Chapter 6

State Consistency – A Matter of Conceptual Modeling?

Elmar J. Sinz

Most conceptual models are expressed as data schemes, some as process schemes and only a few in other modeling languages. Modeling of state consistency, i. e., the goal-oriented alignment of the states of system components does not seem to play a role. However, the more the business world becomes distributed, cooperative, and volatile, this matter gets increasingly important. Has conceptual modeling a 'blind spot' here? The article shows the need for conceptual state consistency modeling. It is based on the concept of compensating transactions, borrowed from data management. Modeling of state consistency is demonstrated using the SOM approach.

6.1 Conceptual modeling and its 'blind spot'

Conceptual modeling has been an important topic in information systems engineering for decades (Frank et al. 2014). In particular, modeling of business reality is used here as a tool for analyzing and designing tasks and resources. The adjective conceptual indicates, on the one hand, that the focus here is on structures that are stable over the longer term and, on the other hand, that these can be seen independently of technical details.[1]. This allows to look at a conceptual model from two perspectives, a domain perspective, and an implementation perspective (Sinz 2021). The domain-oriented perspective describes the real-world context, while the implementation-oriented perspective describes its realization with IT.

Most conceptual models are expressed as data schemes; they capture aspects of the structure of a system. Sometimes process schemes are used, which describe behavioral properties. But what about state consistency, i. e., the goal-oriented alignment of the states of components of a distributed system? State integrity constraints (Schlageter and Stucky 1983, p. 291), formulated in a data schema by means of cardinalities of relationships, only capture the states within the respective system component, but not the overarching states of a distributed and volatile system. Does conceptual modeling have a 'blind spot' here?

One of the long-term development lines of our world is that production of goods and services increasingly takes place in distributed systems. Application systems (IT systems) have changed from monolithic blocks to highly distributed systems, consisting of a multitude of interacting components. The same is true for business operations. Hardly a production process is carried out by a single company; rather many larger and smaller companies are

1 https://www.umo.wiwi.uni-due.de/forschung/forschungsgebiete/konzeptuelle-modellierung/

usually involved. They cooperate toward the goal of producing goods and services. And they are volatile, as companies change over time. Business activity increasingly takes place in so-called ecosystems.

So, why an explicit consideration of state consistency? A simple example is the reconciliation of the financial account of a bank customer with that of the bank. Discrepancies between the two indicate an error, i. e., a lack of consistency of corresponding states, that must be resolved in the interest of both parties. Another example is the matching of an order sent by a customer with the order received by the corresponding supplier. Here, avoiding discrepancies is a prerequisite for smooth business operations.

Other state consistency problems arise at the interface between the service area and the payment area of an IT system. The author himself has experienced the following examples: A ticket vending machine for bus and streetcar tickets collects the money but does not issue a ticket. At a gas station, the transaction is aborted when trying to pay by credit card. The payment is switched to cash, but the credit card is still charged. In both cases, there is obviously a problem of state consistency.

6.2 Scope and understanding of state consistency

For further considerations, the *scope of state consistency* is looked at. The following cases can be distinguished (Sinz 2014):

1. *Business task*. Such a task is, for example, creating and sending a purchase order. State consistency ensures that this task is done completely and is not aborted.
2. *Business transaction.* A business transaction is performed by multiple tasks. To continue the example, the customer's order must be received and correctly processed by the supplier.
3. *Business object*. A business object consists of several tasks. Their attribute values must be consistent with each other. So, the received order must be added to the stock of open orders, otherwise the state of the object will be inconsistent.
4. *Business system.* A business system consists of several interacting business objects. For example, an open order must be passed correctly from the sales object to the production planning object.

The scope of state consistency ranges from a single task to a transaction, an object, and a system. While (1) is typical for a simple database transaction in an application system, (4) expresses the scope focused here. In distributed and cooperative business systems with volatile components, this scope goes far beyond that of conventional database transactions.

Another issue is the *understanding of system consistency.* Can an enterprise that has undelivered orders or unpaid invoices be consistent with respect to its states? If temporary (intermediate) states are to be considered consistent, open items have to be introduced and managed.

If we look at the main interactions of a company on the sales and procurement side as well as the flow of goods/services and the flow of payments, there are four open items: open purchase orders, open delivery orders, receivable items and liability items. These open items are generally accepted in modeling because they pragmatically describe a working company. Therefore, a receivable item for a specific customer does not violate state consistency. On the other hand, it is violated if a purchase order is fulfilled and no receivable item is created.

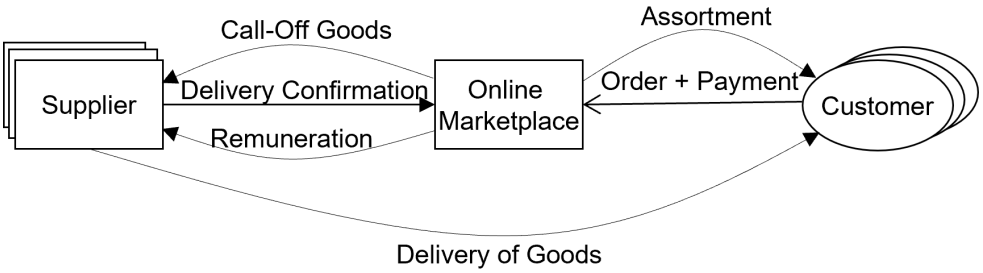

Figure 6.1: *Online marketplace*

6.3 When is modeling of state consistency important?

Modeling of state consistency is not important for all business systems. As mentioned above, there are three characteristics of a business system that, taken together, draw attention to state consistency modeling.

- *Distributed*: A distributed system (Enslow 1978) consists of several subsystems, each of them is autonomous. The subsystems act by themselves, they cannot be controlled from the outside. All subsystems are subject to the goals of the system.
- *Cooperative*: Subsystems work together according to a specific protocol to achieve the goals of the system.
- *Volatile*: Subsystems can change the availability of their services. They can even leave the system while other subsystems come in.

The assumption, that a system with these characteristics always works as planned, cannot be sustained. Deviations from the planned behavior will rather be the normal case. However, if a subsystem cannot provide its service as planned, the state consistency of the overall system is at risk.

The modeling approach required here is an extension of behavioral modeling. It must ask the question, which functions on the objects are necessary to enable state consistency of the entire business system? And what are the processes to restore state consistency to the system when an object does not behave as intended? Since these questions can only be answered from the domain-oriented perspective, they are a matter of conceptual modeling.

An example is an *online marketplace* that presents an assortment for a variety of goods of different suppliers (Figure 6.1). It accepts orders for suppliers and receives the payments from the customers. The orders are passed from the marketplace to the respective supplier, who delivers the ordered goods to the customer and confirms this to the marketplace. The marketplace then remunerates the supplier.

What if the payment goes from customer to the marketplace, but the goods are not delivered by the supplier, maybe they are out of stock, or the supplier has meanwhile left the system? In this case, it is foreseen that the money will be refunded by the marketplace in the same way as the payment was made. Then, the necessary functions of the objects as well as the process must be provided.

The case can be extended. What if the refund does not work, for example, because the customer's credit card has meanwhile expired? In this case, it may be necessary to

consult with the customer to determine an alternative payment method. Again, functions and process have to be provided too.

In summary, conceptual modeling of state consistency aims to describe distributed, cooperative, and volatile business systems with respect to the domain-oriented perspective. The modeling of the real-world context will help to better understand these business systems and develop better IT systems to support them.

6.4 Methodological foundation – A borrowing from database transactions

The tool for ensuring state consistency in IT is the concept of transactions[2] known from database systems. A transaction follows the ACID principle (Härder and Reuter 1983), it is

- *atomic* (A), i. e., it is executed completely or not at all,
- *consistent* (C), it satisfies the given integrity constraints,
- *isolated* (I), parallel user orders do not influence each other, and
- *durable* (D), the effect of a completed transaction is persistent.

The realization of the properties A, I and D is based on the concept of sphere of control (Gray and Reuter 1993, p. 174ff.), i. e., the set of states on which the transaction operates. The sphere of control builds up as the transaction is executed and disappears after the transaction becomes persistent.

A simple (flat) transaction is, for example, the creation of a sales order in a central, database-driven application system. Here, the customer data, the article data and the created order enter the sphere of control one after the other. If an error occurs during this process, the transaction is reset (abort transaction), i. e., it is undone. As soon as it is completed and error-free, it becomes persistent (commit transaction) and can therefore no longer be reset.

But what if the transaction is executed in a distributed system with correspondingly distributed application systems? Because of the autonomy and volatility of the subsystems, concepts such as distributed database systems are not applicable. The supplier who takes the specific customer order only once, wants to keep his IT equipment autonomous. He wants not to get involved in interface agreements. Possibly he wants to use no IT support at all.

The sphere of control of the global transaction (related to the entire business system) thus breaks down into several sub-parts (each related to a subsystem) that can become isolated persistent. To still be able to preserve the ACID property of the global transaction, the usage of compensating transactions is necessary (Gray and Reuter 1993, p. 204ff.). These are installed during the execution of a partial transaction. They are triggered if the effect of the partial transaction, although becoming persistent, must be compensated. This means, a partial transaction (DO) is compensated (UNDO) and, if necessary, replaced by another partial transaction (REDO) (Gray and Reuter 1993, p. 538ff.).

The difference between resetting and compensating transactions is explained by an example: With an accounting program a payment of € 820.00 must be booked. However, the amount turns out to be wrong, the correct amount is € 795.00. If the error is corrected before

2 The term transaction is used homonymous in literature. It denotes, on the one hand, a coordinated transfer of an information package or of goods/services (business transaction). On the other hand, it stands for a set of operations following the ACID principle (database transaction).

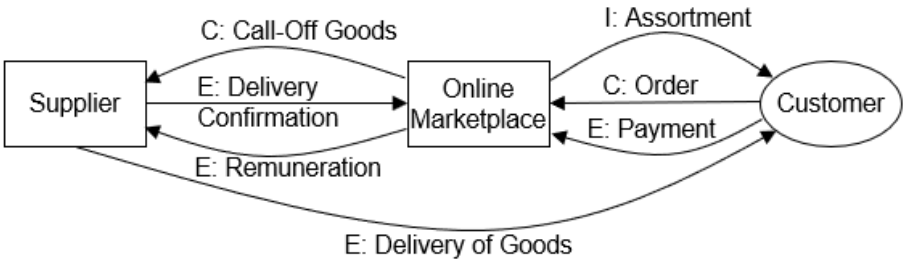

Figure 6.2: Structural view of *online marketplace* using SOM

the transaction has become persistent, it can be reset. Nothing is visible in the accounts or in the general ledger. However, if the error is discovered after the transaction has become persistent (DO), it will be compensated by a cancellation booking (UNDO) and replaced by a correct booking (REDO). The semantic effect of the erroneous booking is thus cancelled, but the error remains visible in the accounts and in the general ledger.

The previous considerations apply to database transactions. But what if you consider real-world transactions. In the example of the *online marketplace*, this is the payment that has already been collected and now must be refunded. In many cases, however, it is not so easy to specify the compensating transaction. As an example, imagine a machining center that performs several machining steps on a workpiece on a single machine. What to do if a hole is already drilled (DO) and then the transaction is to be compensated (UNDO)? Should the hole be filled again or is the workpiece a reject? This question cannot be answered from an IT perspective, but only from a domain-oriented perspective. It is therefore a matter of conceptual modeling.

Another example: A passenger has booked a connecting flight, shows up at the gate, but the plane has already departed. Should the contract be fulfilled by rebooking the passenger on another flight or should the contract be cancelled? Both alternatives are probably associated with costs. Since the case occurs more frequently and can become arbitrarily complex, it is analysed and supported by a conceptual model.

Summarizing, the borrowing from database transactions is the concept of compensating transactions, adopted and extended to real-world transactions.

6.5 An example using the Semantic Object Model

The Semantic Object Model (SOM) is a comprehensive approach to modeling business systems (Ferstl and Sinz 1995; Ferstl and Sinz 2006; Ferstl and Sinz 2013). Business process models are an important part of SOM. A business process model is represented in two views, a structural view, and a behavioral view.

The *structural view* shows the business objects and their relationships (Figure 6.2). For system delimitation, the objects are divided into discourse world objects and environmental objects. The relationships between the objects are business transactions that coordinate the objects. Two forms of coordination are distinguished: non-hierarchical coordination according to the ICE principle (initiating, contracting, and enforcing transaction) and hierarchical coordination according to the CF principle (control and feedback transaction).

Objects and transactions can be refined hierarchically. Objects can be decomposed into sub-objects. The same applies to transactions. For example, an enforcing transaction (E) can be decomposed into an initiating (I), a contracting (C), and an enforcing (E) sub-transaction according to non-hierarchical coordination. During the initiating sub-transaction, the objects exchange information on the goods or services to be delivered. The objects conclude a contract while doing the contracting sub-transaction. They exchange the goods or services while running the enforcing sub-transaction. All this is done with respect to the goal of the global enforcing transaction.

The *behavioral view* shows the tasks assigned to business objects and their relationships (Fig. 6.3). All tasks of an object are based on one and the same object-internal memory. Task relationships between objects correspond to business transactions. The relationship of tasks within an object are based on object-internal events. The behavioral view is basically based on the Petri net concept (Reisig 2010), whereby this is extended and interpreted semantically.

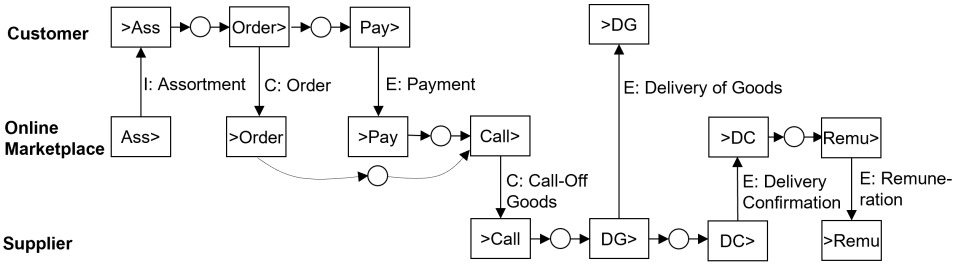

Figure 6.3: Behavioral view of *online marketplace* using SOM

The Semantic Object Model has several features that are useful for modeling the state consistency of business systems:

- Business processes with structural and behavioral views: In the behavioral view the tasks are assigned to objects.
- Binary business transactions: Business transactions are executed by the tasks of two business objects, thus building one database transaction.
- Decomposition of objects and transactions: If an *E* transaction is decomposed according to the ICE principle, the semantics of the whole transaction is transferred to its parts.

Figure 6.4 shows the behavioral view of the *online marketplace* with spheres of control and compensating transactions for the assumed case that a supplier cannot perform the delivery of goods (symbol lightning). Hence, the state consistency of the business system is violated. The handling of the exception, which is an error from the perspective of the whole business system, is now explained:

- At the time of the error, the spheres of control of the objects *customer*, *online marketplace* and *supplier* are partially established (highlighted in light grey), but not completed. They must therefore be reset.
- The tasks that have already become persistent are shown with thick border. Their spheres of control, which were embedded in the global sphere of control, have disappeared. These tasks must be compensated.

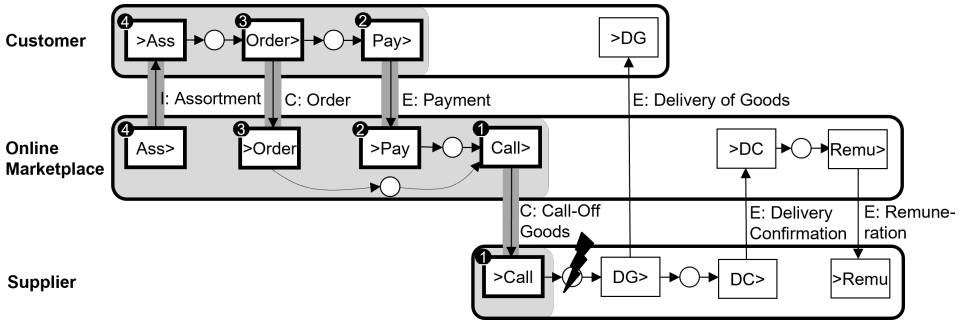

Figure 6.4: *Online marketplace* with spheres of control and compensating transactions

- However, each task of a business object performs a business transaction with a task of another object. So, the two tasks can only be compensated together, they are coupled (dark grey bar between two tasks). For example, the compensation of *>Call* in the object *supplier* also causes the compensation of *Call>* in the object *online marketplace* (step 1).
- The compensation of *Call>* triggered in the object of *online marketplace* leads to further compensation steps here (steps 2 and 3).
- The initiating transaction (step 4) was not considered necessary for compensation, because neither of the two objects entered into an agreement. Its treatment serves the continuation of the process.

In total, four steps are necessary to handle the error 'supplier cannot perform delivery of goods', involving eight tasks (see Table 6.1). Three of these steps are compensations (UNDO), one is used to restart the system (REDO). After these tasks are executed, the state consistency of the system is re-fulfilled, and the continuation of the business activity is initiated.

These tasks were revealed by conceptual modeling of state consistency. It concerns conceptual modeling in the sense used here because the semantics can only be derived from the domain-oriented perspective. The modeling approach and its borrowing from database transactions focus the modeler's attention on the relevant issues.

Step	Task	Description of the task	Type
1	>Call	Reject of call-off goods	UNDO
1	Call>	Register the rejection of call-off goods	UNDO
2	>Pay	Refund of payment to customer	UNDO
2	Pay>	Register the refund	UNDO
3	>Order	Reject of sales order	UNDO
3	Order>	Register the rejection of sales order	UNDO
4	Ass>	Submit an alternative offer to the customer	REDO
4	>Ass	Register the alternative offer	REDO

Table 6.1: Restoration of state consistency after 'supplier cannot perform delivery of goods'

6.6 Conceptual modeling of state consistency – Lessons learned

In general, a conceptual model serves to identify features of a system that would not have been recognized, or not as clearly, without a model. Conceptual modeling of state consistency is about identifying and embedding those functions in process flows that are required to restore the global state consistency of the system when a subsystem has not behaved as planned. If the UNDO function is required, then the model must say how. This is especially important for distributed, cooperative, and volatile systems.

In addition to these execution time aspects, conceptual modeling of state consistency is also helpful at design time. An example is the execution of a travel booking, consisting of flight, car rental and hotel booking. The trip is only offered in full or not at all. In this case the system design is supported by the expected risk for unavailability of a travel component and the cost of rebooking (UNDO/REDO) of already booked components.

Overall, it is expected that through the conceptual modeling of state consistency, a better understanding and more systematic investigation of the business side of a system is achievable. This leads incidentally to specifications of more robust and complete IT systems. Since the modeling can only be done considering the domain-oriented perspective, it is a matter of conceptual modeling.

References

Enslow, P. H. (1978). 'What is a "Distributed" Data Processing System?' In: *IEEE Computer* 11.1, pp. 13–21.

Ferstl, O. K. and Sinz, E. J. (1995). 'Der Ansatz des semantischen Objektmodells (SOM) zur Modellierung von Geschäftsprozessen'. In: *WIRTSCHAFTSINFORMATIK* 37.3, pp. 209–220.

Ferstl, O. K. and Sinz, E. J. (2006). 'Modeling of Business Systems Using SOM'. In: *Handbook on Architectures of Information Systems*. Ed. by P. Bernus, J. Blazewics, G. Schmidt, M. Shaw, P. Bernus, K. Mertins and G. Schmidt. International Handbooks on Information Systems. Berlin, Heidelberg: Springer, pp. 347–368.

Ferstl, O. K. and Sinz, E. J. (2013). *Grundlagen der Wirtschaftsinformatik*. 7th ed. Berlin: De Gruyter Oldenbourg.

Frank, U., Strecker, S., Fettke, P., Brocke, J. vom, Becker, J. and Sinz, E. (2014). 'The Research Field "Modeling Business Information Systems" (Research Note)'. In: *Business & Information Systems Engineering* 6.1, pp. 39–43.

Gray, J. and Reuter, A. (1993). *Transaction Processing: Concepts and Techniques*. San Mateo, CA: Morgan Kaufmann.

Härder, T. and Reuter, A. (1983). 'Principles of Transaction-Oriented Database Recovery'. In: *ACM Computing Survey* 15.4, pp. 287–317.

Reisig, W. (2010). *Petri-Netze. Modellierungstechnik, Analysemethoden, Fallstudien*. Wiesbaden: Vieweg und Teubner.

Schlageter, G. and Stucky, W. (1983). *Datenbanksysteme: Konzepte und Modelle*. 2., neubearb. und erw. Aufl. Teubner-Studienbücher. Stuttgart: Teubner.

Sinz, E. J. (2014). 'Konzeptuelle Modellierung der Zustandskonsistenz verteilter betrieblicher Informationssysteme'. In: *Modellierung 2014, 19.-21. März 2014, Wien, Österreich*. Ed. by H. Fill, D. Karagiannis and U. Reimer. Vol. P-225. LNI. GI, pp. 241–256.

Sinz, E. J. (2021). 'Five questions to be clarified before starting to model conceptually'. In: *EMISAJ – International Journal of Conceptual Modelling* 16, 3:1–3:4.

Acknowledgement

Thanks to Felix Härer (University of Fribourg, Switzerland) and to Dominik Bork (Technical University Vienna) for reading and commenting on an earlier version of this article.