Evaluating System Integrity

Simon N. Foley,
s.foley@cs.ucc.ie

Department of Computer Science,* University College, Cork, Ireland.

Centre for Communications Systems Research, University of Cambridge, Cambridge CB2 3DS, UK.

Abstract

Conventional models of system integrity tend to be implementation-oriented in that they define integrity in terms of specific controls such as separation of duties, well-formed transactions, and so forth. In this paper we propose a formal definition of integrity that is based on the notion of dependability and is implementation independent. Using a series of examples, we argue that separation of duties, assured pipelines, fault-tolerance, and cryptography may be viewed as implementation techniques for achieving integrity.

1 Introduction

Conventional integrity models such as [2, 4, 22] limit themselves to the boundary of the computer system and tend to define integrity in an operational and/or implementation-oriented sense. For example, the Clark-Wilson model [4] recommends that well-formed transactions, separation of duties and auditing be used to ensure integrity. However, the model does not attempt to address *what* is meant by integrity—evaluating a system according to the Clark-Wilson model gives a confidence to the extent that good design principles have been applied. For instance, when we define a complex separation of duty policy, we cannot use the model to guarantee that a user of the system cannot somehow bypass the intent of the separation via some unexpected circuitous route.

Traditional Requirements Analysis [20] typically identifies the *essential* functional requirements that define *what* the system must do. An implementation defines *how* the system operates and must take into consideration the fact that the infrastructure that is put in place to support the requirements may be unreliable. For example, experience tells us that a system's infrastructure should include a suitable backup and restore subsystem. While not part of the essential requirements, it is a necessary part of the implementation since the infrastructure can corrupt data. Infrastructure is everything that serves the requirements—software, hardware, users, user-procedures, and so forth.

*Address for correspondence.

Proceedings of the New Security Paradigms Workshop, Charlottesville, VA, USA, September 1998. ACM press. Forthcoming

In [13], integrity is characterised as just one attribute of dependability, that is, *"dependability with respect to absence of improper alterations"*, and dependability is *"a property of a computer system such that reliance can be justifiably placed on the service it delivers"*. If a system is built on a perfect infrastructure that never fails then it is dependable. Such a system would include functionally correct and reliable computer systems, completely trustworthy users who follow procedures exactly, and so forth. However, in practice, it is not possible to build such an enterprise. Even if the system is functionally correct, the infrastructure is almost always sure to fail: users may be dishonest, not follow procedures properly, and so forth.

In this paper we characterise dependability as a form of refinement—a system is sufficiently robust such that even in the presence of infrastructure failures it can be shown to implement (refine) the top-level requirements. In addition to integrity, authentication and confidentiality are other attributes of dependability [13], and in this paper we argue that our notion of dependability encompasses them.

Section 2 introduces the notion of local refinement and argues how it can be used to characterise dependability. Clark and Wilson identify external consistency—the correct correspondence between data objects and the real world objects they represent—as the abstract requirement that integrity mechanisms such as separation of duty seek to enforce, and we characterise this in terms of local refinement. A series of simple examples are given in Section 3 to illustrate how separation of duties, cryptography, fault-tolerance and assured pipelines may be regarded as implementation techniques used to achieve dependability. Section 4 investigates some general properties of dependability.

Local refinement is formalised in terms of event systems. Rather than building and reasoning about an event-system from first principles, CSP [10] is used in the paper to present the theory and examples in a convenient and unambiguous manner. The Appendix gives a brief summary of CSP and its trace semantics.

2 Integrity and Dependability

Example 1 A simple enterprise receives (equal-value) shipments, and generates associated payments for a supplier. Requirements Analysis identifies events snote and pay corresponding to the arrival of a shipment (note) and it's associated payment, respectively. Enterprise behaviour is speci-

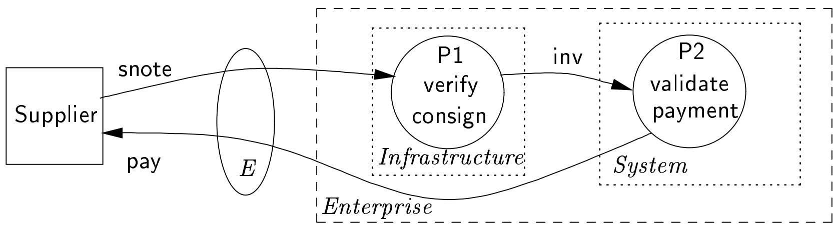

Figure 1: A simple payment enterprise

fied by CSP process $ConsReq_0$, where

$$\begin{aligned} ConsReq_0 &= (\mathsf{snote} \to ConsReq_1) \\ ConsReq_i &= (\mathsf{pay} \to ConsReq_{i-1} \qquad (i > 0) \\ &\quad \mid \mathsf{snote} \to ConsReq_{i+1}) \end{aligned}$$

Figure 1 outlines a possible implementation of this requirement. A clerk verifies shipment notes and enters invoice details (event inv) to a computer system, which in turn, generates payment (pay) for the supplier. This is specified as

$$\begin{aligned} Clerk &= (\mathsf{snote} \to \mathsf{inv} \to Clerk) \\ System &= (\mathsf{inv} \to \mathsf{pay} \to System) \end{aligned}$$

and the enterprise design is specified as $ConsImp = System \| Clerk$.

Intuitively, integrity is maintained if, even in the presence of failures within the infrastructure, the implementation *ConsImp* supports requirement $ConsReq_0$ at it's external interface E with the supplier. △

The example above illustrates that integrity may be characterised as a form of refinement—*ConsImp* refines $ConsReq_0$. In the traces model of CSP, process S is a (safety) refinement of process R if $\alpha R = \alpha S$ and $traces(S) \subseteq traces(R)$, that is, every possible trace of S is permitted by R [10]. For example, the process $P = (\mathsf{snote} \to \mathsf{pay} \to P)$ which alternates between snote and pay, is a refinement of process $ConsReq_0$.

The Supplier (Example 1) is oblivious to 'internal' event (inv) and interacts with *ConsImp* abstracted through interface $\{\mathsf{snote}, \mathsf{pay}\}$, that is, $ConsImp@\{\mathsf{snote}, \mathsf{pay}\}$, where for process S and set of events E,

$$S@E \mathrel{\widehat=} \{\, t : traces(S) \bullet t \restriction E \,\}$$

and $t \restriction E$ is the trace t with events not in E removed. Every trace the supplier can observe from $ConsImp@\{\mathsf{snote}, \mathsf{pay}\}$ is permitted by $ConsReq_0$ and we say that *ConsImp locally refines* $ConsReq_0$ at that interface, that is, $ConsReq_0 \sqsubseteq^{\{\mathsf{snote},\mathsf{pay}\}} ConsImp$.

Definition 1 (Local Refinement) R is locally refined by S at event interface E iff $R \sqsubseteq^E S$, where, $R \sqsubseteq^E S \Leftrightarrow E \subseteq \alpha R \subseteq \alpha S \wedge S@E \subseteq R@E$. △

Example 2 Continuing Example 1, we assume that the computer system will behave reliably (according to *System*). However, it is not reasonable to assume that the clerk will always act reliably according to *Clerk*. In practice, an unreliable clerk ($\overline{Clerk}$) can take on any behaviour involving events snote and inv.

$$\overline{Clerk} = RUN_{\{\mathsf{snote},\mathsf{inv}\}}$$

$$ConsImp2 \mathrel{\widehat=} ConsSys \| \overline{Clerk}$$

We argue that *ConsImp2* is a more realistic representation for the actual enterprise that will be fielded. It more accurately reflects the reliability of its infrastructure than the previous design *ConsImp*. However, for external interface $E = \{\mathsf{snote}, \mathsf{pay}\}$, since $t = \langle \mathsf{inv}, \mathsf{pay} \rangle \in traces(ConsImp2)$, and $t \restriction E = \langle \mathsf{pay} \rangle \notin traces(ConsReq)$ then $ConsReq \not\sqsubseteq^E ConsImp2$, that is, the design is not robust enough to be able to support, in a safe way, the original requirements *ConsReq*. △

In [13], integrity is given as one attribute of *dependability*; other attributes include confidentiality and authentication. Dependability is characterised as a *property of a computer system such that reliance can be justifiably placed on the service it delivers* [13]. We argue that this notion of dependability may be viewed as a class of refinement whereby the nature of the reliability of the enterprise is explicitly specified.

Definition 2 (Dependability) If R gives behavioural requirements for an enterprise and S is its proposed implementation, including details about the nature of the reliability of its infrastructure, then S is as *dependably safe* as R at interface E if and only if $R \sqsubseteq^E S$. △

According to Clark-Wilson [4], external consistency is the *"correct correspondence between data objects and the real world"*. Another way to view this is that an external entity can achieve consistent interactions with the enterprise, even in the presence of failures within the infrastructure of the enterprise. We characterise this notion of external consistency in terms of dependability.

Definition 3 (External Consistency) Let $S \| I$ and $S \| \overline{I}$ describe the behaviour of system S operating within reliable, and unreliable, infrastructure I and $\overline{I}$, respectively. We say that S is *externally consistent* at interface E if $S \| \overline{I}$ is as dependably safe as $S \| I$, that is, $S \| I \sqsubseteq^E S \| \overline{I}$. △

Example 3 Given the nature of an unreliable clerk (Example 2), *ConsImp2* is not as dependably safe as *ConsReq* at the interface E. Similarly, *System* is not externally consistent at interface E, since $System \| Clerk \not\sqsubseteq^E (System \| \overline{Clerk})$. △

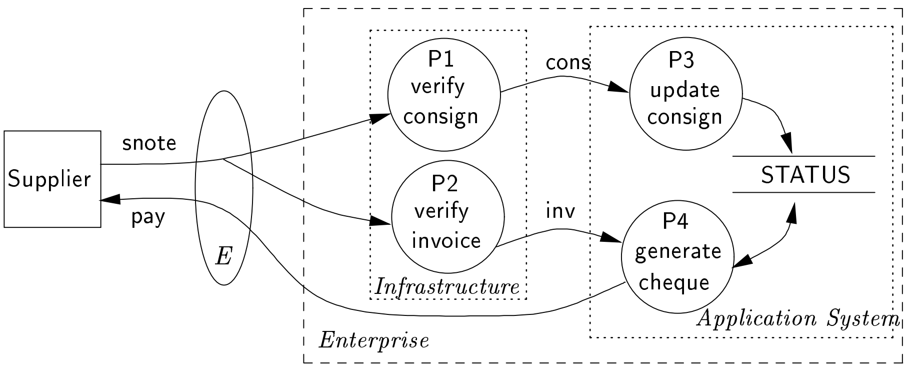

Figure 2: Supporting separation of duties

3 Dependable Systems

3.1 Separation of Duties

Separation of duties is a common implementation technique for achieving integrity. While fault-tolerant techniques replicate an operation, separation of duties can be thought of as a partitioning of the operation.

Example 4 Suppose that when a shipment arrives a clerk verifies the consignment at goods-inwards (entering details cons into the system). When an invoice arrives, a different clerk enters details into the system, and if the invoice matches a consignments, a payment is generated. So long as the operations are separated then a single clerk entering a bogus consignment cons or invoice inv can be detected by the system. For simplicity, we assume that both cons and inv arrive at the same time in snote; this is depicted in Figure 2.

To distinguish shipments, events are prefixed with identifiers drawn from $\mathcal{N}$, the set of shipment-identifiers. For example, $n.\mathsf{pay}$ corresponds to the payment resulting from shipment-note $n.\mathsf{snote}$. While shipment-identifiers are intended to be unique, it is possible that a supplier may re-use identifiers. Thus, $n{:}ConsReq_0$ (process $ConsReq_0$ with events prefixed by n) describes the behaviour required when processing shipments identified by $n \in \mathcal{N}$. The top-level requirement is

$$ConsReq = \|_{n:\mathcal{N}}(n{:}ConsReq_0)$$

The proposed application system allows arbitrary clerks u and v verify the consignment ($n.\mathsf{cons}.u$) and invoice ($n.\mathsf{inv}.v$) for consignment n, after which, payment is generated.

$$AppSys = \Box_{\substack{n:\mathcal{N}\\ u:U}} (n.\mathsf{cons}.u \to \Box_{v:U}\ (n.\mathsf{inv}.v \to n.pay \to AppSys))$$

This system allows the same clerk to perform both operations, and a separation of duty mechanism is required to limit certain behaviours. Specification

$$Sep_u^v = STOP_{\{\mathsf{cons}.u,\mathsf{inv}.v\}} \| RUN_{\{\mathsf{cons}.v,\mathsf{inv}.u\}}$$

separates clerks u and v who may process invoices and consignment, respectively, but not vice-versa. If we assume that the infrastructure has only two clerks $U = \{x, y\}$ then a dynamic separation of duty mechanism, allowing a clerk vary operation between shipments is specified as $DynaSep$.

$$DynaSep = \|_{n:\mathcal{N}}(n{:}Sep_x^y \Box n{:}Sep_y^x)$$

$$StatSep = (\|_{n:\mathcal{N}} n{:}Sep_x^y) \Box (\|_{n:\mathcal{N}} n{:}Sep_y^x)$$

$StatSep$ describes a static separation of duty mechanism requiring a clerk to perform the same operation for all shipments. The overall (reliable) system is described as $SepSys = AppSys \| DynaSep$.

A reliable clerk u processing shipment n is expected to behave according to $n{:}Clerk^u$, where

$$Clerk^u = (\mathsf{snote} \to (\mathsf{cons}.u \to Clerk^u \mid \mathsf{inv}.u \to Clerk^u))$$

However, we make the assumption that, of our two clerks x and y, one may take on an unreliable or arbitrary behaviour. Thus, the unreliable infrastructure behaviour is $\overline{Clerks}$, where

$$\overline{Clerks} = \|_{n:\mathcal{N}}\ n{:}(Clerk^x \| RUN_{\alpha\, Clerk^y} \\ \Box\ Clerk^y \| RUN_{\alpha\, Clerk^x})$$

Since the system and separation mechanism ensures that one failing clerk cannot influence the generation of a payment, without the assistance of the other clerk, then, we can prove that for any $n : \mathcal{N}$ and $n{:}E = \{n.\mathsf{snote}, n.\mathsf{pay}\}$,

$$ConsReq \sqsubseteq^{n:E} (SepSys \| \overline{Clerks})$$

As currently defined, our specification favours the payment-enterprise, not the supplier: payments may be very late, or effectively not be made at all, but are never bogus. If a clerk fails then payment is not made. In reality, the infrastructure contains many additional components; audit logs to record failures and supervisors, who make judgements and rectify these inconsistencies. △

Example 5 Example 4 illustrates how separation of duties may be regarded as an implementation technique for achieving dependability. The implementation also maintains external consistency on shipments, since,

$$SepSys \| Clerks \sqsubseteq^{n:E} SepSys \| \overline{Clerks}$$

where $Clerks = \|_{n:\mathbb{N}} n{:}(Clerk_i^x \| Clerk_i^y)$ characterises a completely reliable infrastructure. △

3.2 Cryptographic Techniques

Our enterprise model is comparable to the network model used in the analysis of cryptographic-based authentication protocols [5, 15, 16]. The authentication protocol corresponds to the reliable system component being studied, while the network corresponds to the infrastructure, with the protocol attacker (Spy) choosing to have normal or abnormal behaviour.

Example 6 Suppose that the system and supplier (Example 1) share a secret cryptographic key (unknown to the clerk). The supplier includes a Message Authentication Code (MAC) with snote to ensure the authenticity of the note and this, in turn, provides authenticity for each invoice entered by the clerk.

Let $\mathcal{M}$ be a datatype representing shipment-identifiers plus associated MAC fields. Let $\mathcal{N}$ be the set of all values from $\mathcal{M}$ that represent cryptographically secured shipment-identifiers, that is, the MAC component corresponds correctly to the identifier. Let $\overline{\mathcal{N}}$ represent all other values in $\mathcal{M} \setminus \mathcal{N}$. The top-level requirement is as before, except that we expect only cryptographically secured shipment-identifiers to be used.

$$ConsReq = \|_{n:\mathcal{N}}(n{:}ConsReq_0)$$

The system will generate payment only for valid invoices that it has not seen before. A system that has processed $P \subseteq \mathcal{N}$ shipment-identifiers has behaviour

$$\begin{aligned} MacSys_P = & (\Box_{n:\mathcal{N}\setminus P}\ (n.\mathsf{inv} \to n.\mathsf{pay} \to MacSys_{P\cup\{n\}})) \\ & \Box \\ & (\Box_{n:\overline{\mathcal{N}}\cup P}\ (n.\mathsf{inv} \to MacSys_P)) \end{aligned}$$

Invoices processed in the past (P), or with invalid identifiers $\overline{\mathcal{N}}$ are processed, but payment is not generated.

A reliable clerk has behaviour $MClerk = \|_{n:\mathcal{N}}(n{:}Clerk)$ (Example 1). An unreliable clerk engages in arbitrary events, generating identifiers in $\overline{\mathcal{N}}$, and using identifiers it has already processed. However, we assume that the clerk cannot forge messages from $\mathcal{N}$.

$$\begin{aligned} \overline{MClerk}_P = & (\Box_{n:\mathcal{N}}\ (n.\mathsf{snote} \to Clerk_{P\cup\{n\}})) \\ & \Box \\ & (\Box_{n:\overline{\mathcal{N}}\cup P}\ (n.\mathsf{inv} \to Clerk_P)) \end{aligned}$$

Given this characterisation of an unreliable clerk we can prove that the resulting enterprise is as dependably safe as the original requirement, that is,

$$ConsReq \sqsubseteq^{n:E} MacSys_{\{\}} \| \overline{MClerk}_{\{\}}$$

Since our notion of dependability is independent of any particular implementation technique, it should be straightforward to combine different techniques. For example, we did not consider how the enterprise might establish the secret key between the supplier and the system. Suppose that a supervisor is given this responsibility. So long as the supervisor (infrastructure) and the snote-processing clerk are different people, then a failure by one cannot result in an unexpected behaviour at the external interface. This should be included as part of the implementation specification. △

The analysis performed in the example above is not unlike the approaches used in the analysis of authentication protocols [5, 15, 16]. A key difference is that we take a refinement approach while the other techniques may be viewed as verifying, what is in effect, a form of external consistency on an interface of an implementation. For example, verifying that external consistency is maintained at the interface of the supplier gives us

$$MacSys_{\{\}} \| MClerk \sqsubseteq^{n:E} MacSys_{\{\}} \| \overline{MClerk}_{\{\}}$$

In the case of an authentication protocol, external consistency is provided on the interfaces that make up the principles involved ('Alice' and 'Bob').

3.3 Confidentiality

Sections 3.1 and 3.2 illustrate that the attributes of integrity and authentication may be formalised in terms of dependency refinement. Confidentiality is a further attribute of dependability [13] and, for the sake of completeness, this section illustrates how multilevel security might be formally characterised in terms of refinement.

Example 7 By our fault model, the reliable part of a multilevel secure system is the TCB while the operating system and applications make up the unreliable infrastructure. The TCB has to be sufficiently robust to be able to provide an externally consistent interface to a low user regardless of the behaviour of a high application, that is, the TCB running a high application A_h is as dependably safe as TCB running any other high application A'_h. Or, in other words, that the TCB is externally consistent at the low interface.

$$\begin{aligned} & \forall\, A_l, A_h, A'_h \mid \alpha A_l = Lo \wedge \alpha A_h = \alpha A'_h = Hi \\ & \quad \bullet\ (A'_h \| TCB \| A_l) \sqsubseteq^{Lo} (A_h \| TCB \| A_l) \end{aligned}$$

This can be shown to simplify to $(TCB \| STOP_{Hi}) \sqsubseteq^{Lo} TCB$, and simplifies further to $(TCB \| STOP_{Hi})@Lo = TCB@Lo$. This corresponds to non-information flow [7, 12] as related to non-deducability [21]. If Lo and Hi partition the entire alphabet of TCB then it simplifies further to non-inference [14]: $TCB@Lo \subseteq TCB$. △

3.4 Fault-Tolerance

Another approach to dealing with unreliable systems (infrastructure) is to replicate the faulty components and make the system fault tolerant. We can make the payment enterprise fault tolerant if we replicate the clerk. We assume that every shipment is processed by $2k+1$ replicated clerks. The system votes(on the $2k+1$ invoices) to decide whether or not a consignment is valid. In this case, the abnormal behaviour of the infrastructure is represented by at least $k+1$ clerks having normal behaviour, and we argue that the resulting enterprise is as dependably safe as $ConsReq$ at interface $n{:}E$.

Non-interference techniques have been previously used to verify fault-tolerance [19, 23]. Faulty behaviour is modelled using special *fault* events and the system is fault-tolerant if the fault events are non-interfering with the critical events of the system. In essence, engaging a *fault* event changes the system from normal to abnormal behaviour, and what may be thought of as external consistency must be preserved on the the critical events that make up the external interface.

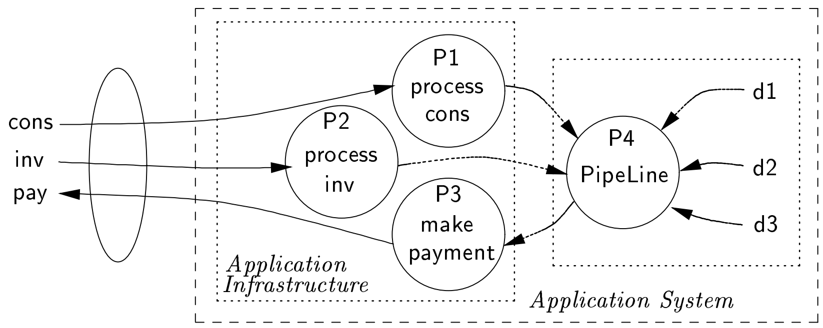

Figure 3: Application running on a TCB

3.5 Security Kernels

In Example 4 we considered the integrity of the enterprise with respect to the external supplier and assumed that *SepSys* was reliable, that is, secure. A conventional secure application system is usually built in terms of untrusted (unreliable) applications running on an underlying trusted computing base (TCB).

Example 8 Consider the application system used by the payment enterprise (Example 4). Figure 3 depicts a design of this system based on a simplistic model of an assured pipeline [3] composed of domains D1, D2 and D3. The applications form the infrastructure which is composed of programs P1, P2 and P3 which may run in domains D1, D2 and D3, respectively. The integrity of an application built on an assured pipeline relies on the separation enforced between domains, and the 'correctness' of the applications along the pipeline.

We specify a model of the assured pipeline—probably over-simplified, but serving as a useful illustration. Event $n.\mathsf{d1}$ represents entry into domain D1 by program $P1$ (processing shipment n). Events $n.\mathsf{d2}$ and $n.\mathsf{d3}$ have similar interpretations. The pipeline enforces a strict ordering on domain entry.

$$Pipeline = \Box_{n:\mathcal{N}} \; (n.\mathsf{d1} \to n.\mathsf{d2} \to n.\mathsf{d3} \to Pipeline)$$

When a cons event is engaged the program enters domain D1, and similarly for inv (these events will eventually be prefixed by shipment identifier).

$$P1 = \Box_{u:U} \; (\mathsf{cons}.u \to \mathsf{d1} \to P1)$$
$$P2 = \Box_{u:U} \; (\mathsf{inv}.u \to \mathsf{d2} \to P2)$$

The payment program P3 behaves slightly differently. Once the pipeline enters domain d3 a payment may be generated.

$$P3 = \mathsf{d3} \to \mathsf{pay} \to P3$$

Our failure model assumes that programs $P1$ or $P2$ may fail and engage arbitrarily events. Failure of program $P3$ can result in multiple payments and therefore it is necessary to treat the payment program P3 as a reliable component. This is not an unreasonable assumption: for example, a typical guard pipeline regards that part that generates the output as trusted [9]. Thus, the infrastructure is modelled as $\overline{Apps} = \|_{n:\mathcal{N}}(n{:}\overline{Trans})$, where $\overline{Trans}$ specifies the unreliable processing of a single shipment.

$$\overline{Trans} = ((P1 \| RUN_{\alpha P2}) \sqcap (P2 \| RUN_{\alpha P1})) \| P3$$

And we can prove that $AppSys \sqsubseteq^{\alpha AppSys} PipeLine \| \overline{Apps}$. $\triangle$

4 Evaluating Dependability

4.1 Dependability and Safety

It follows from its definition that trace refinement preserves dependability, that is,

$$\frac{R \sqsubseteq S}{R \sqsubseteq^{E} S} \quad [\; E \subseteq \alpha R \;]$$

However, the converse does not necessarily hold. For Example 7, we might prove that $TCB \| STOP_{Hi} \sqsubseteq TCB$ which, by the law above, implies that $TCB \| STOP_{Hi} \sqsubseteq^{Lo} TCB$ holds. However such a TCB is not of much use—for every trace t of TCB then $t \upharpoonright Hi = \langle\rangle$—it is not willing to engage in *any* Hi event!

If we take the view that refinement is a property [11], then since trace refinement is expressed as a predicate on traces it can be regarded as a safety property in the usual sense of [1]: the predicate $(t \in traces(R))$ holds for every trace t of S. On the other hand, local refinement is expressed as a predicate on sets of traces and we therefore regard it as an information-flow [12] or security property [18]: the predicate $(\exists\, t' : traces(R) \bullet t' \upharpoonright E = t \upharpoonright E)$ holds for every trace t of S. This also applies to external consistency and is not surprising in light of the examples studied in Section 3. Thus, we see no reason why our definition could not be re-cast in terms of other non-interference style frameworks such as [6, 17]. Doing this would provide access to a wide range of results on unwinding, composition, model-checking, verification, and so forth.

4.2 Incremental Evaluation

We interpret $R \sqsubseteq^{\alpha R} S \| I_S$, to mean that the system S is sufficiently resilient to the faults in I_S to be able to (safely) support the requirements R. This dependable component may then be used in place of R, which in turn, may be used in place of some other more abstract requirement. In general, the following law holds

$$\frac{R \sqsubseteq^{E} S \| I_S \;,\; S \sqsubseteq^{\alpha S} P \| I_P}{R \sqsubseteq^{E} (P \| I_P \| I_S)}$$

Example 9 We have from Example 4 and Example 8 that, for $n \in \mathcal{N}$ and $E = \{\mathsf{snote}, \mathsf{pay}\}$,

$$ConsReq \sqsubseteq^{n:E} AppSys \| DynaSep \| \overline{Clerks}$$
$$AppSys \sqsubseteq^{\alpha\, AppSys} PipeLine \| \overline{Apps}$$

and it follows that

$$ConsReq \sqsubseteq^{n:E} PipeLine \| \overline{Apps} \| DynaSep \| \overline{Clerks}$$

Thus, a TCB composed of the pipeline and dynamic separation of duty mechanism is sufficiently resilient to infrastructure failures (clerks and programs) and supports the original requirement *ConsReq*. $\triangle$

4.3 Composition

Under certain circumstances, if systems S and S' are dependable (according to requirements R and R') then so is their composition.

$$\frac{R \sqsubseteq^{E} S \;,\; R' \sqsubseteq^{E} S'}{R \| R' \sqsubseteq^{E} S \| S'} \quad [\, \alpha R \cap \alpha R' \subseteq E \,]$$

We note, however, that if the side-condition $\alpha R \cap \alpha R' \subseteq E$ does not hold then, $R \| R' \sqsubseteq^{E} S \| S'$ does not necessarily hold since synchronisation on events in $(\alpha R \cap \alpha R') \setminus E$ may result in behaviour restrictions in $R \| R'$ that are not restricted in $S \| S'$.

4.4 Unwinding

Given $t \in traces(P)$ then P/t is the process P after engaging in trace t and P/t may be viewed as a possible state of P. Thus, the set of all reachable states of P is $states(P) = \{t : traces(P) \bullet P/t\}$ and provides a way for us to view P as a state transition system. Engaging event $e \in \alpha P$ in state $p \in states(P)$ results in a new state $p/\langle e \rangle$.

Dependability refinement may be unwound into a condition on states and state transitions. An abstract state $r : states(R)$ is related to its concrete equivalent $s : states(S)$ by a refinement abstraction relation $r \approx s$. To prove that $R \sqsubseteq^{\alpha R} S$ it is necessary to prove that the result of transitions on concrete states are consistent with transitions on abstract states, as related by $\approx$. Formally, we have the rule

$$\frac{\begin{array}{l} \forall\, r : states(R);\; s : states(S);\; e : \alpha S \bullet \\ \quad r \approx s \wedge e \in \alpha R \Rightarrow r/\langle e \rangle \approx s/\langle e \rangle \\ \quad r \approx s \wedge e \notin \alpha R \Rightarrow r \approx s/\langle e \rangle \end{array}}{R \sqsubseteq^{\alpha R} S}$$

It is interesting to compare this with the unwound form for non-interference: $(r \approx s \wedge e \in \alpha R \Rightarrow r/\langle e \rangle \approx s/\langle e \rangle)$ is comparable to a no read-up rule, and $(r \approx s \wedge e \notin \alpha R \Rightarrow r \approx s/\langle e \rangle)$ is comparable to a no write-down rule.

5 Discussion

We think it more appropriate to refer to the kind of property reflected by local refinement as a *safe-dependability property*, rather than an information-flow or security property [18]. Being based on a traces model it is a safety-style property, but as argued in Section 4.1, more expressive. Alternative local refinement relations could be developed. For example, local refinement based on CSP's failures-divergences model would provide the basis of a liveness-dependability property.

A number of observations may be drawn from the examples in this paper. Throughout the paper it has been necessary to treat a $n.\mathsf{pay}$ output as being on a trusted path, that is, any component generating $n.\mathsf{pay}$ has to be reliable. In practice, if we know that only one message can be output at the end of an assured pipeline (as in [9]), then we could regard the P3-make-payment program (Example 8) as a potentially unreliable filter or integrity verification procedure (IVP), whose failure cannot result in the generation of multiple $n.\mathsf{pay}$ outputs.

By choosing to support only one unit of payment (no payment amount) we avoided the problem of a failing program modifying the payment amount. In a practical system such a failure should be detected at some point by appropriate double-entry book-keeping on payments and invoices, and dealt with by generating an additional payment or an invoice. If payment is viewed as something that can occur in stages then we believe that such a system, if specified properly, could be shown to be dependable.

6 Conclusion

By considering the nature of the entire enterprise we provide a meaningful and implementation-independent definition for integrity and dependability in general. This systems view has not been adopted by conventional integrity models, such as Clark-Wilson [4], which limit themselves to the boundary of the computer system and tend to define integrity in an operational/implementation-oriented sense.

In some respects, our definition of dependability blurs the distinction between the attributes discussed in this paper (integrity, authentication and confidentiality); indeed, the Clark-Wilson model incorporates authentication as one component (rule E1) of its model of integrity. Example 3.2 illustrates that, what are in effect authentication techniques, may be used to achieve external consistency, that is, integrity (in the Clark-Wilson sense). Therefore, as in [8], we speculate that the verification of 'security' should be regarded as the verification of correctness. In this paper we use local refinement and a fault model articulates the nature of the possible attacks on the system. This suggests a paradigm for the development of a secure system:

1. Develop top-level Requirements.
2. Design an implementation, incorporating a fault model.
3. Verify that the implementation refines the requirement.

If a top-level requirement is not available then external consistency may be verified.

Acknowledgements

This work was done while I was a member of the CCSR, on leave from University College, Cork. I'd like to express my gratitude to Stewart Lee for inviting me and thank him, and the members of the Cambridge Computer Security Group, for a most enjoyable visit. Thanks also to Kan Zhang for discussions that helped to solidify my understanding of external consistency and to Dieter Gollman who suggested that external consistency might be a form of dependability. This work was supported, in part, by a basic research grant from Forbairt (Ireland).

References

[1] Alpern, B., and Schneider, F. Recognizing safety and liveness. *Distributed Computing 2* (1987), 181–126.

[2] Biba, K. Integrity considerations for secure computer systems. Tech. Rep. MTR-3153 Rev 1 (ESD-TR-76-372), MITRE Corp Bedford MA, 1976.

[3] Bobert, W., and Kain, R. A practical alternative to hierarchical integrity properties. In *Proceedings of the National Computer Security Conference* (1985), pp. 18–27.

[4] Clark, D. D., and Wilson, D. R. A comparison of commercial and military computer security models. In *Proceedings Symposium on Security and Privacy* (Apr. 1987), IEEE Computer Society Press, pp. 184–194.

[5] Focardi, R., Ghelli, A., and Gorrieri, R. Using noninterference for the analysis of security protocols. In *Proceedings of DIMACS Workshop on Design and Formal Verification of Security Protocols* (1997).

[6] Focardi, R., and Gorrieri, R. A taxonomy of security properties. *Journal of Computer Security 3*, 1 (1994).

[7] Foley, S. *A Model and Theory of Secure Information Flow.* PhD thesis, National University of Ireland, 1988.

[8] Good, D. A position on computer security foundations. *IEEE Cipher Newsletter* (Jan. 1989), 24–25.

[9] Greve, P., Hoffman, J., and Smith, R. Using type enforcement to assure a configurable guard. In *Proceedings of the 13th. Annual Computer Security Applications Conference* (1997).

[10] Hoare, C. *Communicating Sequential Processes.* Prentice Hall, 1985.

[11] Jacob, J. The varieties of refinement. In *Proceedings of the 4th Refinement Workshop* (1991), J. M. Morris and R. C. Shaw, Eds., Springer-Verlag, pp. 441–455.

[12] Jacob, J. Basic theorems about security. *Journal of Computer Security 1*, 4 (1992), 385–411.

[13] Laprie(ed.), J. *Dependability: Basic Concepts and Terminology.* Springer Verlag. IFIP WG 10.4-Dependable Computing and Fault Tolerance.

[14] O'Halloran, C. M. A calculus of information flow. In *Proceedings of the European Symposium on Research in Computer Security* (Oct. 1990), G. Eizenberg, Ed., AFCET, pp. 147–159.

[15] Paulson, L. The inductive approach to verifying cryptographic protocols. In *Proceedings of the IEEE Computer Security Foundations Workshop* (1997).

[16] Roscoe, A. Using intensional specifications of security protocols. In *Proceedings of the IEEE Computer Security Foundations Workshop* (1996).

[17] Roscoe, A., Woodcock, J., and Wulf, L. Non-interference through determinism. *Journal of Computer Security 4*, 1 (1995).

[18] Schneider, F. Enforcable security policies. Tech. Rep. TR98-1664, Cornell University, Jan. 1998.

[19] Simpson, A. *Safety through Security.* PhD thesis, Oxford University, Computing Laboratory, 1996.

[20] S.M. McMenamin, J. P. *Essential Systems Analysis.* Prentice Hall, 1984.

[21] Sutherland, D. A model of information. In *Proceedings 9th National Computer Security Conference* (1986), U. S. National Computer Security Center and U. S. National Bureau of Standards.

[22] U. S. Department of Defense. Integrity-oriented control objectives: Proposed revisions to the trusted computer system evaluation criteria (TCSEC). Tech. Rep. DOD 5200.28-STD, U. S. National Computer Security Center, Oct. 1991.

[23] Weber, D. Specifications for fault-torerance. Tech. Rep. 19-3, Odyssey Research Associates, Ithaca, NY, 1988.

A Appendix

Lemma 1 Given processes P and Q and interface E then

$$(P\|Q)@E \subseteq (P@E)\|(Q@E)$$
$$\alpha P \cap \alpha E \subseteq E \Rightarrow (P\|Q)@E = P@E\|Q@E$$

Proof: Found in [7]. □

Theorem 1 Given systems S and P, and their corresponding infrastructures $\overline{I}_S$ and $\overline{I}_P$, then

$$R \sqsubseteq^E S\|\overline{I}_S \wedge S \sqsubseteq^{\alpha S} P\|\overline{I}_P$$
$$\Rightarrow R \sqsubseteq^E (P\|\overline{I}_P\|\overline{I}_S)$$

Proof: If $S \sqsubseteq^{\alpha S} P\|\overline{I}_P$, then $t \in traces(P\|\overline{I}_P) \Rightarrow t \upharpoonright \alpha S \in traces(S)$, implies that $t \in traces(P\|\overline{I}_P\|\overline{I}_S) \Rightarrow t \upharpoonright (\alpha S \cup \alpha\overline{I}_S) \in traces(S\|\overline{I}_S)$. Thus, $S \sqsubseteq^{\alpha S} P\|\overline{I}_P$ implies that $(P\|\overline{I}_P\|\overline{I}_S)@(\alpha S \cup \overline{I}_S) \subseteq traces(S\|\overline{I}_S)$, and since $E \subseteq \alpha R \subseteq \alpha S \cup \alpha\overline{I}_S$, then it follows that $(P\|\overline{I}_P\|\overline{I}_S)@E \subseteq (S\|\overline{I}_S)@E$ and from the hypothesis $(S\|\overline{I}_S) \subseteq traces(R)$, and by transitivity of $\subseteq$ the theorem follows. □

Theorem 2 Given requirements R and R' and systems S and S' and an interface E such that $\alpha R \cap \alpha R' \subseteq E$, then

$$R \sqsubseteq^E S \wedge R' \sqsubseteq^E S'$$
$$\Rightarrow R\|R' \sqsubseteq^E S\|S'$$

Proof: If $S@E \subseteq R@E$ and $S'@E \subseteq R'@E$, then it follows that $S@E\|S'@E \subseteq R@E\|R'@E$. Since, by definition $\alpha R \subseteq \alpha S$ and hypothesis $\alpha R \cap \alpha R' \subseteq E$, we have $E \subseteq \alpha S$ and, similarly, $E \subseteq \alpha S'$. Lemma 1 implies that $(S\|S')@E \subseteq S@E\|S'@E$ and since $\alpha R \cap \alpha R' \subseteq E$ then $(R\|R')@E = R@E\|R'@E$. Thus, $(S\|S')@E \subseteq (R\|R')@E$ and the theorem follows.

We should note that if $\alpha R \cap \alpha R' \subseteq E$ does not hold then, from Lemma 1, $(P\|Q)@E = P@E\|Q@E$ does not necessarily hold and thus $R \sqsubseteq^E S \wedge R' \sqsubseteq^E S' \Rightarrow R\|R' \sqsubseteq^E S\|S'$ does not hold in general. □

Theorem 3 Given requirement R and system S then there exists a suitable abstraction relation $\approx$ such that then

$$
\begin{array}{l}
(\forall\, r : states(R);\ s : states(S);\ e : \alpha S \bullet \\
\quad r \approx s \wedge e \in \alpha R \Rightarrow r/\langle e\rangle \approx s/\langle e\rangle \\
\quad r \approx s \wedge e \notin \alpha R \Rightarrow r \approx s/\langle e\rangle) \\
\Rightarrow R \sqsubseteq^{\alpha R} S
\end{array}
$$

PROOF SKETCH: Semantically, the trace model may only be used to reason about deterministic systems, and its corresponding state-transition machine—a labelled transition system—is deterministic. The usual relationship between trace refinement and (safety) refinement for a labelled transition system implies that:

$$
\begin{array}{l}
(\forall\, r : states(R);\ s : states(S@\alpha R);\ e : \alpha R \bullet \\
\quad r \approx s \Rightarrow r/\langle e\rangle \approx s/\langle e\rangle) \Rightarrow R \sqsubseteq S@\alpha R
\end{array}
$$

where $\approx$ is a suitable abstraction relation. The set $states(S@\alpha R)$ effectively induce a set of equivalence classes on the set $states(S)$ and we can re-construct the abstraction relation $\approx$ to preserve this relationship, such that,

$$
\begin{array}{l}
(\forall\, r : states(R);\ s : states(S);\ e : \alpha R \bullet \\
\quad r \approx s \Rightarrow r/\langle e\rangle \approx s/\langle e\rangle) \wedge \\
(\forall\, r : states(R);\ s : states(S);\ e : \alpha S \setminus \alpha R \bullet \\
\quad r \approx s \Rightarrow r \approx s/\langle e\rangle) \\
\quad \Rightarrow R \sqsubseteq S@\alpha R
\end{array}
$$

where, transitions on events $e \in \alpha S \setminus \alpha R$ are viewed as 'internal' events (to an interface αR) that keep a state (of S) in the same equivalence class.

The unwinding is also a sufficient condition for local refinement. If $S\alpha R \subset R$ then define an abstraction relation such that $(R/t_r \approx S/t_s) \Leftrightarrow t_s \upharpoonright \alpha R = t_r$, for $t_r \in traces(R)$ and $t_s \in traces(S)$, and the unwinding conditions follow. $\square$

B Communicating Sequential Processes

In the traces model of CSP [10] the behaviour of a process is represented by a prefix-closed set of event traces. If P is a process then $traces(P) \subseteq (\alpha P)^*$ gives its traces and αP its alphabet. We use a subset of the CSP algebra to specify system behaviour; the trace semantics of the operators used is given below.

$$
\begin{array}{ll}
traces(STOP_A) & = \{\langle\rangle\} \\
traces(RUN_A) & = A^* \\
traces(a \to P) & = \{\, t : traces(P) \bullet \langle a\rangle \frown t \,\} \cup \{\langle\rangle\} \\
traces((a \to P & \\
\qquad \mid b \to Q)) & = traces(a \to P) \cup traces(b \to Q) \\
traces(P \,\square\, Q) & = traces(P) \cup traces(Q) \\
traces(P \| Q) & = \{\, t : (\alpha P \cup \alpha Q)^* \mid \\
 & \qquad t \upharpoonright \alpha P \in traces(P) \wedge \\
 & \qquad t \upharpoonright \alpha Q \in traces(Q) \,\}
\end{array}
$$

While not used for specifying processes, the after operator is useful for reasoning about processes in an abstract manner.

$$traces(P/t) = \{\, s : (\alpha P)^* \mid t \frown s \in traces(P) \,\}$$

In the paper we also used a indexed form of concurrency and external choice. For example, $(\|_{i:I} P(i))$ corresponds to the concurrent composition of each $P(i)$ indexed over $i : I$. Processes may also be specified recursively, in the form $P = F(P)$. For example, $P = a \to P$, which has a unique fixed-point—a process that repeatedly engages in event a.