Aligning Service-Oriented Architectures with Security Requirements

Mattia Salnitri, Fabiano Dalpiaz, and Paolo Giorgini

University of Trento, Italy
{mattia.salnitri,f.dalpiaz,paolo.giorgini}@unitn.it

Abstract. Aligning requirements and architectures is a long-standing concern in software engineering. Alignment is crucial in the area of systems evolution, wherein requirements and system architectures keep changing after system deployment. We address a specific alignment problem, namely, checking the compliance of a service-oriented architecture—representing a composite service—with security requirements. Service-oriented architectures are dynamic (services can be replaced on-the-fly), and assessing compliance with security requirements is key, since non-compliance may lead to sanctions as well as privacy violation. After motivating and describing the problem, we propose algorithms to check two specific security requirements: non-disclosure and non-repudiation. We illustrate the approach using an e-government scenario.

Keywords: SOA, alignment, evolution, security requirements.

1 Introduction

The alignment between requirements and software architectures (R/A alignment, from now on) is an age-old yet actual problem in software engineering in general, and in requirements engineering in particular [17, 22]. This problem is traditionally addressed at design-time in a top-down fashion [2, 16], by providing methodological guidelines to requirements engineers and system architects for deriving an architecture that satisfies a given requirements specification.

Though useful at design-time, these approaches are only a partial solution when considering that both requirements and architectures are subject to evolution after system deployment. Requirements evolve [11] due to changes in the stakeholders needs, in organizational policies, in norms and laws in the deployment environment, and as a result of feedback about system operation. Architectural evolution [1] can be either internal—the architecture topology changes—or external—the specification of components and interactions is altered.

Architectural evolution is driven by requirements evolution, i.e., architectures need to evolve when they do not satisfy their requirements. Semi-automated runtime compliance verification requires understanding and formalizing the conceptual relationship between requirements and architectural models. The challenge is to relate these two artifacts, that exploit different modeling abstractions, which refer to the problem space and the solution space, respectively.

R. Meersman et al. (Eds.): OTM 2012, Part I, LNCS 7565, pp. 232–249, 2012.

In this paper, we are interested in the alignment between security requirements and service-oriented architectures:

- *security requirements* are crucial, as non compliance may violate norms, e.g., about privacy, or usage / modification of official documents. In turn, this may imply monetary compensations, dissatisfaction of customers, and decreasing reputation;
- *service-oriented architectures* represent an inherently evolving type of architecture, where composite services [3] may evolve—either due to designer intervention or through self-recomposition—when services are replaced or when the composition structure changes.

Our contribution is a semi-automated approach to support runtime alignment between composite services and security requirements. First, the analysts are expected to conceptually connect the architecture specification with the security requirements, i.e., linking concepts in the requirements (goals, actors, security needs) with concepts in the composite service (activity, participant, data flow). Second, automated algorithms check compliance at run-time, whenever changes in the architecture or in the requirements occur. Compared to existing approaches, ours minimizes human involvement after system deployment.

Organization. Sec. 2 presents the baseline languages for security requirements and composite services. Sec. 3 details the R/A alignment problem. Sec. 4 describes the conceptual link between the two models. Sec. 5 illustrates two algorithms for checking security requirements: non-disclosure and non-repudiation. Sec. 6 discusses related work and presents future directions.

2 Baseline

In Sec. 2.1 we introduce the adopted modeling language for security requirements, while in Sec. 2.2 the language to describe the architecture of service compositions.

Example 1 (eGovernment). Land selling involves not only finding a buyer, but also exchanging documents with governmental bodies. The municipality has to certify that the land is residential zoning. We suppose land selling is supported by an eGov application that sends the official contract (including the municipality's certification) to the ministry (who can object), and archives the contract. □

2.1 Security Requirements with STS-ml

Many security requirements modeling languages have been proposed so far. Some extend UML—mainly abuse cases [14] and misuse cases [19]—by adding undesirable usage scenarios of the system. In a similar spirit, Lamsweerde [24] extends goal modeling by adding an anti-model that describes the goals of attackers. Differently, SI* [9] and Secure Tropos [15] aim at modeling security needs. We choose STS-ml [5], a goal-oriented language for service-oriented settings, in which security requirements constrain the interactions between actors.

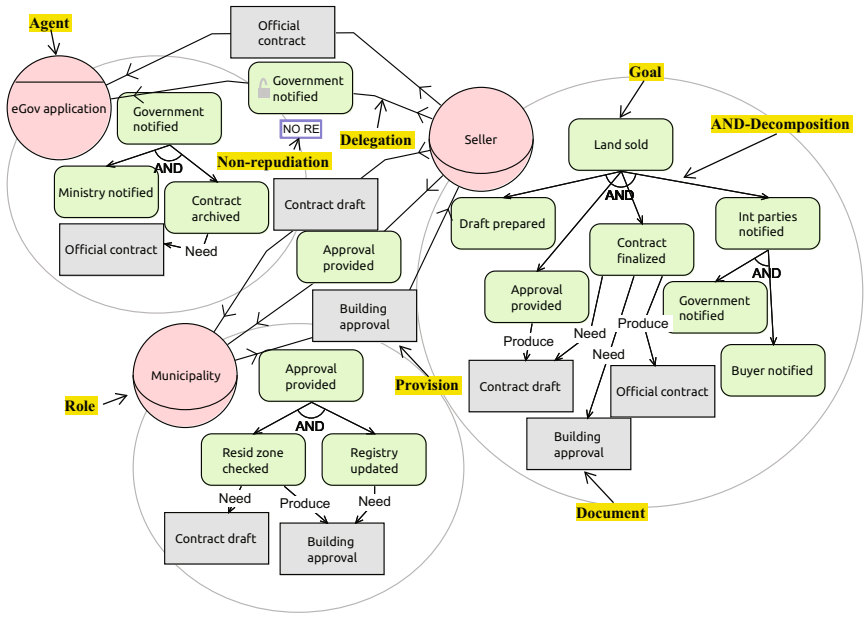

Fig. 1. STS-ml social view for the eGovernment example

In STS-ml, requirements models are expressed by three views: (i) the social view describes the main actors, their rationale, goal delegations, and document provisions (Fig. 1); (ii) the information view defines the relationship between information and documents (Fig. 2); and (iii) the authorization view represents the authorizations about information that actors grant one to another (Fig. 3).

Social View. Two types of actors are modeled: agents—concrete entities that are known at design-time (e.g., *eGov application*)—and roles—that can be played by different agents at runtime (e.g., *Seller*). An actor's rationale is an AND/OR goal tree (e.g., the root goal of the *Seller* is to have her *Land sold*). To achieve their goals, actors need, modify, and produce documents (e.g., *Seller* needs document *Contract draft* to achieve goal *Contract finalized*). Two social relationships link couples of actors: goal delegation and document provision. Goal delegations can be subject to security needs (e.g., non-repudiation, no-delegation, redundancy, ...). In Fig. 1, the *Seller* delegates goal *Government notified* to the *eGov application*, requesting the delegatee not to repudiate the delegation.

Information View. Documents and information are linked one to another. The "Tangible by" relation indicates that an information is represented by a document. In Fig. 2, *Sale information* is made tangible by documents *Official contract* and *Contract draft*. The "Part of" relation defines sub-information and sub-documents. Ownership relates an actor to the information that she owns.

Authorization View. It represents the permissions on information that actors grant one to another. An authorization is indicated as an arrow connecting two

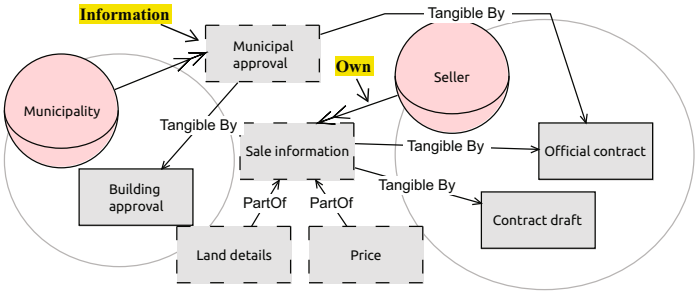

Fig. 2. STS-ml information view for the eGovernment example

actors with a middle box. Such box details the granted permissions (Use, Modify, Produce, Distribute) on documents representing specific information. Authorizations can be limited by defining a scope: one or more goals for whose fulfillment the permissions are granted. In Fig. 3, the *Seller* authorizes the *Municipality* to use *Sale information* in the scope of goal *Approval provided*.

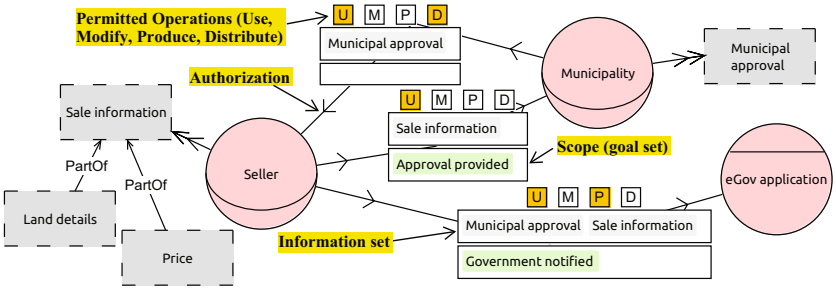

Fig. 3. STS-ml authorization view for the eGovernment example

Starting from the three views that constitute an STS-ml model, a Security Requirements Specification (SRS) is derived using the STS-tool[1]. The specification details the security requirements—social commitments between couples of actors—, as well as a knowledge base, that makes the SRS self-contained.

A social commitment is a quaternary relation C(x,y,p,q), where a debtor actor x commits to a creditor actor y that, if the proposition p is brought about, then the proposition q will be brought about [20]. STS-ml supports a specialized version of commitments to express security requirements; in particular, the consequent q is about the satisfaction of a security need.

Table 1 is a partial SRS for the eGovernment scenario. C_1 is a security requirement about non-repudiation: the *eGov application* commits to the *Seller* that, if goal *Government notified* is delegated, such delegation will not be repudiated. Both C_2 and C_3 concern non-disclosure. In C_3, the *Municipality* commits to the

[1] `http://www.sts-tool.eu`

Table 1. Part of the security requirements specification for the scenario in Figures 1-3

Security requirements:
C_1: C(eGov application, Seller, D=delegation(Seller, eGov application, Government notified), non-rep(D)),
C_2: C(e-Gov application, Seller, $\top$, non-disc(Municipal approval$\wedge$Sale information),
C_3: C(Municipality, Seller, $\top$, non-discl(Sale information), ...
Knowledge base:
part-of(Land details, Sale information), part-of(Price, Sale information), ...
tangible-by(Sale information, Official contract), tangible-by(Sale information, Contract draft), ...
owns(Seller, Sale information), ...

Seller that *Sale information* will not be disclosed. The knowledge base describes relations that enable the understanding of the relationships related to information. For example, it describes the information parts that *Sale information* is composed of, it details the documents that enable exchanging *Sale information*, and information owners.

2.2 BPMN with Security Extensions

We describe the architecture of services composition—i.e., the way services are interconnected—by using an extended version of the Business Process Modeling Notation (BPMN) with support for security annotations (inspired by [18]). The notation we use in this paper can be easily adopted by existing tools, provided that they include support for representing the data flow between activities and enable the definition of custom security-related activities.

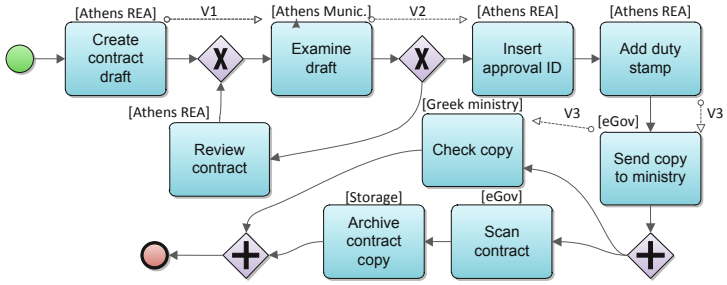

Fig. 4. Security-extended BPMN for the eGovernment example

Fig. 4 shows a security-extended BPMN model for the eGovernment scenario. Such model is a set of activities (e.g., *Create contract draft*) carried out by different performers (e.g., *Athens REA*) following the specified control flow (solid arrows, exclusive and parallel gates). For example, after the real estate company

Athens REA has completed activity *Create contract draft*, the municipality of Athens has to examine the draft. The model also represents the information flow through labeled dashed arrows, whose label indicates the variable that is transferred between two performers. So as to determine how an activity uses a variable, we consider the incoming and outgoing arrows related to that variable:

- *Create contract draft* produces *V1*, as it has an outgoing arrow labeled *V1* but no incoming arrow;
- *Examine draft* uses *V1*, as there is only an incoming arrow labeled *V1*;
- *Send copy to ministry* may modify *V3*, as there are both incoming and outgoing arrows labeled *V3*.

3 The R/A Alignment Problem

We motivate and illustrate the evolution of requirements and architectures, and show its implications on R/A alignment. We refer to specific requirements specifications (business processes) with the labels $SRS_1, \ldots, SRS_n$ ($BP_1, \ldots, BP_n$). After illustrating the problem in Sec. 3.1, we analyze the evolution of security requirements and service compositions in Sec. 3.2 and Sec. 3.3, respectively.

3.1 The R/A Alignment Problem Explained

Fig. 5 shows some possible evolutions of requirements and architectures, and emphasizes their impact on R/A alignment. At design-time, the service composition BP_1 meets the initial security requirements specification SRS_1. When the composition self-reconfigures at runtime, and is replaced by another composition BP_2, alignment with SRS_1 is lost. The requirements analyst relaxes some security requirements of secondary importance to the stakeholders. This leads to a new specification SRS_2 which guarantees R/A alignment. Later, stricter privacy laws are introduced. The analyst revises the requirements models, leading to a new specification SRS_3. BP_2 is now non-aligned with SRS_3. Then, the system architect defines a new composition BP_3 that is aligned with SRS_3.

Non-compliance with security requirements has severe consequences [4]. Loss of privacy and unauthorized disclosure of data are serious threats, especially when loss of confidentiality can potentially open the doors to world-wide access to personal information through the Internet. Another consequence is data integrity: imagine what would happen if the price for a land sale were changed by an unauthorized user! Compliance with organizational processes and standards is also at risk: take the case, for instance, if binding-of-duties, separation-of-duties, and redundancy are not provided as expected. In general, security requirements violations have consequences from an economical perspective [6], lead to penalties imposed by laws, and decrease the reputation of involved organizations.

3.2 Security Requirements Evolution

The effect of the evolution of security requirements is that a new specification SRS_2 replaces the previously valid specification SRS_1, and the two differ in at

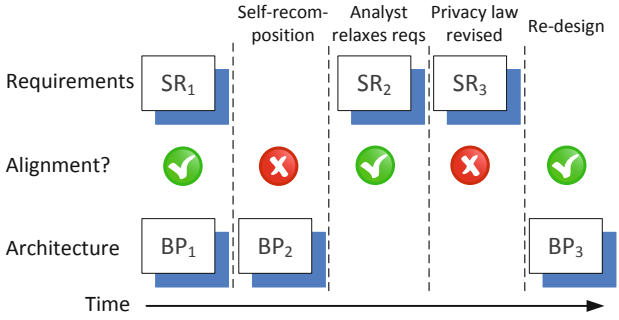

Fig. 5. R/A alignment threatened by the evolution of requirements and architecture

least one element. Since our considered security requirements specification is derived from an STS-ml model, changes in *SRS* are triggered by changes in the social, resource, and authorization views of the corresponding STS-ml model.

The effects of requirements evolution may turn a compliant architecture into a non-compliant one, or vice-versa. These effects arise due to the occurrence of specific events that cause the evolution of security requirements:

- *Revised security needs*: the stakeholders change their desires about security, or the organizational policies are revised. For example, in the eGovernment scenario, the municipality may establish that the seller cannot transfer its authorization on documents representing the municipal approval;
- *Broadened analysis scope*: the requirements analyst may realize that the conducted analysis does not cover all the important stakeholders. For example, the requirements analyst may realize that the STS-ml models do not include any actor representing the competent ministry. In turn, this will lead to further security requirements about confidentiality;
- *Evolution of norms*: the security norms in the legal environment where the service composition is deployed change. New laws imposing stricter security could be introduced, existing norms could be strengthened, weakened, and abrogated. For example, consider a new municipal law stating that the prices of land transactions shall not be notified to governmental bodies;
- *Legal context switching*: the actors participating in security requirements are not always located in the same legal context. If the actor is a role, in particular, different agents—in different locations—may adopt such role; depending on the location, a different set of laws applies. For instance, a Spanish buyer would have to comply both with Greek and Spanish laws.

3.3 Service Composition Evolution

The architecture of a secure composite service BP_2 evolves a previous architecture BP_1 if they differ in either: (i) the activities—including the security-related activities—they are composed of; (ii) the control flow between activities; (iii) the information flow between activities; or (iv) the performers assigned to activities.

A service composition may evolve either autonomously, or due to human intervention. *Manual redesign* occurs when a designer defines an alternative composition to better satisfy her needs. *Self-recomposition* occurs when, based on the monitored runtime data, a workflow engine replaces the composition.

Diverse reasons may trigger the evolution of a composition, among which:

- *No available or no trustworthy service*: there is no available service for a specific activity, or existing providers are not sufficiently trustworthy. For example, if no employee were available to add the duty stamp, a web service that uses a electronic duty stamp could be introduced;
- *Functional failure or under-performance*: the service composition does not deliver the expected outcome, or its performance is not adequate. For instance, the service composition about eGovernment could be too slow for Athens REA, who may decide to outsource the approval process chunk;
- *Security failure*: the service composition fails to correctly deliver the security features declared at the architectural level. For example, if the service responsible for sending a copy of the contract to the ministry publishes it—violating the architectural information flow—confidentiality is violated;

4 Supporting R/A Alignment: A Methodological Approach

We describe a methodological approach to support R/A alignment. It starts at design-time by establishing initial alignment, and is continuously carried out at runtime, so as to identify non-alignment and to re-establish alignment in case of evolutions. The approach is semi-automated, as it includes activities to be conducted by analysts as well as machine-executable algorithms.

Fig. 6 illustrates our approach. Initially, an analyst models and analyzes the security requirements—e.g., using STS-ml, as explained in Sec. 2.1—and derives the *Security Requirements Specification* (*SRS*). As soon as a business process *BP* defining the architecture a service composition is available, the analyst has to devise a conceptual mapping between elements in *SRS* and elements in *BP*. This step (Sec. 4.1) produces a conceptual mapping *CM*, which enables the automated instantiation of the security requirements. Such activity (Sec. 4.2) interprets the *SRS*—which is expressed over concepts in STS-ml such as actors and goals—in terms of the concepts that characterize service composition at hand—such as performers and activities—and produces an *Instantiated SRS* (*ISRS*). This document and the *BP* feed alignment checking, a set of automated algorithms (some described in Sec. 5) that compute the actual R/A alignment. Based on the alignment status, different paths are followed:

- *Non-alignment*: the analyst has to revise either the composition, the *SRS*, or both. While the basic case involves the analyst in a redesign activity, an interesting option is to trigger an automated service re-composition. The applied revisions require the conceptual mapping to be adjusted; then, instantiation and checking are executed on the revised artifacts;

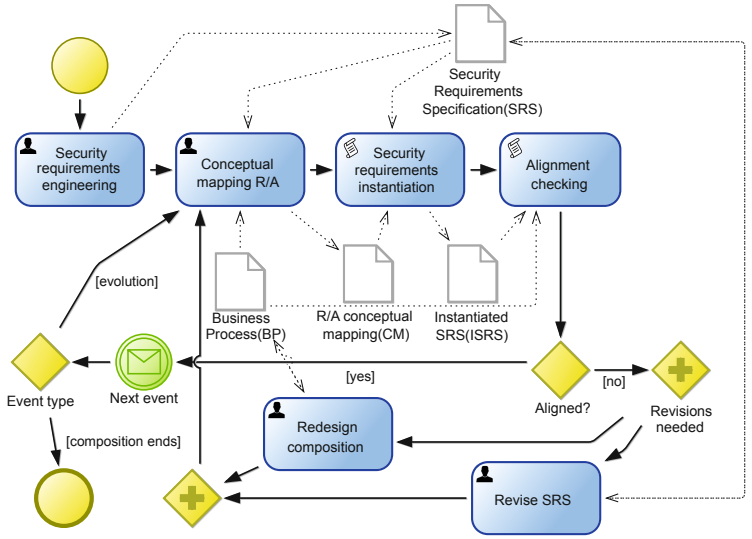

Fig. 6. Our method for supporting R/A alignment in presence of evolution

– *Alignment*: our framework waits for the next event to happen. If the event is the occurrence of an evolutionary action, the process re-starts from the conceptual mapping; if the composition terminates, the process ends.

In this paper, we detail the activities in the upper part of Fig. 6, which constitute our basic framework. We leave to future work the activities concerned with fixing non-compliance and repeating the check when an artifact evolves.

4.1 Conceptual Mapping R/A

This activity is carried out by an analyst to define a conceptual mapping *CM* between the *SRS* and the *BP*. The creation of *CM* is necessary because the architectural model is at a different level of abstraction from the requirements specification. Therefore, the skills of a human analyst to reconcile these abstraction level gap are required. *CM* consists of a set of relations between *BP* and *SRS* elements:

– *Participants to actors*: for each participant in *BP*, the analyst identifies links with the actors appearing in *SRS* as debtor or creditor. To do so, *BP* participants shall be specialized into agents or roles. Two possible relationships are possible: (i) *is-a* relates one or more *BP* roles (agents) to one *SRS* role (agent); (ii) *plays* links one or more *BP* agents to one or more *SRS* roles;
– *Activities to goals*: all activities in *BP* shall be linked to relevant goals in *SRS* (if any relevant goal exists). The relationship we support is called *relates-to*, and indicates that the activity is performed in order to achieve a goal. An activity may be related to several goals, and a goal to several activities;

– *Variables to documents*: each variable appearing in *BP* is linked to zero or one documents in *SRS* via a *represents* relationship, indicating that the variable represents a document.

Table 2. R/A conceptual mapping for the eGovernment scenario

plays(Athens REA, Seller), **plays**(Athens Munic., Municipality), **is-a**(eGov,eGov application), **is-a**(Storage, eGov application)
relates-to(Create contract draft, Draft prepared), **relates-to**(Review draft, Draft prepared), **relates-to**(Examine draft, Resid zone checked), **relates-to**(Insert approval ID, Approval provided), **relates-to**(Add duty stamp, Contract finalized), **relates-to**(Send copy to ministry, Government notified), **relates-to**(Scan contract, Government notified), **relates-to**(Archive contract copy, Government notified)
represents(V1, Contract draft), **represents**(V2, Building approval), **represents**(V3, Official contract)

Table 2 shows a possible *CM* between the *SRS* in Table 1 and the *BP* in Fig. 4. There are two *BP* participants (*eGov* and *Storage*) which are linked (by an *is-a* relation) to the agent *eGov application* in *SRS*. The *BP* participant *Greek ministry* is not linked to any actor in *SRS*. Among the mapped activities, both *Create contract draft* and *Review draft* are related to goal *Draft prepared*.

Security requirements are not directly mapped to security activities. These requirements correspond to patterns and anti-patterns which will be checked using algorithms like the ones described in Sec. 5.

4.2 Security Requirements Instantiation

The conceptual mapping *CM* enables to instantiate *SRS* on the service composition *BP*, i.e., understanding these requirements in terms of *BP* concepts (e.g., participant, activity, variable). Instantiation of security requirements comprises two main steps: (i) instantiation of debtors and creditors of security requirements, according to *CM*, into participants in *BP* (see Sec. 4.2; and (ii) instantiating the security needs, originally expressed on *SRS* actors, to *BP* participants (see Sec. 4.2). As a result, these steps return an instantiated specification *ISRS*.

Debtor and Creditor Instantiation. Each commitment in *SRS* is instantiated, based on *CM*, by checking the corresponding *BP* performers associated with the debtor and creditor of that commitment. If *CM* says that a debtor actor in an *SRS* commitment is linked to two *BP* performers, that commitment will be instantiated twice, as each performer has to create a commitment instance.

Algorithm 1 (function InstantiateDebtor) shows how the debtor instantiation is performed, and returns an instantiated set of commitments *instCommitments*. If the debtor is a role (lines 2-3), the *CM* is searched for, in order to identify all performers who play that role. These performers are added to the set $bpPerformers_{deb}$. Then (line 4), *is-a* relationships involving the debtor

Algorithm 1 Debtor instantiation for a commitment

```
InstantiateDebtor(C(deb,cred,p,q), CM)
1  instCommitments ← ∅
2  if TypeOf(deb) = role then
3      bpPerformers_deb.Add(CM.Search(plays(*, deb)))
4  bpPerformers_deb.Add(CM.Search(is-a(*, deb)))
5  for each perf ∈ bpPerformers_deb do
6      instCommitments.Add(c(perf, cred, p, q))
7  return instCommitments
```

are searched for, and the set $bpPerformers_{deb}$ is enriched. For each of these performers (lines 5-6), an instance of the original commitment is created with that performer as debtor; the commitment instance is added to *instCommitments*.

The returned set of commitments is processed to instantiate the creditor. The algorithm for the creditor is analogous to Algorithm 1, with the only difference that the creditor is replaced (instead of the debtor).

Example 2 (Actors instantiation). Consider commitment C_1:C(eGov application, Seller, non-discl(Sale information)) from Table 1, and the mapping in Table 2. The *SRS* role *eGov application* is linked to two participants: *eGov* and *Storage*. The instantiation of C_1 creates two commitments:

$\mathsf{C}_{1.1}$:C(eGov, Athens REA, non-discl(Sale information))
$\mathsf{C}_{1.2}$:C(Storage, Athens REA, non-discl(Sale information))

The debtor and creditor of these commitments are now referring to *BP*. □

Security Need Instantiation. After the commitments have been instantiated with respect to debtor and creditor, they are instantiated with respect to the security need in the consequent. The general form of a consequent is a parametric predicate: *sec-need*(par_1,...,par_n). Unlike the previous instantiation activities, we cannot devise a generic algorithm for security needs. We detail specific algorithms in Sec. 5. Here, we illustrate the problem on two security need types:

- Separation of duties: C(Seller, Municipality, SoD(Approval provided, Draft prepared)). The instantiation algorithm will create $n \times m$ commitments, where n is the number of activities that relate to *Approval provided* and m the number of activities that relate to *Draft prepared*;
- Need-to-know: C(Seller, municipality, NtK(Municipal approval, Draft prepared)). Even if there are p activities related to the goal ($p > 1$), a single commitment is created. In our example, the instance will contain the set of activities {*Create contract draft, Review draft*}.

5 Compliance Checking

We detail two algorithms to check R/A alignment—to execute alignment checking in Fig. 6—for two security requirements types in STS-ml: non-disclosure of information (Sec. 5.1) and non-repudiation of a delegation (Sec. 5.2).

These algorithms need the conceptual mapping *CM*, the security requirements specification *SRS* (and its instantiated version *ISRS*), and the business process of a composition *BP*. The result is the compliance status of *BP*. A possible extension could return also the cause for non-compliance.

The service composition architecture *BP* is a BPMN model extended with security-related activities. Typically, *BP* would be defined by enriching a regular business process model. The security activities are special types of activities meant to guarantee security and reliability; for instance, sending an acknowledgement, encrypting some data, replicating stored data, etc. In [18], these activities are graphically represented as annotations on regular activities.

In general, a security requirement is satisfied if *BP* exhibits a specific structural pattern, either in terms of the sequencing of activities (the control flow), the information flow, the included security activities, and the assigned performers. Our algorithms enable verifying the presence (absence) of specific patterns (anti-patterns). Our ultimate objective is to build a repository of algorithms to allow checking all the security requirements supported by STS-ml.

5.1 Non-disclosure

In STS-ml, a non-disclosure security requirement is a commitment C(deb, cred, ⊤, non-discl(info)), where actor *deb* commits to actor *cred* that information *info* will not be distributed to other actors than *cred* or the owner of *info*.

Algorithm 2 Non-Disclosure instantiation

```
INSTANTIATEND(C(deb, cred, ⊤, non-discl(info)), CM, SRS)
1  instCommitments ← ∅
2  documents ← SRS.SEARCH(tangible-by(info, *))
3  for each doc ∈ documents
4  do bpVariables ← CM.SEARCH(represents(*, doc))
5     for each var ∈ bpVariables
6     do instCommitments.ADD(C(deb, cred, ⊤, non-discl(var)))
7  return instCommitments
```

Algorithm 2 (INSTANTIATEND) takes in input a non-disclosure requirement, *CM*, and *SRS* (after debtor and creditor instantiation). It searches *SRS* for all the documents that make tangible the information (line 2). For each document (lines 3-6), *CM* is searched for variables representing that document (line 4). A non-disclosure commitment instance is created (line 6) for each of those variables.

The commitments returned by Algorithm 2 feed Algorithm 3, which checks whether that commitment is satisfied or not by *BP*. If any commitment is not satisfied, R/A alignment does not hold. Given a commitment instance C(deb, cred, ⊤, non-discl(var)) and the process *BP*, Algorithm 3 determines whether there is at least one activity, performed by the debtor *deb*, that transfers variable *var* to an activity executed by a performer that differs from the variable owner

Algorithm 3 Non-Disclosure Verification

```
VerifyND(C(deb, cred, ⊤, non-discl(var)), BP, SRS, CM)
 1  actByDeb ← BP.ActivitiesBy(deb)
 2  actByCred ← BP.ActivitiesBy(cred)
 3  actUsingVar ← BP.ActivitiesUsing(var)
 4  doc ← CM.Search(represents(var, *))
 5  if doc ≠ null
 6    then info ← SRS.Search(tangible-by(*, doc))
 7         for each i ∈ info
 8         do own ← SRS.Search(owns(*, i))
 9            actByOwner.Add(BP.ActivitiesBy(own))
10  actByOthers ← actUsingVar \ actByDeb \ actByCred \ actByOwner
11  for each a_i ∈ actByDeb
12  do for each a_j ∈ actByOthers
13     do if var ∈ output(a_i) ∩ input(a_j)
14           then return non-compliant
15  return compliant
```

or the creditor *cred*. In order to check this, a number of sets of activities are defined in the algorithm by querying *BP* (e.g., *actByDeb* indicates all activities performed by *deb*), *CM* (e.g., *doc* is the document that the variable represents, if any) , and *SRS* (e.g., *info* is the set of information the document makes tangible).

Example 3 (Checking non-disclosure). Suppose an evolution of the security requirements occurs. Take *BP* as in Fig. 4, *CM* as in Table 2, and the evolved requirement C_3:C(eGov application, Seller, ⊤, non-discl(Sale information)). By executing the instantiation algorithms for debtor (Algorithm 1), creditor, and security need (Algorithm 2), we obtain the following four commitment instances:

$\mathsf{C}_{3.1}$:C(eGov, Athens REA, ⊤, non-discl(V1))
$\mathsf{C}_{3.2}$:C(eGov, Athens REA, ⊤, non-discl(V3))
$\mathsf{C}_{3.3}$:C(Storage, Athens REA, ⊤, non-discl(V1))
$\mathsf{C}_{3.4}$:C(Storage, Athens REA, ⊤, non-discl(V3))

By running Algorithm 3 on the four commitments, it returns compliance for $\mathsf{C}_{3.1}$, $\mathsf{C}_{3.3}$, and $\mathsf{C}_{3.4}$, while it returns non-compliance for $\mathsf{C}_{3.2}$. Indeed, *eGov*'s activity "Send copy to ministry" transfers *V3* to activity "Check copy" performed by the *Greek ministry*, who is neither the owner of *V3* nor the creditor of $\mathsf{C}_{3.2}$.

In order to re-establish R/A alignment, either *BP* is re-designed so that *V3* is not transferred to the *Greek ministry*, or the non-disclosure requirement is relaxed, by allowing sale information to be re-distributed. □

5.2 Non-Repudiation

Non-repudiation in STS-ml is defined as a commitment C(d, c, del=delegate(c,d,g), non-rep(del)), where actor d commits to actor c that the delegator (c) will be provided with a proof that the delegation of goal g is acknowledged by d (so that d cannot able repudiate the delegation later on).

Algorithm 4 Non-Repudiation instantiation

```
InstantiateNR(C(d, c, del=delegate(c,d,g), non-rep(del)), CM)
1  instCommitments ← ∅
2  activities ← CM.Search(relates-to(*, g))
3  for each act ∈ activities
4  do inst ← C(d, c, del=delegate(c,d,act), non-rep(del))
5     instCommitments.Add(inst)
6  return instCommitments
```

Function InstantiateNR (Algorithm 4) takes in input a non-repudiation commitment and *CM*. *CM* is searched for all activities that are linked to the delegated goal g via a *relates-to* relationship. For each of these activities (lines 3-5), a commitment instance is created where the debtor d commits not to repudiate the delegation of that activity.

Algorithm 5 Non-repudiation verification

```
VerifyNR(actNR, perf, actCurr, found, visited)
 1  switch TypeOf(actCurr)
 2    case ack :
 3      if (actCurr ∈ AckFor(actNR) ∧ Perf(actCurr) = perf)
 4        then return true
 5    case end :
 6      return not found
 7    case default :
 8      if (actCurr = actNR)
 9        then found ← true
10  next ← NextActivities(actCurr) \ visited
11  if next = ∅ then return true
12  switch TypeOf(NextElement(actCurr))
13    case gway-excl :
14      return ⋀_{a∈next} VerifyNR(actNR, perf, a, found, visited ∪ actCurr)
15    case gway-incl :
16      return ⋁_{a∈next} VerifyNR(actNR, perf, a, found, visited ∪ actCurr)
17    case activity :
18      VerifyNR(actNR, perf, GetFirst(next), found, visited ∪ actCurr)
```

Each instantiated non-repudiation commitment—which refers to a specific activity *actNR*—is verified on *BP* by the recursive function VerifyNR (Algorithm 5). The commitment is satisfied when, for each path from start to end in *BP* that includes that activity, there is an *acknowledge* for *actNR* made by the executor of *actNR*. The performer of the *acknowledge* activity certifies the delegator that the delegation has taken place. A possible implementation is notifying a workflow engine about the delegation acceptance.

The parameters of VerifyNR are the activity for which an ack is needed (*actNR*); the debtor in the non-repudiation commitment (*perf*); the currently

examined activity in *BP* (*actCurr*); a boolean variable indicating if, in the path the algorithm is exploring, *actNR* was encountered (*found*); and the set of activities the algorithm has already encountered (*visited*). This last variable enables dealing with cycles in *BP*. Given a commitment for the non-repudiation of *act*, in which the debtor is *deb*, the algorithm is initially invoked as VerifyNR(*act*, *deb*, `start`, `false`, {`start`}).

If the current activity is an *ack* for the searched activity executed by *perf*, it returns true (lines 2-4). If it is the end activity, the algorithm returns false if the activity was found but the ack was not found, while it returns true if the activity was not found (lines 5-6). If the current activity is the searched one, then the variable *found* is set to true: an ack is required (lines 7-9). If there are no subsequent activities, this means the algorithm has reached the end of a cycle; since cycles are supported only via exclusive gateways, the algorithm returns true, as such value does not affect the computation of compliance (lines 10-11). The recursive calls of the algorithm depend on the type of the next encountered element (line 12). If an exclusive gate is found, the algorithm is recursively called for all subsequent activities, and the compliance results are conjuncted, as compliance is needed in all paths (lines 13-14). If a parallel gateway is encountered, all outgoing paths are followed, and the results are disjuncted, as one ack suffices (lines 15-16). In case of an activity, the algorithm examines it (lines 17-18).

Example 4 (Checking non-repudiation). Suppose the architecture of the composite service evolves. We check the evolved *BP* in Fig. 4 with the non-repudiation commitment C_1:C(eGov application, Seller, D=delegate(Seller, eGov application, Government notified), non-rep(D)). The instantiation process—which includes Algorithm 4—returns the following commitments:

$C_{1.1}$:C(eGov, Athens REA, non-rep(Send copy to ministry))
$C_{1.2}$:C(eGov, Athens REA, non-rep(Scan contract))
$C_{1.3}$:C(eGov, Athens REA, non-rep(Archive contract copy))
$C_{1.4}$:C(Storage, Athens REA, non-rep(Send copy to ministry))
$C_{1.5}$:C(Storage, Athens REA, non-rep(Scan contract))
$C_{1.6}$:C(Storage, Athens REA, non-rep(Archive contract copy))

By running Algorithm 5 on all the commitments, it returns that $C_{1.3}$, $C_{1.4}$, and $C_{1.5}$ are compliant: the debtor is not the performer of the activity, thus no acknowledge is required. $C_{1.1}$, $C_{1.2}$, and $C_{1.6}$ are not compliant: the activity specified in the commitment is executed but there is no corresponding acknowledge. To align the *BP* with $C_{1.1}$, $C_{1.2}$ and $C_{1.6}$, either *BP* is modified adding the needed acknowledge activities, or security requirement C_1 is removed. A concrete solution is to add an acknowledge activity *Send copy to ministry* between *Add duty stamp* and *Send copy to ministry*.

6 Discussion

We have proposed a methodological and semi-automated approach to align service-oriented architectures—specifically, service compositions—with security

requirements specifications—in particular, social commitments in STS-ml. While our approach addresses the entire R/A alignment problem, this paper focuses on checking alignment. Such activity is key in the era of software evolution, where both requirements and software systems are subject to unpredicted changes.

Our approach includes three steps: (i) an analyst creates a conceptual mapping between requirements and architecture—e.g., the activities that relate to a certain goal; (ii) algorithms are executed to check R/A alignment; and (iii) in case of non-alignment, evolutionary actions are taken by the analysts—or the system itself—to revise either the architecture or the requirements.

We have provided algorithms to check R/A alignment for two security requirements types: non-disclosure of information and non-repudiation of delegations. We have already investigated other types (need-to-know, fall-back and true redundancy) which were not shown due to space limitations.

Our approach complements methods to derive architectures from requirements (e.g. [2, 10, 16, 23]), which can be applied at design-time to define a suitable architecture for a given set of requirements. Our approach provides a continuous on-line alignment checking, which enables coping with evolution of both requirements and architecture.

Compliance is a hot topic in information security [12]. The effects of non-compliance are well known and existing empirical studies have investigated [21] how compliance is perceived by employees in organizations.

However, only recently the compliance between requirements and business processes has lead to concrete research efforts. Liu et al [13] describe how to check the compliance between a set of formally expressed regulatory requirements and business processes. They created a tool which transforms (i) business process models, expressed with Business process Execution language (BPEL), in pi-calculus; (ii) regulatory requirements, expressed with Business Property Specification Language (BPSL), in linear temporal logic. This tool verifies the business process against these compliance rules by means of model-checking technology. Our approach takes a different yet orthogonal standpoint as it considers security requirements over interactions.

Ghose and Koliadis [8] enrich BPMN with annotations, then transform models created using such language into Semantic Process Network (SPNets). This allows for defining a class of proximity relations that highlight the extent to which evolutions of an original business process deviate. Unlike us, they focus only on the structural difference between processes, and they don't take into account security requirements.

Other approaches (e.g., [7]) tackle the evolution problem from a legal perspective. They propose a systematic approach to help organizations align their business processes with (privacy) laws. Unlike ours, however, their approach is off-line and mainly design-time.

Some algorithms are implemented in the Security Requirements Compliance Module (SRCM) tool. It currently supports non-repudiation and several authorization-related requirements: non-modification, non-usage, non-production and non-disclosure. Future versions of the module will support other require-

ments such as redundancy, separation of duties, binding of duties, integrity, etc. The tool takes as input three XML files: a specific version of BP, the SRS, and the CM. SRCM returns another XML file which contains the SRS commitments grouped in three sets: satisfied, violated, or undecidable. SRCM is implemented as an OSGi bundle[2], so as to facilitate integration with other tools.

Our future work includes extending the framework in many ways. First, we will devise algorithms to support different types of security requirements. Second, we will provide a complete implementation of the tool to support the analyst in the mapping phase as well as to check R/A alignment in a continuous online fashion. Third, we will investigate how service-oriented architectures can self-evolve to guarantee R/A alignment. Fourth, we will empirically evaluate the effectiveness of our approach on industrial case studies in the Air Traffic Management domain (from Aniketos). Fifth, we will extend our approach to include verifying alignment between architecture and implementation. We plan to use techniques such as bytecode verification to determine whether the security properties in the business process are correctly implemented.

Acknowledgement. The research leading to these results has received funding from the European Union Seventh Framework Programme (FP7/2007-2013) under grant no 257930 (Aniketos) and 256980 (NESSoS).

References

1. Barais, O., Le Meur, A.F., Duchien, L., Lawall, J.: Software Architecture Evolution. In: Mens, T., Demeyer, S. (eds.) Software Evolution. LNCS, pp. 233–262. Springer, Heidelberg (2008)
2. Bastos, L.R.D., Castro, J.F.B.: Systematic Integration Between Requirements and Architecture. In: Choren, R., Garcia, A., Lucena, C., Romanovsky, A. (eds.) SELMAS 2004. LNCS, vol. 3390, pp. 85–103. Springer, Heidelberg (2005)
3. Casati, F., Ilnicki, S., Jin, L., Krishnamoorthy, V., Shan, M.-C.: Adaptive and Dynamic Service Composition in eFlow. In: Wangler, B., Bergman, L.D. (eds.) CAiSE 2000. LNCS, vol. 1789, pp. 13–31. Springer, Heidelberg (2000)
4. Crook, R., Ince, D., Lin, L., Nuseibeh, B.: Security Requirements Engineering: When Anti-Requirements Hit the Fan. In: Proc. of RE 2002, pp. 203–205. IEEE (2002)
5. Dalpiaz, F., Paja, E., Giorgini, P.: Security Requirements Engineering via Commitments. In: Proc. of STAST 2011 (2011)
6. Garg, A., Curtis, J., Halper, H.: Quantifying the Financial Impact of IT Security Breaches. Information Management & Computer Security 11(2), 74–83 (2003)
7. Ghanavati, S., Amyot, D., Peyton, L.: Compliance Analysis Based on a Goal-oriented Requirement Language Evaluation Methodology. In: Proc. of RE 2009, pp. 133–142 (2009)
8. Ghose, A., Koliadis, G.: Auditing Business Process Compliance. In: Krämer, B.J., Lin, K.-J., Narasimhan, P. (eds.) ICSOC 2007. LNCS, vol. 4749, pp. 169–180. Springer, Heidelberg (2007)

[2] `http://www.osgi.org/`

9. Giorgini, P., Massacci, F., Mylopoulos, J., Zannone, N.: Modeling Security Requirements through Ownership, Permission and Delegation. In: Proc. of RE 2005, pp. 167–176. IEEE (2005)
10. Hall, J.G., Jackson, M., Laney, R.C., Nuseibeh, B., Rapanotti, L.: Relating Software Requirements and Architectures using Problem Frames. In: Proc. of RE 2002, pp. 137–144. IEEE (2002)
11. Harker, S.D.P., Eason, K.D., Dobson, J.E.: The Change and Evolution of Requirements as a Challenge to the Practice of Software Engineering. In: Proc. of RE 1993, pp. 266–272. IEEE (1993)
12. Julisch, K.: Security Compliance: the Next Frontier in Security Research. In: Proc. of the 2008 Workshop on New Security Paradigms, pp. 71–74. ACM (2008)
13. Liu, Y., Müller, S., Xu, K.: A Static Compliance-Checking Framework for Business Process Models. IBM Systems Journal 46(2), 335–361 (2007)
14. McDermott, J., Fox, C.: Using Abuse Case Models for Security Requirements Analysis. In: Proc. of ACSAC 1999, pp. 55–64. IEEE (1999)
15. Mouratidis, H., Giorgini, P.: Secure Tropos: A Security-Oriented Extension of the Tropos methodology. International Journal of Software Engineering and Knowledge Engineering 17(2), 285–309 (2007)
16. Nuseibeh, B.: Weaving together requirements and architectures. Computer 34(3), 115–119 (2001)
17. Nuseibeh, B., Easterbrook, S.: Requirements Engineering: a Roadmap. In: Proc. of FOSE 2000, pp. 35–46. ACM (2000)
18. Rodríguez, A., Fernández-Medina, E., Piattini, M.: A BPMN Extension for the Modeling of Security requirements in Business Processes. IEICE Transactions on Information and Systems 90(4), 745–752 (2007)
19. Sindre, G., Opdahl, A.L.: Eliciting Security Requirements with Misuse Cases. Requirements Engineering 10(1), 34–44 (2005)
20. Singh, M.P.: An Ontology for Commitments in Multiagent Systems: Toward a Unification of Normative Concepts. Artificial Intelligence and Law 7(1), 97–113 (1999)
21. Siponen, M., Pahnila, S., Adam Mahmood, M.: Compliance with Information Security Policies: An Empirical Investigation. Computer 43, 64–71 (2010)
22. van Lamsweerde, A.: Requirements Engineering in the Year 2000: A Research Perspective. In: Proc. of ICSE 2000, pp. 5–19 (2000)
23. van Lamsweerde, A.: From System Goals to Software Architecture. In: Bernardo, M., Inverardi, P. (eds.) SFM 2003. LNCS, vol. 2804, pp. 25–43. Springer, Heidelberg (2003)
24. van Lamsweerde, A.: Elaborating Security Requirements by Construction of Intentional Anti-Models. In: Proc. of ICSE 2004, pp. 148–157. IEEE (2004)