Engineering Applications of Artificial Intelligence 70 (2018) 52–66

Contents lists available at ScienceDirect

Engineering Applications of Artificial Intelligence

journal homepage: www.elsevier.com/locate/engappai

Secure information sharing in social agent interactions using information flow analysis

Shahriar Bijani [a,*], David Robertson [b], David Aspinall [b]

[a] Computer Science Department, Shahed University, Persian Gulf Highway, Tehran, Iran
[b] Informatics School, University of Edinburgh, 10 Crichton St. Edinburgh, UK

ARTICLE INFO

Keywords:
Multi-agent systems (MASs)
Open systems
Language-based security
Information leakage
Information flow analysis
Lightweight Coordination Calculus (LCC)

ABSTRACT

When we wish to coordinate complex, cooperative tasks in open multi-agent systems, where each agent has autonomy and the agents have not been designed to work together, we need a way for the agents themselves to determine the social norms that govern collective behaviour. An effective way to define social norms for agent communication is through the use of interaction models such as those expressed in the Lightweight Coordination Calculus (LCC), a compact executable specification language based on logic programming and pi-calculus. Open multi-agent systems have experienced growing popularity in the multi-agent community and gain importance as large scale distributed systems become more widespread. A major practical limitation to such systems is security, because the very openness of such systems opens the doors to adversaries to exploit vulnerabilities introduced through acceptance of social norms.

This paper addresses a key vulnerability of security of open multi-agent systems governed by formal models of social norms (as exemplified by LCC). A fundamental limitation of conventional security mechanisms (e.g. access control and encryption) is the inability to prevent information from being propagated. Focusing on information leakage in choreography systems using LCC, we suggest a framework to detect insecure information flows. A novel security-typed LCC language is proposed to prevent information leakage.

Both static (design-time) and dynamic (run-time) security type checking are employed to guarantee no information leakage can occur in annotated agent interaction models. The proposed security type system is discussed and then formally evaluated by proving its properties. Two disadvantages of the pure dynamic analysis are its late detection and its inability to detect implicit information flows. We overcome these issues by performing static analysis. The proposed security type system supports non-interference, i.e. high-security input to the program never affect low-security output. However, it disregards information leaks due to the termination of the program.

1. Introduction

Security is a major practical limitation to the advancement of open systems and open multi-agent systems (MASs) is no exception. Although openness in open MASs makes them attractive for different new applications, new problems emerge, among which security is a key issue. Unfortunately, there remain many potential gaps in the security of open MASs and little research has been done in this area.

A MAS could be defined as a subcategory of a software system, a high level application on top of the OSI[1] networking model; therefore the security of MASs is not a completely different and new concept; it is a sub-category of computing security. However, some traditional security mechanisms resist use in MAS directly, because of the social nature of MASs and the consequent special security needs (Robles, 2008). Open MASs are particularly difficult to protect, because we can provide only minimum guarantees about the identity and behaviour of agents.

Confidentiality is one of the main features of a secure system that is challenging to be assured in open MAS. Open MASs are convenient platforms to share knowledge and information, however usually there exists some sensitive information that we want to protect. The openness of these systems increases the potential for unintentional leaking of

* Corresponding author.
E-mail addresses: bijani@shahed.ac.ir (S. Bijani), dr@inf.ed.ac.uk (D. Robertson), david.aspinall@ed.ac.uk (D. Aspinall).
[1] Open Systems Interconnection.

https://doi.org/10.1016/j.engappai.2018.01.002
Received 20 January 2017; Received in revised form 13 August 2017; Accepted 1 January 2018
0952-1976/

```
Interaction Model := {Clause,...}

Clause := Role::Def

Role := a(Type, Id)

Def := Role | Message | Def then Def | Def or Def | null<- C | Role <- C

Message := M => Role  |  M => Role <- C  |  M <= Role  | C <- M <= Role

C:= Constant | P(Term,...) | ¬ C | C ∧ C | C ∨ C

Type := Term

Id := Constant | Variable

M:= Term

Term:= Constant | Variable | P(Term,...)

Constant:= lower case character sequence or number

Variable := upper case character sequence or number
```

Fig. 2.1. LCC language syntax; principal operators are: messages (`and`), conditional (`<–`), sequence (`then`) and committed choice (`or`).

sensitive information. Thus, it is crucial to have mechanisms that guarantee confidentiality and to assure that the publicly accessible information during the interactions is what we deliberately want to share.

Information leakage denotes disclosure of secret information to unauthorised parties via insecure information flows. Information leaks in agent interactions occur when secret data are revealed through message transfers, constraints or assigning roles to agents.

An *electronic institution* (Esteva et al., 2001) or an *interaction model* is an organisation model for MASs that provides a framework to describe, specify and deploy agent interaction environments (Joseph et al., 2006). It is a formalism which defines agents' interaction rules and their permitted and prohibited actions. While interaction models can be used to implement security requirements of a multi-agent system, they also can be turned against agents to breach their security in a variety of ways, as we will show in this paper.

To employ a language-based approach to secure interaction models, we need to select an agent language. We chose the *Lightweight Coordination Calculus* (LCC) as the agents' communication language (see Section 2 for a summary of LCC).

Common security techniques such as conventional access control, encryption, digital signatures, virus signature detection and information filtering are necessary but they do not address the fundamental problem of tracking information flow in information systems, therefore, they cannot prevent all information leaks. Access control mechanisms only prevent illegal access to information resources and cannot be a substitute for information flow control (Sabelfeld and Myers, 2003). Encryption-based techniques guarantee the origin and integrity of information, but not its behaviour.

This paper is laid out as follows. First, different types of insecure information flows in open MAS governed by interaction models are introduced. Second, a security type system is proposed by defining *security types* and the *type inference rules*. Then, the security type system is evaluated by proving some of its properties. Next, the dynamic and the static approaches in the interaction type checking are reviewed and *non-interference* and *declassification* are discussed.

2. Lightweight Coordination Calculus (LCC)

In our security analysis *Lightweight Coordination Calculus* (LCC) is used to implement agents' interaction models and formulate attacks. LCC (Robertson, 2005), is a declarative language used to specify and execute social norms in a peer to peer style. LCC is a compact executable specification based on logic programming.

An interaction model in LCC is defined as a set of clauses, each of which specifies a role and its process of execution and message passing. The LCC syntax is shown in Fig. 2.1.

Each role definition specifies all of the information needed to perform that role. The definition of a role starts with: `a(roleName, PeerID)`. The principal operators are outgoing message (=>), incoming message (<=), conditional (<–), sequence (then) and committed choice (or). Constants start with lower case characters and variables (which are local to a clause) start with upper case characters. LCC terms are similar to Prolog terms, including support for list expressions. Matching of input/output messages is achieved by structure matching, as in Prolog.

The right-hand side of a conditional statement is a constraint. Constraints provide the interface between the interaction model and the internal state of the agent. These would typically be implemented as a Java component which may be private to the peer, or a shared component registered with a discovery service.

Role definitions in LCC can be recursive and the language supports structured terms in addition to variables and constants so that, although its syntax is simple, it can represent sophisticated interactions. Notice also that role definitions are "stand alone" in the sense that each role definition specifies all the information needed to complete that role. This means that definitions for roles can be distributed across a network of computers and (assuming the LCC definition is well engineered) will synchronise through message passing while otherwise operating independently.

Robertson (2005) defined the following clause expansion mechanism for agents to unpack any LCC interaction model they receive and suggested applying rewrite rules to expand the interaction state:

$$C_i \xrightarrow{M_i, M_{i+1}, P, O_i} C_{i+1} s, \ldots, C_{n-1} \xrightarrow{M_{n-1}, M_n, P, O_n} C_n$$

where C_n is an expansion of the original LCC clause C_i in terms of the interaction model P and in response to the set of received messages M_i, O_n is an output message set, M_n is a remaining unprocessed set of messages.

The rewrite rules allow an agent to conform to the interaction model by unpacking clauses, finding the next step and updating the interaction state. The rewrite rules are defined in the LCC interpreter, which should be installed on each agent running LCC codes. For more information about the LCC expansion algorithm see Robertson (2005) and Robertson et al. (2009).