Purpose-Based Policy Enforcement in Actor-Based Systems

Shahrzad Riahi, Ramtin Khosravi, Fatemeh Ghassemi

Purpose-based Policy Enforcement in Actor-based Systems

Shahrzad Riahi, Ramtin Khosravi, and Fatemeh Ghassemi

School of Electrical and Computer Engineering,
College of Engineering, University of Tehran,
Tehran, Iran
{sh.riahi, r.khosravi, fghassemi}@ut.ac.ir

Abstract. *Preserving data privacy is a challenging issue in distributed systems as private data may be propagated as part of the messages transmitted among system components. We study the problem of preserving data privacy on actor model as a well known reference model for distributed asynchronous systems. Our approach to prevent private data disclosure is to enforce purpose-based privacy policies which control the access and usage of private data. We propose a method to specify purposes based on workflows modeled by Petri nets in which transitions correspond to message communications. We first use model checking to verify whether the actor model behaves conforming to the purpose model. Then, the satisfaction of the policies are checked using data dependence analysis. We also provide a method to evaluate the effectiveness of policies through checking of private data disclosure in the presence of privacy policies. Since these checks are performed statically at design time, no runtime overhead is imposed on the system.*

Keywords: Actor-based systems, Privacy, Purpose, Data disclosure, Formal Verification, Rebeca

1 Introduction

Actor [1] is a well known model for concurrent and distributed systems, in which objects (called actors) encapsulate data and communicate via asynchronous message passing. In such systems, data of an actor can flow among other actors through message passing. Since the actors can send private data to each other as part of the transmitted messages, in systems where privacy is a concern, it is essential to protect private data from disclosure. Actor model can be used for modeling real world distributed systems, so disclosure of private data in the model indicates the privacy violation in the real world. Solove [2] classifies different types of privacy violations in four classes: information collection, information processing, information dissemination, and invasions.

Our concern in this paper, is the third case which is affected by actor communication model. A special form of information dissemination is disclosure, which,

according to [3], means "making private information known outside the group of individuals expected to know it". In actor-based systems, if there is no sufficient control on the transmitted messages and their included data, disclosure of private data may happen.

A useful method to prevent private data disclosure would be to enforce system-wide privacy policies which control the access and usage of these data in the system. In this way, private data are used only as intended. The purpose of using a private data is an important aspect of privacy protection. Purpose refers to the intention behind accessing or using data items. In other words, as stated in [4], "purposes often refer to an or a set of abstract actions". For example, patient's health record can be accessed for the purpose of treatment, research, insurance, and so on. To incorporate purpose in privacy policies, privacy constraints can be explained as access or usage control policies which contain purpose. This type of privacy policies is called purpose-based policies. Based on [5], [12], and [6], purpose-based policies can be categorized in two groups: data-centric and rule-centric policies. Data-centric policies focus on data and specify the purposes for which a data item can be used. A Rule-centric policy specifies that a subject can perform an action on a private data item with a certain purpose.

How the purpose of using a data item or performing an action is identified, is an important part of data-centric and rule-centric policy enforcement. Most existing work on specification and enforcement of purpose, do not consider semantics for purposes. Some work like [11], [12] and [6] consider that "an action is for a purpose if and only if the action is part of a plan for achieving that purpose", and define the purpose semantics using formalisms based on planning. Nevertheless, to the best of our knowledge, there is no work that specifies and enforces purposes for actor-based systems. We consider the idea of planning for specification and enforcement of purpose for actor-based systems and model plans using workflows (Sect. 4).

In this paper, we focus on avoiding disclosure of private data in the actor-based systems by enforcing purpose-based policies at system design time. We assume that the actor-based system is modeled by Rebeca modeling language. Rebeca [13] (Reactive Objects language) is an actor-based modeling language, with a formal foundation, that is used to model concurrent and distributed systems. It is important to note that our method does not depend on the choice of the language and can be tailored to any kind of actor-based modeling or programming language with the assumption that the messages are delivered in the order they are sent.

Having the system model, a set of data-centric policies, a set of rule-centric policies, and the description of the purposes, we first model the purposes in a manner suitable for actor systems and then check whether the system satisfies the given policies. We use a two-step mechanism for purpose-based policy enforcement. In the first step (called purpose enforcement), we verify whether the system works exactly based on the defined purposes (Sect. 5) and in the second step (called policy enforcement), data-centric and rule-centric policies are

checked (Sect. 6). We use model checking for purpose enforcement and data dependence analysis for policy enforcement. In addition to purpose and policy enforcement, we introduce another method, called data disclosure analysis, which determines each actor in the system can access which private data of other actors (Sect. 7). So it can be used as an evaluation of the effect of purpose-based policies on avoiding data disclosure.

If the purpose enforcement or policy enforcement step determines that the model does not behave according to the intended purpose model or policies, we guide the modeler to correct the model by providing counterexamples. But if data disclosure happens, despite the system satisfies the purpose-based policies, the purposes and policies must be reviewed.

2 Related Work

The existing methods for specifying and enforcing purpose can be categorized in three groups: self-declaration, role-based approach, and action-based approach.

In the self-declaration approach (e.g. [8], [14]), the subject (i.e. initiator of an access request) explicitly expresses the purpose of its action. This approach is based on trusting the requester to honestly declare its purpose of action. But this approach is unable to detect if a malicious user claims a false purpose. In role-based approach (e.g. [10], [7], [9]), the purpose is identified based on the subject's role in the system. This approach cannot exactly identify the purpose of an action, because members of the same role may practice different purposes in their actions.

The main problem of these two approaches is that they do not consider that the purpose of an action may be determined by "its relationships with other interrelated actions" [6]. Action-based approaches consider that "an action is for a purpose if and only if the action is part of a plan for achieving that purpose" [6]. Tschants et. al. [11] define the purpose semantics using a formalism based on planning and using a modified version of Markov Decision Processes to model this planning. With this formal semantics, they automate auditing for purpose restrictions. Jafari et. al. [12] use a modal logic language to define purpose semantics. They present a model-checking algorithm for evaluating purpose constraints in a workflow-based information system (which is modeled by a workflow formalism based on Petri nets) and use this model checker for enforcing purpose-based policies using a workflow reference monitor. Masellis et. al. [6] define semantics of purpose-aware policies based on a first-order temporal logic and design a runtime monitor for enforcing purpose-aware policies. They consider that the semantics of a purpose is its associated workflow and specify workflows using Linear-Time Temporal Logic (LTL).

We use the same idea that the semantics of a purpose is its associated workflow. [11] uses Markov Decision Processes, [12] a modal logic language, and [6] Linear-Time Temporal Logic, as the formalism for purpose semantics, but we formalize purposes using an interpretation of Petri nets tailored for actor systems (why we have chosen Petri nets is explained in Section 4). Another dif-

ference with previous work is that [6] uses run-time monitoring for enforcing purpose-aware policies, [11] tries to audit purpose restrictions, and [12] uses a workflow reference monitor to enforce purpose-based polices. We use model checking and data dependence analysis for purpose-based policy enforcement. This is performed statically at design time, so no runtime overhead is imposed on the system. This way, we can make sure that all the actors do nothing that violates the purpose-based policies and there is no disclosure of private data in this system.

3 Preliminaries

3.1 Running Example

In this section we describe an educational institute as the running example which will be used throughout this paper. We consider students request this institute for two purposes: educational consulting (called Consulting purpose) and class registration (called Registration purpose). This system includes five actors: student which requests the system for one of the two purposes, and four employees (Em1, Em2, Em3 and Em4) with different responsibilities. We consider ContactInfo (including student's name and phone number), EduRec (including student's educational record), and CPersonal (including student's complete personal information) as private data of student. We assume that if an employee knows a student's private data item without permission, then she can abuse it.

3.2 System Model in Rebeca

A Rebeca [13] model consists of a set of reactive classes (called rebec) which are concurrently executed and communicate via asynchronous message passing. Each rebec has three main parts: known rebecs (rebecs which can be receivers of this actor sending messages), state variables, and message servers. Each rebec has a FIFO queue to automatically receive messages. When a message is taken from the queue, the corresponding message server is executed atomically. We define a new type of state variables, called *private data*, and assume that each actor has its own private data (e.g. postal code, medical records, telephone number, and so on). Fig. 1 shows an incomplete portion of our running example modeled in Rebeca[1].

3.3 Purpose-based Privacy Policy

The purpose-based policies, including data-centric and rule-centric policies, are specified in an actor system as below:

[1] The complete Rebeca code for our running example is accessible from `http://ramtung.ir/privacymodel.zip`.

```
reactiveclass Student {
   knownrebecs {
      Employee1 Em1;
      Employee3 Em3;
   }
   statevars {
      @Private string ContactInfo;
      @Private string EduRec;
   }
   msgsrv StartConsulting() {
      // prepare consulting request
      Em1.ConsultingReq(ContactInfo, EduRec);
   }
   msgsrv ConsultingResult(string result) {
      // process consulting result
   }
   // other message servers
}
```

```
reactiveclass Employee1 {
   knownrebecs {
      Student St;
      Employee2 Em2;
      Employee3 Em3;
   }
   msgsrv ConsultingReq(string contact
                                ,string cv){
      //forward consulting request to Em2
      Em2.NewConsulting(cv);
   }
   msgsrv provideresult(string result){
     // forward the consulting result,
        received from Em2, to St
      St.ConsultingResult(result);
   }
   // other message servers
}
```

Fig. 1. An incomplete portion of our running example modeled in Rebeca

1. A data-centric policy is defined as a pair of a data item, which is an actor's private data item, and the purpose for which it can be used. For example, (Student's ContactInfo , Registration) specifies that the ContactInfo of a student can be used for the purpose of Registration.
2. A rule-centric policy is defined as a tuple of a subject (one of the actors in the system), a data item (an actor's private data item), an action, and the purpose. For example, (Em1 , Student's EduRec , Send , Consulting) specifies that Em1 can send the student's EduRec for the purpose of Consulting.

As we will explain in Section 4, it is sufficient for our analysis to only consider message sending as actions in an actor-based system, since we do not explicitly express the action in a rule-centric policy and assume that all the actions appeared in the rule-centric policies correspond to sending of messages in the system. The given data-centric and rule-centric policies for the running example are presented in Table 1 and Table 2 respectively.

Table 1. Data-centric policies for the running example

Actor's Private Data	Purpose
(Student,EduRec)	Consulting
(Student,ContactInfo)	Consulting
(Student,ContactInfo)	Registration
(Student,CPersonal)	Registration

Table 2. Rule-centric policies for the running example

Sender	Actor's Private Data	Purpose
Em1	(Student,EduRec)	Consulting
Em1	(Student,ContactInfo)	Registration
Student	(Student,EduRec)	Consulting
Student	(Student,ContactInfo)	Consulting
Student	(Student,ContactInfo)	Registration
Student	(Student,CPersonal)	Registration

As mentioned before, we use workflows to describe purposes in the actor model. Actions in these workflows are messages communicated among actors in the system. The workflow of Consulting purpose is defined as follow:

1. Student gets her request for educational consulting to Em1. This request includes ContactInfo and EduRec of student.
2. Em1 forwards this request to Em2. This request includes student's EduRec.
3. Em2 queries Em3 for current state of the classes, if needed, and provides the consulting result.
4. Em1 delivers the consulting result to student.

The workflow of Registration purpose is defined as follow:

1. Student gets her request for registration to Em1. This request includes ContactInfo of student.
2. Em1 forwards this request to Em3 (including student's ContactInfo).
3. Em3 requests student for complete personal information and queries Em4 for payment information.
4. When Em3 receives both of the student and Em4 responses, the registration is done.

4 Purpose Model

As mentioned in Section 2, the self-declaration and role-based approaches for specifying and enforcing purpose, do not assume a semantics for the purpose, so a major problem is the ambiguity in the interpretation of purposes. Hence, we use action-based approach and address the mentioned problem by relating the actions using the workflow-based plan.

Masellis et al. [6] refer to workflow as "collections of activities (called tasks) together with their causal relationships, so that the successful termination of a workflow corresponds to achieving the purpose which it is associated to". There are various types of workflow definition languages, that can be categorized in two groups. The first group includes models such as Petri nets [17] and process algebra [18], which have a proper formal semantics. The second group includes approaches like Web Services Business Process Execution Language (BPEL) [19] and the Business Process Modeling Notation (BPMN) [20]. These languages often have no proper formal semantics [16].

Our goal is to formalize the notion of purpose in a manner suitable for actor models. According to [15], it is possible to formalize most aspects of privacy policies by abstracting all activities as communications between actors. Workflows are normally expressed at the requirements (or business level), which comprise the tasks and the control flow among the activities in the system. During the design process, these tasks must be mapped into elements of the system model. In our case, since the actor model is based on the object-oriented paradigm, tasks are mapped into methods (message processors) of the actors based on the principles of data encapsulation. Furthermore, a behavioral model of the purpose is constructed based on the control flow specified in the workflow, to describe the order of interactions among the actors. We call this workflow "message flow".

In the case of sequential models, the behavioral model may be expressed as UML sequence diagrams. However, since actors communicate asynchronously,

the underlying behavioral model must be able to clearly express concurrent computation. Hence, we choose Petri nets to describe the global control flow of the system. Aalst [22] presents several good reasons for using Petri nets to specify workflows: "formal semantics despite the graphical nature, state-based instead of event-based, abundance of analysis techniques".

In addition to the above reasons, there are transformations from workflows described in modeling languages like BPMN and BPEL to Petri net models ([16] surveys these transformations). We consider the standard definition of Petri net [17] that consists of two finite disjoint sets of places and transitions together with a flow relation. In a Petri net, the places, transitions and flow relations are graphically represented by circles, squares and directed arcs respectively. We borrow Definition 1 and Definition 2 from [21]:

Definition 1 (Petri net). *A Petri net is a triple (P , T , F) where:*

1. *P is a finite set of places.*
2. *T is a finite set of transitions (P and T are disjoint sets).*
3. *$F \subseteq (P \times T) \cup (T \times P)$ is a set of arcs (or flow relations)*

In a Petri net, places model intermediate states and transitions model tasks. In [16], the mapping of some workflow patterns to Petri nets are presented. A Petri net that models a workflow definition is called a workflow net (WF-net).

Definition 2 (WF-net). *A Petri net PN = (P , T , F) is a WF-net (workflow net) if and only if:*

1. *PN has two special places source and sink. The source place has no input arc and the sink place has no output arc (a token in the source place corresponds to a new instance of workflow, and a token in the sink place corresponds to a completed instance of workflow).*
2. *If we add a new transition to PN which connects sink place with source place, then the resulting Petri net is strongly connected.*

We specify purposes using message flows, and model the message flows using a modified version of WF-net, referred to as message flow net (MF-net), in which transitions are messages.

Definition 3 (Message flow net (MF-net)). *A message flow net MFN = (P , T , F) is a WF-net in which:*

1. *Each transition corresponds to a specific message in actor system.*
2. *Flow relations specify the order in which the actors are allowed to take their messages.*

Each message is modeled as a triple (s, m, r) in which s is the name of the sender actor, m is the message name, and r is the name of the receiver actor. A transition (s, m, r) in a MF-net means actor r takes message m from its message queue (and starts the execution of the corresponding message server) which is sent by actor s. The MF-net models of Consulting and Registration purposes specified in running example, are shown in Fig. 2 and Fig. 3 respectively.

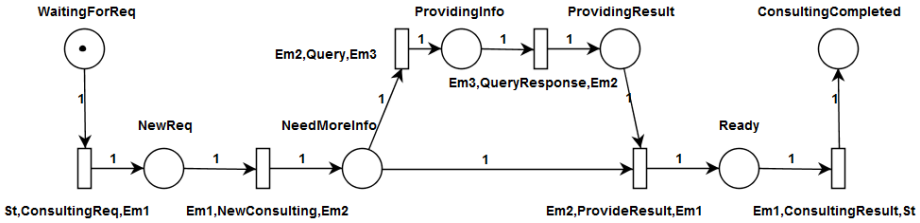

Fig. 2. Consulting purpose in educational institute modeled in MF-net

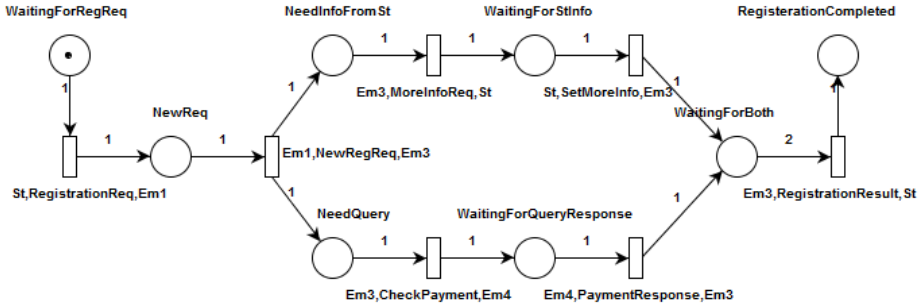

Fig. 3. Registration purpose in educational institute modeled in MF-net

5 Purpose Enforcement

As discussed in the previous section, each purpose is modeled by a MF-net. We add a new actor to the system for each MF-net, called *purpose actor* or *p-actor*, aiming to verify whether the actor system behaves according to the corresponding purpose. Each p-actor checks the state of the MF-net and decides whether an execution of a message server conforms to the corresponding purpose.

5.1 Constructing Purpose Actor

Since p-actors are defined to check the conformance of the transmitted messages to the purposes of the system, when an actor takes a message from its message queue, a copy of this message, parameterized with its sender and receiver, is sent to the corresponding p-actor. Therefore, we define one message server in the p-actor for each message in the MF-net. For simplicity, we assume that a message can only be part of one purpose.

The state of a MF-net: A state in a MF-net (as in Petri net) is represented by the distribution of tokens over the places (also referred to as marking). For keeping the state of the MF-net in the corresponding p-actor, we define an integer variable for each place that represents the number of tokens in that place. So, the state of the MF-net is modeled by the values of this set of integer variables which are the state variables of the p-actor. The variables $p_1, ..., p_n$ in Fig. 4, are the variables corresponding to the places of a Petri net.

The behavior of a MF-net: The behavior of a MF-net is modeled with conditional statements in the body of each p-actor's message server. Fig. 4 shows the description of a simple MF-net behavior. We can model different types of workflow patterns in this way.

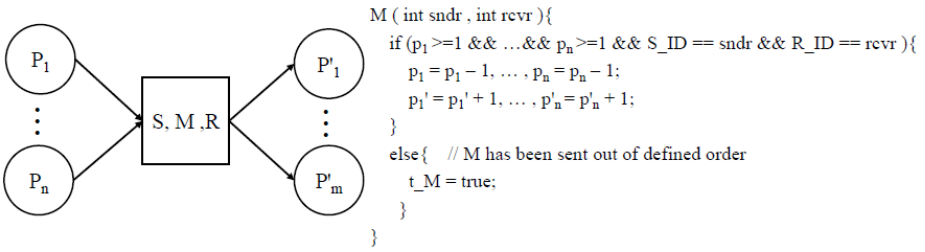

Fig. 4. Modeling a transition in the purpose actor

We call the conditional statement inside the p-actors' message servers the *transition condition*. It is noticeable that the execution of M in right side of Fig. 4 is atomic.

5.2 Purpose Verification

For each transition in a MF-net model, one boolean variable (initially false) is included as a state variable of the corresponding p-actor, and if the transition condition does not hold, then this boolean variable (e.g. t_M in Fig. 4) is set to true, representing an error has occurred. So, the property that must be checked is the invariant property $(\neg t_1) \wedge ... \wedge (\neg t_n)$ $(t_1, ..., t_n$ are the mentioned boolean variables for the transitions).

We use model checking to verify whether the system satisfies the above invariant property. If it is not satisfied, counterexamples are reported for the correction of the model. We use RMC (Rebeca Model Checker) [24] to model check our running example. The p-actor for Consulting purpose, is presented in Fig. 5.

We can define multiple instances of one MF-net in its corresponding p-actor for different instances of its execution, and distinguish them by a workflow ID.

6 Policy Enforcement

Now that we have a system that works exactly according to the defined purposes, we aim to check whether the data-centric and rule-centric policies hold in the system. As data is an important aspect of these policies, we need a mechanism which can trace the flow of data in both actors' message servers as well as sending messages to other actors. To achieve this, we use data dependence graph analysis.

```
reactiveclass RegistrationPurpose
{
    knownrebecs{
        Student St; Member1 Em1;
        Member3 Em3; Member4 Em4;
    }
    statevars{
        int WaitingForRegReq;
        int NewReq;
         ...
        int ConsultingCompleted;

        boolean t_RegistrationReq;
        boolean t_NewRegReq;
         ...
        boolean t_RegistrationResult;
    }
    msgsrv initial(){
        WaitingForRegReq = 1;
        NewReq = 0;
         ...
        ConsultingCompleted = 0;

        t_RegistrationReq = false;
        t_NewRegReq = false;
         ...
        t_RegistrationResult = false;
    }
```

Place variables

Transition variables

Initial marking

```
msgsrv RegistrationReq (int sndr, int rcvr){
    if (WaitingForRegReq > 0 && sndr == StID &&
                                rcvr == Em1ID){
        WaitingForRegReq = WaitingForRegReq - 1;
        NewReq = NewReq + 1;
    }
    else
        t_RegistrationReq = true;
}
msgsrv NewRegReq(int sndr, int rcvr){
    if (NewReq > 0 && sndr == Em1ID &&
                                  rcvr == Em3ID){
        NewReq = NewReq - 1;
        NeedInfoFromSt = NeedInfoFromSt + 1;
        NeedQuery = NeedQuery + 1;
    }
    else
        t_NewRegReq = true;
}
         ...

msgsrv RegistrationResult (int sndr, int rcvr){
    if (WaitingForBoth > 1 && sndr == Em3ID &&
                                    rcvr == StID){
        WaitingForBoth = WaitingForBoth - 2;
        ConsultingCompleted = ConsultingCompleted + 1;
        WaitingForRegReq = WaitingForRegReq + 1;
    }
    else
        t_RegistrationResult = true;
}}
```

Fig. 5. Purpose actor for Consulting purpose

6.1 Data Dependence Graph

In [23], a special dependence graph based on Rebeca [13] semantics is introduced and used as an intermediate graph representation for slicing a Rebeca model. We modify this dependence graph and use it for verifying data-centric and rule-centric policies and analyzing the disclosure of private data in the actor systems.

Rebeca Dependence Graph

Rebeca Dependence Graph (RDG) introduced in [23], has three types of nodes, including reactive class entry, message server, and statement (for Rebeca state variables and statements) nodes, and four types of edges, including data dependence, control dependence, member dependence, and parameter-in edge/activation edges. Table 3 presents how [23] models Rebeca features by RDG. Activation, formal-in and actual-in nodes are of statement nodes which are defined to model message passing.

In addition to the above dependencies, there is one more dependency called intra-rebec data dependency. According to [23], “this dependency exists between the last statement of a message server which is assigning value to a variable and the first use of that variable in another message server”. In RDG, intra-rebec data dependency is modeled using data dependence edges.

Table 3. Mapping Rebeca features to RDG according to [23]

Rebeca features	RDG nodes	RDG edges
Reactive class	A reactive class entry node	The reactive class entry node is connected to each of its state variables and message servers by the member dependence edges.
Message server	An entry node and a set of nodes representing its statements	The existing dependencies within the body of the message server modeled by data dependence edges and control dependence edges.
Message passing	An activation node	The activation node is connected to the entry node of the related message server by an activation edge.
Parameters of the messages	Formal-in and actual-in nodes	Parameter-in edges connect the formal-in and actual-in nodes.

Modified Rebeca Dependence Graph

We introduce a modified version of Rebeca dependence graph which is suitable for our policy enforcement. The modified Rebeca dependence graph differs from the original version [23] in the following ways:

1. For modeling the actors' private data, we add a new type of node, called private data node.
2. According to [23], a data dependence edge exists "between two statement nodes if assigning value to a variable at one statement might reach the usage of the same variable at another statement". We categorize assignment of value to a variable in two cases: reversible and irreversible. In reversible assignment the operands can be extracted from the result. For example in $a = b \times 10$ we can extract value of b from the value of a. In irreversible assignment, the operands cannot be conducted from the result. For example in $a = b \mod 3$, the exact value of b cannot be conducted from the value of a. We only use data dependence edges for reversible assignments.
3. We consider the activation nodes, which correspond to send statements, as a separate type of nodes.

So, in a modified Rebeca dependence graph $DDG = \{V, E\}$, $V(DDG) = V\text{-}RC \cup V\text{-}MS \cup V\text{-}PD \cup V\text{-}ST \cup V\text{-}AC$, and $E(DDG) = E\text{-}CD \cup E\text{-}DD \cup E\text{-}MD \cup E\text{-}PI$. The description of these sets are given in Table 4 and Table 5.

An incomplete portion of the data dependence graph for our running example is shown in Fig. 6 (due to space restriction, we eliminate some parts of this graph).

6.2 Data-centric and Rule-centric Policy Enforcement

For policy enforcement, we first construct the data dependence graph (DDG) for Rebeca model, and then apply Algorithm 1 to determine whether the system

Table 4. DDG nodes

Name	Description
V-RC	Set of reactive class nodes
V-PD	Set of private data nodes
V-MS	Set of message server nodes
V-ST	Set of statement nodes
V-AC	Set of activation nodes

Table 5. DDG edges

Name	Description
E-CD	Set of control dependence edges
E-DD	Set of data dependence edges
E-MD	Set of member dependence edges
E-PI	Set of parameter-in edges

satisfies the data-centric and rule-centric policies. This algorithm, gets a data dependence graph, an actor's private data item (pv), the sets of data-centric and rule-centric policies as the inputs. First, the set of all nodes which can affect the given private data is computed (lines 1-5). To check data-centric policies, all message servers that have access to pv (possibly passed through a series of messages or assignments) are selected (lines 6-7). Then, the corresponding purpose of each such message server (determined by FindPurpose(v)), is checked against the data-centric policies (lines 8-10). To check rule-centric policies, all send statements that potentially send pv as a parameter (again, possibly indirectly) are selected (lines 11- 12). Then, the permission of such communication is checked against the rule-centric policies (lines 13-15). For complete data-centric and rule-centric policy enforcement, this algorithm must be run for all private data in the system.

Algorithm 1 Purpose-based policy enforcement algorithm

Input: A dependence graph $DDG = \{V\text{-}RC \cup V\text{-}MS \cup V\text{-}PD \cup V\text{-}ST \cup V\text{-}AC,\ E\text{-}CD \cup E\text{-}DD \cup E\text{-}MD \cup E\text{-}PI\}$, one actor's private data item in form of (owner , pv), the set of data-centric policies DCPolicy and the set of rule-centric policies RCPolicy
Output: Does DDG satisfy DCPolicy and RCPolicy for (owner , pv)?

```
1: S ← ReachableFrom(DDG, pv)      // Using Depth First Search
2: For each v ∈ S
3:     For each u ∈ V(DDG)
4:         If ((u, v) ∈ E-CD  ∧  u ∉ S)
5:             S ← S ∪ {u}
6: For each v ∈ S
7:     If (v ∈ V-MS)      // If v is a message server node
8:         If ((pv, FindPurpose(v)) ∉ DCPolicy){
9:             DCPCounterExample ← (pv, FindPurpose(v))
10:            Return False }
11: For each v ∈ S
12:    If (v ∈ V-AC)      // If v is an activation node
13:        If ((FindActor(v), pv, FindPurpose(v)) ∉ RCPolicy){
14:            RCPCounterExample ← (FindActor(v), pv, FindPurpose(v))
15:            Return False }
16: Return True
```

Fig. 6 shows an example execution of Algorithm 1. The inputs of this example are the data dependence graph of our running example, the sets of policies shown in Table 1 and Table 2, and student's CPersonal private data item. As shown in Fig. 6, CPersonal can be used in *Query* message server with the purpose of Consulting. As the pair (student's CPersonal, Consulting) is not a member of Table 1, the algorithm indicates a violation of the data-centric policy. This violation occurs because *SetMoreInfo* message server (with Registration purpose) assigns CPersonal to a state variable while *Query* message server (with Consulting purpose) uses this state variable and sends its value to another actor. Although the actor is eligible to access its own state variable, its access should be controlled when it contains private data.

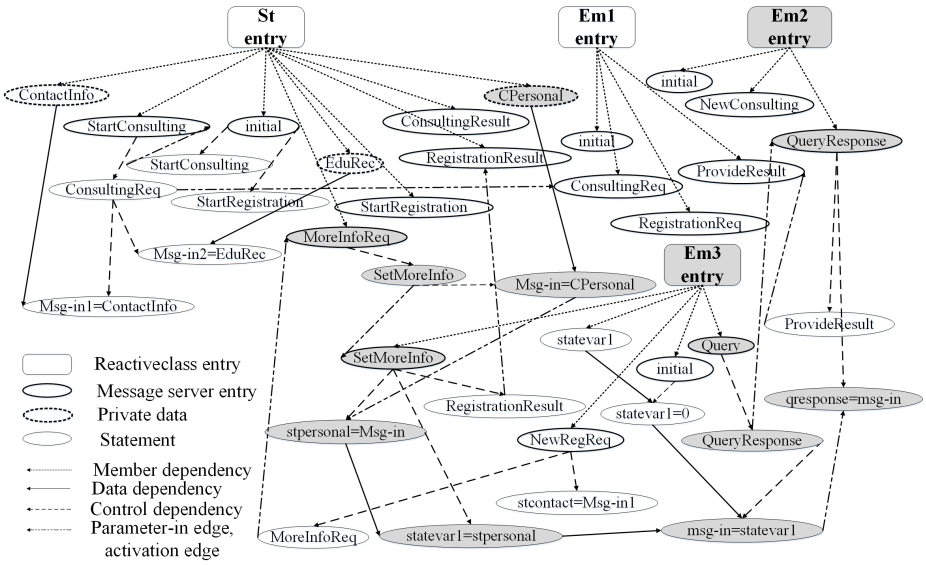

Fig. 6. Applying Algorithm 1 to DDG of running example for the sets of policies shown in Table 1 and Table 2, and student's CPersonal private data item. The gray nodes are nodes which have access to CPersonal private data item.

7 Data Disclosure Analysis

In addition to policy enforcement, we can analyze the disclosure of private data in an actor system. This analysis needs to determine each actor in the system can access which private data of other actors. This access can be done in one of the following forms:

1. Direct receive: the owner of private data directly sends its private data to another actor.

2. Indirect receive: an actor sends private data of another actor to a third actor.
3. Receive by inferring: the actors can infer other actors' private data based on some inference rules (these rules are defined based on data model or message model of the system).

In this paper we only consider the first two forms, and receive by inferring is remained as our future work.

For data disclosure analysis, we introduce Algorithm 2 based on data dependence graph analysis. In this algorithm, first the set of all nodes which can affect the given private data item, is computed (lines 2-6), and then the parent of each message server node in this set, is added to the output set (lines 7-9). The function $Parent(v)$ returns an actor node which v is a member of it. The output of this algorithm is the set of actors which can know the input private data item.

Algorithm 2 Data disclosure analysis algorithm

Input: A dependence graph $DDG = \{V\text{-}RC \cup V\text{-}MS \cup V\text{-}PD \cup V\text{-}ST \cup V\text{-}AC,\ E\text{-}CD \cup E\text{-}DD \cup E\text{-}MD \cup E\text{-}PI\}$, one actor's private data item in form of (owner , pv)
Output: *ActorsKnownpv* (the set of actors which can know pv)

```
1: ActorsKnownpv ← ∅
2: S ← ReachableFrom(DDG, pv)      // Using Depth First Search
3: For each v ∈ S
4:     For each u ∈ V(DDG)
5:         If ((u, v) ∈ E-CD  ∧  u ∉ S)
6:             S ← S ∪ {u}
7: For each v ∈ S
8:     If (v ∈ V-MS)      // If v is a message server node
9:         ActorsKnownpv ← ActorsKnownpv ∪ {Parent(v)}
10: Return ActorsKnownpv
```

If the result of this algorithm indicates the existence of data disclosure, despite the system satisfies the purpose-based policies, the purposes and policies must be reviewed.

8 Conclusion and Future Work

In this paper, we provided a way for purpose-based policies enforcement in actor-based systems with the aim of avoiding disclosure of private data in such systems. We modeled purposes using Petri nets, and make sure that the system works exactly according to them by model checking and if needed, correction of the system model. Then the data-centric and rule-centric policies are checked by analysis of the data dependence graph of the system. Data disclosure analysis algorithm has also been introduced, which can be used for evaluating of the effect of purpose-based policies on disclosure of data. However, this analysis can be used for each actor model to specify the distribution of data among actors.

Using our method, we can statically check that in a distributed asynchronous system there is no privacy violation. Since we model purposes using workflows, our method is usable for practitioners. All of our analysis are performed statically at system design time so, no runtime overhead is imposed on the system.

In future work, we intend to consider receive by inferring, as well as direct and indirect receive, for data disclosure analysis, which needs to apply required inference rules in our analysis. We also interest to provide a runtime monitoring mechanism for purpose-based policy enforcement in actor systems. This extends the scope of our method to the systems in which policies may change during time.

References

1. Agha, G.A.: ACTORS - a model of concurrent computation in distributed systems. MIT Press series in artificial intelligence. MIT Press (1985)
2. Solove, D.J. : A Taxonomy of Privacy. University of Pennsylvania Law Review, Vol. 154, No. 3, pp. 477–560 (2006)
3. Tschantz, M., Wing, J.: Formal Methods for Privacy. In: 2nd World Congress on Formal Methods, pp. 1–15. Springer-Verlag, Berlin, Heidelberg (2009)
4. Rath, A.T., and Colin, J.N.: Modeling and expressing purpose validation policy for privacy-aware usage control in distributed environment. Proceedings of the 8th International Conference on Ubiquitous Information Management and Communication, ACM New York, NY, USA (2014)
5. Jafari, M., Safavi-Naini, R., Sheppard, N.P.: Enforcing Purpose of Use via Workflows. In: Proceedings of the 8th ACM workshop on Privacy in the electronic society (WPES '09), pp. 113–116. ACM New York (2009)
6. Masellis, R.D., Ghidini, CH., Ranise, S.: A Declarative Framework for Specifying and Enforcing Purpose-Aware Policies. In: Foresti, S. (eds.) Security and Trust Management. LNCS, vol. 9331, pp. 55–71. Vienna, Austria (2015)
7. Masoumzadeh, A., Joshi, J.B.D.: PuRBAC: Purpose-Aware Role-Based Access Control. In: Meersman, R., Tari, Z. (eds.) On the Move to Meaningful Internet Systems: OTM 2008. LNCS, vol. 5332, pp. 1104–1121. Springer, Heidelberg (2008)
8. Jawad, M., Alvarado, P.S., Valduriez, P.:Design of PriServ, a privacy service for DHTs. In: PAIS '08 Proceedings of the 2008 international workshop on Privacy and anonymity in information society , pp. 21–25. ACM New York, USA (2008)
9. Byun, J., Bertino, E., Li, N.: Purpose based access control of complex data for privacy protection. In: Proceedings of the tenth ACM symposium on Access control models and technologies, pp. 102–110. New York, USA (2005)
10. Ni, Q., Bertino, E., Lobo, J., Brodie, C., Karat, C., Karat, J., Trombeta, A.: Privacy-aware role-based access control. ACM Transactions on Information and System Security (TISSEC). Vol. 13 ,Issue 3, 24:1–24:31 (2010)
11. Tschantz, M.C., Datta, A., Wing, J.M.: Formalizing and Enforcing Purpose Restrictions in Privacy Policies. In: IEEE Symposium on Security and Privacy (SP), pp. 176–190. IEEE (2012)
12. Jafari, M., Safavi-Naini, R., Fong, P.W.L, Barker, K.: A Framework for Expressing and Enforcing Purpose-Based Privacy Policies. ACM Transactions on Information and System Security (TISSEC). Vol. 17, Issue 1, 3:1–3:31 (2014)
13. Sirjani, M., Movaghar, A., Shali, A., de Boer, F.: Modeling and verification of reactive systems using Rebeca. Fundamenta Informaticae 63, 385–410 (2004)

14. Kabir, M.E., Wang, H.: Conditional purpose based access control model for privacy protection. In: Proceedings of the Twentieth Australasian Conference on Australasian Database, pp. 135–142, Vol. 92, Australia (2009)
15. Ronne, J.: Leveraging Actors for Privacy Compliance. In: Proceedings of the 2nd edition on Programming systems, languages and applications based on actors, agents, and decentralized control abstractions (AGERE! 2012), pp. 133–136. ACM New York, (2012)
16. Lohmann, N., Verbeek, E., Dijkman, R.: Petri Net Transformations for Business Processes A Surveys. In: Jensen, C., Aalst, W.M.P. (eds.) Transactions on Petri Nets and Other Models of Concurrency II. LNCS, vol. 5460, pp. 46–63. Springer Berlin Heidelberg (2009)
17. Reisig, W.: Petri Nets, An Introduction. EATCS Monographs on Theoretical Computer Science, Springer-Verlag Berlin Heidelberg (1985)
18. Baeten, J.C.M., Weijland, W.P.: Process Algebra. Cambridge tracts in theoretical computer science, vol. 18. Cambridge University Press, Cambridge (1990)
19. Web Services Business Process Execution Language Version 2.0, OASIS Standard, April 11, 2007, OASIS (2007), http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.pdf
20. OMG: Business Process Modeling Notation (BPMN) Version 2.0., Object Management Group (2011), http://www.omg.org/spec/BPMN/2.0/
21. Aalst, W.M.P.: The Application of Petri nets to Workflow Management. The Journal of Circuits, Systems and Computers, vol. 8, no. 1, pp. 21–66 (1998)
22. Aalst, W.M.P.: Three Good Reasons for Using a Petri-Net-Based Workflow Management System. Information and Process Integration in Enterprises, Vol. 428, The Springer International Series in Engineering and Computer Science, pp. 161–182 (1998)
23. Sabouri, H., Sirjani, M.: Slicing-based Reductions for Rebeca. In: Proceedings of FACS08, pp. 209–224. Elsevier ENTCS Post-proceedings (2008)
24. RMC (Rebeca Model Checker) tool (2016), http://www.rebeca-lang.org/wiki/pmwiki.php/Tools/RMC