Towards Self-Protecting Ubiquitous Systems *Monitoring Trust-based Interactions*

Colin English, Sotirios Terzis and Paddy Nixon

University of Strathclyde
Department of Computer and Information Sciences
{Firstname.Lastname}@cis.strath.ac.uk

Abstract. The requirement for spontaneous interaction in ubiquitous computing creates security issues over and above those present in other areas of computing, deeming traditional approaches ineffective. As a result, to support secure collaborations entities must implement self-protective measures. Trust management is a solution well suited to this task as reasoning about future interactions is based on the outcome of past ones. This requires monitoring of interactions as they take place. Such monitoring also allows us to take corrective action when interactions are proceeding unsatisfactorily. In this vein, we first present a trust-based model of interaction based on event structures. We then describe our ongoing work in the development of a monitor architecture which enables self-protective actions to be carried out at critical points during principal interaction. Finally, we discuss some potential directions for future work.

1 Introduction

As advances in mobile and embedded technologies coupled with progress in ad-hoc networking fuel the shift towards ubiquitous computing systems it is becoming increasingly clear that security is a major concern. While this is true of all computing paradigms, the characteristics of ubiquitous systems amplify this concern by promoting spontaneous interaction between diverse heterogeneous entities across administrative boundaries [5]. Entities cannot therefore rely on a specific control authority and will have no global view of the state of the system. To facilitate collaboration with unfamiliar counterparts therefore requires that an entity takes a proactive approach to *self-protection*. We conjecture that trust management is the best way to provide support for such self-protection measures.

Trust management in its evidence based form [3, 7, 8] allows evidence to be recorded about the outcome of past interactions to enable better reasoning about future interactions. Thus trust management does not provide security in the traditional sense of ensuring protection of resources, rather its purpose is to mitigate the risks that are present in interactions. In the current paradigm, once an interaction has been initiated, it executes and we trust the other principal to behave as expected. After completion of the interaction, we evaluate the outcome and can punish unexpected behaviour by reducing our trust in the principal.

Although we are less likely to interact with them in the future, it is possible that damage from the completed interaction may already have occurred.

What if applications could protect themselves further? What if instead of only observing the final outcome of an interaction, an application was able to monitor progress throughout its execution, in order to be able to protect itself when things are going wrong. By following the progress of an interaction, we facilitate the implementation of safeguards for corrective action, which in extreme cases may even involve the premature termination of an interaction, to prevent damage we suspect may occur. This may help to increase to acceptability of evidence based trust management solutions in the larger security community.

Our aim is to develop such a monitor for general use over a range of applications. To achieve this, an underlying interaction model for applications is needed, capable of representing the types of interactions we are concerned with and how these take place. The model must consider the observable factors both within the interaction itself and its environment enabling the representation of the interaction state. The monitor then can follow this interaction model, and allow the application to intervene at critical points. This paper outlines our work in developing the monitor, detailing our interaction model in section 2 and aspects of the monitor in section 3. We then discuss ongoing work and conclude in section 4.

2 The Interaction Model

The basis for our model comes from work undertaken in the SECURE project [1], which facilitates collaboration between entities through a decision making process. This process allows evidence to be gathered to construct a trust value, which is then used in conjunction with cost factors to facilitate a risk analysis process upon which an interaction decision can be based. There is a formal theoretical trust model based on *event structures* underlying this decision process [2].

More specifically the model acknowledges that interactions will have certain relevant properties from the perspective of trust. During the interaction there will be things that may be observed which determine the interaction's outcome and as such are of interest to future trust decisions. To represent such *observables*, the model uses events, which could be generated internally or externally to the application. An interaction is therefore modelled as an event structure, a set of all observable events, E_T, within which certain relations exist. These relations reflect that the occurrence of some events may be necessary for the occurrence of others (necessity relation), and the occurrence of some will prohibit the occurrence of others (conflict relation). The outcome of an interaction is therefore represented as a subset $C \subseteq E_T$ called a configuration, holding the set of events actually observed, which is consistent with these relations.

Modelling outcomes as configurations enables a history (H) to be built from a series of interactions, represented by a set of configurations where members represent the outcomes of individual instances of the interaction. Through such a history H, trust values can be represented as mappings of configurations to triples (s, i, c), in the form of counts reflecting how many configurations in H support (s), contradict (c) or are inconclusive (i) for each configuration. Whether an observed outcome contributes to s, i, or c for each configuration is determined by the necessity and conflict relations between events in the configurations. These counts can therefore be used to estimate the likely outcome of the next such interaction with the principal in question.

Further to the events that are of relevance to trust, many interactions will incorporate observable properties of relevance to the cost of outcomes of the interaction. We incorporate these into the interaction model as events in a set E_C, which can be generated either internally by the application or by some external source aware of environmental conditions. Although these events do not contribute to configurations, they are of importance to the overall decision process, as they affect the risk analysis upon which decisions rely.

As an example consider the following scenario of a distributed file storage service. This service allows users to subscribe to host files on many different file servers, each for a specific duration. Users may decide to remove their files from a particular server before the end of the subscription period if they become dissatisfied with the quality of service offered. Alternatively they may decide to replicate their files on multiple servers to make them more available and so on. Observables (events) come from the application as a result of attempted file actions or pings, or some external source such as an intrusion detection system. For example, events of interest might be $fileaccess$, $fileok$, $\overline{fileok}$, $ping$, $serveravailable$ and $\overline{serveravailable}$, where the overline implies negation. We should point out that the following relationships hold between the events; $fileok$ and $\overline{fileok}$ are in conflict and both are in necessity relation to $fileaccess$, while $serveravailable$ and $\overline{serveravailable}$ are also in conflict and both in necessity relation to both $fileaccess$ and $ping$. Hence an example of a valid configuration is $\{fileaccess, fileok, ping, \overline{serveravailable}\}$, whereas $\{fileaccess, fileok, \overline{fileok}\}$ is invalid due to the conflict between $fileok$ and $\overline{fileok}$ for a specific $fileaccess$. Furthermore, an example of a cost relevant event is a critical update of a file, $fileupdate(critical)$, meaning the value of the file has increased for the user. Notification of such an event could come from the application. Another risk relevant event from an external source is *changeconnection*. This event denotes that depending on the kind of network connection used, e.g. home dail-up connection or workplace LAN, the cost of accessing a file changes.

It is important to bear in mind that there are many possible interactions in a ubiquitous computing environment and it may be difficult for the application developer to characterise all of them. However, the model is based on a decision process that requires developers to consider specific interactions and thus the model need only cope with those that the developer is concerned with. From a self-protection point of view there are certain properties that applications must have:

- There must be a rich set of observable characteristics to be monitored.
- There exists sensible points at which an interaction initiated by a single trusting decision can allow corrective action to be taken.

– The interaction must offer scope for reaction to observed events.

Although we show a limited E_T above, this could be greatly expanded into a more rich set. The example is both interruptible and has duration, as the service allows subscription for a period of time, during which the file may be removed at any time.

3 The Monitor

So we have an interaction model with events for trust observables reflecting likelihoods of outcomes and events for cost observables dictating costs of the outcomes. How does the monitor utilise this to follow the progress of such interactions? It must track the state of the interaction at any time throughout its execution. As the observable events of the model deal separately with cost concerns and trust concerns, our monitor maintains a representation of both the interaction *trust-state* and its *cost-state*.

Trust-state is a representation of how the interaction is proceeding in terms of the events in E_T. The configurations that represent outcomes in the model can also represent partial executions of the interaction and hence its trust-state. The monitor takes a *snapshot* of the current interaction. This is a set, or more specifically a configuration, containing the observables from E_T seen so far during the execution. These snapshot configurations will be subsets of those configurations that represent final outcomes of interactions. Cost-state is a set maintained by the monitor which contains the events from E_C that have occurred throughout the course of the interaction. The snapshot and cost-state are updated as the result of event notifications for the specific interaction.

The trust-state and cost-state reveal the currently available information on the likelihoods and costs of outcomes respectively. This can be used to improve our understanding of the risks in the progressing interaction compared to the risk analysis used in the initial decision. To facilitate this the monitor must combine trust-state and cost-state, with consideration for the concerns of the application, to create a representation of overall *risk-state*. This is a subjective view of how satisfactorily the interaction is proceeding from the perspective of the truster, with respect to its initial risk estimation.

Figure 1 shows how observed trust and cost events influence the risk-state of the interaction through the specification of conditions that reflect the trust and cost concerns of the application given the risk estimation at the decision stage. The following section will elaborate on this by detailing the construction of the state space.

3.1 Constructing the Risk-State Space

The risk-state space is constructed by the application in terms of a set of conditions. The number and content of the conditions is up to the application to define. However, given that the application can have concerns relating to both

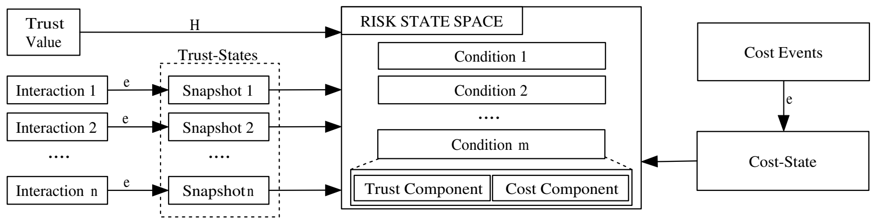

Fig. 1. Risk-State Space Construction.

trust and costs, these conditions will most likely have both a trust component and cost component. These components are specified as predicates, involving the current trust-state and cost-state of the interaction. The application can specify cost component predicates based on knowledge of how certain events in E_C affect the cost/benefit of certain outcomes. The trust component predicates can be generated by the application based on which configurations are of concern to the application. As an example, there may also be particular outcomes that the application would like absolute protection from regardless of how small their likelihood initially was. Furthermore, other evidence may become available during an interaction. Therefore snapshots from other ongoing interactions of the same or similar type with the same or other principals can be incorporated into the trust component of conditions. Therefore if other interaction instances are unsatisfactory, this may be an indication that the current instance will follow suit. As an example from the file server scenario, if the application is concerned about file integrity regardless of H, it may define a predicate that becomes true when, across all snapshots, a file has been corrupt on four out of five file accesses.

Conditions may necessitate the comparison of the current snapshot to H or to other snapshots. We can determine whether any of the snapshots support or contradict particular configurations, or even express uncertainty via the inconclusive comparisons. This is key to specifying and evaluating conditions. For example, the current snapshot may contradict certain configurations, limiting the outcomes the interaction may proceed to. It is thus possible to define a condition on all good outcomes having been ruled out or when the snapshot supports too undesirable an outcome. An example condition from the file server scenario with both a trust and a cost component is when in the current snapshot, a particular file server has been unavailable on the past two occasions and at the same time, the connection cost is quite high.

Each time the snapshot or the cost-state for the current interaction is updated, this triggers the re-evaluation of the condition set. The set of all conditions defined constitutes the risk-state space while the truth values of these conditions dictate the risk-state at any time throughout the interaction. Therefore when a condition's predicate becomes true it effects a change in the risk-state which is of concern and may trigger an action. Thus changes in risk-state can be seen to reflect critical points in the interaction defined by the application, where some action might be taken.

3.2 Triggering Actions Based on State

Further to the monitor requiring the application to define the risk-state space, it must also define the actions to be taken upon specific changes in state. We are currently examining *Event-Condition-Action* (ECA) rules [6] for inspiration for a means to communicate this information to the monitor. These rules state that the occurrence of one or more events will trigger evaluation of a condition, which if true will trigger one or more actions. However, we wish to define actions to be triggered based on risk-state rather than the individual conditions to allow for more complex combinations of conditions to be the trigger. We envisage this being done using ECA rules on the risk-state itself. A very basic ECA rule to represent final outcome monitoring could be based on a risk-state space consisting of only one condition, that becomes true when the snapshot of the interaction matches a final outcome configuration. This is therefore a binary state space, and an ECA rule could specify that when the risk-state's single condition becomes true, an action is triggered that communicates an updated H to the application.

The specified actions will be application specific, possibly in the form of callback methods or messages. The content of these message may vary and can, for example, provide warnings to allow rollback or termination of the interaction or other information dictated by the application. It is the responsibility of the application to decide on the course of action to follow upon receipt of a message. In our file server example, feedback could allow the application to replicate a file on another file server or start some kind of recovery process.

3.3 The Monitor Architecture

The monitor we are developing must operate as a general, self-contained software component to enable it to be used across a range of applications. To this end we have developed a high-level architecture, seen in figure 2, to support the concepts defined so far in this section.

The architecture highlights the responsibilities of the different components and the interfaces between them. The application takes responsibility for specifying the condition set based on trust and cost concerns for each interaction to be monitored, communicating this to the monitor. It must also communicate to the monitor the ECA rules as a means to specify actions and also the locations of the necessary event sources to be subscribed to. This allows the monitor to retain its generality. Based on the information supplied by the application, the monitor must keep track of the cost-state and trust-state/snapshots for all interactions that it has been instructed to. The monitor also takes responsibility for maintaining H, as this can be updated based on the snapshot tracker when the outcome of an interaction is observed. Aside from storing this information, the monitor uses it in evaluation of the condition sets to build a representation of the

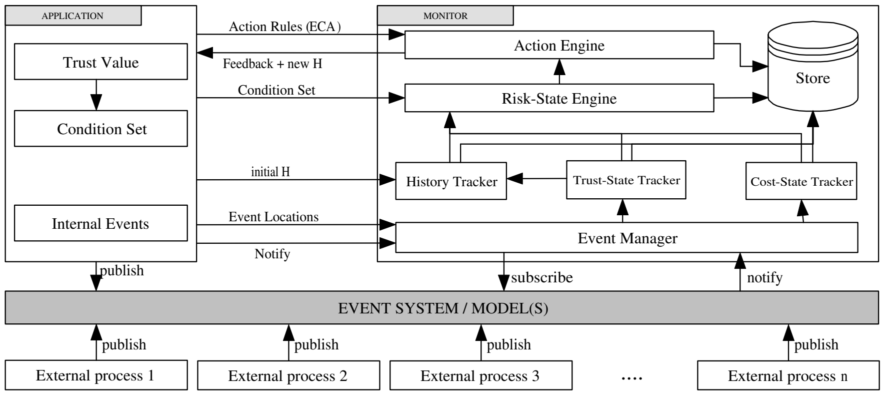

Fig. 2. The Monitor Architecture.

risk-state of each interaction and performs the actions as specified. The monitor does not reason about the risk-state, rather it just filters events and evaluates the conditions to determine and maintain a representation of the interaction's current risk-state.

Several underlying event models could be implemented, for example event channels using push or pull methods. The monitor may have to support a variety of event models to allow it to cope with a variety of event sources. Measures for the composition of events from different sources should be included, to allow evaluation of complex conditions. For example multiple environmental events might affect the cost of a single outcome. This is another aspect we are currently investigating. It is important to note that it may not be possible to monitor all events that may be of interest. Inaccessibility of certain external resources for event generation, loss of event feed or even just excessive delay can all contribute to this problem. The mechanism for dealing with this will be implementation dependent. The necessity relation as defined in the model can be useful here since if we observe an event which relies on the occurrence of others, those others must have occurred also. This allows us in some sense to fill in missing observations.

4 Conclusions and Ongoing Work

In this paper, we have briefly described our initial steps towards a monitor capable of providing support for self-protecting systems utilising trust. It does so by following the progress of interactions initiated through a trust-based decision process. The monitor is independent of application specific factors, requiring the application to specify its concerns in terms of both cost and trust observables and what action to take under certain circumstances in relation to this. We feel that this approach will improve the robustness of trust management solutions, increasing their acceptability through facilitating improved self-protection.

We have outlined our model of interaction and briefly introduced our initial ideas on the monitor. As this constitutes work in progress, we are currently working on a variety of issues this paper presents. To develop the monitor and architecture further, we are examining not only the aforementioned ECA rules, but also work on probes and gauges in autonomic computing [4] for external event generation. This will give us an insight into how the monitor can generate the necessary code for evaluation of the predicates. As a step towards implementation, we are making initial investigations of two scenarios, the file storage service briefly highlighted above and an online auction, to examine specification of events and generation of conditions for specific trust values. This will also provide an insight into the operational aspects of the monitor, such as event models and communication means, and provide the means to evaluate the self-protecting abilities of the monitor.

Acknowledgements

This work is supported by the EU project SECURE: Secure Environments for Collaboration among Ubiquitous Roaming Entities (IST-2001-32486). The authors would also like to acknowledge contribution by Waleed Wagealla.

References

1. V. Cahill, E. Gray, J.-M. Seigneur, C. Jensen, Y. Chen, B. Shand, N. Dimmock, A. Twigg, J. Bacon, C. English, W. Wagealla, S. Terzis, P. Nixon, G. Serugendo, C. Bryce, M. Carbone, K. Krukow, and M. Nielsen. Using Trust for Secure Collaboration in Uncertain Environments. In *Pervasive Computing Magazine*, volume 2, pages 52–61. IEEE Computer Society Press, 2003.
2. Marco Carbone, Karl Krukow, and Mogens Nielsen. Revised computational trust model. *SECURE Deliverable 1.3*, 2004.
3. Tyrone Grandison and Morris Sloman. Trust management tools for internet applications. In Paddy Nixon and Sotirios Terzis, editors, *Proceedings of the First International Conference on Trust Management*, volume 2692 of *LNCS*, pages 91–107, Heraklion, Crete, Greece, May 2003. Springer.
4. Philip N. Gross, Suhit Gupta, Gail E. Kaiser, Gaurav S. Kc, and Janak J. Parekh. An active events model for systems monitoring. In *Working Conference on Complex and Dynamic Systems Architecture*, December 2001.
5. Tim Kindberg and Armando Fox. System software for ubiquitous computing. *IEEE Pervasive Computing*, 1(1):70–81, January-March 2002.
6. Norman Paton. *Active Rules in Database Systems.* Springer Verlag, 1998.
7. Li Xiong and Ling Liu. Building trust in decentralized peer-to-peer electronic communities. In *Proceedings of the 5th International Conference on Electronic Commerce Research(ICECR-5)*, Montreal, Canada, October 2002.
8. Bin Yu and Munindar P. Singh. An evidential model of distributed reputation management. In *Proceedings of the first international joint conference on Autonomous agents and multiagent systems*, pages 294–301, Bologna, Italy, 2002. ACM Press.