TCO-aware provisioning of information security infrastructure

Philip Robinson, Bryan Stephenson

HP Laboratories
HPL-2008-195

Keyword(s):
TCO automated security infrastructure provisioning

Abstract:

As IT infrastructure proliferates, higher-level security controls must become interpretable and enforceable by machines, largely without human intervention. We establish a single provisioning model and process for addressing security and the Total Cost of Ownership (TCO). The innovation is the combination of these two aspects of provisioning hosted software in the same process and model. Provisions are made for comparing alternatives and justifying design and mechanism-selection decisions, based on their impact on the TCO of the system being protected. While TCO calculations may be estimations and qualitative, quantitative arguments can still be incorporated to validate all security infrastructure provisioning decisions made with respect to both security and TCO.

External Posting Date: November 21, 2008 [Fulltext] Approved for External Publication
Internal Posting Date: November 21, 2008 [Fulltext]

TCO-aware provisioning of information security infrastructure

Philip Robinson
SAP Research
Belfast, Northern Ireland
philip.robinson@sap.com

Bryan Stephenson
HP Laboratories
Palo Alto, California
bryan.stephenson@hp.com

Abstract

As IT infrastructure proliferates, higher-level security controls must become interpretable and enforceable by machines, largely without human intervention. We establish a single provisioning model and process for addressing security and the Total Cost of Ownership (TCO). The innovation is the combination of these two aspects of provisioning hosted software in the same process and model. Provisions are made for comparing alternatives and justifying design and mechanism-selection decisions, based on their impact on the TCO of the system being protected. While TCO calculations may be estimations and qualitative, quantitative arguments can still be incorporated to validate all security infrastructure provisioning decisions made with respect to both security and TCO.

1 Introduction

With advances in storage and networking technology, as well as the rise of trusted providers of these resources, businesses and individuals are starting to access infrastructure and software as services remotely, as opposed to installing them locally within their own domain. This motivates a shift of IT management to a concept of *Everything as a Service*[15]. This shift has the potential to reduce customer costs but also the potential to decrease their security. From the provider's perspective, they have to care about their own internal costs and security, as well as those of all their customers, their varying needs and varying assessments of risk. This describes the problem of security infrastructure provisioning. Provisioning and management often rely on a team of experts in various areas such as networking, storage, applications, OS, security, performance, availability and scalability. However, using separate processes and different models can lead to the situation where the fulfillment of one operational goal breaks the other. For example, performing provisioning of security infrastructure configuration of a system followed by provisioning for costs can destroy the security properties. Our focus and contribution in this paper is on building an infrastructure model that supports the analysis and design of both security and costs. Secondly, by capturing the infrasructure model as predefined templates, the operational burdens on administrators are reduced. They work on a higher-level of template selection and parameterisation, which help them to deal with the complexities and specialities of the system and its environment. The specification and handling of these templates is implemented within a previously published systems management framework known as the Model Information Flow (MIF) [3]. The MIF seeks to reduce cost and accelerate delivery and evolution of the system using a model-driven holistic approach to simultaneously treat all functional and non-functional requirements, thus enabling a much better and faster understanding of the effects that potential system designs or changes would have on those functional and non-functional requirements. We therefore contribute here by integrating security concerns throughout the MIF process, where security is a non-functional requirement in the MIF.

It requires considerable skill to define the processes required to secure the initial system deployment and ensure that proper security is maintained throughout the system's lifetime. This makes the *Total Cost*

of Ownership (TCO) of the system expensive. Ding, Yurcik and Yin[7], in their 2005 economic analysis of outsourcing Internet security, estimate that a security engineer for the role of security device management costs between $8000 to $16,000 per month. Lampson [13] states that the main reason many systems are vulnerable to attack is that security is expensive and a nuisance to maintain. Additionally, he makes the clear statement that *simpler setup is the most important step toward better security*. Accordingly, our goal is to enable the efficient and secure delivery, operation, and governance of complex distributed applications at very large scale. We show that it is possible to formally model in one location all security controls for a complex service and, using an incremental approach, systematically refine these models to executable infrastructure configurations. Using a series of model transformations combined with deployment engines, the complete set of desired security controls can be automatically deployed and configured, using a variety of security mechanisms. Programatic verification that all desired security controls are in place in the running infrastructure is also made possible. A process for secure hosted service delivery and service life-cycles is then established, which is based on incremental configuration and is *TCO-aware* at each stage. TCO-awareness suggests that provisions are made for comparing alternatives and justifying design and mechanism-selection decisions, based on their impact on the TCO of the system being protected. While TCO calculations may be estimations and qualitative, we still use quantitative, empirical arguments to validate any provisioning decisions made.

In section 2 we provide background on the Model Infromation Flow (MIF) and Total Cost of Ownership (TCO). In section 3 we describe the Observer-Controller-Compartment model which we use to represent the single model for analysis of security and TCO. In section 4 we describe the architecture and process used for implementing the model. Section 5 shows a case study. Section 6 discusses our approach relative to related work in this area, and section 7 concludes.

2 Background

2.1 The Model Information Flow

The *Model Information Flow* or MIF [3] was developed as part of a joint research project between SAP Research and HP Labs. The MIF is a conceptual framework and associated model-driven tooling for service lifecycle management. It links three viewpoints:

1. The design of the business processes;
2. The applications that implement those business processes;
3. The IT infrastructure (virtual and physical) hosting these applications.

This enables a change in one viewpoint to be linked to changes in other viewpoints. For example it links a change in the business process to the necessary changes in application configurations and infrastructure topologies. Goals of this approach include reducing cost; improving security, performance, availability, and other desirable aspects; and making it much easier and more predictable to evolve the system over time.

The principle steps of the MIF are shown below in Figure 1. It starts on the left hand side with a catalogue of business processes from which one is chosen and proceeds with the design of the application and infrastructure, onto the deployed and running system. Cycles can occur at every step, but are not shown for reasons of simplicity. Security requirements are modelled and refined at each step as described in Section 4.2. The high-level security requirements are refined in a traceable manner into a selection of security mechanisms and their appropriate configurations. The refinement takes the TCO of the security mechanisms into account at each refinement stage.

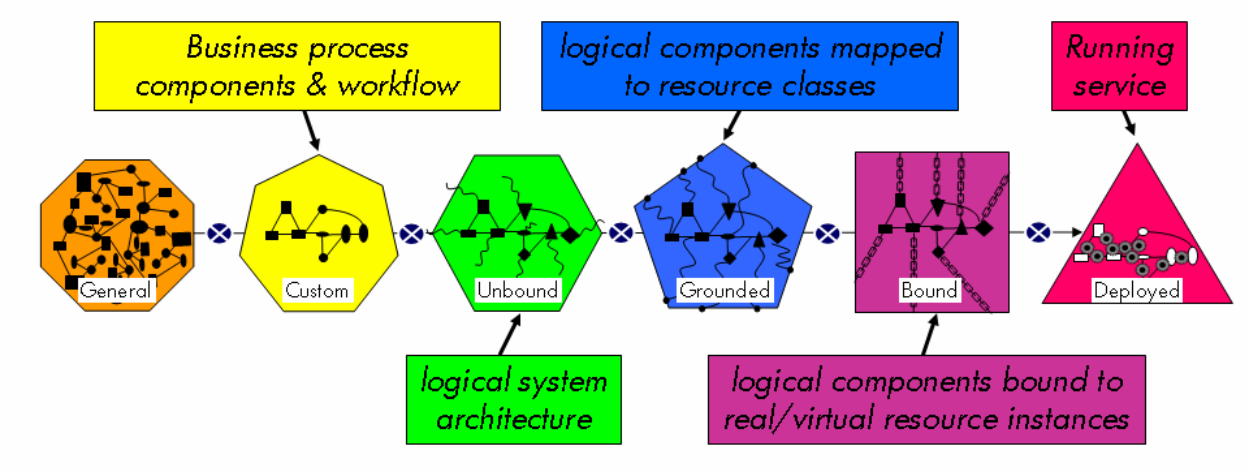

Figure 1: The Model Information Flow

2.2 Total Cost of Ownership

The *Total Cost of Ownerhip (TCO)*, as first introduced by Gartner in 1987[9], is a measure of one-time, initial costs and recurring maintenance and operations costs of an IT product/solution. If the TCO is larger than the *Return on Investment (ROI)*, then a customer is advised not to purchase the solution. From a service provider's perspective, if the service's TCO is larger than their revenues from the service, then they lose money on the service. TCO considers interdependent factors such as time, money and people involved in setting up and operating the solution, but there is no single standard formula or metrics. However, from the literature, we understand Equation1 to be representative of the general understanding of TCO determination:

$$TCO(E) = cost_0(E) + \frac{\sum_{t=1}^{t=n} cost_t(E)}{T} \quad (1)$$

Equation 1 states that the TCO for a set of infrastructure elements $E = (e_1, ...e_n)$ is equal to the intial costs $cost_0$ of setting up E plus the sum of the estimated cost $cost_t$ of maintaining E at different milestones t, given an estimated, overall system lifetime T.

Patel and Shah [14] provide a cost model for Data Centers, which identifies factors such as power, cooling and maintenance as the main contributors to TCO from the Data Center's perspective. Absolute costs, cost functions or cost classes can then be selected in order to determine $cost_t(E)$. The design process of the MIF generates a candidate design or *grounded model* for a proposed service infrastructure which will deliver the business process at the levels of performance, availability, security, etc. specified by the customer. The *grounded model* includes precise specifications of all physical and virtual resources needed to build the service at the requested levels of service, for example the list in table 1.

The TCO of each needed infrastructure element can be determined in any manner the service provider chooses. TCO of each VM is often determined by using a fixed cost to intially provision the VM ($50 in this example) and adding a cost factor for each quanta of variable resource specified, such as CPU shares, memory, and disk storage. TCO of software applications, monitoring services, and other software or services is often calculated by amoritzing the cost of the licenses over the license period (or cost of the service over the service period), and then adding factors to account for things like less than 100% utilization of the licenses, periodic manual administration tasks, etc. The TCO of all the infrastructure elements can be summed to find the TCO for the service. This calculated total TCO for the service provider to deliver the service can be adjusted to account for things like account management overhead, customer discounts, a markup for profit, etc. We however wish to emphasize that it is possible for an analyst to use any methods they wish to calculate

QTY	Element	Initial Cost	Cost per day
1	VM with 10 CPU shares, 8 GB memory, and 400 GB disk	50	50
4	VM with 4 CPU shares, 3 GB memory, and 100 GB disk	50	15
1	VM with 7 CPU shares, 2 GB memory, and 200 GB disk	50	25
1	Specified Sales and Distribution software	0	25
1	Specified DB software	0	5
1	IPTables software	0	3
1	Application Monitoring service	0	10

Table 1: Bill of Materials for proposed service

TCO and price the services for the customers. The real challenge we address is that of making estimates about the impact of security requirements on the TCO and how to reuse this analysis in the derivation of suitable deployment specifications.

2.3 Towards a Single Provisioning Model and Process for TCO and Security

In order to reuse the analytical work done by a *TCO analyst*, we follow the declarative, model-based approach introduced by the Model Information Flow (MIF) framework [3]. Using this approach, a security analyst, engineer or administrator defines high-level models of the security infrastructure, which may be realized using different configurations of infrastructure elements $C = (config(E)_1, ..., config(E)_n)$. Each $config(E)$ varies in its TCO calculation from the customer or provider perspective, as exemplified in Table 1.

Recall that there is assumed to be a preliminary negotiation beforehand that sees the customer agreeing to a cost model that fits with their expected ROI and the provider agreeing to a configuration and cost model that meets their profitability expectations. On selecting a configuration, the system is able to refine a general security infrastructure model to a deployed security infrastructure model. However, each configuration varies with respect to its security guarantees and the associated TCO. The intentions of the model-based approach are to reduce human error and foster a more understandable and rationalizable approach to security, as well as its relation to TCO. The security infrastructure models are created alongside the business process models and operational models to ensure changes in business process also include security considerations. A crucial benefit of the approach is that it provides a framework within which both people and software can reason about and manage change. Changes in the requirements of the system, resulting from changes in business need (including costs), lead to matched changes in the application and infrastructure models and configurations. For example, an administrator can change the security infrastructure model to require isolation of the database. This causes the automatic generation of a new configuration description, which now includes a firewall between the database servers and the application servers with the proper parameter settings to provide the required level of protection. The administrator doesn't need to re-design the new network topology, decide which firewall ports need to be open, or determine any other details of the firewall configuration. The models capture such information and enable software to design the new infrastructure and start it running. Importantly, these security changes are not done irrespective of other operational concerns e.g. TCO. Rather, the general model of the system is modified, such that tools which handle other areas of concern, like performance, can evaluate the impact of the changes and make the necessary adjustments. Continuing the example, the addition of another firewall into the new service delivery infrastructure design may cause additional delays in the end-to-end response time. The performance modelling tools will address this when determining the back-end infrastructure needed, and if necessary will generate a slightly higher performance design for the back-end to compensate. This should ensure that end-to-end response time is not affected by the addition of the firewall, and the overall user experience and system performance

characteristics are still within the specified requirements. Again, the administrator doesn't need to repeatedly bother about these details; the MIF models and tooling perform the necessary calculations and generate a new infrastructure design which still meets all requirements.

3 Security Infrastructure Model for Supporting TCO Analysis

3.1 The Elements of Security Infrastructure Provisioning

In this section we define the basic modeling elements and theorems used in our approach to TCO-aware security infrastructure provisioning. The main concept from which the elements and theorems are derived is that of *compartmentalization*. We then refer to the TCO of security infrastructure as the operational costs associated with initializing and maintaining a logical or physical *Compartment* around a set of infrastructure elements. The Compartment concept extends the work of Franke and Robinson [8], but we do not place an emphasis on Service Level Agreements (SLAs) here. However, the topics of SLA management and maintaining a low TCO are complementary, as the agreed TCO may be based on terms and guarantees in a SLA.

In order to provide a formal baseline for our methodology, the terms Compartment, Infrastructure Element, Security Control Policy, and Security Control Element are first defined below:

Definition 1: A *Compartment* C is a logically or physically enforced boundary that 'encapsulates' a set of infrastructure elements E. A Compartment C has the following composition:

- 0 to n `Infrastructure Elements` $E = (e_1, ..., e_n)$ whose existence and operations are physically or logically encapsulated by the Compartment. Encapsulation refers to either aggregation, composition or embedding. Using object-oriented concepts, aggregation implies that an infrastructure element e can exist beyond its encapsulating Compartment's lifetime, while Composition suggests that the destruction of the Compartment is also the end of the infrastructure element's lifetime. However, if an infrastructure element is embedded in the Compartment, loss or compromise of the respective element results in the loss of some core functionality and capability of the Compartment e.g. consider loss or compromise of a reference monitor.

- 1 to n `Gateways` $CG = (cg_1, ..., cg_n)$ that restrict the transfer and communication of and with the infrastructure elements in the Compartment from external elements or Participants. An ideal case is $n = 1$ following the Single Access Point pattern from Yoder and Baracalow [21], but, for purposes of performance, flexibility, heterogeneity or scalablility, this is not always feasible.

- 1 to n known (non covert) `Channels` $CC = (cc_1, ..., cc_n)$, which are strictly for communication i.e. event transfer between elements within the same Compartment.

- 1 to n `Participants` $CP = (cp_1, ..., cp_n)$ that have certain privileges within the Compartment ranging from ownership and administration to simple read access to data dt stored in different infrastructure elements. The participant's identity $cp.id$ is maintained by one or more infrastructure elements e, such that, in some cases `cp.id` $\leftrightarrow$ `e.id`

- 1 to n `Keystores` CK, which contain either symmetric or asymmetric keys used for authentication and secure communication.

- 0 to n immediate `Parent Compartments` $C \uparrow$, where $C_i \uparrow C_j$ states that a compartment C_i is parent of and encapsulates C_j, such that C_j has the behavior of an infrastructure element from the viewpoint of C_i. Note that we allow n parents, as the possibility for overlapping and partial

encapsulation occur in the real world (we however have not fully investigated the impact of this case but still make provisions in the framework, keeping the default as n=1). That is, n here does not indicate layers or levels of nesting.

- A set of parameters $CF = (cf_1, ..., cf_n)$ that define facts about the Compartment's operational features. It is noted that these features inevitably change the security and TCO assumptions about C:
  - `Physicality`: is a boolean parameter that states if the Compartment has a physical or virtual manifestation. For example, consider the variation of a physical machine vs a virtual machine. A physical compartment is often more secure than a virtual one but less flexible and more expensive[8].
  - `Accessibility`: is a scalar parameter that describes the accessibility level of a Compartment with respect to a top-most, outer-most root parent Compartment $C \uparrow^{root}$ e.g. a service provider or Data Center could be a root. Note that the accessibility of a Compartment is hypothetically an inverse of the capability required by an attacker.
  - `Mobility`: is a boolean parameter that describes if the Compartment can be moved from its current physical location and/or logical parent Compartment. Mobility of Compartments often introduces additional risks as all assumptions upon which security are built may no longer hold.
  - `Lifetime`: defines the time-based or expiry constraint on the runtime of the Compartment. An indefinite lifetime $t = *$ indicates a permanent Compartment in place that may be providing some ongoing infrastructure support.
  - `Element Capacities`: upperbounds on how many infrastructure elements of specific types can be contained by the Compartment. The unit used to measure capacity depends on the type of Compartment and the type of infrastructure elements it can contain.
  - `Gateway Capacities`: describes the number of infrastructure elements and events that can be concurrently transferred through a Gateway. Note that the semantics of *transfer* tries to capture both electronic, networked communication and serialization, as well as the actual, physical movement of elements depending on their `Mobility` feature.

Definition 2: an *Infrastructure Element (e)* is any self-contained hardware or software entity in an IT infrastructure of a particular type `etype` that is uniquely addressable with an identifer `e.id`, performs and requests specific services (e.g. communications, processing, storage, monitoring etc) in the infrasructure, is accessible via well-specified functional and management interfaces and throughout the runtime of the infrastructure has some responsibilities with respect to the maintenance of a set of data. Note that a Compartment is itself an infrastructure element and the same properties are assumed.

- a *target-element (te)* is an infrastructure element that will reveal knowledge or change its state by manipulating its current data dt, as a result of an action ac being committed.
- *a source-element (se)* is an infrastructure element that initiates an action ac in order to change its knowledge and/or the state of one or more target elements in the infrastructure.

The system landscape we consider is heterogeneous, distributed and dynamic, such that centralized, policy-based management and control is desirable, as is supported by many of the systems management concepts we reference[11, 5, 6, 20]. We now define the semantics and high-level syntax of the *security control policies* we incorporate for provisioning:

Definition 3: A *Security Control Policy p* is a directive executed in a security infrastructure, implementing the following quintuple `((source se, target te, action ac, data dt, [context Q]))`.

This definition of security control policies is similar to the Event-Condition-Action (ECA) policies defined by Ponder[5], but `source` and `target` refer explicitly to `source-elements (se)` and `target elements (te)` in our conceptualization. We limit the `action (ac)` parameter to `ac` $\leftarrow$ `('read', 'write', 'delete', 'execute', 'open', 'get', 'put')`. The data elements `(dt)` included in a policy refer to either a reference to data maintained by a target `te` or a payload (e.g. arguments) associated with the action `ac` to be performed. Finally, we leave the optional `context [Q]` as a free-form expression at this level of description, but assume it to be interpretable and executable by the relevant element in the security infrastructure. However, the `time t` provided by a timer in the Compartment is a concrete form of context that is often important for various security controls. The next section proceeds to describe a pattern used to abstract the elements of a security infrastructure and their interactions in order to protect a Compartment.

3.2 Compartments and the Observer-Controller Security Pattern

This section introduces our extension of the observer/controller (o/c) pattern. Schöler and Müller-Schloer[17] also present a version of an o/c pattern with very similar semantics to ours, such that one can refer to their work for conceptual details unrelated to security. Our realization of the o/c pattern models the entities and interactions that enforce the Compartment's encapsulation properties of infrastructure elements. Observers and Controllers are special, trusted infrastructure elements referred to as *security control elements (SCE)*, such that $E^{SCE} \subset E$ and the remaining infrastructrure elements are referred to as the set of *managed elements* E^{ME}, where $E^{ME} \cap E^{SCE} = \emptyset$. That is, we ensure that there is a clear distinction between the responsibility of trusted and untrusted/ security-capable and security-incapable elements in the infrastructure. Similarly, with reference to Equation 1, $TCO(E) = TCO(E^{SCE}) + TCO(E^{ME})$. The goals of using the o/c pattern are twofold: (1) to model the infrastructure required to enforce the security control policies of a Compartment C and (2) to refine the calculation of $TCO(E^{SCE})$ with respect to costs associated with instances of the pattern. Before proceeding, we first re-emphasize and refine the security principles of a Compartment as Axioms (1 - 6), previously defined by Franke and Robinson [8], in order to clarify the enforcement assumptions and objectives of an instance of the o/c pattern:

- **Axiom 1**: data maintained by infrastructure elements E in one Compartment C cannot be read outside the Compartment without knowledge of a secret (e.g. symmetric key, trusted private key, username/password) stored within the Compartment
- **Axiom 2**: the source `se` and target `te` of any event `ev`, its action `ac` and data `dt` must be authenticatable in accordance with Axiom 1, else the event cannot be allowed in or out of the Compartment by default
- **Axiom 3**: all events or elements transferred in and out of the Compartment C must go via one of the Compartment's registered Gateways CG; all internal events go via non-covertly created Channels CC between elements encapsulated in the Compartment;
- **Axiom 4**: any action ac on a target element te in a Compartment can only be granted if the the Gateway cg or Channel cc via which the event is transferred has 1 or n explicit security control policies $(p_1, ..., p_n)$ that permit the action (we avoid negative policies)
- **Axiom 5**: fail events ev^{-1} of elements E_i in a Compartment C_i do not cause the elements E_j of

[1] We do not go into detail of fail events in this paper but include them for completion

another Compartment C_j to fail and remain unknown, given that $C_i \cap C_j = \emptyset$ holds - note that this covers the case of nesting as $C_i \uparrow C_j \Rightarrow (C_i \cap C_j = C_j)$.

- **Axiom 6**: every Gateway and Channel in a Compartment is observable and controllable by some set of trusted elements e.g. in the case of a Gateway, $E^{SCE}(cg) \subseteq E^{SCE}$

Given the above Axioms and their requirements placed on the trusted security control elements E^{SCE}, we now proceed to specify the modeling of these elements and their interactions using the o/c pattern.

Definition 4: a *Security Control Element* is a special, trusted infrastructure element that provides protection services of *event monitoring* and *policy enforcement* in a system infrastructure.

- an *Observer* obs_i is a Security Control Element that monitors/listens for events or interactions of a specific type in its set of observations O^i. On observing an event $o^i \in O^i$, it notifies one or more Elements registered or encoded as requiring the knowledge of o.

- a *Controller* ctr_j is a Security Control Element that registers with one or more Observers for a set of events $O^j = (o^j_1, ..., o^j_n)$ and, on being notified of the occurence of an event $o^j \in O^j$, executes a control action act^j according to a policy p^j, where $act^j \leftarrow (assert, allow, deny, commit)$.

The interaction model for the o/c pattern is purposely simple, to avoid highly complex interaction protocols that may lead to flaws: observers recieve/intercept events $se \rightarrow obs : ev$, observers notify registered controllers $obs \rightarrow ctr : ev$, controllers make policy decisions and act on their decisions by either changing the event state $ctr \rightarrow ev : ev^*$ or that of its target $ctr \rightarrow te : te*$. We also assume that all interactions are a-priori secured between elements in E^{SCE}, or that they are physically bound or equivalent i.e. refer to the same device or process but define different functional views. We further define a typology and dependency mapping on the classes of security controls we consider, depicted in Figure 3.2 and described afterwards.

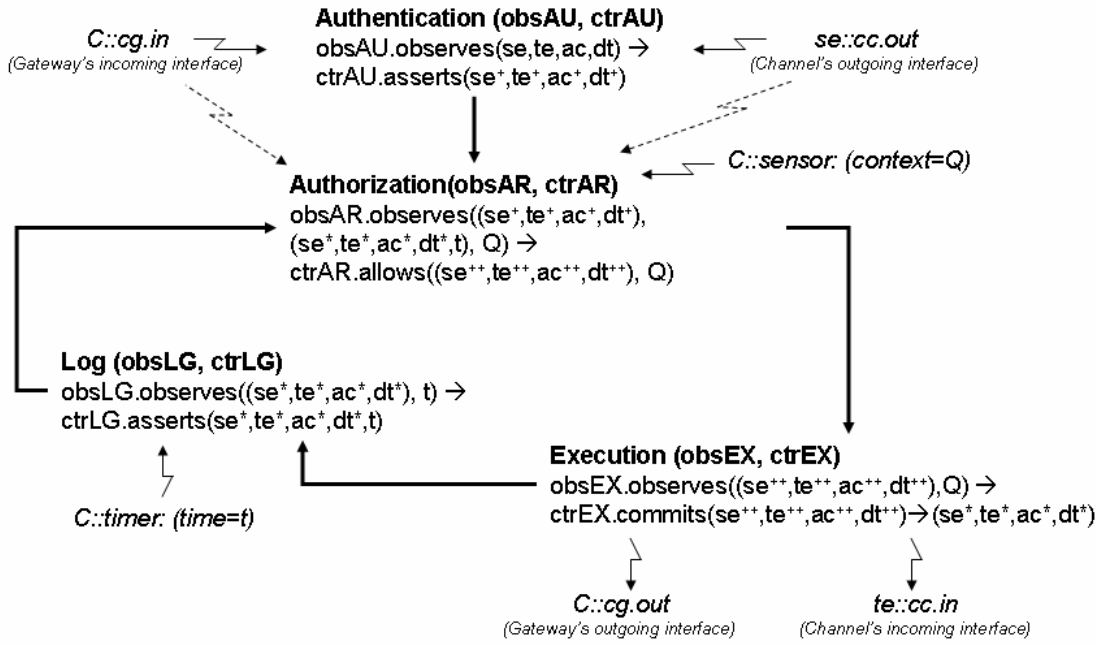

Figure 2: Typology and dependency mapping on security controls

Authentication: (1) includes an observer of type **obsAU** that listens for raw events of type `(se,te,ac,dt)` originated from a compartment's gateway `C::cg.in` or an internal output channel from a source element `se::cc.out`; (2) **obsAU** forwards to a registered controller of type **ctrAU**; (3) **ctrAU** is capable of positively or negatively asserting the identities of the events source $se \rightarrow se^{+}$, target $se \rightarrow se^{+}$ and `ac=ac.id`, as well as the sources of `ac.se` and `dt.se`, validating their authenticity, using keys $(k_{se,C}, k_{te,C}) \in CK$;

Authorization: (1) includes an observer of type **obsAR** that listens for authenticated events of type $(se^{+}, te^{+}, ac^{+}, dt^{+})$; these are typically observed from an authentication controller of type `ctrAU` but may originate from a pre-authenticated channel cc^{+} or gateway cg^{+} e.g for support of single-sign-on; (2) **obsAR** may also recieve information about current conditions and context Q from a sensor element in the compartment, provided by one or more elements, as well as an execution event history $(se^{*}, te^{*}, ac^{*}, dt^{*}, t)$ of the same type; (3) **obsAR** forwards authenticated events and histories to a registered authorization controller of type **ctrAR**; (4) given that **ctrAR** possesses or can derive a positive authorization policy of type $p = (se, te, ac, dt, [Q])$, **ctrAR** creates an authorized event $(se^{++}, te^{++}, ac^{++}, dt^{++})$ and forwards this to a specified target within or external to the Compartment.

Execution: (1) includes an observer of type **obsEX** that listens for authorized events $(se^{++}, te^{++}, ac^{++}, dt^{++})$; **obsEX** may be embedded in a Gateway or a managed element. (2) **obsEX** forwards to a registered controller of type **ctrEX**, which triggers the invocation of the action ac^{++} and creating an execution event $(se^{*}, te^{*}, ac^{*}, dt^{*})$; (3) the execution event is then forwarded to the outgoing interface of a targeted gateway `C::cg.out` or incoming interface of a targeted channel `te::cc.in`

Log: (1) includes an observer of type **obsLG** that listens for execution events from an execution controller **ctrEX** and timestamp t from a timer embedded in the Compartment; (2) **obsLG** forwards execution events to a log controller `ctrLG`, which is able to securely maintain an execution event history such that *non-repudiation* is also protected against;

The main reason for separating the functionalities of observer and controller rests in the real world knowledge that there are various configurations possible. For example, for any of the control types previously described, there could be 1-1, 1-n, n-1 or n-n configurations for a realization of the o-c pattern. Furthermore, depending on the business process and deployment environment, the nature and frequency of authentication, authorization, execution and logging controls will vary. Each of these would result in different system topologies, dependencies and resource costs. In pointing that out, we now return to the calculation of $TCO(E^{SCE})$, the cost of the security infrastructure, and define this in Equation 2

$$TCO(E^{SCE}) = cost_0(AU, AR, EX, LG, \phi) + \frac{\sum_{ev=1}^{ev=n} \omega_{ev}.cost_{ev}(AU, AR, EX, LG, \phi)}{T} \qquad (2)$$

Equation 2 can be traced to Equation 1, but there are some new details that need to be explained:

Firstly, we simplify the modeling of a business process BP as a partially-ordered event set specification $BP = ev_0 \circ ev_1 ... \circ ev_n$, where $\circ \leftarrow$ `(sequence, parallel, choice)`, ev_0 represents the initialization of BP and ev_n is always the terminating event.

Secondly, the $cost_{ev}(AU, AR, EX, LG)$ function is the calculated or estimated cost associated with provisioning the security infrastructure for authentication (AU), authorization (AR), secure execution (EX) and logging (LG) respectively for a particular $ev \in BP$. This includes the costs for observing and controlling the particular event types, given the type of technical infrasructure (see Table 1).

Thirdly we introduce the *dependency ratio* ϕ to account for the number of dependencies between security control elements and control types that need to be maintained per event. For example, an authorization control depends on an authentication control.

Finally, the cost weight parameter ω_{ev} defined in Equation 3 is used to distribute the costs across events in BP based on (1) estimated event frequency $EventFrequency$, (2) value of data in payload $DataValue$ and (3) expected response time $ResponseTime$. Note that as the response time decreases, the TCO is expected to increase as, in most cases, the amount of bandwidth, processing and power will need to be increased. However, while it may be the case that removing security controls to achieve such a response time is necessary, the removal of dependencies and controls still balances out the calculation.

$$\omega = \frac{EventFrequency \times DataValue}{ResponseTime} \tag{3}$$

In addition to calculating the cost of a security configuration, it must be possible to provide some arguments for its robustness, as well as proof that a deployed configuration represents what the engineer or administrator intended at each stage of the MIF.

3.3 TCO-aware security infrastructure provisioning theorems

This section now presents three theorems that integrate the concepts defined so far into a formal reasoning framework for TCO-aware security infrastructure provisioning. The technical implementation of this formal framework as a process is then given in Section 4.1. The reasoning framework builds on the 6 Axioms defined in Section 3.1 and shows (1) how to determine the capability of the weakest attacker, (2) provide a refinement guarantee for a security infrastructure model and (3) maintain the estimated costs for a security infrastructure model throughout the MIF.

Theorem 1: The likelihood of compromise for any two Compartments C_i and C_j can be determined and compared according to the variations in the properties and features of the Compartment Model defined by Definition 1.

Proof: To prove this, we prove that the compromise likelihood determination of a Compartment depends on each property and feature of the Compartment: (1) if a Compartment has 0 elements, then the attraction to attackers is minimal. (2) The more gateways or channels that exist on a Compartment, the more points of attack are provided; (3) the more participants in a Compartment, the higher the probability that at least one will act maliciously; (4) The more keys that need to be contained in a keystore CK, the higher the probability of compromise.

Theorem 2: a lower-level model C^m is as secure as a higher-level Compartment model C^{m-1}, if and only if **only** Gateways, channels, events and control policies that occur in C^{m-1} are expressed in C^m and Axioms 1 - 6 still hold.

Proof: From rules of refinement, if C^m can only produce outcomes that C^{m-1} can produce, then C^m is a refinement of C^{m-1}, stated $C^m \sqsubseteq C^{m-1}$. From Axiom 1, all outcomes caused by event transfer must occur via Gateways or Channels. From Axiom 2, an event only becomes an outcome if there is an explicit policy in place that allows it.

Theorem 3: Variations in costs of a lower-level deployed Compartment model $C^{deployed}$ are traceable to the estimated costs of higher-level bound C^{bound}, grounded $C^{grounded}$, unbound $C^{unbound}$, custom C^{custom} and general $C^{general}$ models.

Proof: This follows from the proof of Theorem 2, in that the cost of a security infrastructure from Equation 2 varies according to elements that are expressed in a semantically consistent way across all model types.

The goal of the modelling approach is not to totally abstract the complexity of system security, as this can lead to unjustified decisions. The idea is that security and TCO analysis are iterative, step-wise processes, where introducing more details should lead to improved results. For example, the model does not go into detail of how complex issues such as role inheritance and aggregation of permissions are handled, as these would require further refinement of the Compartment model and authorisation constraints. This can however be done by changing the granularity of a Compartment and adding roles as authorisation constraints. This is consistent with the security-preserving refinement and composition theorem of Backes, Pfitzman and Waidner[1], where they state that if a larger system is designed based on a specifiaction of a subsystem, and the implementation of the subsystem is later plugged in, the entire implementation of the larger system is as secure as its design.

4 Architecture and Process for TCO-Aware Security Infrastructure Provisioning

This section describes the technical architecture for TCO-aware security infrastructure provisioning. It describes the architectural goals, the components and finally the operations performed in the architecture, based on a series of model-transformations. Firstly, our main architectural goals are listed below:

1. traceability of security requirements at each stage of the MIF
2. rational decision making with respect to TCO and security tradeoffs
3. reusability and separation of concerns with respect to security
4. support for incremental configuration and provisioning of security controls in the MIF

The next two sections technically describe the architecture and operational process that have been designed to achieve the above goals, using the TCO and O/C concepts established in the previous sections.

4.1 Architecture Description

Figure 3 shows a pictorial view of an architecture used in the proof of concept research prototype. The Operational Resource Fabric consists of network switches, routers, and blade servers each capable of running several Virtual Machines (VMs). Customers access a UI (User Interface) in the Access Services to select and configure the business process from the Business Process Catalogue which they want the service provider to deliver. This configuration of the business process includes specification of performance, availability, security, and other non-functional requirements for it. Based upon these selections, the Design Service 1) selects the necessary software products or modules from the Software Product Catalogue, 2) selects the appropriate compartment models from the Compartment Model Catalogue, and 3) creates a candidate design for an infrastructure which will deliver the business process with the specified non-functional requirements. This candidate design is priced for the customer, who may decide to change some of the requirements to arrive at a different candidate design possibly at a different price. When the customer approves a design for deployment, the Provisioning Service reserves the needed IT infrastructure from the Infrastructure Catalogue, at the proper time installs and configures the software and infrastructure, and starts the Monitoring Service. The hosted business process is now ready for use by the customer's users.

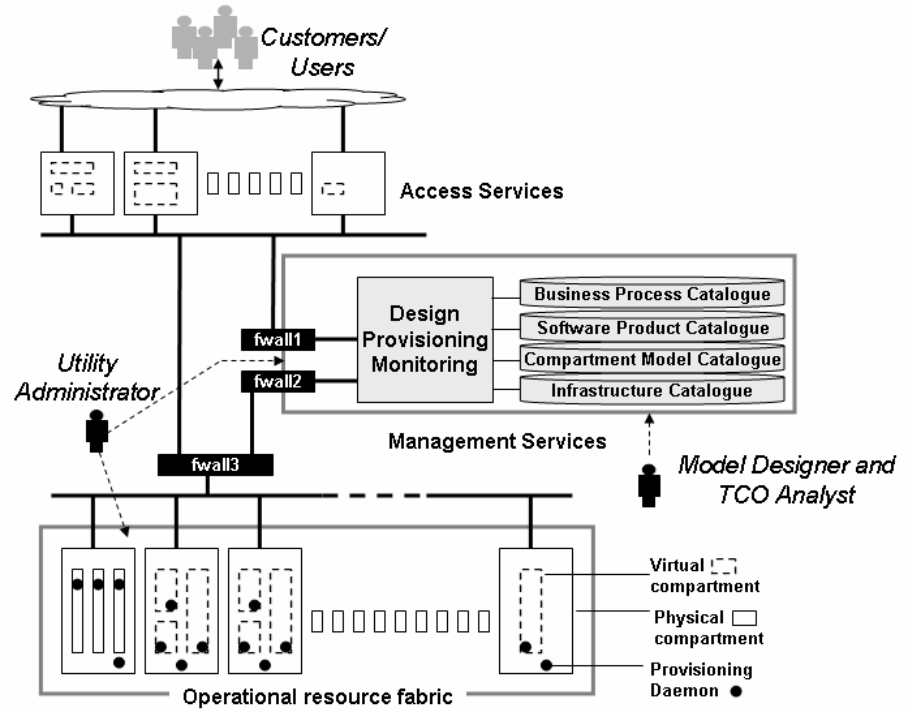

Figure 3: Architecture components and their integration in the infrastructure

We now describe each of the components identified in Figure 3 with respect to their functionalities, as well as their impact on the configuration of the security infrastructure.

Business Process Catalogue: A collection of business processes which the service provider is able to offer and which the customer can configure to some degree, for example modifying an ordering process to include an optional credit check.

Software Product Catalogue: A collection of available software products or product modules which may be installed and configured by the provisioning service.

Compartment Model Catalogue: A collection of models of compartments, with each compartment describing how the contained infrasructure is protected by a set of security controls. Each compartment model may offer different levels of protection.

Infrastructure Catalog: A list of the available IT infrastructure elements which the service provider can use to build new service instances. This includes things like physical and virtual computer systems, disk storage, firewalls, load balancers, subnets, IP addresses, and other infrastructure elements found in a datacenter.

Design Service: This service takes the customer's configured business process and its requirements and designs a candidate infrastructure which will deliver the business process with the specified requirements.

Provisioning Service: This service takes the selected software products or modules (from the Software Product Catalogue), the selected compartment models (from the Compartment Model Catalogue), and selects infrastructure elements (from the Infrastructure Catalog) to create a running service instance based upon the models.

Monitoring Service: This service monitors the running service instance, sending alerts and taking automated actions when necessary.

In addition, each of the components above maintains and provides bits of information for the final security infrastructure configuration, with respect to event and control types. As examples of security control policies, consider a simple example of a compartment which protects web application servers where the following events are observed:

1. *receiveApplicationLogonRequest* - This event models the logon request to the application.
2. *receiveApplicationRequest* - This event models application traffic (other than logon requests) flowing from outside of the compartment to the application servers in the compartment, for example an HTTP request from a user.
3. *receiveOperatorRequest* - This event models operator traffic flowing from outside the compartment into the compartment, for example an OS login session used for system administration.
4. *sendMonitoringRequest* - This event models monitoring traffic flowing from inside the compartment to a monitoring system outside the compartment, such as an SNMP trap or application alert.
5. *sendDNSResolutionRequest* - This event models contacting an external DNS server to reverse resolve an IP address into a Fully Qualified Domain Name (FQDN).

There may be several controls desired to govern the receiveApplicationRequest event. As an example, we show user authentication and network filtering controls. Other controls such as authorization constraints can also be modelled and enforced. The condition required to be true in order to allow the GET action of the receiveApplicationRequest event can be described as follows:

```
(     ( isMemberOfSet(user, stronglyAuthenticatedUsersSet) )
  AND ( protocol == "TCP" )
  AND ( (destinationPort == "80") OR (destinationPort == "443") )
  AND ( isMemberOfSet(destinationIP, applicationServerIPAddressSet) )
)
```

The subcondition in the first line above states the condition that the user specified in the request must already be strongly authenticated, which happens during an earlier successful receiveApplicationLogonRequest event. The second line states the condition that the protocol field in the IP header of the packet must be TCP. The third line states the condition that the destination port field in the TCP header must be either 80 or 443. The fourth line above states the condition that the destination IP address in the IP header must be one of the application server IP addresses. Note that both application-level and network-level conditions are described together. It is our intent that all conditions pertinent to determine the correct action to take in response to this event are described in one place to facilitate defining, locating, and programmatically evaluating and verifying the complete set of security controls for the compartment. The series of model refinements described in the next subsection take each subcondition and ensure that an appropriate security mechanism is selected and configured to enforce the subcondition, thus ensuring that the complete condition is enforced by the set of security mechanisms.

4.2 Model Refinement Process

The MIF moves in several stages from a high level of abstraction (a business process and its requirements) to a low level of abstraction (running IT systems providing the business process). At each stage, models are

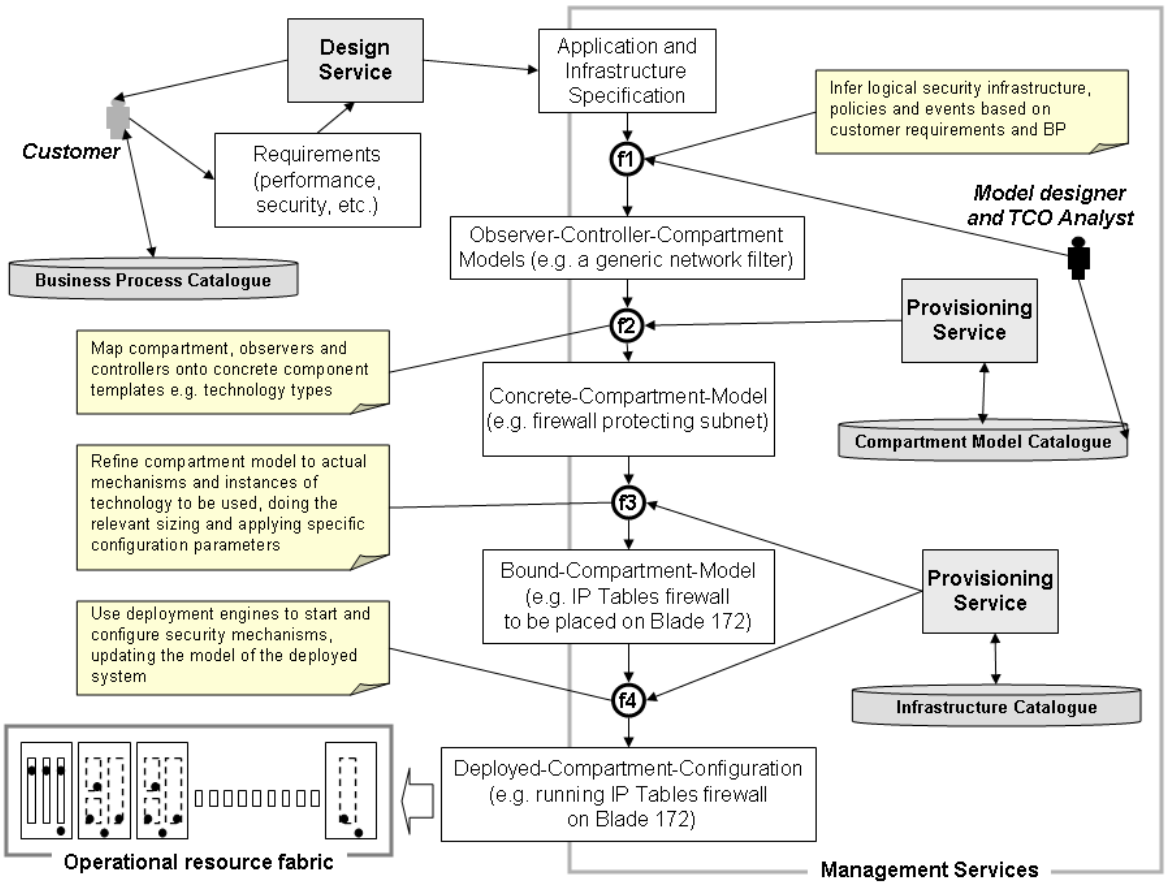

Figure 4: The staged model-refinement process for supporting security infrastructure provisioning

refined and/or templates are selected to move to the next lower level of abstraction. For example, the highest level of abstraction to describe security properties for a web application server compartment might be the single natural language sentence ”Enforce our standard Corporate Web Application Server compartment policy.” This can be refined to the next level of detail, which might be a few more descriptive natural language sentences like the following paragraph:

> ”Implement a network filter on the compartment boundary which allows only 1) serving HTTP requests, 2) resolving external DNS names, 3) sending monitoring traffic, and 4) administrative access. Implement an application authentication feature which requires each user to have a valid certificate issued from our Certificate Authority in order to logon.”

This is transformed into the set of events necessary to model the requests entering and leaving the compartment, as described in section 4.1. The collection of events will determine the collection of security controls which need to be in place. The network filtering controls specifically for the receiveApplicationRequest event are described by some of the subconditions for this event, listed earlier:

```
...
AND ( protocol == "TCP" )
AND ( (destinationPort == "80") OR (destinationPort == "443") )
AND ( isMemberOfSet(destinationIP, applicationServerIPAddressSet) )
```

The above refinements or transformations are represented by function "f1" in Figure 4.2 and entail some human involvement, in that one or more experts such as the Model Designer must create each lower level of detail, verify the correctness of the transformation or mapping, and place the compartment model into the Compartment Model Catalogue. This expert knowledge is captured in reusable models and templates for massive leverage across the service provider's infrastructure and customers.

The remaining selection or transformation functions f2, f3, and f4 in Figure 4 are primarily automated using software, in that software agents use the models to design and instantiate the infrastructure used to provide the modelled service. The Provisioning Service uses a selection function f2 in Figure 4 to select and parameterize an infrastructure template which specifies the network topology, location and type of firewalls, and other security mechanisms. The Provisioning Service uses a transformation function f3 to transform the complete set of modelled security controls from the o/c compartment models into device-specific configurations for the selected security mechanisms, and an actual location in the datacenter is selected to host each security mechanism instance. Continuing this example, if IPTables software were chosen as the network filtering mechanism, and only two application servers at IP addresses 15.2.3.4 and 15.2.3.5 were used, and eth1 were the "outside" interface name, then the IPTables configuration file generated by the transformation function f3 would have four lines like the following to allow the receiveApplicationRequest events into the compartment:

```
iptables -A FIREWALL -i eth1 -d 15.2.3.4 -p tcp --dport 80 -j ACCEPT
iptables -A FIREWALL -i eth1 -d 15.2.3.5 -p tcp --dport 80 -j ACCEPT
iptables -A FIREWALL -i eth1 -d 15.2.3.4 -p tcp --dport 443 -j ACCEPT
iptables -A FIREWALL -i eth1 -d 15.2.3.5 -p tcp --dport 443 -j ACCEPT
```

At f4 in Figure 4, the Provisioning Service will deploy the security mechanisms, such as IPTables software, on appropriate devices or machines (virtual or physical) and start them running with the appropriate configurations. The application servers and software and the monitoring service are also deployed and started. The *deployed model* of the service is updated to reflect these deployments, and then we have a running service ready for use.

5 Case Study and Evaluation

Assume a bank wants to begin an online loan processing service hosted by a utility service provider [20]. The business process models will include a web presence which allows both private individuals and commercial companies to apply for loans online and upload or view documents related to their loan applications. The bank's organizational structure has different departments handling the personal and commercial loans, so they model a system which has one compartment which stores personal loan data and allows access from certain roles in the personal loan department, and a separate compartment which stores commercial loan data and allows access from certain roles in the commercial loan department. Customers also have limited access to view and submit loan applications and related documents restricted to their accounts. At this level of abstraction, called the Customized Process Model in the MIF, the realizations of the compartments are undetermined, but some of the security properties and access requirements which must apply to the compartments are determined, see Figure 5.

This is transformed to an Unbound Model, Figure 6, which is a functional system specification. In this example, the main decisions at this step are 1) to select the particular database software to use in each compartment to store the information and 2) to mediate customer access using a Web Gateway. At this point in the MIF-based system design, we still have not chosen the technology to implement the compartments.

Next we transform to the Gounded Model, Figure 7, which is a deployable system specification. For this example, let us assume that the performance modelling tools determine that the most cost-effective

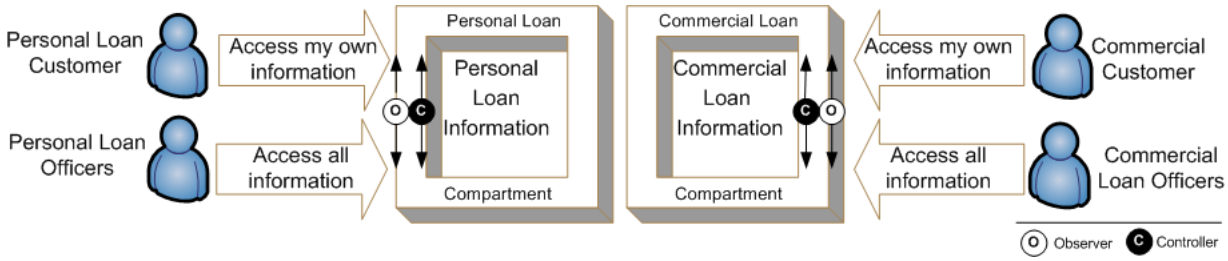

Figure 5: A graphical view of the Customized Process Model

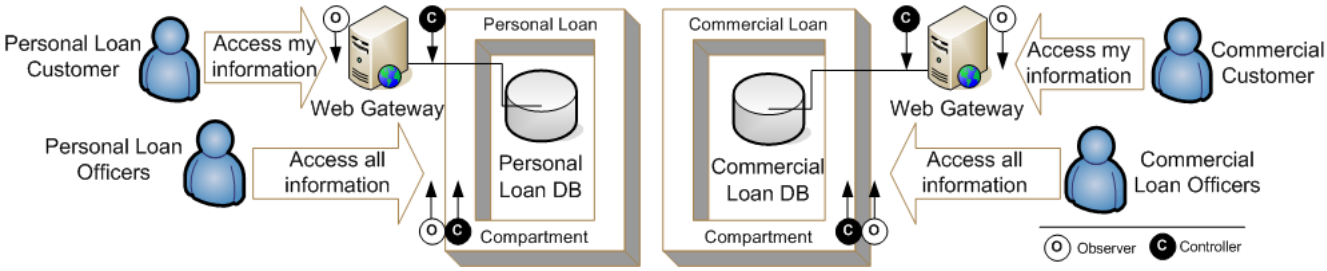

Figure 6: A graphical view of the Unbound Model

realization of the compartment for commercial loans which meets all requirements and that can be built with the available infrastructure is database software running on a Linux OS on a Virtual Machine (VM) on a blade system using SAN (Storage Area Network) storage. In this case, the security mechanisms which implement the observers and controllers are 1) a stateful packet inspection filter in the Linux OS to control network access to the OS, 2) access controls on the SAN which limit storage access, and 3) the access control rules of the database which limit who can access or alter records.

The performance modelling tools determine that a load balancer, three physical machines on a LAN, and SAN storage are best able to handle the larger volume of requests to the database for personal loans. The security mechanisms which implement the observers and controllers which provide the isolation for the personal loan compartment are 1) a stateful packet inspection firewall which controls network access to the subnet containing the machines, and 2,3) the same SAN and database controls used for the commercial loan compartment. A different set of security mechanisms delivers the same security properties in each compartment using the same model of those properties. A key innovation is the ability to generate and automatically deploy different realizations of a compartment, based upon both the available infrastructure and the business requirements. Each compartment realization can meet the same business requirements, including security properties, but with different levels of performance or availability.

6 Discussion and Related Work

6.1 Resolving Cost, Performance and Security Concerns

Irvine and Levin[10] define a taxonomy and costing method for network security services towards supporting both the Quality of Service and security requirements of customers. Security services are labeled as authentication, confidentiality, integrity, non-repudiation etc., classified under three service areas - (i) end system, (ii) intermediate (e.g. routers, switches) and (iii) the wire - and indicated as fixed or variable services. Each security service that appears in a security configuration for a given network topology is then associated with a cost measure specified in their taxonomy. In order to support the customer's Quality of

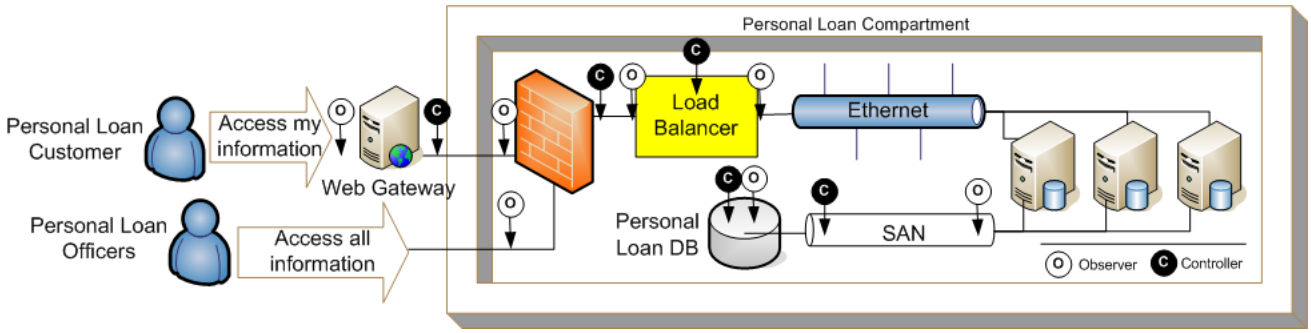

Figure 7: A graphical view of the Grounded Model for the personal loan compartment

Service and security requirements, a separate level of security request is sent to a Resource Management System. Our taxonomy and consideration of security costs are based on compartmentalization and the specialization of Security Control Elements that satisfy the security requirements of the customer's business process.

One of the contributors to elevated cost and decreased performance of security infrastructures is the complexity of their configurations. Jaeger [11] describes an approach to assessing the configuration complexity of access control configurations, introducing a set of complexity metrics. The metrics reflect the complexity and costs associated with inheritance and aggregation of authorizations. We consider complexity in a similar manner, but define this with respect to the number of new instances and relationship of controllers and observers caused by the application hosting request of a customer. From another perspective, complexity can be seen as a positive aspect of a security configuration, as it makes it more difficult for an attacker to circumvent, given that it has been implemented without flaws. Schechter [16] presents an econometric approach to security risk analysis, with the argument that an economically rational attacker will be disuaded from attacking a system that has too much risk for them. With our provisioning approach, we encourage and support both customers and providers to include an estimated value of data and functionality in their agreed TCO upper-limits. In doing so, a high-cost security infrastructure and compartmentalization is only provisioned for highly-rated data that is labelled as potentially very attractive for attackers. Incorporating this knowledge in software models remains a promising direction for software engineering.

6.2 Model-driven and Aspect-Oriented Security

The foremost, existing approaches to model-based and model-driven security engineering are SecureUML defined by Basin, Doser and Lodderstedt [2] and UMLSec created by Juerjens [12]. They however do not consider how to provide comprehensive methodologies for considering the various cost and performance concerns that need to be made during the provisioning of systems. Their techniques may be useful for capturing security models and eventually their formal representation as more concrete configuration templates. Nevertheless, the backwards path from security configuration decision to business, cost and performance requirements would still have to be integrated into these schemas in order that they can satisfy the modeling requirements of our approach. Other work such as by Viega et al. on Aspect-Oriented Programming (AOP) for Security [19], treat security as a separable engineering concern and describe how to then integrate them using the crosscutting method of Aspect Oriented Programming [19]. There are advantages for flexibility, extensibility and modularity gained with AOP techniques, however, as explained by Shah and Hill [18], the problems of development complexity, design-time representation and no support for requirements traceability remain as challenges for AOP approaches to Security Engineering and, by association, provisioning.

6.3 Security Infrastructure Description and Provisioning

Microsoft states model-based management as the foundation for their Dynamic Systems Initiative [4]. They intend to provide System Modeling Language (SML) tools and models which link the three layers of business process, application configuration and infrastructure configuration. At this time, they do not have clear plans to address the system lifecycle as the MIF does. It is also unclear how they will deal with the vast array of choices available when designing a large IT system, as the MIF does using flexible configurable templates.

Porto de Albuquerque et al.[6] devise a framework and integrated set of tools for policy modeling and refinement for network security systems. While they pay attention to the correctness of refinement and composition, they have not modelled nor investigated TCO in the level of detail that we have.

7 Conclusion

We have proposed a single model and process for combined analysis and assessment of security and TCO during provisioning. It is thus referred to as TCO-aware security infrastructure provisioning. The approach is still preliminary but presents some underpinnings for formally representing and understanding the trade-offs between security and cost. It was also shown that the proposed theoretical framework for reasoning about the TCO of a security infrastructure can also be integrated into a more general model-based provisioning framework known as the Model Information Flow (MIF)[3]. The main concept was that of a Compartment, which we have formally specified and demonstrated through various examples. Secondly, we then went on to define an architecture and process for supporting security architecture provisioning using our concepts, and demonstrated its viability in a relevant business scenario.

We see the contributions of our work going towards both software engineering and systems management, particularly in an age when highly distributed and loosely-coupled systems based on services are becoming the trend in IT. We maintain that addressing security concerns as early as possible in a system's lifecycle saves money and leads to more robust and reliable systems. As a system's lifecycle progresses from an abstract system specification to a concrete running infrastructure, the compartment concept is used to ensure that security concerns are addressed at each stage. The ability to assess the weakest attacker for different Compartment instances is indeed a novelty for considering measurability of security, leading to a new breed of security metrics.

We continue to implement this approach in the joint research project. We have flexible templates which define alternative topologies for the system layout and enable the simultaneous treatment of certain functional, performance, availability, and security attributes of the system. We are extending the o/c pattern to implement the model-driven deployment and configuration of host-based firewalls and network firewalls to build compartments using either protected virtual machines or protected subnets as a first proof of concept. There are however many more security concepts that can be modelled and assessed using this framework. Future work will show the full integration of this in solutions for automated and dynamic sizing of shared infrastructure.

References

[1] M. Backes, B. Pfitzmann, and M. Waidner. A general composition theorem for secure reactive systems. In *TCC*, pages 336–354, 2004.

[2] D. Basin, J. Doser, and T. Lodderstedt. Model driven security: From uml models to access control infrastructures. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 15(1):39–91, 2006.

[3] G. Belrose, K. Brand, Edwards, Graupner N., J. S., Rolia, and L. Wilcock. Adaptive infrastructure meets adaptive applications. Technical report, HP Labs, 2007.

[4] Microsoft Corporation. Microsoft dynamic systems initiative. MSDN Technet, 2007.

[5] N. Damianou, N. Dulay, E. Lupu, and M. Sloman. The ponder policy specification language. In *POLICY '01: Proceedings of the International Workshop on Policies for Distributed Systems and Networks*, pages 18–38, London, UK, 2001. Springer-Verlag.

[6] J.P. de Albuquerque, H. Krumm, and P.L. de Geus. Policy modeling and refinement for network security systems. In *Proceedings of the Sixth IEEE International Workshop on Policies for Distributed Systems and Networks (POLICY'05)*, pages 24–33, Washington, DC, USA, 2005. IEEE Computer Society.

[7] W. Ding, W. Yurcik, and X. Yin. Outsourcing internet security: Economic analysis of incentives for managed security service providers. In *WINE*, pages 947–958, 2005.

[8] C. Franke and P. Robinson. Autonomic provisioning of hosted applications with level of isolation terms. In *Engineering of Autonomic and Autonomous Systems (EASe)*, 2008. to appear.

[9] Gartner. Gartner total cost of ownership. Website: last accessed 29-01-2008, 2008.

[10] C. Irvine and T. Levin. Toward a taxonomy and costing method for security services. In *ACSAC '99: Proceedings of the 15th Annual Computer Security Applications Conference*, page 183, Washington, DC, USA, 1999. IEEE Computer Society.

[11] T. Jeager. Managing access control complexity using metrices. In *SACMAT '01: Proceedings of the sixth ACM symposium on Access control models and technologies*, pages 131–139, New York, NY, USA, 2001. ACM.

[12] J. Jrjens and P. Shabalin. Tools for secure systems development with uml: Security analysis with atps. In *Fundamental Approaches to Software Engineering*, LNCS, pages 305–309. Springer Berlin/Heidelberg, 2005.

[13] B. W. Lampson. Computer security in the real world. *Computer*, 37(6):37–46, 2004.

[14] C. Patel and A. Shah. Cost model for planning, development and operation of a data center. Technical report, HP Laboratories, 2005.

[15] Shane Robison. The next wave: Everything as a service. Executive Viewpoint: www.hp.com, 2008.

[16] S. E. Schechter. Toward econometric models of the security risk from remote attack. *IEEE Security and Privacy*, 3(1):40–44, 2005.

[17] Thorsten Schöler and Christian Müller-Schloer. An observer/controller architecture for adaptive reconfigurable stacks. In *ARCS*, pages 139–153, 2005.

[18] V. Shah and F. Hill. An aspect-oriented security framework. In *3rd DARPA Information Survivability Conference and Exposition (DISCEX-III 2003)*, pages 143–145. IEEE Computer Society, 2003.

[19] J. Viega, J. T. Bloch, and P Chandra. Applying aspect-oriented programming to security. *Cutter IT Journal*, 14(2):31–39, 2001.

[20] C. Wolter and A. Schaad. Modeling of task-based authorization constraints in bpmn. In *BPM*, pages 64–79, 2007.

[21] J. Yoder and J. Barcalow. Architectural patterns for enabling application security. In *4th Conference on Patterns Language of Programming (PLoP'97)*, 1997.