. The payload size and the corresponding objects for Event 2 can be observed in Table 1. The payload section of this event contains the object ticket, which in turn consists of two sub-objects, $Passenger$ and $Seat$, whose internal structure is further defined by some scalar attributes.

Regarding object-centric specifics, object-centric event logs enable the definition of behavioral semantics across multiple interacting entities. Unlike traditional flat logs that associate each event with a single case, object-centric event logs allow an event to refer to multiple objects, each potentially of a different type. This richer structure makes it possible to model and analyze behaviors that span across object lifecycles and types.

Formally, let $e \in E$ be an event and $obj(e) \subseteq O$ the set of objects it involves. A sequence of events can then be projected onto any individual object o or its object type τ, yielding a perspective of system's behavior in terms of objects.

For our case study we hypothesize that two equivalent objects (as per Definition 4) that appear in distinct events may correspond to different copies of the same type or to the same object instance observed multiple times. Formally, we assume the identity of any such two objects as follows:

Assumption 1 (Object Identity with Equivalence)

$$\forall e_1, e_2 \in E, \forall o_1 \in e_1, \forall o_2 \in e_2 : \begin{pmatrix} act(e_1) = act(e_2) \\ e_1.InvID = e_2.InvID, \\ o_1 = o_2 \end{pmatrix} \Rightarrow [o_1 \; is \; o_2]$$

Two theoretical scenarios emerge from the formulated Assumption 1. The first considers the possibility that, from a technical aspect, two payloads may refer to distinct objects, despite sharing identical characteristics, thus presenting them as indistinguishable in terms of internal structure. Nevertheless, from a logical perspective, such payload should be treated as several representations of the same object. If the system explicitly maintained object references (e.g., memory addresses or unique identifiers), the uncertainty would be resolved, enabling a precise determination of whether two events refer to the same or merely equivalent objects. Given the constraints of the current database schema, it was necessary to adapt the underlying table structure.

4.2 Identifying Dynamic Objects

Following the discussions, it was concluded that merely identifying structural or primitive objects is insufficient. A technical peculiarity of log generation lies in the presence of multiple serialized copies of the same object within the event log. These copies need to be identified and treated as representations of a single object within the process. This issue will be referred in this section as the importance of the potential dynamic objects, which are subject to iteration over time. Recognizing object evolution is essential for understanding real-world process behavior, as many business scenarios require tracking changes to objects throughout their lifecycle.

At this stage of the study, we partially accept Assumption 1, which posits that structural objects sharing identical internal structure and content can be regarded as contextually identical. Let A be a set of observed activities, function $act^{-1}()$ be a reverse function of $act()$ which returns for a specific activity a a set of events e, such that $act(e) = a$, that is preimage of A. We will refer to o.type as to the o.name, i.e. name of the object will define its object type. Then, the approach of identifying equivalent dynamic objects will have the following algorithm:

1) **Event Class Selection.** A subset of activities of interest $A' \subseteq A$.
2) **Contextual Object Identification.** For a given activity $a \in A'$ and an object o that appears in its context (that is in events $e \in act^{-1}(a)$ where $o \in obj(e)$), the object's type τ is noted.
3) **Expert Identification.** This results in a pair (a, τ), representing a business action and an object type that appears within it. A domain analyst then defines a set of attributes for objects of type τ that, when changed, are considered to be the object's evolution along its lifespan.

Therefore, the criteria of two dynamic objects $o_1 \in obj(e_1)$ and $o_2 \in obj(e_2)$ equivalence will be:

Definition 6 (Dynamic Objects Equivalence) *Let A_{a^*,τ^*} be the set of mutable attributes defined by domain expert for pair (a^*, τ^*). Then, objects $o_1 \in obj(e_1)$ and $o_2 \in obj(e_2)$ are equivalent by definition iff*

- $act(e_1) = act(e_2) = a^*$
- $o_1.type = o_2.type = \tau$, *where* $\tau \in (a^*, \tau^*)$ *and* $act(e_1) = a^*$
- $\forall a_1 \in o_1.attributes, a_2 \in o_2.attributes \; a_1.type, a_2.type \in \tau \backslash \tau^* : a_1 = a_2$ (!)

(!) This criterion allows the mutable attributes of o_1 and o_2 to differ, which is recorded as an evolutionary state change. If the mutable attributes are identical, it signifies that the object, while capable of change, has retained its state in this instance. Failure to meet these criteria results in an "unknown" status of equivalence, we neither confirm nor deny that o_1 and o_2 are the same object.

5. Object-Centric Sequence Diagram Notation

To provide a rigorous definition of the Object-Centric Sequence Diagram (OCSD) and ensure compatibility with UML 2.x sequence diagram semantics, we present its formal syntax in Backus–Naur Form (BNF) [15]. This notation strictly follows the structure and style of the standard UML sequence diagram grammar [16], while extending it with object-centric process mining and SOA-specific primitives derived from Section 4. The OCSD notation reuses all core UML interaction constructs, including lifeline, message, execution specification, state invariant, interaction use and others, and extends them to represent object identifiers, object types, object states, dynamic object equivalence, and SOA-specific interaction metadata (InvID, request/response/exception types, latency, and payload objects).

The complete syntax of OCSD is defined in BNF on Fig. 4.

This formalization adheres to the standard UML sequence diagram metamodel while addressing the unique requirements of object-centric event logs (OCEL) and SOA-based systems:

1) **Object-Centric Lifeline.** The *ocsd-lifeline-ident* extends the UML lifeline with a unique *object-id* (PayloadID.AttributeID), *object-type* (AName), and runtime *object-state* to explicitly track object lifecycles and evolution as defined in Section 4.
2) **SOA-Aware Message.** Messages include SOA-specific fields *(Interface_ID.OperationName, InvID, RQ/RS/SE)* to model distributed service interactions.
3) **Full UML Compatibility.** All standard UML interaction fragments are preserved unchanged [16], ensuring compatibility with standard UML tools.
4) **Dynamic Object Support.** The *selector* and *object-state* constructs support grouping equivalent object instances and tracking state changes, aligned with dynamic object equivalence in Section 4.

```
OCSD   ::=  'sd'  ocsd-name  '{'  lifeline+  message+
     interaction-fragment* '}'

-- 1. Object-Centric Lifeline (UML Extension)
ocsd-lifeline-ident ::= [ object-id '[' selector ']' ] ':'
object-type [ '{' object-state '}' ] | 'self'
object-id                ::= identifier | PayloadID '.'
AttributeID
object-type              ::= AName | structural-type |
primitive-type
structural-type     ::= 'struct' '{' AName '}'
primitive-type       ::= 'string' | 'number' | 'boolean' |
'date'
object-state        ::= attribute (',' attribute)*
attribute           ::= AttributeName '=' AttributeValue
selector             ::= expression | instance-set | type-
set
instance-set        ::= '{' object-id (',' object-id)* '}'
type-set             ::= '{' object-type (',' object-type)*
'}'

-- 2. Object-Centric & SOA Message
message              ::= sender '->' receiver ':' message-
label [ '{' message-constraint '}' ]
sender              ::= ocsd-lifeline-ident
receiver            ::= ocsd-lifeline-ident
message-label       ::= activity '(' object-list ')'
activity            ::= Interface_ID '.' OperationName
object-list         ::= object-id (',' object-id)*
message-constraint    ::=  action-type  ','  InvID  (','
latency)?
action-type         ::= 'RQ' | 'RS' | 'SE'
latency             ::= duration 'ms'
duration            ::= number
```

```
-- 3. Standard UML Interaction Fragments
interaction-fragment  ::=  execution-spec  |  state-
invariant  |  interaction-use  |  gate  |  destruction-
occurrence
execution-spec          ::= 'exec' ocsd-lifeline-ident
':' activity [ '(' start-time ',' end-time ')' ]
state-invariant          ::=  ocsd-lifeline-ident  '('
constraint ')'
constraint            ::= boolean-expression | object-
state-equality
object-state-equality::= 'state' '=' object-state
interaction-use         ::= 'ref' interaction-ident [
'strict' ]
gate                  ::= 'gate' direction message-name
direction             ::= 'in' | 'out'
destruction-occurrence  ::=  'destroy'  ocsd-lifeline-
ident

-- 4. Basic Terminals
identifier      ::= [a-zA-Z_] [a-zA-Z0-9_]*
number          ::= [0-9]+
string          ::= '"' [^"]* '"'
expression       ::= identifier  |  number  |  string  |
boolean | function-call
boolean         ::= 'true' | 'false'
ocsd-name       ::= identifier
interaction-ident ::= identifier
start-time      ::= timestamp
end-time        ::= timestamp
timestamp       ::= ISO8601-datetime
```

Fig. 4. OCSD Grammar Definition.

6. Software Implementation Details

Having established the theoretical underpinnings of object-centric sequence diagram (OCSD) generation, in this section we propose the software implementation details of the proposed OCSD visualization approach. The tool is realized as a Python-based prototype that processes the relational event log and generates human-interpretable UML-style sequence diagrams. Critically, the prototype is designed to address the core challenge of SOA event analysis: tracking dynamics, equivalent object instances across distributed service interactions, while preserving temporal and contextual details absent in existing OCEL visualizations.

The prototype supports four distinct visualization modes that demonstrate analysis from the single object instance, inter-instance interaction, single object type, inter-type interaction perspectives. Specifically:

- **Single Object Instance.** Visualize the full lifecycle of a single object instance (Definition 3) across SOA service events, including its associated InvID cases and request-response (RQ/RS) pairs.
- **Object Instance Interaction.** Analyze temporal and structural interactions between multiple distinct object instances, with annotations for inter-instance object counts and service interaction details.
- **Single Object Type.** Aggregate lifecycle behavior for a specific object type (Definition 5) using core attributes and equivalent dynamic object instances (per Definition 6).
- **Object Type Interaction.** Visualize aggregated interactions between multiple object types, with metrics computed across all equivalent instances of each selected type and support for inter-case service interaction analysis.

Each mode is based on the four core algorithms *(DetermineCoreAttributes, GroupEquivalentObjects, CollectCaseEventsForObject, PairRequestsResponses)* and tailored to the SOA event log structure and object equivalence rules defined in Section 4.

6.1 Algorithmic Core Integration

Key algorithmic adaptations include:

1) **DetermineCoreAttributes.** Computes core attributes for an object type (AName) by intersecting the attribute sets of all structural object instances of that type, excluding embedded sub-objects ($ParentID \neq -1$).
2) **GroupEquivalentObjects.** Generates serialized immutable attribute fingerprints for dynamic object instances (per Definition 6) using domain-expert specified immutable attributes for (activity, object type) pairs. Equality checks exclude embedded sub-objects and only compare scalar attributes of structural objects.
3) **CollectCaseEventsForObject.** Extracts events for equivalent object instances by filtering the events for matching case ID and object values, and retains all events in a case with a $timestamp \leq$ the target object's event timestamp.
4) **PairRequestsResponses.** Matches RQ/RS events by activity and case ID, with unpaired RS events. This is critical for SOA systems where service requests/responses may be distributed across log entries.

6.2 Single Object Instance Mode

This mode generates a OCSD for a single structural object instance, capturing the instance's full lifecycle across SOA service events. It is designed to answer SOA-specific questions such as: *What is the lifetime of a specific object? How frequently a specific object is modified?*
and etc..

The single object instance mode processes a single structural object instance through five core steps. First, users select a (PID, AID) instance from the PayloadObject table (constrained to $AValue = NULL, ParentID = -1$) and provide domain-expert defined immutable attributes. The *GroupEquivalentObjects* algorithm then returns a singleton group for the instance (its serialized fingerprint is the (PID, AID) tuple), aligning with the prototype's unified pipeline. Next, *CollectCaseEventsForObject* filters the TracingEvent table for events linked to the instance's PayloadID and InvID cases, retaining the first event of each case and all prior events (sorted by EventTimestamp). The *PairRequestsResponses* algorithm matches RQ/RS pairs by OperationName and InvID, calculating latency as the time difference between RQ and RS EventTimestamp values. Finally, the prototype generates a sequence diagram annotated with chronological SOA service events (RQ/RS event IDs pairs on the nodes), OperationName and object instance in parentheses above the arrow if such appeared directly in the observed event.

In this mode we obtain an OCSD of the object instance's SOA service interaction lifecycle (Fig. 5), enabling debugging of individual object-service interactions in SOA systems.

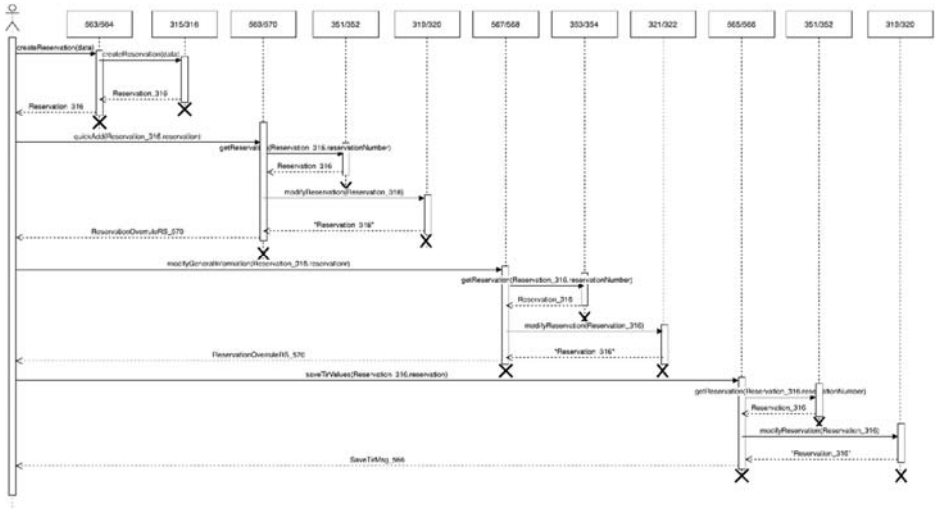


Fig. 5. Fragment of the OCSD for object ReservationMsg_316.

6.3 Object Instances Interaction Mode

This mode generates an OCSD for interactions between multiple distinct structural object instances (each (PID, AID) tuple), building on the single object instance mode and adding SOA-specific inter-instance metrics. It addresses questions such as: *When is the invoice created for a specific booking? Which objects of the booking were modified and how before the final payment?*

When the workflow is similar to the single object instance analysis, the main difference here will be adding a main object instance for focal observation and collecting data from the preprocessed tables for several object identifiers (PID, AID).

A multi-instance OCSD (Fig. 6) enables analysis of object interactions across distributed SOA services and cases.

6.4 Single Object Type Mode

This mode generates an OCSD for a single object type, aggregating lifecycle behavior across all equivalent dynamic object instances of that type.

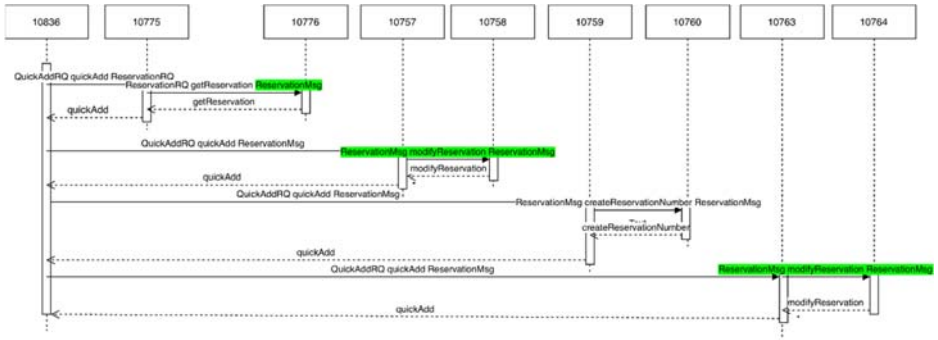


Fig. 6. OCSD for objects interaction of ReservationMsg_316 and QuickAddRQ_10836.

It is designed for high-level SOA analysis questions such as: *What is the all-shared behaviour of the objects for this type? What kind of behaviour can be expected, what should be marked if observed?* First, users select an object type (AName) from the PayloadObject table and provide domain-expert defined (activity, object type) immutable attributes (Definition 6). The *DetermineCoreAttributes* algorithm then identifies core attributes for the type by intersecting the attribute sets of all structural object instances with the target AName. On the next step *GroupEquivalentObjects* generates serialized fingerprints for all dynamic instances of the type (using the immutable attributes) and groups equivalent instances (Definition 4 and Assumption 1). For each equivalent group, CollectCaseEventsForObject extracts case events from the TracingEvent table. The resulting OCSD for the object type (Fig. 7) enables monitoring of aggregate object type performance and its variations in distributed SOA systems, thus, promptly inspect anomalies in object behaviour.

6.5 Object Types Interaction Mode

This mode generates an OCSD for interactions between multiple object types (AName values). It addresses high-level analysis questions such as: *What is the expected interaction behaviour between selected object types? Which intermediate object's behaviour is unpredictable in terms of first having a booking and obtaining a payment from it? Is the high-level service logic correct in terms of objects interaction? and etc.*

The process of diagram generation is similar to the single object type one with addition of the main object type selection step. An aggregated multi-type OCSD presents analysis of cross-type object dependencies in SOA systems and identification of bottlenecks in object flow between services.

7. Conclusion

This work presents a novel object-centric sequence diagram (OCSD) visualization approach for Service-Oriented Architecture (SOA) systems, bridging the gap between Object-Centric Process Mining (OCPM) theory and software engineering practice. Building on formal definitions of structural/equivalent/dynamic objects and a transformed SOA event log schema (Section 3), the work makes three core contributions to OCPM and SOA system analysis, addressing key limitations of classical process mining and existing OCEL visualizations:

- **Formal OCSD Definition for SOA.** We define the OCSD as a UML-style sequence diagram that preserves contextual details of service-object interactions, which is a critical improvement over existing OCEL visualization approaches. The OCSD is grounded in the SOA event log structure and formal object equivalence rules (Definitions 4,6; Assumption 1), enabling precise tracking of dynamic object instances across distributed SOA services and cases (InvID).

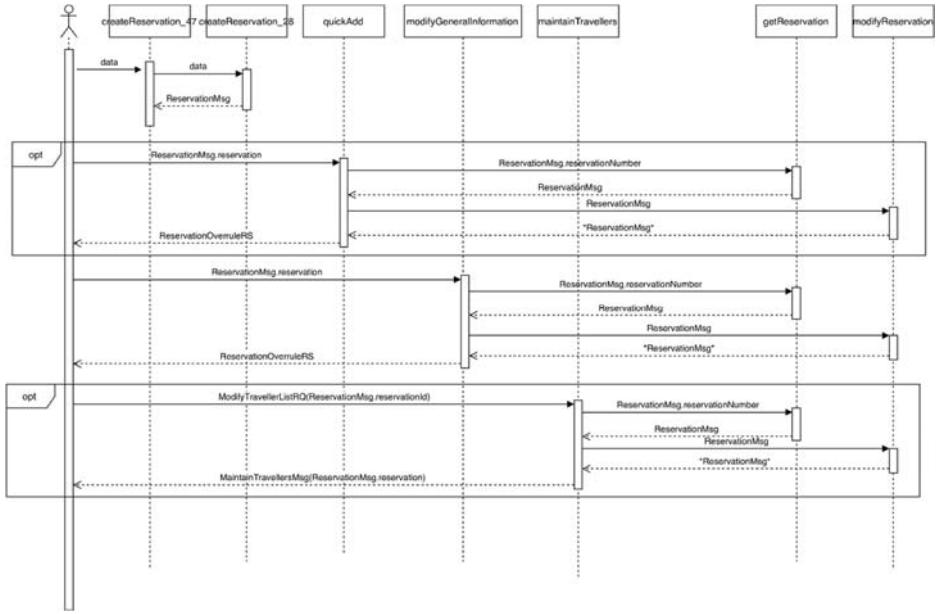


Fig. 7. OCSD for object type ReservationMsg.

- **Four-Mode Visualization Prototype.** We implement the OCSD approach as a Python-based prototype with four consistent visualization modes (Single Object Instance, Object Instance Interaction, Single Object Type, Object Type Interaction). The prototype enforces critical object analysis rules (exclusion of embedded sub-objects from top-level equality checks, domain-expert defined mutable/immutable attributes for dynamic objects) and generates OCSDs that are both human-interpretable and machine-processable.
- **SOA-Specific OCPM Analysis Capabilities.** The work extends classical process mining by integrating payload-extracted object attributes into SOA system analysis. The OCSD approach reveals hidden SOA service interaction patterns (e.g., redundant object accesses, temporal bottlenecks in RQ/RS pairs, cross-type object dependencies) that are invisible in flateventlog analysis. A real-world runningexample demonstrates that the OCSD prototype enables practitioners to diagnose inefficiencies in distributed service-object interactions and optimize SOA system performance.

In summary, this work addresses the core limitation of classical process mining for SOA systems. The OCSD approach turns object-centric event data into actionable insights for SOA system engineers, enabling data-driven diagnosis and optimization of distributed service-object interactions.

References

[1]. Cui Y., Zada D.-I., Shahzad S., Nazir S., Khan S., Hussain N., Asshad M. Analysis of service-oriented architecture and scrum software development approach for IIoT. Scientific Programming, 2021, pp. 1-14. DOI:10.1155/2021/6611407.

[2]. IEEE Task Force on Process Mining. Process mining manifesto. In BPM 2011 Workshops, 2011. DOI: 10.1007/978-3-642-28108-2_19.

[3]. van der Aalst W.M.P. Object-Centric Process Mining: Unraveling the Fabric of Real Processes. Springer, 2023. DOI: 10.3390/math11122691.

[4]. Berti A., van der Aalst W.M.P. OC-PM: Analyzing object-centric event logs and process models. In Proceedings of the ICPM 2022, 2022. DOI:10.1007/s10009-022-00668-w.
[5]. Khayatbashi S., Miri N., Jalali A. Advancing object-centric process mining with multi-dimensional data operations. arXiv preprint, 2025. DOI:10.21203/rs.3.rs-8444775/v1.
[6]. Shershakov S.A., Rubin V.A. System runs analysis with process mining. Modeling and Analysis of Information Systems, 2015, vol. 22, pp. 818-833. DOI: 10.18255/1818-1015-2015-6-818-833.
[7]. van der Aalst W.M.P., Berti A. Discovering object-centric petri nets, 2020. DOI: 10.3233/STAL200004.
[8]. Peeva V., Porsil M., van der Aalst W.M.P.. Object-centric local process models. pp. 376-388.
[9]. van der Aalst W.M.P., Li G., Montali M. Object-centric behavioral constraints, 2019, pp. 48-56. DOI: 10.1145/3297280.3297287.
[10]. Xiu B., Guannan Li, and Yixiang Li. Discovery of object-centric behavioral constraint models with noise. IEEE Access, 2022, pp. 88769-88786. DOI: 10.1109/ACCESS.2022.3199345.
[11]. Park G., van der Aalst W.M.P. Monitoring constraints in business processes using object-centric constraint graphs. pages 479-492, 2023. DOI:10.1007/978-3-031-27815-0_35.
[12]. Davydova K.V., Shershakov S.A. Mining hybrid UML models from event logs of SOA systems. Proceedings of the Institute for System Programming of RAS, 29(4):155–174, 2017. DOI: 10.15514/ISPRAS-2017-29(4)-10.
[13]. Shershakov S.A. Multi-perspective process mining with embedding configurations into db-based event logs. In Proceedings of the 2021 International Conference on Process Mining, 2021. DOI: 10.1007/978-3-030-71472-7_5.
[14]. Kabierski M., Basmer M., Patecka A., Sahling K., Bala S. Data quality in object-centric event data: Issues classification and evaluation. In Proceedings of the 2025 International Conference on Process Mining, 2025. DOI: 10.1007/978-3-031-82225-4_23.
[15]. Backus J.W., Naur P., et al. Revised report on the algorithmic language ALGOL 60. Communications of the ACM, 6(1):1–17, 1963. DOI: 10.1145/366193.366201.
[16]. UML Diagrams. UML sequence diagrams. https://www.uml-diagrams.org/sequence-diagrams.html, accessed: 10.03.2026.

Информация об авторах / Information about authors

Сергей Андреевич ШЕРШАКОВ – кандидат компьютерных наук, доцент, академический руководитель магистерской программы «Системная и программная инженерия» факультета компьютерных наук НИУ ВШЭ. Сфера научных интересов: извлечение и анализ процессов, встраиваемые системы, объектно-ориентированное программирование, Software Process Mining, сервис-ориентированная архитектура (СОА).

Sergey Andreevich SHERSHAKOV – Cand. Sci. (Comp. Sci.), Assoc. Prof., Academic Supervisor of the Master Programme "System and Software Engineering" on Faculty of Computer Science, HSE University. Research interests: process mining, data mining, embedded systems, object-oriented programming, Software Process Mining, SOA.

Цзиньюй ЧЖОУ – выпускник факультета компьютерных наук НИУ ВШЭ. Сфера научных интересов: анализ процессов, визуализация, анализ систем на основе сервис-ориентированной архитектуры (СОА).

Jinyu ZHOU – graduate student, Faculty of Computer Science, HSE University. Research interests: process mining, visualization, SOA system analysis.