

Edith Cowan University

## Research Online

---

Australian Information Security Management  
Conference

Conferences, Symposia and Campus Events

---

11-30-2010

### Threat Modelling with Stride and UML

Michael N. Johnstone  
*Edith Cowan University*

Follow this and additional works at: <https://ro.ecu.edu.au/ism>

 Part of the [Information Security Commons](#)

---

#### Recommended Citation

Johnstone, M. N. (2010). Threat Modelling with Stride and UML. DOI: <https://doi.org/10.4225/75/57b670493477c>

DOI: [10.4225/75/57b670493477c](https://doi.org/10.4225/75/57b670493477c)

8th Australian Information Security Mangement Conference, Edith Cowan University, Perth Western Australia, 30th November 2010

This Conference Proceeding is posted at Research Online.  
<https://ro.ecu.edu.au/ism/88>

## Threat Modelling with Stride and UML

Michael N. Johnstone  
School of Computer and Security Science  
Edith Cowan University  
Perth, Western Australia  
m.johnstone@ecu.edu.au

### Abstract

*Threat modelling as part of risk analysis is seen as an essential part of secure systems development. Microsoft's Security Development Lifecycle (SDL) is a well-known software development method that places security at the forefront of product initiation, design and implementation. As part of SDL, threat modelling produces data flow diagrams (DFDs) as key artefacts and uses those diagrams as mappings with STRIDE to identify threats. This paper uses a standard case study to illustrate the effects of using an alternative process model (UML activity diagrams) with STRIDE and suggests that using a more modern process diagram can generate a more effective threat model.*

### Keywords

Security, Threat Modelling, UML, Vulnerability, Software Engineering, Information Systems Security, Conceptual Modelling.

### INTRODUCTION

A threat, according to Bishop (2005, p4) is “a potential violation of security.”, whilst a vulnerability, is “A flaw or weakness in a system's design, implementation, or operation and management that could be exploited to violate the system's security policy.”. (RFC 2828, 2000, p190). Consequently, an exploit is an attack that takes advantage of a vulnerability thus realising a threat. Clearly, the aim of secure development is to identify and mitigate threats before they become exploitable vulnerabilities in production systems.

Secure development, like much of software engineering, is still in its infancy. Evidence of the latter claim can be found in Jayaratna (1994), who suggests that there are over 1000 extant methodologies. There are not 1000 different ways to build a bridge or an aircraft, nor do avionics engineers decide to modify an aircraft just before it is due to fly out of an airport-an interesting contrast with software engineering. How effective some of those methodologies are at building systems is perhaps open to question, but that is beyond the scope of this paper. Shostack and Stewart (2008, p89) provide some evidence for the former claim in that they assert that most software is insecure. This could be because, as Wysopal et al. (2007) note, security requirements are often omitted from requirements specifications altogether.

Information systems (IS) security research has been extensively studied. Dhillon and Backhouse (2001) used the framework of Burrell and Morgan to categorise existing IS security research, while Baskerville (1993) used a “generation” taxonomy culminating in logical transformational methods. Siponen (2005) analysed existing IS security methods, also using a generational taxonomy (although different to that of Baskerville). Siponen argued that future work on IS security methods should move towards social and adaptable (empirically grounded) methods. This focus on IS security research has resulted in security-oriented development methods being created and empirically tested to some extent. The most notable of these methods are SQUARE (Mead and Stehney, 2005), CLASP (OWASP, 2006) and SDL (Howard and Lipner, 2006; Microsoft, 2010). These methods place security requirements at the forefront of all stages of the development lifecycle but, as with any new method, have not yet been extensively field-tested.

SDL has a nine-step process for risk analysis which uses dataflow diagrams (DFDs) to identify potential threats to a system prior to building the system. This is interesting because dataflow diagramming would appear to be a somewhat outmoded process modelling technique that was developed in the 1970s (see DeMarco, 1978 for the original DFD technique, Gane and Sarson, 1979 for a variation).

The focus of this paper is not to examine the reasons why DFDs are appropriate process diagrams for threat models (as there is no evidence why this choice was made for the SDL) but to use an alternative process diagram, the activity diagram of the UML, and compare the two diagrams for effectiveness in a threat modelling scenario.

### THE SDL RISK ANALYSIS PROCESS

Figure 1 shows how secure development practices have been embedded into the SDL by amending a traditional software development process. Threat modelling is an explicit part of the SDL and the steps of the former will be explained in this section.

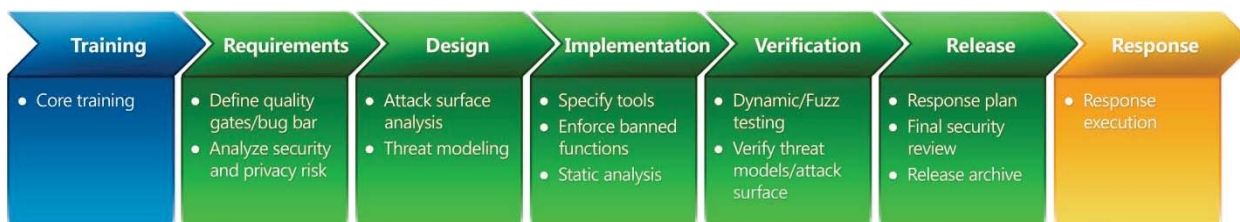


Figure 1: Secure software development process model at Microsoft (Microsoft, 2010).

Howard and Lipner (2006, p105) define the following threat model steps: 1. Defining use scenarios; 2. Gather a list of external dependencies; 3. Define security assumptions; 4. Create external security notes; **5. Create DFDs of the application being modelled;** **6. Determine threat types;** **7. Identify the threats to the system;** 8. Determine risk; and 9. Plan mitigations. It is steps 5-7 that are the focus of this research.

Howard and Lipner (2006) assert that step 5, creating correct DFDs is crucial and that failure here means that the rest of the threat modelling process will be incorrect, a view supported by Hernan et al. (2006). Howard and Lipner use the (now ubiquitous) pet shop e-commerce case, a top-level DFD of which is shown in figure 2.

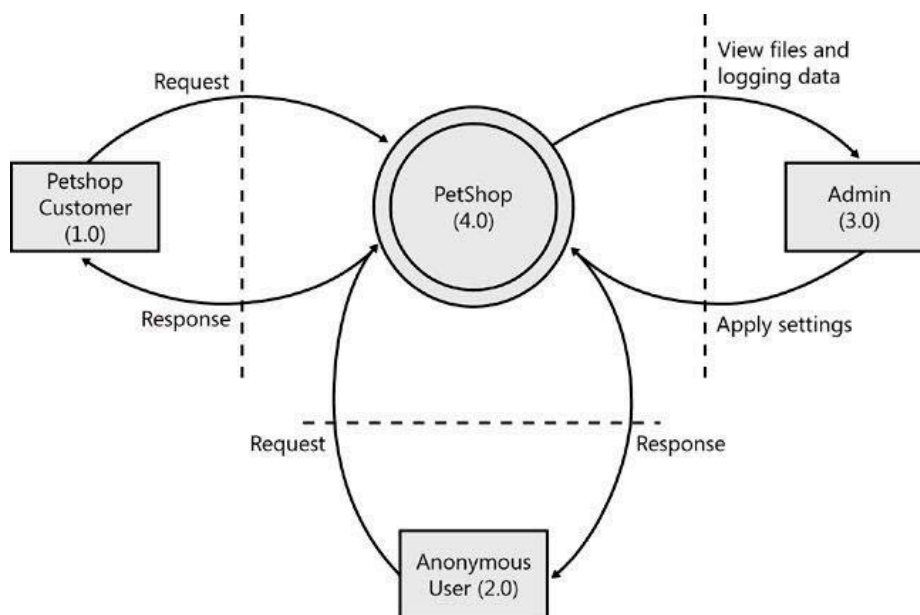


Figure 2: Pet Shop Context Diagram (Howard and Lipner, 2006, p110).

In figure 2, the data flows are represented by arrows leading to processes (circles). Circles within circles signify abstractions of complex processes which will be decomposed in other DFDs. Data sources are represented by rectangles, while data files are straight lines (see figure 3 for an example of data files). SDL makes one significant security modelling extension to DFDs, viz, dashed lines representing trust boundaries.

Figure 3 provides the detail of order processing for the Pet Shop case study. This diagram does not precisely match the original case study-it has been modified slightly to include an audit log. In common with many on-line ordering systems, there are options for processing orders immediately or for deferring such processing until a later time-a useful function for a resource-intensive process, which can speed up servicing of requests by performing resource levelling in much the same way that batch processing works on mainframe systems. It is interesting to note that when asynchronous orders are queued, inventory isn't checked, however, Howard and Lipner (2006) never claimed that the model was realistic, just that it was correct for the purpose of threat modelling.

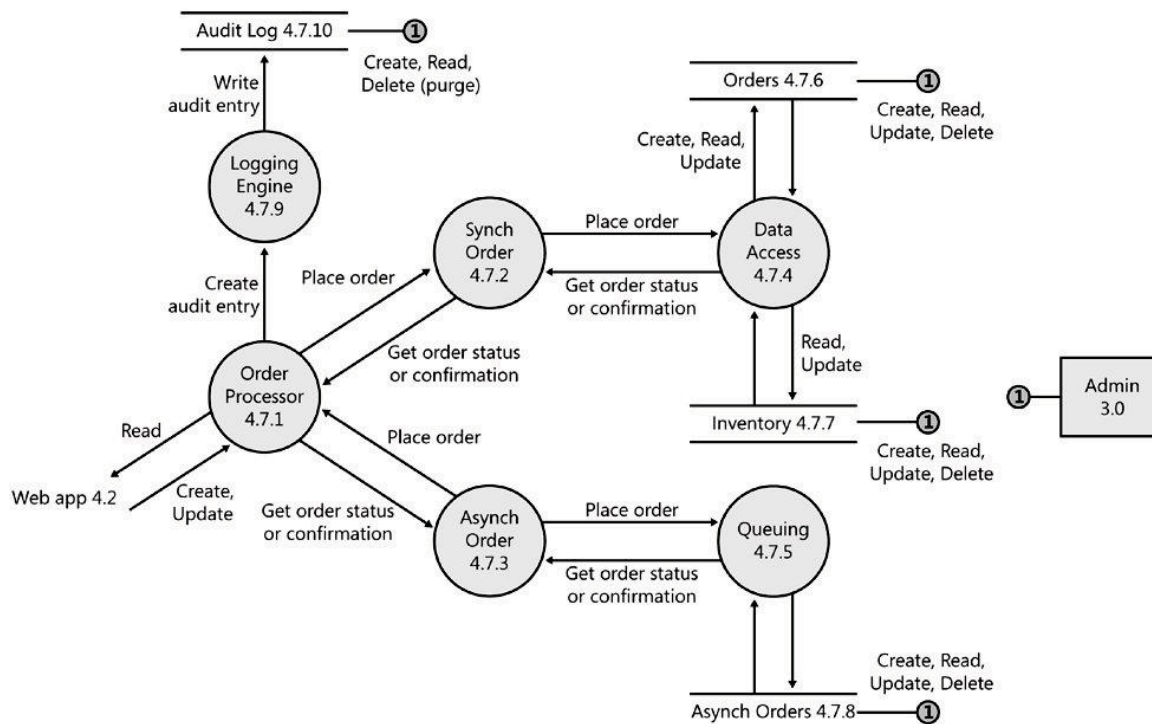


Figure 3: Pet Shop Order Processing DFD (Howard and Lipner, 2006, p113).

Assuming that step 5 was performed correctly, step 6, determine threat types can be implemented in several ways. One common taxonomy is CIA or confidentiality, integrity and availability. Howard and Lipner (2006) claim that Microsoft’s taxonomy, STRIDE (an acronym for Spoofing identity, Tampering, Repudiation, Information disclosure, Denial of service and Elevation of privilege) is more complete. This is not entirely true as the CIA taxonomy has been extended to cover authentication and non-repudiation.

Table 1: DFD Elements List (Howard and Lipner, 2006).

DFD Element Type	DFD Item Number
External Entities	Pet Shop customer (1.0)
	Anonymous user (2.0)
	Administrator (3.0)
Processes	Web application (4.2)
	User profile (4.5)
	Membership (4.6)
	Order processor (4.7.1)
	Sync/Async Order processors (4.7.2 and 4.7.3)
	Data-access or queuing components (4.7.4 and 4.7.5)
	Auditing engine (4.7.9)
Data Stores	Web application configuration data (4.1)
	Web pages (4.3)
	Use profile data (4.8)
	Membership data (4.9)

	Order and async orders data (4.7.6 and 4.7.8)
	Inventory data (4.7.7)
	Audit-log data (4.7.10)
Data Flows (partial list)	Web application reading configuration data (4.1 -> 4.2)
	Web pages read by Web application (4.3 -> 4.2)
	Anonymous user request/response (2.0 -> 4.2 -> 2.0)
	Pet Shop customer request/response (1.0->4.2->1.0)
	Admin reading, creating, updating Web application configuration data (3.0->4.1->3.0)
	Admin reading, creating, updating, deleting Web pages (3.0->4.3->3.0)
	Web application reading, creating, updating an order (4.2-> 4.7.1->4.2)

Step 7, identify the threats to the system is done in three stages. First, construct a candidate list of the atomic DFD elements. Atomic in this context means that complex processes are not included, but that the elements that comprise the complex process are articulated and recorded in the list. This is not unlike the composite data dictionary used as a repository of knowledge about a system being modelled in systems analysis. Second, reduce the candidate list by considering whether two or more DFD elements behind the same trust boundary can be considered to be equivalent-this would be so if the threat analysis to two or more elements produces the same outcome, the elements handle similar data and they use the same technology (see table 1 for an example). Finally, apply STRIDE to each of elements in the reduced list by mapping the elements to the STRIDE categories (table 2).

Table 2: Mapping STRIDE to DFD Elements (Howard and Lipner, 2006).

DFD Element Type	S	T	R	I	D	E
External Entity	X		X			
Data Flow		X		X	X	
Data Store		X	†	X	X	
Process	X	X	X	X	X	X

In table 2, a cross indicates that the element is subject to the relevant threat represented by STRIDE. For example, external entities (such as a customer) are subject to spoofing identity and repudiation attacks. The dagger mark represents the notion that whilst data stores are not usually subject to repudiation attacks, this is not the case for the audit log, which could potentially be manipulated by an attacker. In the next section, the same artefacts will be generated, but substituting UML activity diagrams for data flow diagrams.

## UML PROCESS MODELS

In the UML 2.0 there are 13 types of diagrams, several of which can be used to model dynamic aspects of systems, viz, activity diagrams, sequence diagrams, communication diagrams and state diagrams. Activity diagrams (ADs) are process models and thus model the workflow of a system, but work in conjunction with the other dynamic models. This section will begin by explaining some basic properties of activity diagrams, and then will progress to generating activity diagrams relating to the Pet Shop case as described above. Following this, the STRIDE matrix will be generated for the relevant artefacts.

Table 3 describes the major elements of activity diagrams. Similar to DFDs, ADs can be nested, thus promoting understandability via abstraction. Activity diagrams also share some notation with other UML models, e.g. the

initial/final nodes and fork/join nodes can also be found in UML state diagrams. At their simplest, ADs resemble flowcharts, but the expressive power of ADs is significantly enhanced (as compared to flowcharts) by additional constructs, specifically guard conditions (which are not the same as conditions in a flowchart), nesting as a mechanism for abstraction, fork/join for parallel processing and partitions for indicating where an activity takes place.

Table 3: Activity Diagram Elements (adapted from Ambler, 2010).

Element	Description
Initial node	A filled in circle is the starting point of the diagram.
Activity final node	A filled circle with a border is the end point.
Activity	The rounded rectangles represent activities (physical or electronic) that occur.
Activity edge	The arrows on the diagram.
Fork	A black bar with one flow going into it and several leaving it. This denotes the beginning of parallel processing.
Join	A black bar with several flows entering it and one leaving it. All flows going into the join must reach it before processing may continue (end of parallel proc.)
Condition	Text enclosed in square brackets “[ ]” on a flow, defining a guard which must evaluate to true in order to traverse the node.
Decision	A diamond with one flow entering and several leaving.
Merge	A diamond with several flows entering and one leaving. The implication is that one or more incoming flows must reach this point before processing continues, based on any guards on the outgoing flow.
Partition	Partitions (also called swimlanes) indicating who/what is performing the activities.
Flow final	The circle with the “x” through it. This indicates that one split of a (parallel) process stops at this point.

Figure 5 is a first-cut activity diagram of order processing for the Pet Shop case study. Figure 6 shows another version of an activity diagram for order processing in the Pet Shop case. There are several useful aspects to the diagram, viz. the flexible viewpoint, the use of partitions or swimlanes and the use of guard conditions. Using an organisational or systems-oriented viewpoint of the activities shows which business unit is responsible for the activities, whilst in an e-commerce environment such as the Pet Shop case, a more technology-oriented view (see figure 7) can show which tier of a multi-tier architecture is responsible for the activities. The latter is possibly more useful for risk analysis.

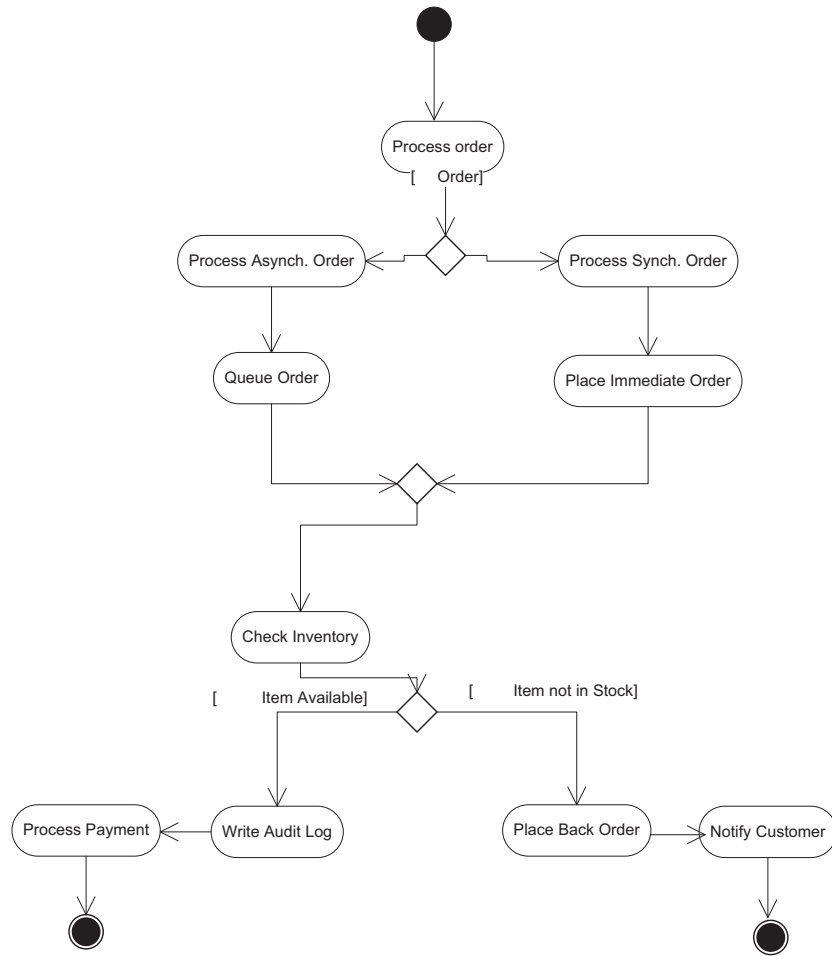


Figure 5: First-cut Activity Diagram of Pet Shop Order Processing.

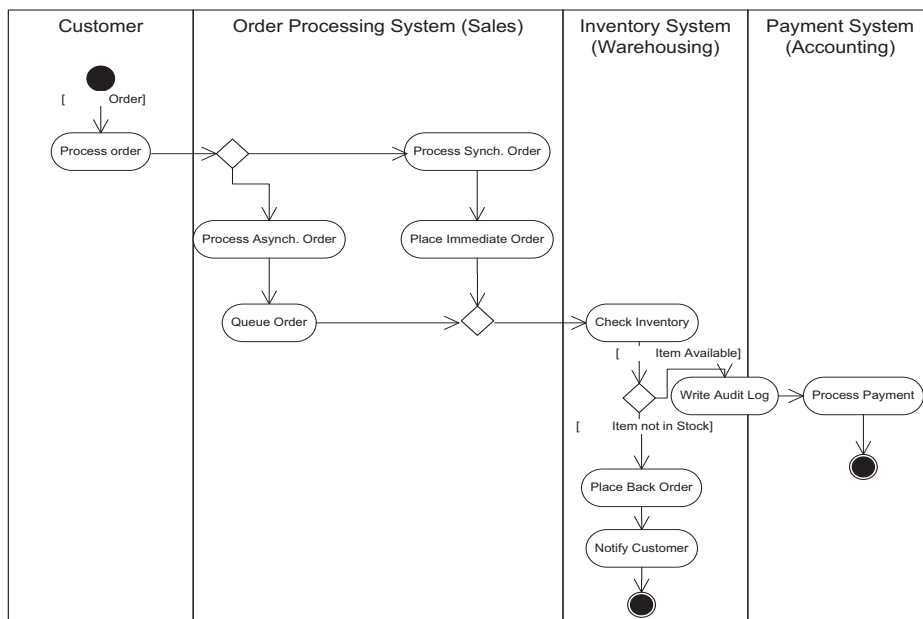


Figure 6: Organisation View Activity Diagram.

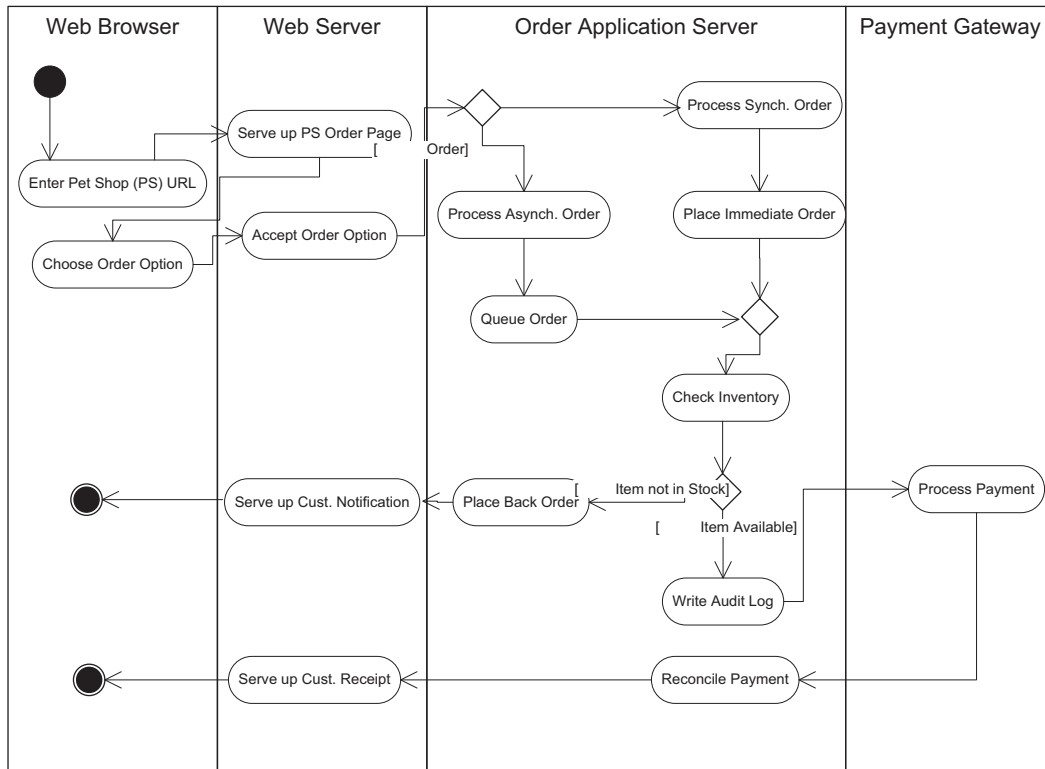


Figure 7: Technology View Activity Diagram.

Table 4: Activity Diagram Element List (Pet Shop Order Processing).

AD Element Type	AD Item
Actors	Pet Shop customer
	Anonymous user
	Administrator
Activities	Choose ordering method
	Queue order
	Place order
	Sync. or asynch. order processors
	Check inventory
	Place back order
	Write audit log
Conditions	Order (exists)
	Item not in stock
	Item in stock
Activity Edges	Web pages served by Web server



	Customer/Anonymous user request/response
	Admin CRUD operations on Web pages
	Web application updating audit log
	Web application reading, creating, updating an order

Table 4 focuses mostly on order processing but includes a generic operation (admin functionality) common to all web-based systems. Table 5 shows that using an AD in place of a DFD generates the same risk analysis, thus validating the substitution of the UML diagram. Interestingly, the dagger in table 2 is replicated in table 5 because the guard condition “item available” must be true for the audit log to be updated.

Table 5: Mapping STRIDE to Activity Diagram Elements.

AD Element Type	S	T	R	I	D	E
Actor*	X		X			
Activity Edge		X		X	X	
Condition		X	†	X	X	
Activity	X	X	X	X	X	X

\* Actors, whilst part of the UML, are not part of activity diagrams. They are used, however, in activity diagrams as objects when determining business workflows, so it is reasonable to assume that the actors customer, admin and anonymous user exist, even though they are not explicit.

## COMPARISON OF MODELS

Comparing DFDs and ADs is in one sense reasonably straightforward as they are both members of the same generic family of models (i.e. they are both process models). They can be compared by examining their expressive power, ability to compose (vertical partitioning), ability to recognise system boundaries (horizontal partitioning) and their connection to security-oriented concerns (specifically risk analysis).

Expressiveness and abstraction are often considered to be qualities under some tension i.e. qualities that are at either end of a spectrum of flexibility. A language that is abstract tends to work at a macro level compared to a language that has more constructs and thus the latter has a greater potential to model closer to reality (a sometimes desirable property). Based on the number of available language elements alone (11 vs. 5-including trust boundaries for DFDs), ADs are more expressive than DFDs. This is also evident from examining figures 3 and 7, which are both representations of the same process.

Both models can express some form of composition. DFDs represent composition by means of concentric circles which signify abstractions of processes, details of which are provided in other, more complex DFDs. ADs allow nesting, which assists in hiding complexity in much the same way as in DFDs, the only difference being that the nested AD can be shown either as an abstraction or as a complete AD with a boundary within the current diagram.

DFDs were extended by Howard and Lipner (2006) to include trust boundaries for risk analysis modelling. Partitions (or swimlanes) can perform the same function in ADs. Notice that in figure 6, there is a clear delineation between different systems (hence a potential trust boundary). Similarly, in figure 7, the partitions are expressed as different tiers of a multi-tier architecture common to many web-based systems-again, these partitions indicate system boundaries, albeit at quite a low level, that are obvious trust boundaries where elevation of privilege can potentially occur.

Non-overlapping guard conditions in ADs control which of a set of alternatives will be traversed once an activity is complete. This can be seen in figure 6 where following the “check inventory” activity, either the item requested is in-stock in which case the transaction is written to the audit log and the payment processed. Alternatively, if the item is not in stock, then a back-order is generated and the customer notified. There is no analogue of this element in DFDs.

ADs offer the ability to perform activities in parallel via the fork/join elements. Note that this is quite different to decision/merge nodes, where one of a number of choices will complete. In a fork/join, all activities must complete for the join to progress. These elements were not used in figure 6, however the diagram could have been drawn with a fork at “process order/send invoice” and a join at “place or queue order/receive payment”. There are no analogues of these elements in DFDs.

In summary, both modelling techniques have equivalent elements in the areas of vertical and horizontal partitioning, but activity diagrams appear to be more expressive, have guard conditions and allow parallel processing of activities. This suggests that activity diagrams have greater utility as risk analysis models in SDL.

## CONCLUSION AND FURTHER WORK

This study explored the use of process diagrams in threat modelling within the SDL. The complex nature of a modern software modelling notation was unveiled and reasons why modern software process models should be used put forward.

Specifically, this study used a well-known case study to show how a different process modelling technique (viz. activity diagrams) could be substituted for data flow diagrams in the SDL's risk analysis process. It was argued that such a substitution provided benefits in that ADs can be used in place of DFDs with no loss of information and further, the extra notational elements of ADs could model more complex situations than DFDs. A secondary driver for this research was that most computer science/software engineering courses teach UML but not DFDs.

Whilst this study showed that activity diagrams have greater utility as threat models in SDL as compared to data flow diagrams, the use of a single, simple case study suggests that ADs should be tested with other, more complex cases. Also, the most common method for generating an activity diagram involves examining the corresponding use case. It would be useful to consider security-oriented analogues of use cases (e.g. misuse cases or abuse cases) as they would directly focus the risk assessment process on security aspects of a system, which would potentially make step 7 of the SDL risk analysis (identify the threats to the system) a largely mechanical process amenable to full automation, thus providing much-needed tool support in this area.

It is acknowledged that a limitation of this work is that it is unwise to generalise from a single case study therefore further work could be to field-test this idea to see how well ADs contribute to threat models of real-world systems. Further, there are other dynamic models both within the UML (e.g. state diagrams) and external to the UML (e.g. Business Process Modelling Notation) and it may be that other process models generate better threat models than ADs. Bishop (2005) certainly takes the view that a secure system can be defined in terms of authorised/unauthorised states, which suggests that a state-based technique might be more valuable than a workflow-based technique. If this thesis were to be explored, there are a plethora of state-based modelling techniques of varying complexity (e.g. finite state machines, Petri nets, CPNs, UML state diagrams), all of which could be tested across a selection of case studies. Whilst this might be a useful exercise, as it would provide evidence of effective techniques for threat models, clearly the product of the number of potential cases and the number of alternative state models suggests an approach that can be generalised more easily with some confidence. Ontology may provide such an approach as properties of various models can be proven via mathematics, rather than by a multiple case design. Therefore an ontological analysis and comparison of the various state modelling techniques is a logical first step, which would reduce the candidate list of techniques substantially. This could be followed by a mapping of the ontologically complete state diagramming technique to threat modelling, which would confirm (or not) their suitability and effectiveness in that domain. This could then be confirmed by using empirical studies.

## REFERENCES

- Ambler, S. (2010). *Introduction to UML 2 Activity Diagrams*. Retrieved September 29, 2010, from <http://www.agilemodeling.com/artifacts/activityDiagram.htm>
- Baskerville, R. (1993). Information Systems Security Design Methods: Implications for Information Systems Development. *ACM Computing Surveys*. 25(4):375-414.
- Bishop, M. (2005). *Introduction to Computer Security*. Boston, MA: Addison Wesley.
- DeMarco, T. (1978). *Structured Analysis and System Specification*. Englewood Cliffs, NJ: Prentice-Hall.
- Dhillon, G. and Backhouse, J. (2001). Current directions in IS security research: towards socioorganisational perspectives. *Information Systems Journal*. 11(2):127-153.
- Gane, C. and Sarson, T. (1979). *Structured Systems Analysis: Tools and Techniques*. Englewood Cliffs, NJ: Prentice-Hall.

Hernan, S., Lambert, S., Ostwald, T. and Shostack, A. (2006). "Uncover Security Design Flaws Using The STRIDE Approach". *MSDN Magazine*, November.

Howard, M. and Lipner, S. (2006) *The Security Development Lifecycle. SDL: A Process for Developing Demonstrably More Secure Software*. Redmond, WA: Microsoft Press.

Jayaratna, N. (1994). *Understanding and Evaluating Methodologies*. London: McGraw-Hill.

Mead, N.R. and Stehney, T. (2005). Security Quality Requirements Engineering (SQUARE) Methodology. *Proc. Software Engineering for Secure Systems: Building Trustworthy Applications (SESS'05)*, May 15-16, 2005, St Louis, MO, USA.

Microsoft (2010). *Microsoft Security Development Lifecycle Version 5.0*. Redmond VA: Microsoft.

OWASP. (2006). *OWASP CLASP (Comprehensive, lightweight application security process) Project*. Retrieved August 20, 2009, from <http://www.owasp.org>.

RFC 2828 (2000). *Internet Security Glossary*. Internet Engineering Task Force. Retrieved September 22, 2009, from <http://www.ietf.org/rfc/rfc2828.txt>

Siponen, M.T. (2005). *Analysis of modern IS security development approaches: towards the next generation of social and adaptable ISS methods*. Oulu, Finland: Department of Information Processing Science, University of Oulu.

Shostack, A. and Stewart, A. (2008). *The New School of Information Security*. Upper Saddle River, NJ: Addison Wesley.

Wysopal, C., Nelson, L., Dai Zovi, D. and Dustin, E. (2007). *The Art of Software Security Testing*. Upper Saddle River, NJ: Addison Wesley.