Integrity in Very Large Information Systems
Dealing with Information Risk Black Swans

Beat Liver and Helmut Kaufmann

Credit Suisse Information Technology, Zurich, Switzerland
beat.liver@credit-suisse.com, helmut.l.kaufmann@gmail.com

Abstract. Multi-national enterprises, like financial services companies, operate large and critical information systems around the globe on a 24/7 basis. In an information-based business, even a single inadequately designed, implemented, tested and operated business application can put the existence of the enterprise at risk.

For adequately securing the integrity of business critical information and hence ensuring that such information is *meaningful, accurate* and *timely*, we present our risk assessment and controls framework: First, we introduce our criticality rating scheme that is based on the recoverability from integrity failures. For dealing with dependencies among applications, we present our approach based on services given a Service-Oriented Architecture (SOA). Second, we provide an overview of our design-related controls including a data analytics approach to continuously audit the most critical information assets. Finally, we present our learnings from a first implementation of the presented framework.

Keywords: Information risk management; integrity; business critical systems; data analytics; Service-Oriented Architecture.

1 Introduction

Information security in general aims at protecting the confidentiality, integrity and availability of information. Integrity protection is mainly understood as the assurance that all data modifications are *authorized* [6]. Our understanding is broader: Integrity also requires that information is *meaningful*, *accurate* and *timely* and that it is modified only in acceptable ways (see also [8,9]). In the financial industry, severe integrity failures are rare, but have already resulted in material harm to companies and even the financial markets. This is illustrated by the following real-world examples:

- A trading software bug generated wrong market orders resulting in a loss of 440 million USD within 30 minutes [13].
- After a software change, a payment order processing batch failed. Sorting out and restoring operations took several weeks [14].
- A trader inadvertently entered an order to sell 610'000 shares at 16 Yen a piece instead of 16 shares at 610'000 Yen. The resulting order was partially canceled. However, it resulted in a loss of up to 100 million USD [15].

C. Salinesi, M.C. Norrie, and O. Pastor (Eds.): CAiSE 2013, LNCS 7908, pp. 641–656, 2013.

The root causes for the above failures cannot be determined impeccably from the available public sources. In the last example, a plausbile explanation is a combination of human error, bad design and testing. The point these examples illustrate well, is that integrity failures can have disastrous consequences.

In the above examples, material financial losses resulted from mis-processing by business critical systems. The recent accumulation of similar incidents caught the attention of the regulators and triggered a public discussion on integrity failures and their root causes, such as (i) systems' ages, complexity and technology; (ii) risk management issues, such as underestimated risks as well as inadequate and ineffective controls; and (iii) impact of organizational and economic factors, especially outsourcing/off-shoring and cost pressure [20].

Generally, risk-related standards require that *business critical systems must be adequately protected.* While people tend to agree that business critical systems must be safeguarded, they often fail in identifying these systems. And even if truly critical systems are identified, you are almost guaranteed to find yourself in a discussion around the term *adequate safeguards*: If you are not doing enough – whatever *enough* means – an organization can be seriously harmed. If you are doing too much, an organization's resources are wasted.

This paper presents Credit Suisse's approach to safeguard the integrity of very large information systems, of which many are considered to be business critical. Figure 1 illustrates our overall information risk management framework and highlights the scope of this paper. Section 2 introduces our easy-to-understand

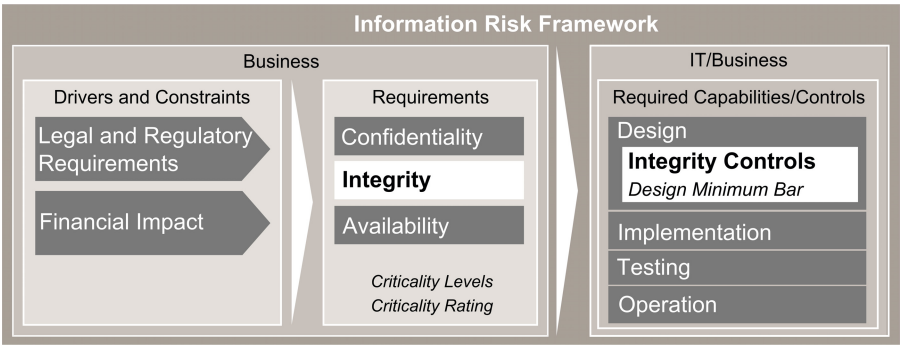

Fig. 1. Information Risk Framework Overview

rating scheme to correctly determine the integrity criticality of systems and services. Section 3 summarizes our so-called *minimum bar for integrity*, a minimum standard regarding integrity design controls, which ensure risk-adjusted safeguarding of systems, i.e., commensurate to a system's criticality. We briefly discuss related work in Section 4. Section 5 summarizes our learnings from a first enterprise-wide compliance assessment against these standards as well as a proof of concept for independent integrity controls.

To illustrate our approach, we use foreign exchange orders throughout the paper: A client buys 100'000 USD against CHF at an exchange rate of 0.9228 USD/CHF on November 6th, 2012 at 08:26 UTC. This so-called spot trade results in two settlement payments on November 8th: The client receives 100'000 USD from the bank and pays 92'280 CHF to the bank.

2 Integrity Criticality Rating

In traditional information security, a system's *criticality* corresponds to the impact of confidentially breaches, loss of integrity and availability [6,8]. Often, there is a fixed set of 3-5 criticality levels, ranging from 'non-critical' to 'business critical'. In our rating scheme,

- information systems are called *business critical* if a single failure might put a firm's existence at risk. In a regulated financial services environment, large-scale financial losses pose that risk;
- the number of *levels* is determined by the ability (i) to distinguish levels and (ii) to identify meaningfully different sets of safeguards per level.

2.1 Integrity Criticality Rating of Applications

Integrity-Criticality Rating Schema. To determine the integrity criticality of a system, we first determine whether legal/regulatory requirements exist or not. Legal and regulatory requirements are, e.g., privacy laws, liquidity or records retention requirements. Hence, we divide business applications in two classes based on whether such requirements apply or not. Second, we classify the applications according to financial impact by considering whether (i) material financial losses are possible and (ii) whether all possible material financial losses are recoverable or not. Table 1 summarizes the resulting rating scheme. For instance, the 'normal' integrity criticality level means basically that neither losses nor non-compliances are acceptable. These strictly-defined non-functional requirements determine the minimum bar (see also Section 3), which an application has to meet.

As an example, let's apply this rating scheme to an Order Management application for foreign exchange orders. Orders are trades between clients and the bank, i.e., they are legally binding external financial commitments and therefore integrity-relevant from a financial and compliance perspective.

Table 1. Integrity-criticality levels

Integrity-criticality Level	Financial Impact	Legal/Regulatory Reqs.
I-1, critical	Irrecoverable losses possible	Yes/no, traceable
I-2, normal	Recoverable losses possible	Yes, traceable
I-3, non-critical	Losses impossible	No

For applying this rating scheme in general, we first classify the data processed by an application. For this purpose, we identify the business objects that are relevant from one or more of the following perspectives:

- *Compliance relevant*, i.e., the application is used for legal/regulatory purposes, e.g., the calculation of the Basel II/III capital requirements.
- *Financially relevant*, i.e., it results in asset ownership changes and, in particular, *legally-binding external financial commitments* and elements thereof.
- *Risk management controls relevant*, e.g., systems management applications used for safeguarding confidentiality, integrity and availability.

Given this integrity-relevance classification, all applications that are not processing integrity-relevant data are rated as *non-critical.* Second, all applications that are not processing legally binding external financial commitments and elements thereof are rated as *normal critical.* In case of external commitments, we define the *recoverability* of possible financial losses using following guidelines:

- Losses are impossible, if transactions involving external commitments are subject to compensatory business controls by the involved counter-parties. E.g., a trusted third party, like Continuous Linked Settlement (www.cls-group.com), clears and settles transactions only if they receive matching orders from all trade parties.
- Losses are possible and recoverable, if the business controls are sufficient to detect and correct all possible errors. Questions that need consideration in this context: How fast is a large number of small errors detected? How fast are a few large errors detected?

  Business controls often have control points at which data validity is verified and, hence, the time period between two control points has to be considered in an assessment. Note: In reality, the recoverability of financial losses is often considered given, if the recovery costs and the residual losses do not exceed a certain financial threshold.

It is important to understand, that our rating scheme intentionally requires only a rough quantitative assessment of possible losses. We recommend to use a recoverability threshold and loss statistics for taking estimation errors into account. More important than a particular threshold and the exact estimation is the identification of worst-case scenarios including market turbulences with extreme market volatilities. Such turbulences might also be caused by an integrity breach – as above real-world examples illustrate.

For example, the foreign exchange Order Management application is rated at least as I-2 due to compliance requirements. For determining the possible losses and their recoverability, we consider worst-case failure scenarios: Say, losses might result from offering prices below market rates and erroneous settlement payments. Let us assume that all orders are hedged immediately and all settlement payments are conducted through a trusted third party. Hence, the possible losses are recoverable and the (final) rating is I-2.

Business and IT Criticality Rating. For an application, business determines the protection requirements, called *Business Criticality Rating*, using the above rating schema and taking defined business controls into account. Business controls consist of manual activities to supervise the correct working of applications and to handle exceptions. Given the Business Criticality Rating, IT determines the *IT Criticality Rating* based on the business requirements, the actual design and the inter-dependencies of applications. If the IT rating differs from the business rating, it is necessary to redesign and, if impossible, to reconsider the Business Criticality Rating. For instance, our Order Management application offers prices in all tradable currency pairs based on an interbank market data feed. The generated prices are monitored by a trader, which is possible thanks to an automated rate-tolerance check. If IT comes to the conclusion (for example during design or testing) that this check does not work as intended, the criticality rating of the application must be revised, i.e., set to I-1 as non-recoverable losses are possible.

2.2 Criticality Rating of Services

A system typically depends on data and functions that are provided by other applications. For instance, the Order Management application depends on systems providing market data and performing credit-risk checks and order settlement. This leads to *criticality dependencies* that must be managed. In particular, unnecessary propagations of high criticality levels must be prevented. For dealing with criticality-dependencies, we use the *Service-Oriented Architecture (SOA)* [2,19] and introduce *risk-adjusted services*.

Risk-Adjusted Services. A *service description* defines, like an API, the service interface in terms of service operations. A service operation is described by the IN- and OUTput data and the functions provided. For instance, our Order Management application offers a trade capture service with `Create`, `Read`, `Update` and `Delete` operations for foreign exchange orders. For including criticality in service descriptions, we must understand the function of a service operation (e.g., `createFxOrder)`. Towards this end, we classify the functionality of services into EVENT and ACTION service operations [10], whereby a service provider reports on events and responds to service requests, respectively. *Integrity criticality* is a non-functional requirement on a service, i.e.,

- an EVENT service delivers data of a particular quality; and
- an ACTION service provides data and function at a particular quality.

The criticality rating of a service operation is based on the criticality rating of the application sub-system implementing the service. The resulting rating is specified by tagging the service operation with the respective criticality level in the service description. For instance, an EVENT service `getMarketData` that provides inter-bank foreign exchange rates sourced from market data providers would be rated as I-2, because the data is informational and not representing

an offered price. Depending on the division of labor within the bank, our Order Management application might consume tradable prices from the wholesale business unit using an EVENT service `getFxPrice`. This service is classified I-1 assuming that delivering erroneous tradable prices results in possible irrecoverable losses. Furthermore, the Order Management application itself provides ACTION services, such as `createFxOffer` and `createFxOrder` to request a quote and to capture an order, respectively. These two services are classified as I-2, because these services are creating financial commitments and given our Order Management application is classified as I-2.

We also declare the criticality of the data that the service provider expects and delivers by tagging the data attributes of the exchanged data accordingly. The exchanged data is defined by the payload description for the IN and OUT messages of a service operation. A foreign exchange order consists of data attributes defining the financial contract details, contract settlement details, sales markups and customer comments. Arguably, the latter two attributes might be classified as not integrity relevant.

For explaining how risk-adjusted services lead to risk-adjusted sub-systems, we discuss the criticality-level of functional and physical sub-systems:

- A *functional sub-system* is a functional component that is not instantiated as an isolated run-time component. Without this isolation, a functional subsystem must meet the criticality-level of the system. For example, an application serving multiple tenants consists of a functional sub-system per tenant. If such tenants have different business integrity criticality ratings, the applications and its functional sub-systems are one system due to a lack of isolation. Hence, the highest rating of a tenant determines the rating of the whole system.
- A *physical sub-system* is a physically isolated sub-system of a system. Hence, it must meet only its own criticality level taking into account the dependencies that other sub-systems of the application have on it. For example, a service replica and multiple application-instances serving different tenants are physical subsystems, which might be operated and even designed and implemented according to their particular criticality level. In our example, we might consider using two physical instances of the Order Management application to support tenants with different criticality levels.

For managing services, a physical sub-system results in a single service description with multiple *service implementations* – one per sub-system. Each *service implementation description has* its own criticality level. For instance, `createFxOrder` service replicas offer all the same service interface, but their service is either an I-1 or an I-2 one.

Consuming Adequate Services. Given we have a declared criticality in the service descriptions, prospective service consumers simply have to identify and to use *adequate* services from the service repository. Let's first define the term *adequate* and then discuss how we document the selection for management purposes.

Understanding integrity criticality as a quality – the degree of dependability, we conclude that a service consumer must either (i) consume services with at least the same criticality level or (ii) implement *compensations* for consuming lower-grade services. We distinguish between the following two classes of compensations:

- *Data Services*: For EVENT and ACTION services solely delivering data, a compensation can be implemented by an *input plausibility* validation detecting and handling erroneous and missing data.
- *Processing Services*: For ACTION services that have real-world effects that are not reported in the response, a compensation is a so-called *final inspection* of the result (by the consumer).

For instance, our Order Management application might consume a lower-grade market data service `getFxRates`, if it uses an automated input plausibility validation and, in case of a validation exception, a trader manually quotes prices. For using a lower-grade down-stream `createPaymentOrder` service, the Order Management application has to validate the resulting payments.

Note that the service consumers are registered in a service repository. This includes the registration of the required integrity criticality-level and the necessary and implemented compensations. The latter is simply a textual description with a reference to the relevant application documentation.

3 Integrity Minimum Bars

A system's protection does not come for free. Therefore, any protection mechanisms should be employed in line with a system's criticality. To facility this approach, we have introduced so-called *minimum bars* that define a unique standard set of controls for each criticality dimension and level, which an application must fulfill. Minimum bars must meet a number of criteria, such as:

- Each standard must apply to a *single* application such that an IT architect can design an application by acting locally while ensuring integrity globally.
- Each standard is applicable to all kinds of programming languages and to systems developed in-house as well as by third parties.
- Wherever possible, standards must be satisfied by using a standard infrastructure capability. This simplifies the design and assessment, as a system can rely on the controls implemented by the standard infrastructure.
- Standards must be minimal, where (i) implementing higher standards is possible, if justified by other reasons than information security (e.g., efficiency); (ii) implementing alternative controls requires a risk assessment and a proof of adequacy; and (iii) controls perceived irrelevant for a system require a formal exception, if not implemented.

3.1 Ensuring Authorized Modification

The first integrity protection objective is that all *data modifications are authorized*, which requires non-*repudiation* and *tamper-resistance*. Respective controls

are mainly in the space of access control. For example, foreign exchange interbank rates are integrity-relevant. Updating such rates is hence subject to standard access control mechanisms.

3.2 Ensuring Valid Results

Our set of integrity design standards aims at ensuring that information is meaningful, accurate and timely (and hence also modified in acceptable ways). The basic idea is that an application defines first what data must be valid. Second, that its data processing maintains validity. Third, that an application validates its input and output.

These design standards apply to all systems with a criticality I-1 or I-2. The reason is that the same design can be used for both levels, where the controls for testing and operations are differentiated.

Data Integrity Standards. A first standard mandates that the integrity-relevance of business-object attributes is classified and documented in the data model. For defining useful validations, business objects are typed and the actual and more specific business rules must be defined. In our example, the business object is a foreign exchange order with its financial and settlement details. Business rules are, for instance, that traded amounts above a particular threshold are manually quoted; and that trades captured by junior traders require a sign-off by a supervisor.

A second standard mandates that integrity-relevant data is consumed from its golden source using services. Furthermore, it requires that integrity-relevant data – typically, at the level of a business object – is versioned and globally uniquely identifiable.

For instance, all applications processing foreign exchange orders source counter-party and settlement instructions from the same golden sources. Versioning is important because the order processing requires, for instance that a counter-party and a settlement instruction is 'active' and not in an on-boarding state. Other version updates, such as domicile and address changes do not change the state, but are nevertheless relevant for order processing. Therefore, services, such as `createConfirmation` and `createPayment` have to specify how they deal with version updates of reference data. For instance, `createConfirmation` sends a confirmation letter to the current address of the client instead of the one when the order was captured.

Processing Integrity Standards. A first standard requires that integrity-relevant data processing produces an audit trail using the standard logging infrastructure. A second standard requires a reconciliation of integrity-relevant data exchanged between two systems. A third standard requires idempotent batches, services and user interfaces, if they support integrity-critical data processing. A fourth standard requires the use of our *standard consistency patterns* and the respective supporting infrastructure. These patterns are, in particular,

non-remote transactions, reservations and modify tickets. The standard also recommends to use asynchronous instead synchronous service interactions, wherever possible. Finally yet importantly, service interactions might be delayed or fail, for which monitoring and exception handling is mandated.

Validation Integrity Standards. A first standard mandates an automated input/output plausibility validation of all integrity-relevant data communicated in machine and user interfaces. These validations aim at determining the plausibility of the input/output, e.g., input values being in a reasonable range and relation to each other. This standard implies that applications perform their own input/output validation and consume services for complex business validations, like suitability and credit-risk checks. A second standard requires that all validation exceptions must be handled by either a sign-off, degraded mode of operation or a failure. For instance, invalid market rates might result in a degraded mode of operation where all prices are manually quoted. A third standard mandates that service descriptions specify the degree of required and provided data validation.

In the first standard, we distinguish between *deterministic* and *heuristic input/output validations*, where the latter takes into consideration a *processing context* defined in terms of attributes, such as user, counter-party, asset class, etc. At the source of data entry and creation of a business object, a fine-granular processing context is available and hence can be used for heuristic validations. Therefore, the third standard requires that golden sources perform fine-granular validation and deliver authoritatively-validated data. There exist hence two degrees of validation: 'normal' and authoritative. For instance, the Order Management application performs fine-granular input validation, such as 'is this captured amount in the usual range for this user and client?': Capturing an order with a much larger traded amount triggers a request to the user to confirm the order. The down-stream applications receive authoritatively-validated orders and hence they are not obliged to consider whether an order is usual or unusual.

Recovery Integrity Standards. A first standard mandates an appropriate backup of integrity-relevant data. A second standard requires that an application processing integrity-relevant data is restorable, e.g., from its backups. This ensures that all restored integrity-critical data and functions are valid as per post-restoration validation. It is important to understand that integrity requires that the selected availably criticality – defining an upper bound for the duration of possible outages and data losses – does not lead to integrity issues given the capability of the business controls to deal with such a situation. Otherwise, integrity requires an upgrade of the availability criticality, i.e., shorter outage durations.

3.3 Final Validations and Independent Controls

Limitations and Motivation for Additional Controls

Propagation of criticality: The presented rating scheme has the drawback that it leads to a large number of applications that are in the highest class, as illustrated in Fig. 2. Often a business critical *source application* for an external

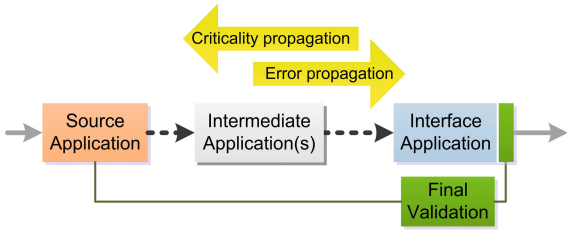

Fig. 2. Error and criticalit propgation vs. final validations

commitment is supported by a chain of down-stream applications. And, this chain is terminated by an *interface application* that eventually communicates such an external commitment to counter-parties that are outside of the firm. In a first analysis, such interface applications were identified as business critical given that integrity failures are materialized at this point. The reason is that errors occurring after the source are propagated to an interface application. Unless such an interface application is able to detect such errors, the whole processing chain has to be treated as business critical. Detecting such errors is possible by introducing *final validations* at the interface that check the correspondence of business activities between the source and interface. For instance, an interface sending out an external-payment message checks that the message content corresponds with the respective payment order captured at the source, such as an on-line banking application.

Mandating that interfaces are performing final validations would allow us to simplify the above rating scheme by (i) applying the rating procedure to distinguish between I-1 and I-2 to source applications only; and (ii) mandating that I-1 interfaces perform final validations (on the I-1 feeds). Instead of this simplified rating procedure, the current version supports only a down-grading of the criticality, if an application is subject to final inspections (by the respective source).

The devil (of controls assurance) is in the details: If all applications along a processing chain design, implement, test and operate their integrity controls completely, consistently and correctly, the presented minimum bars are sufficient. In practice, explicit integrity standards are beneficial on one hand, because integrity receives the necessary attention. On the other hand, controls for I-1 applications are generally very expensive to implement/operate and flawless execution of these controls is almost impossible to guarantee. Past material integrity breaches illustrate the latter point: for instance, a duplicate check failed due to a race condition resulting in a loss of 500'000 USD. For either reducing the effort and/or increasing the assurance, we are developing the concept of independent integrity controls including (i) source behavioral monitoring; and (ii) source to interface correlation. Similar to multi-version programming, these controls are to be designed, implemented and operated by a party independent from the application developers. Given the initial objective to detect only severe integrity breaches, the correlations can be approximations.

Independent Integrity Controls for Final Validations. As a proof-of-concept, we implemented two correlation use cases using audit trails. Fig. 3 illustrates the resulting architecture. Our order processing applications, for in-

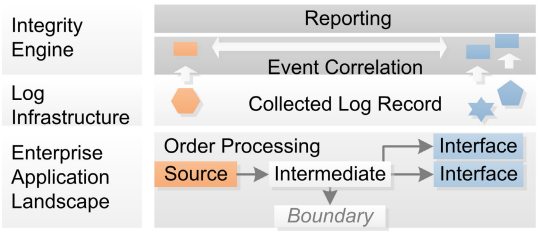

Fig. 3. Integrity engine proof of concept architecture

stance, generate audit log records, which are collected and stored by our log infrastructure. This infrastructure is based on SPLUNK (www.splunk.com) and we implemented our correlations using the product's analytics capability. For this purpose, the relevant log records are retrieved and normalized first. Second, the correspondence of source and interface events is validated by identifying and comparing the relevant source and interface log records. Finally, the correlation results are reported in a dashboard.

Identifying and Defining Validation Correlations: Trading typically involves a trading, a clearing and a settlement business sub-process. In case of foreign exchange, we need to identify trade capture events and relate them to clearing and settlement events. Such a trading business process is often distributed over multiple applications, each of which having its own business objects. For systematically identifying and defining correlations, we use *communicating finite state machines*. For an example, see Fig. 4(b).

For a source application, we define the business object and its life-cycle states. Then, we define the relevant interface applications and their business objects with their life-cycle. Finally, we identify the correlations by communications between state-transitions of the source and interface finite state machines. Fig. 4(b) represents a simplified example for foreign exchange: a client order is captured on the Order Management application, which results in an order in state `Active`.This order is confirmed by a letter generated by a Print Delivery application and the Single Account application pre-books a credit for the bought amount. At settlement date, the client order becomes `Mature` and, hence, the Single Account application credits the account (i.e., replacing the pre-booking with the actual booking). In addition, the counter amount has to be paid to the bank, for which an external payment is received via a Messaging Gateway application. The example illustrates that business process boundaries are either internal or external (and in the latter case involving an interface application). The internal boundaries trigger often new business processes, some of which work

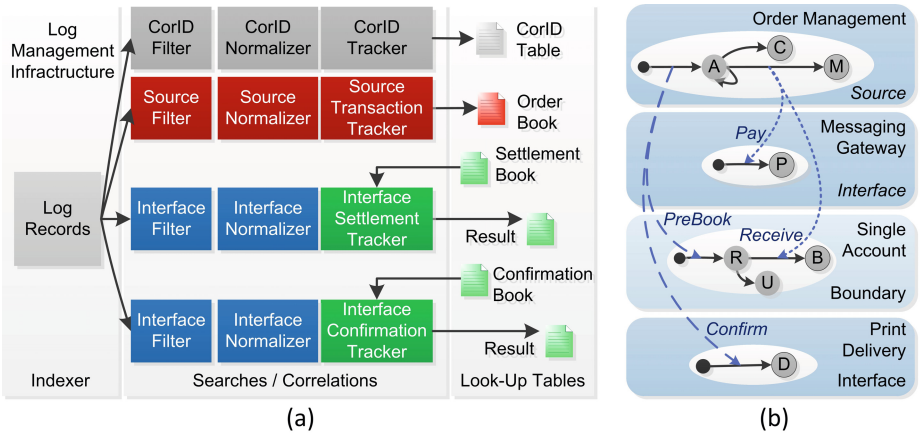

Fig. 4. (a) modular correlation implementation and its (b) finite state machine model

on aggregates. For instance, foreign exchange orders are aggregated into positions and hedged in the interbank market. The integrity control for the hedging business process includes an approximate check of the correct aggregation.

Modular Implementation of Correlations: Given the above finite state machine correlation model including the conditions for state transitions, we implement the correlation rules using our correlation engine based on a few patterns, as illustrated in Fig. 4(a). In the upper half, we search the source order events and keep the order's current state in the SPLUNK lookup table `OrderBook`. In case the order processing has no unique business case identifier, we search for the related correlation identifiers starting from the source: a chain of related identifiers can be established, which might involve computing mappings, such as hashing. For deadline-driven interface events, we create a view of the `OrderBook` for a particular deadline. For instance, foreign exchange spot orders are settled two business days after the trade date. In Fig. 4(b), this view is a lookup-table `SettlementBook`, which is used by `InterfaceSettlementTracker` to correlate the payment messages with the source foreign exchange orders. The matured order correlates with `Pay` and `Receive` in Fig. 4(b). A settlement event might be either missing, duplicated or not corresponding (e.g., amount differences). This example illustrates how to identify and implement validation correlations in a modular way. This is important for maintainability and automating the implementation (for a given correlation engine).

4 Related Work

Integrity Information-Risk: In [6], integrity is defined as the *property of safeguarding the accuracy and completeness of assets*. However, the standard and the accompanying best-practices [7] are of rather generic nature. Furthermore, the

controls primarily ensure that all modifications are authorized. This is the most widely used facet of integrity [8]. Like in [8,9], we consider also the requirement that information is accurate, consistent and meaningful.

Finally yet importantly, we are, in contrast to [6], not considering the probability of a failure. The reason is that severe integrity failures are rare events and practically not predictable (see also [16]). Such failures including the above real-world examples are arguably black swans [17] - these are unexpected events that can be rationalized in hindsight, but are hard to foresee. Consequently, we are not using classical risk analysis techniques [23], such as failure mode effect analysis (FMEA). For assessing possible financial losses and their recoverability, we require, first, to determine the financial assets at risk and, second, to assess the business controls processes. During our work, we considered initially to construct an integrity data and controls flow graph to carry out a formal effects analysis of "corrupted" data attributes. A first pilot with the integrity data and controls flow graph as well as the underlying dependency graph showed that such a formal approach is not suitable for an organizational deployment without considerable investments, especially in tools.

Integrity Criticality Rating: Regarding risk ratings, the industry standard [6] requires that *"information shall be classified in terms of its value, legal requirements, sensitivity and criticality to the organization"*. Our rating scheme provides a modular procedure that effectively classifies integrity risks based on the impact of an integrity failure. Modular means here that, first, risk can be assigned on a per-application-basis. Second, the dependency analysis is limited to the services offered and provided by the assessed application (i.e., the direct dependencies). Our scheme is not universal, because safety-critical systems are out of scope.

Embedded Integrity Design Standards: Best practice on *correct processing in applications* is the most relevant one (see section 12.2 in [7]). We extended these best-practices in the following ways:

1. Standards for information definition and labeling including data-flow controls using services (e.g., criticality, validation quality, etc.). Our data labeling and flow control policies and patterns are aimed at application designers. Hence, we have currently not formalized them. The different facets of integrity as well as their formalization and enforcement as information flow policies are discussed in, for instance, [3].
2. Specific patterns related to idempotency [5] and data consistency suitable for a Service-Oriented Architecture (e.g., non-remote transactions).
3. An extensions of input/output plausibility validations with heuristic plausibility checks (behavioral monitoring).
4. Last but not least, an emphasis on the business operating procedures and hence the business controls for an application.

Independent Integrity Design Standards: Following multi-version programming, we drafted a second set of *independent integrity controls* that are independent

from our established integrity controls (outlined previously). This paper presents the validation of the processing of legally-binding financial external commitments at the boundary of the firm: for instance, order entry and payment business events are correlated. This is similar to the transaction verification in *continuous auditing* [11,12], but we use *audit trails* and *log analysis* [4] and hence apply data analytics to improve the information security in the integrity dimension. The same integrity facet is considered by applying data analytics to ensure the correctness of authentication data [18].

From a *data provenance* perspective [21,22], we are collecting provenance information in the form of audit records through our existing audit log infrastructure. In contrast to data provenance, we are neither explicitly nor implicitly constructing a data provenance graph - a directed acyclic graph explaining the sources and derivations of the data. Like in some data provenance approaches, we are evaluating the data quality by comparing the relevant source and interface events for financial commitments, such as trading and payment orders. Data provenance techniques might be of interest to integrity-critical information based on data warehouses. Hence, future work should relate our log-analysis approach to the service-oriented data provenance approach [22].

5 Experiences and Learnings

The presented rating scheme and standards are currently being rolled out within Credit Suisse. The simplification of the presented rating scheme based on final validations depends on the further development of the final validation concept, the initial proof-of-concept as well as cost/benefit considerations.

Rating: When we piloted a new questionnaire to support the criticality rating, we learned that availability and confidentiality are well-understood concepts. However, integrity – especially beyond the meaning of authorized modifications – is not. We have invested considerable time in phrasing the questions, make them easy to understand and unambiguous. Getting it right is especially important for integrity as a single failure in this dimension might result in the enterprise defaulting.

Minimum Bar Assessment: In order to determine the level of residual integrity risk within critical applications, we established an extensive questionnaire to assess the standing of an application against its defined minimum bar. 200 applications were assessed against their minimum bars. We investigated confidentiality, integrity and availability capabilities using roughly 120 questions per application. Thereof, 1/3 of the questions related to integrity and covered aspects regarding the design and implementation of an application, its testing as well as daily operations in the data center. From this initial assessment, we learned that the application owners partially faced difficulties in correctly assessing their applications. The original set of questions was hard to understand. In collaboration with the IT Risk Management department and the application owners, we have

re-phrased many questions, making them easier to understand. Additionally, we held training sessions, which are now available as a replay for the education of new staff or as a refresher. We believe that such dedicated training sessions facilitate the understanding of integrity, which in turn increases the acceptance of integrity controls and the related assessments.

Independent Final Validation: A proof-of-concept demonstrated that the correlation of source and interface events is possible. For in-band final validations, the non-functional requirements are different and we will have to re-consider the correlation engine architecture. We are also evaluating the SPLUNK database connector for obtaining scalable look-up tables. The model for these correlations was developed manually, which does not scale. And, a large effort to develop correlation rules makes it also brittle, if rules change frequently. Finally yet importantly, demonstrating the validation of aggregations (e.g., such as positions aggregating foreign exchange market orders) was not in scope. However, we have demonstrated in another project how to implement such aggregation controls with Mathematica (www.wolfram.com) using the position aggregation for foreign exchange spot, swap and forward orders as an example.

6 Conclusions and Outlook

What does this paper demonstrate? How to deal with integrity in a broader sense; how to standardize the rating to achieve a more consistent risk assessment; as well as design standards to ensure that integrity is considered appropriately. Finally, it points to a direction for cost-effective assurance of control implementations using independent integrity controls.

We are currently working on the institutionalization of the standards, improving them on an ongoing basis, issuing supporting guidelines, conducting training sessions and delivering self-study training material. For the further development of the independent controls and final validations, we are investigating machine learning approaches to reduce the modeling effort for correlation rules.

References

1. Murer, S., Bonati, B., Furrer, F.J.: Managed Evolution: A Strategy for Very Large Information Systems. Springer (2011)
2. Krafzig, D., Banke, K., Slama, D.: Enterprise SOA: Service-Oriented Architecture Best Practices. Prentice Hall Publisher (2004)
3. Birgisson, A., Russo, A., Sabelfeld, A.: Unifying facets of information integrity. In: Jha, S., Mathuria, A. (eds.) ICISS 2010. LNCS, vol. 6503, pp. 48–65. Springer, Heidelberg (2010)
4. Oliner, A., Ganapathi, A., Xu, W.: Advances and Challenges in Log Analysis. Communications of the ACM 55(2), 55–66 (2012)
5. Helland, P.: Idempotence is not a medical condition. Communications of the ACM 55(5) (2012)

6. ISO/IEC 27001:2005, Information technology – Security techniques – Information security management systems – Requirements 2nd Edition (2005)
7. ISO/IEC 17799:2005, Information technology – Security techniques – Code of practice for information security management, 2nd Edition (2005)
8. Pfleeger, C.P., Pfleeger, S.L.: Security in Computing, 4th edn. Prentice Hall (2007)
9. Mayfield, T.: Integrity in automated information systems, National Computer Security Center, Technical Report 79-91 (1991)
10. OASIS Reference Model for Service Oriented Architecture 1.0, Official OASIS Standard (2006)
11. Vasarhelyi, M.A., Alles, M., Kogan, A.: Principles of Analytic Monitoring for Continuous Assurance. Journal of Emerging Technologies in Accounting 1(1), 1–21 (2004)
12. Chan1, D.Y., Vasarhelyi, M.A.: Innovation and practice of continuous auditing. International Journal of Accounting Information Systems 12(2) (2011)
13. Risks Digest: Forum On Risks To The Public In Computers And Related Systems (moderated by Neumann, P.G.), vol. 26(97) (2012)
14. Risks Digest: Forum On Risks To The Public In Computers And Related Systems (moderated by Neumann, P.G.), vol. 26(92) (2012)
15. Risks Digest: Forum On Risks To The Public In Computers And Related Systems (moderated by Neumann, P.G.), vol. 21(81) (2001)
16. Ross, S.J.: Information Security Matters: Keynes, Shelley, Taleb and Watts. ISACA Journal 4 (2012)
17. Taleb, N.: The Black Swan: The Impact of the Highly Improbable. Penguin (2008)
18. Clement, M., et al.: Data analytics for information security: from hindsight to insight, Research Report, Information Security Forum (2012)
19. Murer, S.: 13 Years of SOA at Credit Suisse: Lessons Learned-Remaining Challenges. In: Ninth IEEE European Conference on Web Services, ECOWS (2011)
20. Financial Times, FSA challenges bank chairmen over IT (September 4, 2012)
21. Simmhan, Y.L., Plale, B., Gannon, D.: A survey of data provenance in e-science. SIGMOD Rec. 34(4), 31–36 (2005)
22. Moreau, L., et al.: The Provenance of Electronic Data. Communications of the ACM 51(4), 52–58 (2008)
23. Bennett, J.C., Bohoris, G.A.: Risk analysis techniques and their application to software development. European Journal of Operational Research 96(3), 467–475 (1996)