Determining Trust in Cooperating Agent-based Systems

Mark Hepburn
School of Computing
University of Tasmania
Sandy Bay, Tasmania, Australia
Mark.Hepburn@utas.edu.au

David Wright
School of Computing
University of Tasmania
Sandy Bay, Tasmania, Australia
David.Wright@utas.edu.au

ABSTRACT

A common problem in intelligent-agent-based systems is that of cooperation. It is imperative that such systems can reliably determine which agents may be safely engaged with, and which should be avoided. We present an analysis framework in which the trustworthiness of potential partners is determined based on their ability to corrupt the agent.

1. INTRODUCTION

A common goal of Artificial Intelligence research is the development of intelligent agents [1] that can independently perform one or more useful tasks, such as remote exploration or news gathering for example.

Many intelligent agents perform a simple information gathering service; for example, trawling the web for related information and collating the results. As the technology improves, though, many agents are being entrusted with purchasing decisions, for example in online shopping. It is not unreasonable to expect many online transactions in the future to be accomplished between intelligent agents. An orthogonal point is that many agents are *mobile*; that is, they may migrate to a different execution platform, either to distribute computational load or perhaps for increased access to local information.

As this area develops, and as the complexity of the tasks performed by independent agents increases, these tasks will increasingly be performed in cooperation with other intelligent agents, each with their own specialized area of expertise. Unfortunately, this cooperation is open to abuse, and as the responsibilities accorded to such intelligent agents increases the risks are magnified.

We are concerned that, particularly with the economic risks inherent, without a solid framework on which intelligent agents can base decisions on whether to cooperate or not this promising technology may face stiff opposition. We identify two types of risk, and present an approach to verifying interactions based on their ability to introduce corrupted data (directly or indirectly).

2. MOTIVATION

We focus on software-based intelligent agents (that is, mobile code, rather than, say, remote exploration vehicles), however it should be noted that the analysis framework is sufficiently general to be extended in other directions as well.

2.1. PROBLEM DOMAIN

As a basis for analysis, we assume a system in which communication amongst intelligent agents occurs along named channels (corresponding, for example, to a socket or shared memory), and in which agents may migrate (obvious in the case of physical agents; otherwise corresponding to mobile code, such as Java applets). We further assume that networks of communication are not fixed; they may dynamically reconfigure themselves by transmitting the name or address of links to other intelligent agents. The ad-hoc nature of such networks makes the analysis more complicated, but is also more realistic and flexible.

These assumptions are captured in the pi-calculus [2]; a core modelling language usually used in programming language research, but that has also been applied to fields such as business process modelling [3]. In particular, we use a variant adapted to allow higher-order analysis (that is, mobile code) [4].

2.2. CORRUPTED COMMUNICATION LINKS

Under the analysis framework just suggested, the most immediate danger is of a corrupted communication channel. By "corrupted", we assume a path which cannot guarantee the integrity of data transmitted along it, with no means of determining if this has happened or not. This may be as a result of "noise" on the channel, or it may be due to the deliberate corruption of data or the injection of false values by a malicious third-party.

There is a second possibility, which we will also consider: the data may have become corrupted, but the agent is capable of determining (on a case-by-case basis) when this has occurred, due to digital signatures, error-detecting codes, or the like. In this case the channel is neither corrupted nor completely secure; it lies in the middle ground.

Note that there is some subtlety that must be considered here: it is perhaps easy to avoid the use of corrupted channels in a fixed setting, but extra care must be taken in the presence of ad-hoc networks where it may be difficult to determine if a new link the agent has just obtained is trustworthy or not.

2.3. MALICIOUS AGENTS

A more subtly pernicious problem arises when considering the impact of interaction with "rogue" intelligent agents. At first glance this looks relatively simple: any data received from an agent known to be untrustworthy should be rejected. Closer examination, however, reveals a serious complication.

Consider the case of interaction with a single intelligent agent, assumed to be trustworthy, but one that accomplishes *its* tasks in cooperation with others. If the secondary agent is malicious, it may conceivably pass corrupted information along to our agent, via the first. In this case, we must also distrust the first agent.

A related problem exists when considering how to treat a collection of intelligent agents (for example, sharing the same virtual machine). If there is a particular agent known to be malicious, but that is isolated from communication with the others in the collection, then it should have no impact when determining the trustworthiness of the group.

The question arises as to *how* an agent may initially be considered untrustworthy. We identify two particular routes. The first is due to the possibility of agent migration: if an intelligent agent is transported, via a corrupted communication channel, from one machine to execute on another then the agent in question must be considered untrustworthy due to the possibility it has been replaced with a malicious agent in transit. A second route may occur due to an agent's history: a bad experience with a particular intelligent agent would logically lead an agent to distrust that agent from then on (we do not consider this route formally; we merely assume that our intelligent agent knows some agents to be trustworthy and others not to be — the programmer of a particular agent may also introduce this initial information)

2.4. Approach

As stated previously, we use the pi-calculus as a modeling language for our analysis. We structure our analysis as a system of type annotations; the determination of the integrity of a potential partner agent then becomes a task of type inference. The type annotations themselves are based on a Boolean algebra, which enables expressive and complicated networks of dependency to be easily encoded. This also enables us to cheaply benefit from the many existing results in the area.

2.5. Paper Structure

The paper is structured as follows. In section 3 we introduce the modelling language used, with a brief introduction of the basic type system. We then proceed in section 4 to present an overview of our approach. In section 5 we briefly review some related work, then in section 6 we conclude.

3. Modelling Language

In order to facilitate a formal analysis of the domain, we choose to encode interactions amongst intelligent agents using a well-understood process calculus; specifically, the pi-calculus [2].

3.1. The Pi-Calculus

The pi calculus was first presented by Milner in 1989, as a means of reasoning about concurrent computation, based on the primitive of communication of names. It has proven sufficiently abstract and powerful, however, that it has now been applied to other domains such as business process modelling [3]. Sangiorgi [4] later described a variant that enabled communication of processes as well as names. It was also shown that the higher-order calculus could be encoded in the first-order, however the natural expressivity of the higher-order pi-calculus sees it enjoy wide usage. We use it to model intelligent agent interactions; agents are modelled as processes, and communication between intelligent agents is modelled naturally using the pi-calculus primitives.

The core primitive of the pi-calculus is communication. Communication occurs along named channels (such as sockets or shared memory). In its pure form, all data objects are channel names (thus new links can be established by sending the name of a new channel), although it is a small stretch from there to also include primitives such as integers.

3.1.1. Examples

We will first introduce the language via some illustrative examples.

As a first example, consider the simple case of an intelligent agent running in parallel (that is, in the same environment) with a second: this is expressed concisely with the "composition" operator, the vertical bar:

$$agent_1 \mid agent_2$$

Secondly, as an introduction to cooperation, consider an agent sending a single value — the number 3 — to a second agent. Letting the name of the communication channel be *x*, and the variable to which 3 will be bound (effectively, the formal parameter) be called *y*, then this situation is represented as follows:

$$\bar{x}[3].agent_1 \mid x(y).agent_2$$

This then reduces, in a single step, to:

$$agent_1 \mid agent_2\{3/y\}$$

The notation $agent\{3/y\}$ denotes $agent$ with every free instance of *y* replaced by 3.

Finally, we provide an example demonstrating the descriptive potential of the language for expressing cooperation amongst intelligent agents.

Assume an agent $agent_1$ that needs some information from the agent $server$ but doesn't have the necessary address. It does, however, have the address of a cooperative agent called $agent_2$ which *is* aware of the server's address. We might express this as follows:

$$p(z).z(y).agent_1 \mid \bar{p}[x].agent_2 \mid \bar{x}[3].server$$

The agents may then cooperate to achieve their goal. First, $agent_1$ receives the address of $server$ from $agent_2$ (along their shared communication channel p), resulting in the following arrangement:

$$x(y).agent_1 \mid agent_2 \mid \bar{x}[3].server$$

Then, as in an earlier example, $agent_1$ — now that is has the server's address in its possession — may receive the desired information from the server:

$$agent_1\{3/y\} \mid agent_2 \mid server$$

(For simplicity, this example assumes that the variable z doesn't occur anywhere else in $agent_1$). It is this natural ability to describe ad-hoc, re-configurable networks of communication that give the pi-calculus its appeal in situations involving cooperation.

3.2. THE HIGHER-ORDER PI-CALCULUS

It is a simple conceptual jump to go from transmitting primitive data types, to transmitting code. The only changes we must make are to allow processes to be transmitted (that is, admit processes in an output clause in the syntax), and to introduce variables that may be bound to processes (as it is necessary to distinguish between intelligent agents and channels of communication).

We now present the complete syntax (Figure 1). In the following, intelligent agents (processes) are represented by P and Q, channel names are represented by x, and agent variables by X. The option of either name *or* variable is ranged over by V, and the option of name or agent by K. The vector notation $\vec{V}$ denotes a sequence of zero or more names or variables $V_1...V_n$ (the same notation is also used to describe sequences of names, processes, and so on).

$$P \ ::= \ 0 \ \mid \ P|Q \ \mid \ P+Q \ \mid \ !P \ \mid \ (\nu V)P \ \mid \ x(\vec{V}).P \ \mid \ \bar{x}[\vec{K}].P \ \mid \ X$$

Figure 1: Language Syntax

A brief explanation follows; for more details the reader is referred to Sangiorgi [4]. All (non-variable) agents are composed on top of the inactive process 0. As mentioned previously, $P \mid Q$ represents an agent P executing in parallel with an agent Q (the language does not specify location in any way; they may be two threads on the same virtual machine, or completely independent agents on opposite sides of the world). The summation construct $P+Q$ describes the non-deterministic choice between P and Q; it is typically used to encode branching (although technically non-deterministic, the decision usually boils down to which has prior opportunity to execute). Replication is expressed as $!P$, describing forking (it may be defined recursively as $P \mid !P$). The construct $(\nu V)P$ is used to introduce the fresh name or variable V in the scope of P. Input is written as $x(\vec{V})P$ which inputs data (including other processes) along the channel x and binds them to the variables (formal parameters, in effect) $\vec{V}$ in the agent P. Output is similarly written as $\bar{x}[\vec{K}]P$, with a bar over the top of the channel name and square brackets around the outputs in order to easily distinguish between input and output constructs.

Due to space limitations, we will focus our attention on the core constructions of parallel execution and communication (input and output). For further details, the reader is referred to [5] and [6].

3.3. TYPE SYSTEMS

Type systems are generally designed to prevent certain kinds of run-time error from occurring, such as instantiation of variables to values of an incorrect type. The most immediately obvious form of run-time error in the pi calculus is arity errors on channels: for example, sending a pair of values to an agent expecting only one. So, our type system must prevent these errors from occurring. This could be achieved by associating an integer (indicating the channel arity) with each channel name. It soon becomes apparent, however, that we must also concern ourselves with the *types* of the data transmitted. That is, for example, we must distinguish between transmission of a single name used to carry a single argument, and of a single name used to carry three at a time. In a higher-order calculus we must also distinguish between a variable that may be bound to a name, and a variable that may be bound to an agent.

We begin by specifying a type for a communication channel as a sequence of types. For example, if the channel x may carry integer pairs we may assign it the type (int int). The process that outputs the name x and the number 3 then exits, written $\bar{y}[x\ 3].0$, is only well typed (assuming x has the type above) if y has the type $((\text{int int})\ \text{int})$. The type of a *process* (intelligent agent) is written with square brackets to differentiate it from a channel type. For example, we would write the following to specify that the process just discussed was well typed:

$$\bar{y}[x\ 3].0 : []$$

For simplicity, we do not consider recursive types.

4. INFERRING TRUSTWORTHINESS

Our analysis framework takes the form of an annotated type system, in which the annotations convey the information regarding trustworthiness. An intelligent agent, before engaging in cooperation with a given potential partner agent, should first perform a type-deduction on the model of the system to determine the overall trustworthiness of the potential interaction.

4.1. Trust Annotations

We annotate the basic type system as mentioned in section 3.3 with additional attributes conveying information about the integrity or trustworthiness of a particular name or intelligent agent. These annotations are modelled on a Boolean algebra, facilitating the reuse of existing results (such as unification algorithms, for example).

We let *trusted* types be tagged with the symbol T, and untrusted with U, with variables ranged over by i and j. With annotation expressions ranged over by b, the usual operations of conjunction, disjunction, and negation are defined as expected:

$$U \cdot b = U \quad T \cdot b = b \quad \hat{T} = U$$
$$U + b = b \quad T + b = T \quad \hat{U} = T$$

Figure 2: Type Annotations

We also define the ordering $U \leq b \leq T$.

4.2. Core Type Rules

When determining the trustworthiness of a network of intelligent agents, it is our contention that a judgement can only be made from a given perspective. In our case, this perspective is the set of ports through which our agent can communicate: if an untrustworthy intelligent agent can gain access to one of these ports, the network must be treated as untrustworthy. In order to represent this perspective, we carry a constant set of information throughout the deduction: the set of names that form the agent's view into the network. Any determination of the combined trustworthiness of a network is then done relative to this context.

A complete deduction actually consists of two steps. The first is the regular application of all the type rules. The second is an application of a special type rule — which is always the final rule in a deduction — which calculates the trustworthiness of the network relative to our agent's perspective.

In order to do this calculation, it is necessary to collect some additional information during the deduction. We refer to this as the *execution context*: it is a set of channel names, associated with the trustworthiness of the intelligent agents that are known to use that name. Thus, if the execution context for a particular network of intelligent agents contains the mapping $x : T$, we can guarantee that only trustworthy agents will use it and thus our intelligent agent (performing the deduction) may safely communicate along the channel x. Alternatively, if the execution context contained the mapping $x : U$ then any intelligent agent using it is potentially introducing corrupted information, so our agent should avoid the use of the channel x.

As remarked earlier, we will only discuss the core cases of parallel execution and communication. For all details (including formal presentation and all auxiliary results), the reader is referred to [5].

4.2.1. Parallel Execution

With the case of parallel execution of intelligent agents, the main quandary is what the *overall* trustworthiness should be. If we consider two agents, one considered trustworthy but the other known not to be, what should the combination be? The safe choice is to say 'untrustworthy' (that is, the lowest common denominator; note that this may be achieved by *multiplying* their annotations, as in Figure 2). This choice, however, is overly restrictive if the untrustworthy intelligent agent is unable to interfere either with the trustworthy agent or our own intelligent agent. The actual type rule is as follows:

$$\frac{\Gamma \succ^{E} P : []^{b} ; C_{P} \Theta \succ^{E} Q : []^{c} ; C_{Q}}{\Gamma, \Theta \succ^{E} P \mid Q : []^{d} ; C_{P}, C_{Q}}$$

In the above, Γ and Θ are type environments (mapping channel names to types), E is out intelligent agent's interface (the set of names through which it communicates; it is a constant through the deduction), C_P is the execution context of the agent P (and likewise C_Q), and the final annotation d is the product (as calculated through the rules in Figure 2) of the mappings — contained in the execution contexts — of all the names in the intelligent agent's interface E. This means that if there is any name used by both our agent and an untrustworthy agent under examination, then the entire network must be considered untrustworthy. Likewise, however, there may be an untrustworthy agent in the deduction, but if it has no avenue of communication with our agent then the network may still safely be considered trusted — as desired in our intuitions.

The rules for communication are considerably more complicated (at least notationally, if not conceptually) and we thus restrict ourselves to an informal overview, rather than a formal presentation.

4.2.2. Input

Firstly, consider the case of inputting data along a given channel. Our primary requirement is that untrustworthy data is never bound to a trusted variable. Conversely, however, it seems logical that we may safely bind a trusted input to an untrustworthy variable (with the only consequence being that a formerly trusted value is now treated with suspicion). This corresponds to a notion of *sub-typing*, where a trusted type is a sub-type of an untrustworthy type, and is relatively easily expressed. Our second major requirement is that data received from an untrusted channel must obviously itself be considered corrupted (the converse is not true; a secure channel may be used to transport corrupted values). We achieve this by requiring that the annotation of all input values be *multiplied* (see the rules of Figure 2) by the annotation of the channel itself (as 'trustworthy' corresponds to the multiplicative identity).

There is a further subtlety that must be considered, however, that is not so immediately obvious. We must require that all parameters to the input occur in a *context* (as expressed by the mapping in the agent's execution context) that is *at least* as trustworthy as the agent's current trustworthiness. The rationale behind this is essentially to prevent a trusted process acquiring a name used by another process in an untrusted context, and thus becoming untrustworthy (obviously not a desirable result; see section 4.3).

4.2.3. OUTPUT

Our output rule is similar in concept. We require the type of the values carried by the channel to be multiplied by the annotation of the channel, as with the input case. For the output rule, however, we reason that it should be possible to send a trusted value along an untrustworthy channel — it will merely be considered untrustworthy by the receiving agent, due to the input rule. This is similarly accomplished using a sub-typing clause, but the in opposite direction to the input rule (that is, the sub-type clause is covariant in the output case; corresponding to the well-known result in function sub-typing).

4.3. DEFINING TRUSTWORTHINESS

Space concerns dictate that we cannot include all the formal results, in particular the many auxiliary results required, however we sketch the main security results.

The first important result is a statement of *subject reduction*; in other words, that types are preserved under reduction. This states that a network of intelligent agents, deduced to have a certain trustworthiness, will have exactly the same trustworthiness after executing a few steps. This is an important result as it establishes the sanity of the type rules: a network of agents deduced to be trusted can be guaranteed not to evolve into a network that would be considered untrustworthy.

On top of this basis are built two security results.

The first is a guarantee of data integrity: it states that agents will not use untrustworthy data in a computation demanding a trusted input. It is couched in terms of an observability property: briefly, that an intelligent agent built solely out of trusted data will never receive untrustworthy inputs.

Finally, we can state what is guaranteed by the determination of the trustworthiness of a network of intelligent agents. The main security result states that a network of agents, determined to be trusted from a particular perspective (that is, with respect to a limited set of communication ports), will never evolve into a network such that an untrustworthy intelligent agent may communicate with any of those ports. Thus, a malicious agent may run in a sandbox (isolated from communication) and be considered trusted, but if there is any potential for a malicious agent to communicate through that set of ports, even through a naïve third-party intelligent agent, then the entire network must be considered as malicious.

4.4. IMPLEMENTATION

We have designed a type inference algorithm that is capable of finding the most general type for a given set of intelligent agents (and thus, how trustworthy they may be from a given perspective).

While there is insufficient space to present the full algorithm, its operation is relatively simple. It basically walks backwards through the deduction tree for the given collection of agents, applying the type rules in reverse as it does so. Unknown types and annotations are assigned variables, which are unified with other types / annotations if required to match by the type rule in operation. If a rule includes a sub-typing clause, then fresh variables are generated and constraints collected. These constraints may be solved on completion of the inference process.

It is noted that this algorithm, in the absence of other information, will generate a type that is expressed in terms of variables. As our results prove that this type is the most general (that is, *any* valid type for the system may be derived by variable substitution), it is a simple matter to reason about the potential interaction by evaluating the type with different values for the variables.

5. RELATED WORK

The work presented here essentially boils down to integrity analysis, in a concurrent and ad-hoc setting. While work in the area of integrity analysis is surprisingly scarce, it has been observed that integrity analysis is in fact dual to secrecy analysis — on which there is a great body of research.

Work on secrecy analysis using the higher-order pi-calculus is limited, however similar goals have been investigated in [7]. Work on security (secrecy) using the ambient calculus (a process calculus designed to express locality and mobility) has been undertaken in [8].

An analysis using distributed notions of trust for the purposes of authentication was presented in [9]. An analysis of trust in agent-based systems, based on past behaviour of other agents, was presented in [10] (they do not consider the integrity of the cooperation however).

6. CONCLUSIONS AND FUTURE WORK

We have presented an overview of a formal method for guaranteeing safe interactions amongst intelligent agents, based on attacks from known malicious agents or from data being corrupted during transmission. It allows an intelligent agent to decide whether or not to commence cooperation with a network of agents, even if the complexity and ad-hoc nature of the network makes a simple determination impossible. The analysis is based on an annotated type system, using the higher-order pi-calculus as the modelling language.

6.1. FUTURE WORK

The system as it currently stands assumes a "white box" model; that is, it is assumed that the source code of collaborating intelligent agents is known. This needs to be extended to handle agents of unknown construction before it may be widely deployed.

Another area for improvement is in the deduction process: currently, if the interface of the intelligent agent performing the deduction changes (for example, by opening another port for communication) the entire deduction must be repeated. It would be desirable to improve on this so that the deduction is expressed as a function of the interface exposed, and the function merely recomputed if the interface is changed.

It would also be worth investigating other process calculi. The pi-calculus has proven well suited for describing cooperation between intelligent agents, however it is fundamentally incapable of describing agent *locality*, which may be an important factor to consider (for example, the situation of an agent migrating across a firewall to interact with another agent is difficult to express in the pi-calculus). Other languages specifically designed to express locality, such as the ambient calculus [11] may prove more suitable for this task. Further investigation is required however.

7. ACKNOWLEDGEMENTS

This paper benefited greatly from the feedback of doctors Peter Vamplew and Vishv Malhotra. Hepburn was partially supported during this research by a Tasmanian Postgraduate Research Scholarship.

REFERENCES

[1] S. J. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, Prentice Hall, 2003.

[2] R. Milner, "The Polyadic Pi-Calculus: A Tutorial", in *Logic and Algebra of Specification*, Springer-Verlag, 1993.

[3] D. Yang and S Zhang, "Approach for Workflow Modelling using Pi-Calculus", 2003.

[4] D. Sangiorgi, "Expressing Mobility in Process Algebra", PhD. Thesis, 1993.

[5] Mark Hepburn, "Integrity Analysis and Coercion in Distributed Systems", PhD. Thesis (submitted), 2004.

[6] Mark Hepburn and David Wright, "Execution Contexts for Determining Trusted in a Higher-Order Pi-Calculus", Technical Report, School of Computing, University of Tasmania, 2003.

[7] J. Vivas and N. Yoshida, "Dynamic Channel Screening in the Higher-Order pi-Calculus", in Proceedings of FWAN'03, 2003.

[8] C. Bodei and F. Levi, "Security Analysis for Mobile Ambients", 2000.

[9] R. Yahalom and B. Klein and T. Beth, "Trust-based Navigation in Distributed Systems", 1994.

[10] S. Marsh, "Optimism and pessimism in trust", 1994.

[11] L. Cardelli and A. Gordon, "Mobile Ambients", in Proceedings of FoSSaCS'98, 1998.