Automated VAlidatioN of Trust and Security of Service-oriented ARchitectures

FP7-ICT-2007-1, Project No. 216471

www.avantssar.eu

Deliverable D5.4
Assessment of the AVANTSSAR Validation Platform

Abstract

We report on the assessment of the application of the AVANTSSAR Validation Platform to the problem cases in terms of coverage and efficiency. The results of the assessment demonstrate the achievement of all of the project objectives. We have been able to formalize (in ASLan++, HLPSL++, annotated BPMN, and ASLan) 94 problem cases from 10 application scenarios, and the platform successfully analyses 78 problem cases. All of the success criteria set out in the Description of Work are thus fulfilled by the platform, exceeding by several times the required number of formalized and validated problem cases.

Deliverable details

Deliverable version: *v1.1*
Classification: *public*
Date of delivery: *22.03.11 (v1.0: 05.02.2011)*
Due on: *31.12.2010*
Editors: *all*
Total pages: *57*

Project details

Start date: *January 01, 2008*
Duration: *36 months*
Project Coordinator: *Luca Viganò*
Partners: UNIVR, ETH Zurich, INRIA, UPS-IRIT, UGDIST, IBM, OpenTrust, IEAT, SAP, SIEMENS

Contents

List of Tables

1 Introduction

In this deliverable, we assess the technical achievements of the project by testing the AVANTSSAR Validation Platform (AVANTSSAR Platform, for short [9]) against the AVANTSSAR Library of problem cases [11].

Given the three *application areas*

- e-business,
- e-government, and
- e-health,

we consider 10 *application scenarios*:

- Loan Origination,
- Anonymous Shopping,
- Visa Application,
- Car Registration,
- Public Bidding,
- Digital Contract Signing,
- Electronic Health Records,
- Process Task Delegation,
- Access Control Management,
- SAML Single Sign-On.

Notice that an application scenario may be characterized by different *scenes* expressing different aspects of the application scenario. Every application scenario addresses different *security aspects*, i.e. security relevant aspects such as Federation, Authorization Policies, etc. The formal description of (part of) an application scenario in order to capture a security aspect originates a *problem case*. Thus, every security aspect can be seen as a set containing all problem cases, from different scenarios, that refer to that particular security aspect. We call each such set a *family of problem cases*.

In other words, an application scenario is composed of one or more scenes that focus on different use cases of the considered service, protocol or the like. Each scene contains at least one goal formalizing a security property or aspect that formalizes a problem case[1] and may differ from other scenes for the

[1] Obviously, in order to reason on a problem case, it is not sufficient to consider a goal. If one wants to validate a system against a problem case, one must consider also the system components. For what concerns ASLan, this means that one should take into account all sections (signature, types, inits...) and the specific goal. For this reason, it is common to have different files that differ only for the goal section.

considered security aspect (problem case) and/or for the use case formalized. A concrete example is represented by the Anonymous Shopping scenario (all components of this scenario mentioned in the following paragraphs are described in § 2.1.2):

- The Anonymous Shopping scenario is composed of 3 scenes that focus on the formalization of a protocol that aims to guarantee correct authentication while preserving the privacy of users.
- The scenes formalize different use cases; for example, in one scene a customer purchases goods directly from a winery using her credential and in another she first obtains credentials from an intermediary entity called "FCS". Another difference between the different Anonymous Shopping scenes (as one can see from Table 2) is the set of problem cases considered.
- The problem cases considered in the Anonymous Shopping scenario are the ones indicated in Table 2 and are mainly characterized by the set of goals expressing the corresponding security aspect or property.

The library of problem cases comprises a total of 94 problems cases from 10 application scenarios, specified in one of the application-level languages ASLan++, HLPSL++, annotated BPMN, or in the more low-level specification language ASLan. Every specification may represent multiple problem cases if it addresses different security aspects. The 94 problem cases considered are expressed by 26 specifications where:

- 15 are ASLan++ specifications,
- 5 are HLPSL++ specifications,
- 1 is a BPMN specification with annotations,
- 1 is a textual specification, and
- 4 are ASLan specifications.

Among the 26 specifications, 4 involve orchestration and formalize 13 problem cases that have to be orchestrated prior to validation.

In the Description of Work, we planned to assess the coverage and efficiency of our modelling languages and tools, and, in particular, we foresaw the assessment to be carried out also by considering the degree of simplification brought by the techniques developed in WP 3. However, these techniques have been central to the success of the project, inasmuch as the analysis cannot be performed without employing them. Thus, while qualitatively these

techniques enable validating models that could not have been analyzed otherwise, this cannot be quantified in a numerical performance increase. We therefore refine the evaluation criteria as follows:

Validation Coverage: we present the coverage of security aspects by means of the formally specified and validated problem cases from the application scenarios. For each formalized scenario and problem case, we state the limitations experienced during modeling and verification, and the assumptions made in order to enable automated validation.

Efficiency: we present CPU time and memory required by the tools to analyze the specification files on standard commercially available computers. The actual machine used to run the tools is an Intel(R) Xeon(R) CPU E5430 at 2.66GHz with 2 GB of RAM, and ongoing validations have been interrupted after an hour of computation. This timeout has been set in order to collect results in a reasonable time and it was also used to further try out the efficiency of the tools against complex specifications.

According to the commitment in the Description of Work, the project is to be considered successful if the following targeted success criteria are met:

- At least 15 problem cases covering all application scenarios are described.
- At least 12 problem cases are formalized in ASLan.
- At least 10 problem cases are formally validated.
- At least 10 problem cases are validated in an automated manner.

The results of the assessment of the AVANTSSAR Platform are given in Table 1. The table shows the number of problem cases formalized and validated following two different criteria of selecting problem cases from the specification files used for the assessment of the AVANTSSAR Platform. The column focusing on the 26 scenes takes into account all problem cases formalized by all specifications (the same criterion has been used for Table 2). But different scenes belonging to the same scenario may contain the formalization of quite similar problem cases, if not exactly the same ones. For this reason, we selected a refined subset of problem cases, by choosing only those problem cases that were still not considered for the scenario. As shown in the column “10 Application Scenarios", this selection leads to 57 problem cases described and formalized and 51 problem cases formally and automatically validated.

Table 1: Results of the AVANTSSAR Platform

OBJECTIVES	**RESULTS**	
	26 Scenes	***10 Applications Scenarios***
15 problem cases described 12 problem cases formalized	**94 problem cases described and formalized**	**57 problem cases described and formalized**
10 problem cases formally validated 10 problem cases automatically validated	**78 problem cases formally and automatically validated**	**51 problem cases formally and automatically validated**

In total, we have been able to formalize in ASLan++, HLPSL++, annotated BPMN, or ASLan 94 problems cases from 10 scenarios, and the tool successfully analyzes 78 problems in 584 minutes of CPU time.

All details about validation times are presented by Table 7 in § 3. The table considers all specification files that were tested during the platform assessment. This means that it contains also values obtained from specification files used to check the executability of all scenes belonging to different scenarios. In the worst case, the Platform has consumed 30 minutes on a single problem case. By summing up the total time of each validator from Table 7, we obtain 4244 minutes. If we remove all the timeouts (61) that occurred during the assessment, we obtain 584 minutes of CPU time used to successfully analyze the 78 problem cases.

All of the success criteria set out in the Description of Work (in terms of coverage and efficiency) are thus fulfilled by the platform, exceeding by several times the required number of formalized and validated problem cases.

2 Validation Coverage

Table 2 shows the problem cases formalized and validated by the AVANTSSAR Platform. It contains, for each application scenario, information about the connector used to translate high level specifications into ASLan and, if it is the case, about the orchestrator. For what concerns the families of problem cases, "f" indicates that a formalization of the problem case is present in the specification but was not validated, whereas "v" indicates its validation. Therefore, Table 2 provides an overview of the formalization and validation status of the library of problem cases. This information is then summarized in Table 3, which focuses exclusively on the number of problem cases formalized and validated.

Table 2: Formalization and validation status

Areas	Scenarios	Scene	Specification	Connector	Orchestration	Families of Problem Cases						
						Federation	Authorization Policies	Accountability	Trust Management	Workflow Security	Privacy	Application Data Protection	Communication Security
E-Business													
Banking Services	Loan Origination	1	lop-scene1.aslan	No	No					v			
		2	lop-scene2.aslan	NWBPM	No		v			v	v		
Electronic Commerce	Anonymous Shopping	1	IDMXScene1_Safe.aslan++	ASLan++	No			v			v		
		2	IDMXScene2_Safe.aslan++	ASLan++	No	v					v		
		3	IDMXScene3_Safe.aslan++	ASLan++	No		f				f		
E-Government													
Citizen and Service Portals	Visa Application	1	PTD_VisaBank.aslan++	ASLan++	No		v	v	v	v	v	v	v
	Car Registration	1	CRP.dyn.aslan++	ASLan++	Yes		v	v	v	v	v	v	v
Document Exchange Procedures	Public Bidding	1	pb_scene1.aslan++	ASLan++	No		f	f				f	f
		2	pb_scene2.aslan++	ASLan++	No		v	v				v	v
		3	pb-elig.aslan++	ASLan++	No		v						
		4	PB_alt.aslan	No	Yes		v	v				v	v
	Digital Contract Signing	1	dcs-scene1.aslan++	ASLan++	No		f	f		f		f	f
		2	dcs-scene2.aslan++	ASLan++	No		v	v		v		v	v
		3	dcs-scene3.aslan++	ASLan++	No		v	v		v		v	v
		4	DCS.ORCH.aslan	No	Yes		f						
		5	DCS-GoalStyleInput.ORCH.aslan	No	Yes		f						
E-Health													
Personal Health Information	Electronic Health Records	1	ECR.aslan++	ASLan++	No			v	v	v	v	v	v
	Process Task Delegation	1	PTD.aslan++	ASLan++	No	v	v			v	v	v	v
		2	PTD_PC.aslan++	ASLan++	No	v	v	v		v	v	v	v
	Access Control Management	1	eHRMS.txt	No	No		f				f	f	
	SAML Single Sign-On	1	SP_init-FC-one_channel.hlpsl++	HLPSL++	No	v					v		v
		2	SP_init-BC-two_channels.hlpsl++	HLPSL++	No	v					v		v
		3	IdP_init-FC.hlpsl++	HLPSL++	No	v					v		v
		4	IdP_init-BC.hlpsl++	HLPSL++	No	v					v		v
		5	SAML-based_SSO_for_GoogleApp.hlpsl++	HLPSL++	No	v					v		v
		6	SAML-based_SSO_for_GoogleApp.aslan++	ASLan++	No	v					v		v

Table 3: Problem cases summary

Scenario	Scene	# Problem cases	
		formalized	validated
Loan Origination	1	1	1
	2	3	3
Anonymous Shopping	1	2	2
	2	2	2
	3	2	0
Visa Application	1	7	7
Car Registration	1	7	7
Public Bidding	1	4	0
	2	4	4
	3	1	1
	4	4	4
Digital Contract Signing	1	5	0
	2	5	5
	3	5	5
	4	1	0
	5	1	0
Electronic Health Records	1	6	6
Process Task Delegation	1	6	6
	2	7	7
Access Control Management	1	3	0
SAML Single Sign-On	1	3	3
	2	3	3
	3	3	3
	4	3	3
	5	3	3
	6	3	3
Total		**94**	**78**

2.1 Application Scenarios

2.1.1 Loan Origination

Security requirements coverage The Loan Origination scenario focuses on the access control policy and how it affects the workflow execution. We

consider two different scenes that focus on different aspects of the scenario. In particular, *scene 1* considers human tasks with non-deterministic effects, a conditional permission assignment, and delegation rules for the delegation of certain tasks, whereas *scene 2* takes into account data and distinguishes between human and automated tasks, and thus introduces the concept of principal propagation, which is a form of access control over automated tasks. More in detail, scene 1 involves six human tasks with a conditional permission assignment specified via multiple Horn clauses. Scene 2 is characterized by nine human tasks and two automated tasks, and a static permission assignment expressed by using nullary Horn clauses, i.e. Horn clauses with an empty body. The behavior of both scenes is made even more dynamic by rewrite rules enabling delegation mechanisms. In particular, delegation considerably increases the search space and makes it harder for the tools to support the specifications.

While scene 1 is specified in ASLan, scene 2 is translated automatically to ASLan using the SAP BPM Connector. As shown in Table 2, the two scenes of the Loan Origination scenario cover three families of problem cases (i.e. privacy, workflow security, and authorization policies). More in detail, scene 1 covers the workflow security family of problem cases by means of an object-based separation of duty property. Scene 2 covers

- the privacy family of problem cases by means of a need-to-know property,
- the workflow security family of problem cases by means of separation and binding of duty properties, and
- the authorization policies family of problem cases by means of properties for data confidentiality and access control over automated tasks.

As a result, the two scenes of the Loan Origination scenario originate four different problem cases.

Modeling limitations Though the access control policy specified are representative of real world business processes, ASLan does not provide some advanced features for the specification of complex access policies. In fact, as pointed out in Deliverable D5.3 [11], the use of Horn clauses provides remarkable algebraic properties but does not support the specification of some authorization policies that require the use of negative facts or conditions. As an example, consider a business process where tasks can be executed by every agent except those who belong to a specific group, i.e. the excluded owners. In the specification, it makes sense to specify the excluded owners

rather than the entitled owners in case the number of excluded owners is considerably lower than the number of entitled owners. In this scenario, Horn clauses do not allow the specification of the following access policy "the agent executing a task must not be an excluded owner for that task" as it would require a negated fact in the body of the Horn clause. This limitation has to be considered when the specification of the LOP is extended to support a dynamic permission assignment between roles and tasks, e.g. the permission to execute a task is given to two different roles according to the truth or false value of a fact. As a first attempt to address these kind of requirements CL-AtSe has been enhanced to support the specification of conditions inside Horn clauses.

Other extensions of the specification could have been considered to encompass, e.g., communication aspects. These would allow to consider other security and trust requirements, however they were not the focus of the Loan Origination scenario in the context of the AVANTSSAR project as they are already tackled in other case studies.

Results This scenario is supported by the back-ends SATMC and CL-AtSe as they support the types of Horn clauses used within the case study, i.e. Horn clauses without body and universally quantified Horn clauses. We have run the tools against all the properties characterizing the problem cases tackled within the two scenes (more details can be found in Deliverable D5.3 [11]). SATMC returns `ATTACK_FOUND` for all the properties of the four problem cases defined. CL-AtSe returns `ATTACK_FOUND` for the problem case covering privacy in scene 1 and for the "access control over automated task" property considered in the authorization policies family of problem case within scene 2. For all the other properties, and thus the "data confidentiality" property and the other two problem cases of scene 2, i.e. workflow security and privacy, CL-AtSe times out. In fact, the great flexibility provided by the delegation mechanism causes the state space to explode. OFMC currently only supports the analysis of a fragment of the specification as it does not fully support Horn clauses, and in particular Horn clauses without body, which are widely used to specify the access control model (one of the main features of this scenario).

2.1.2 Anonymous Shopping

Security requirement coverage The Anonymous Shopping scenario focuses on the problem of managing personal data in order to both protect user privacy and allow online transactions with organizations that request information about users. While on the one hand privacy is very important,

on the other hand many transactions require authentication, authorization, and accountability. There is seemingly a partial conflict between the goals of properly identifying users while protecting their privacy. The three scenes of the Anonymous Shopping scenario address different interesting situations that may occur in the described context: "Anonymous yet accountable shopping" where a customer can purchase some goods revealing to the supplier a minimal amount of information about (her)himself; "A frequent customer system" (FCS) where the supplier is a member of a consortium so the customer needs only to obtain a credential from the `FCS` entity representing the association of suppliers, and then (s)he can purchase from all suppliers using that single credential; and the "Anonymous Bonus System" where the customer receives a discount after a certain amount of purchases from a single supplier. For what concerns the problem cases coverage, the Anonymous Shopping scenario consists of three scenes that, as shown in Table 2, cover four families of problem cases (i.e. Federation, Authorization Policies, Accountability and Privacy). Scene 1 is the only one that covers accountability because modeling it also in other scenes would have caused a substantial growth of their complexity. Indeed, in order to formalize accountability it would be necessary to add in scene 2 and 3 the `Judge` entity and the `WineryComplain` entity, and this would obviously lead to a larger number of transitions in the ASLan files. On top of that, the purpose of scene 2 and 3 are respectively to cover federation, by introducing the `FCS` entity, and authorization policies, by prevention and detection of illegal use of one-show credentials. Coverage of privacy is assured throughout all scenes of the Anonymous Shopping scenario because of the usage of the Identity Mixer, an anonymous credential system developed by IBM Research – Zürich, as described in [6, 10, 11].

Modeling limitations A major goal of Identity Mixer is that one cannot observe more information about users than they have deliberately released, including that the different transactions of a user cannot be linked. In other words, when a user proves the possession of a credential to a server, then the server does not learn more about the credential than those facts that the user deliberately chooses to prove. More generally speaking, the user maintains the control over the information it reveals and to whom. Further, dishonest servers cannot tell whether particular actions have been performed by the same users or by different ones. This holds even when several dishonest servers collaborate. For instance, consider two dishonest agents S_1 and S_2 that collaborate and assume S_1 has issued a credential for some user U (acting under some pseudonym) with a property P (e.g. that the owner

is over 18). U (acting under a different pseudonym) proves to S_2 that it possesses a credential from S_1 with property P. We require that, even combining the knowledge of S_1 and S_2, each of the certificates with property P that S_1 has issued is equally likely to be the one U has shown to S_2. These privacy goals are quite difficult to formalize since they are not properties of single execution traces but rather based on the question whether the intruder can observe a difference between certain pairs of traces [8, 5, 14]. In automated protocol verification, there are only few approaches that address privacy properties [20]. For the moment, we have therefore focused only on the formal analysis of secrecy properties as part of the more general privacy ones, and give here only an informal account of the full-fledged privacy properties. All an intruder is able to learn from the zero-knowledge proof

$$\mathsf{spk}(\mathcal{S}; P; \phi; Stmt)$$

are the public values P, the signed statement $Stmt$, and the fact that the proved property ϕ holds on $\mathcal{S}$ and P (for further details about zero-knowledge proof and other Identity Mixer components see [10, 11]). The idea is thus that the intruder cannot distinguish two such terms that differ only in the $\mathcal{S}$ part and this would reflect that the intruder can only observe those properties that are deliberately released by the respective participant. Note that throughout our formalization, for simplicity, we have used deterministic functions, i.e., when a user performs several times the same zero-knowledge proof or verifiable encryption with exactly the same arguments, the result will exactly be the same — and this can indeed be observed. It is not difficult to include a fresh random value into every relevant function as an additional argument, and this correctly models the non-deterministic behavior of the real operations. Given non-deterministic functions, one can indeed formally define what it means that two terms are unequal but not distinguishable for the intruder, and, based on that, prove the privacy. As stated, this is beyond the scope of current automated verification technology.

In order to test Anonymous Shopping specifications against the families of problem cases listed in Table 2, we have made some abstractions from the original description of scenes. For example, as already stated, Scene 1 is the only one which covers accountability because modeling it also in other scenes would have caused a substantial growth of their complexity (i.e. more transitions, so that the backends would not be able to terminate within a reasonable time). Another simplification was made in Scene 3, where we have modeled the Wineshopper entity, formalizing the customer's actions, in a sequential way, namely all purchases are done one after another without changing the Identity Mixer Pseudonym used to show the necessary proofs

and credentials to the Winery entity, formalizing the supplier (see [10,11] for further details on Identity Mixer components).

Results The complete ASLan formalization of the Anonymous Shopping case study, obtained by the translation from ASLan++, is composed of seven files:

- `IDMXBase.aslan++`
- `IDMXScene1_{execTrace, Safe}.aslan++`
- `IDMXScene2_{execTrace, Safe}.aslan++`
- `IDMXScene3_{execTrace, Safe}.aslan++`

Each `execTrace` file is identical to its `Safe` counterpart, except from the `goal` section. Indeed, the aim of the former is to test the executability of all transition steps (and not the problem cases), while the purpose of the latter is to test the specification against the security requirements tailored for that scene. For what concerns `execTrace` files, SATMC times out on all scenes, CL-AtSe times out on scene 3, and OFMC times out on both scene 2 and 3. All other executability checks (i.e. proof that the model is indeed fully executable) have been found during the assessment tests of the AVANTSSAR Validation Platform that have been run with a timeout of 1 hour. The three `Safe` files of the Anonymous Shopping case study were also checked with all backends but the outcome of the assessment confirmed the fact that specifications are quite complex: SATMC timed out an all files, CL-AtSe returned `INCONCLUSIVE` on scene 1 and 2 and timed out on scene 3, OFMC returned `NO_ATTACK_FOUND` on scene 1 and timed out on the other two scenes. CL-AtSe returned `INCONCLUSIVE` because it does not support the Horn clauses we have defined in the `IDMXBase.aslan++` to model partial invertibility of some function symbols (i.e. iknows fact on the head) and thus the tool abstracted them away causing the `INCONCLUSIVE` result even though no attack was found. In order to obtain more results, we have run further tests with OFMC, on a machine Intel(R) Core(TM)2 Duo E7300 CPU at 2.66 GHz with 3 GB of RAM, without setting any timeout. During those tests, we were indeed able to obtain some new results: OFMC has taken 76 minutes and 4 hours respectively on `IDMXScene2_execTrace.aslan` and `IDMXScene2_Safe.aslan` file returning the expected attack, for the `execTrace` file, and `NO_ATTACK_FOUND` for the `Safe` one. For what concerns scene 3, OFMC found the executability attack in about 6 hours by limiting the search space by setting the option `--depth` to 18 on the `execTrace` file, but we have killed the OFMC process

after 24 hours, with the same search bound option, on the `Safe` file. This is the reason why we put "f" in Table 2 for the problem cases of scene 3.

2.1.3 Visa Application

Different countries are now providing online facilities for visa application. These procedures require that the citizen presents certain particular documents, depending on the type of visa being requested. Documents required may include evidence provided by the current employment, bank statements, or the school or university of the applicant. Thus, two or more different workflows, one at the Visa Application and others at Third Parties (e.g., a bank, university, ...) must be coupled together.

In this scenario, we employ Process Task Delegation (PTD) for workflow coupling. Variants and specific applications of the generic PTD protocol are used for the transmission of e-Health Records, the ordering of treatments, Single Sign-On, etc., and the Visa Application, as referred to now.

The specification for the Visa Application is formalized in ASLan++.

Security requirements coverage The purpose of PTD in the Visa Application is to eliminate the existing gaps in distributed electronic workflows and provide a high degree of security and trust. However, in comparison with the strong security aspects provided by PTD in the e-Health scenario (see § 2.1.8) here only a rather weak and repudiable version of user consent is needed. No document that the user has not explicitly consented to is sent to the Visa Portal. The service providers must be unable to repudiate that they issued documents they did.

A reliable user-re-identification (i.e., that it is the *same* user) instead of full SSO capabilities is required. The flow here is linking the two different workflows, each one based on an independent authentication of the user. Also the authorizations of the user in the different domains are independent of each other. Only the basic trust needs to be mediated by the trust servers. In the background, there is a key transport protocol associated to the communication sent via the user.

Only the user is allowed to retrieve his documents: no entity is allowed to retrieve documents belonging to another user. The Visa Portal only may accept documents that are written by trusted and authorized service providers.

Thus, the families of problem cases covered are authorization policies, accountability, trust management, workflow security, privacy, application data protection, and communication security.

Modeling limitations The specification is written in ASLan++ and translated automatically to ASLan using the ASLan++ Connector. The model contains 7 entities (including `Environment` and `Session`). It uses neither loops nor sets nor clauses, and it contains only one `if`-statement. The model can be found in `PTD_VisaBank.aslan++`.

We have modeled the Visa-Application scenario only with one institution (Bank). Sessions can be instantiated in two flavors, where in the first case the *applicant chooses the bank*, and in the second case the *visa portal selects the bank*. Simultaneous sessions may comprise different flavors.

We did not yet model all three institutions (Bank, School, Airline) since by the end of the project the efficiency of the validation platform was not sufficient for more complex scenarios.

Due to the resulting similarity of this simplified model to the generic PTD model, the limitations described in § 2.1.8 are applicable to this reduced scenario as well. In particular we were able to implement non-repudiation via digital signatures to provide accountability, though we would like to have more abstract formal means to state goals of non-repudiation explicitly.

Results Translation and executability tests of the Visa Application model have been performed successfully. All three backends are capable of finding expected attacks for the executability checks.

We used a Celsius W380 with an Intel Core i5-650 Processor (2 Cores / 4 Threads), 3.20 GHz each Core and 3 GB RAM under Windows XP. The translator was the ASLan++ Connector (Version 0.6.0, `-gas`).

Cl-AtSe (Version 2.5-7e, `--not_hc`) took about three seconds, testing 3755 transitions and reaching 782 states. OFMC (Version 2010d, with the options `--classic --untyped`) took about a second with the `--exec`-option to verify executability of 19 Rules, and about the same time to find the attack when analyzing the model, visiting 17 nodes. SATMC in its online version (Version: 3.2.7) took also about a second and 18 steps to analyze 2935 atoms and 7338 clauses.

Cl-AtSe and OFMC were able to analyze this model with one session in about the same time, finding no attack. So we could verify all security properties, i.e. all channel and security goals for that one session. SATMC in its online version in a few seconds exhausted its 50 steps and returned an inconclusive result.

Model checking with more than one session even for this much simplified scenario still proved to be complex. Only Cl-AtSe in a limited way was able to cope with it.

Due to the resulting similarity of this simplified model to the generic PTD

model, the results described in § 2.1.8 are applicable to this reduced scenario as well, as are the limitations of the generic PTD model also described in that section.

We have modeled the Visa-Application scenario only with one institution (Bank) and not with all three institutions (Bank, School, Airline). Therefore possible security aspects required for the full scenario have not yet been considered. We planned to do this when the efficiency of the tools has been sufficiently improved, which by the end of the project was not the case. The full scenario therefore may still contain security flaws, despite the model checkers finding no attack on the generic PTD (and therefore also this) scenario.

More details on the Visa Application model can be found in [11].

2.1.4 Car Registration

Security requirements coverage The Car Registration scenario combines access control policies with workflow execution. The specification is written in ASLan++ and translated automatically to ASLan using the ASLan++ Connector. A single scene is modeled and is described in detail in [11]. As shown in Table 2, the modeled scene covers all families of problem cases except Federation. Authorization Policies, Accountability, Trust Management, Workflow Security, Privacy, Application Data Protection and Communication Security are formalized and validated.

Modeling limitations There are six ASLan++ entities in the specification, and five of them use loops. Three of them loop infinitely due to the nature of the services they render (`CentralRepository`, `RegistrationOffice` and `Employee`), while two combine loops with processing of sets (`CA` and `Head`). Additionally, due to the access control policies, branches are used in many places. All these elements taken together result in a complex specification which, after translation to ASLan, proves difficult to be analyzed; the Horn clauses used for access control policies also raise certain problems for the validation platform.

In order to mitigate the encountered problems and keep under control the complexity, specific simplifications and compromises were done. As we will see in the following paragraphs, they do not affect the semantics of what was modeled and the validity of the obtained results. Rather they consist in expressing the same thing in such a way that it can be handled more easily by the validation platform. More details can be found in [11].

Restructuring of ASLan++ code. The ASLan++ code is written in such a way that the translator from ASLan++ to ASLan can optimize

the generated ASLan output and significantly reduce the number of rewrite rules. For example, nested `if`s, which express branching with more than two alternatives, were rewritten using `select`s.

Removal of Universally Quantified Horn Clauses. Universally quantified Horn clauses were introduced into the ASLan language and described in [7]. They are Horn clauses where variables occurring in the left-hand side are not a proper subset of the right-hand side variables. In practice the backends cannot cope with such Horn clauses, or can do it but the analysis takes too long. All universally quantified Horn clauses were rewritten to eliminate the use of free variables from the left-hand side.

Limiting Recursion in Horn Clauses. The Horn clauses that model trust relations and knowledge flow are inspired by DKAL. In their initial form they used recursion which led to infinite loops in some of the backends. This was avoided by rewriting the recursion in such a way that the application of the respective Horn clauses did not trigger the generation of infinite terms.

Removal of Non-critical Operations. One non-critical operation was left out of the model: the possibility that an employee of a registration office postpones a car registration request for later processing. The removed operation does not influence any of the security properties that were validated.

Besides these limitations raised by the model itself, there were also some limitations regarding the security properties that could be validated. These do have impact on the result of the verification, in the sense that even if the validation platform did not find any attack on the specification, not all desired security properties were covered.

Inability to Verify Certain LTL Formulas. At the moment when the verification was performed, some backends were unable to handle certain LTL formulas. An example are LTL formulas related to liveness, which use the <> (*future*) operator. Another example is the formula related to data consistency. More details can be found in [11].

Results The single scene that is modeled involves one citizen, one employee and one registration office. This scene is sent to the validation platform in various flavors. The access control policies are modeled both dynamically, through Horn clauses, and statically, by introducing appropriate policy facts at the very beginning of the execution. For testing various paths that can be

Table 4: Variations of the Car Registration Protocol model that are sent to the backends for analysis

Description	**Dynamic policies**	**Static policies**
Executability check for the acceptance of a valid car registration request	CRP.dyn.acc.aslan++	CRP.stat.acc.aslan++
Executability check for the rejection of an invalid car registration request	CRP.dyn.rej.aslan++	CRP.stat.rej.aslan++
Executability check for the meta level Horn clauses	CRP.dyn.meta.aslan++	-
Validation of security properties	CRP.dyn.aslan++	CRP.stat.aslan++

taken in the workflow, several executability checks are performed. Table 4 gives an overview of all variations of the model that are sent to the backends.

All three backends are capable of finding expected attacks for the executability checks. Table 5 lists the execution times for each backend and for each executability check. The hardware specifications of the machines that were used for running the tests are those mentioned in the Efficiency section (§ 3). CL-AtSe has the best performance, followed by SATMC and then by OFMC. For OFMC we used the `--depth` option for limiting the search space, and even with this option the time needed to find the attacks is several orders of magnitude higher compared to the other two backends (for three of the versions with dynamically modeled access control policies the exact running time is not known, because OFMC was stopped after one hour). The reason is that OFMC has some limitations related to subtyping and composed types, and in order to receive accurate analysis results it has to be run in untyped mode. More details about this can be found in [9].

When it comes to verifying security properties, each backend behaves differently:

CL-AtSe is capable to fully analyze the specification and give a conclusive verdict, with the conclusion that no attack is found. For the model with static control policies the answer is given in a matter of seconds,

Table 5: Analysis times (in minutes) for finding executability traces for the Car Registration scenario

Model	CL-AtSe	OFMC	SATMC
CRP.dyn.acc.aslan++	00:02.483603	-	00:06.605589
CRP.stat.acc.aslan++	00:00.694782	-	00:03.899333
CRP.dyn.rej.aslan++	00:02.420148	-	00:04.969597
CRP.stat.rej.aslan++	00:00.651390	14:16.376475	00:02.632181
CRP.dyn.meta.aslan++	00:02.242193	02:06.479810	00:03.539045

while for the one with dynamic control policies it takes several hours.

OFMC does not terminate in reasonable time. With the search depth limited through the `--depth` option, OFMC does not find any attack in one hour. Again the long running time of OFMC is due to its limitations related to subtyping and to composed types.

SATMC has only a semi-decision procedure for this specification, in the sense that if an attack is found then SATMC terminates, otherwise it returns an inconclusive answer, which means that no attack was found until a certain search depth was reached. For the default search depth of 80 steps, SATMC finds no attack, and the answer is given in a matter of minutes.

Orchestration In order to perform orchestration on the Car Registration specification, the *Citizen* entity is transformed into an *Orchestration Client* so that the orchestrator synthesizes an *Orchestration Goal* that coordinates the rest of the entities towards a successful processing of a car registration request. The transformation is performed automatically by the ASLan++ connector, starting from the same ASLan++ model that is used for verification.

The orchestration is performed successfully and the result can be sent to the validation platform for verifying the security properties. CL-AtSe is able to verify the orchestrated model that uses static access policies, while on the one that uses dynamic access control policies it does not terminate in the one hour limit imposed during the assessment tests. OFMC and SATMC do not terminate in the one hour limit on both versions of the model, with static and with dynamic access control policies. The reason for the too long running time of the backends is that the orchestrated model is much more

complex than the original model, with many extra transitions added for the *Orchestration Goal*.

2.1.5 Public Bidding

Security requirements coverage The Public Bidding scenario [6] describes a call for tender process, managed by a Bidding Portal. A company uses such a portal to publish a call for tender; then several Bidders can view it, and submit an offer. These offers are then evaluated, and a winner is chosen. The Portal has access to backends to manage the different security aspects of the process, such as signature, encryption and storage of the different documents. As shown in Table 2, this scenario covers the authorization policies, the accountability, the application data protection and the communication security families of problem case. Four scenes have been modeled from this scenario. The first one is the complete formalization of the case-study in ASlan++, with all the related security properties. The second scene provides an optimized ASLan++ model to allow formal verification (as explained in the next paragraph). Scene 3 offers a model of the scenario where several elements, such as the channels and the messages format, are formalized by using a different approach. In particular, this scene focuses on authorization policies. The last scene gives a model suitable for orchestration and then validation of the four families of problem cases mentioned above.

Modeling limitations

Some abstractions made on the specification. Since the different backends do not support some features, and the Public Bidding is a long process, some optimizations and simplifications were made in the specification in order to enable the automatic validation.

- Optimizing the model.
  Two ASLan++ versions of the Public Bidding case were modeled only for analysis: the first one is a complete formalization of the application, used for testing the features of the ASLan++ language. However, since this process is really long (96 steps after translation in ASLan) and cannot be handled in a suitable time by the backends, a simplified version was created, removing some elements of the original application which were not relevant for the formal verification, such as some entities not needed to check the security properties, or some messages not involved in the verification, such as the receipts. These simplifications,

which allow one to have a model with 27 steps after the translation into ASLan, are detailed in Deliverable 5.3 [11].

- Fixing the number of Bidders.
  In the application, the number of Bidders who want to submit an offer is not known before the process starts. Such a behavior cannot be modeled. In ASLan++, one has to specify each entity before the process starts, so the maximum number of Bidders is known in advance. But this does not change the validation results. If the security properties are valid for one Bidder, they will be too for every Bidder who has exactly the same behavior.

- Fixing the number of eligible Bidders.
  Some steps of the Public Bidding application are human tasks. For example, the Bid Manager has to establish the list of eligible Bidders, by checking if the documents are fulfilled correctly. This cannot be modeled with ASLan++. That is why we had to specify in advance the list of eligible Bidders.

Dealing with sets. In the Public Bidding scenario, we need to handle sets of messages, i.e., to send them and receive them. As we cannot send or receive sets in ASLan++, we have opted for modeling message sets as a concatenation of messages. The use of concatenation raises two issues:

- The first one is at the recovery of a message, after its reception: we had to use new types and new variables to be sure to recover the original message. And such a method becomes difficult when you have encrypted messages, if the receiving entity cannot decrypt them. For more information, see [11].

- The second one is the increase of verification time. For concatenation, loops are used: before sending, to create the message, and at reception, for recovery. But concatenation allows to sign and encrypt messages in an easy way.

Results The verification of this complex scenario is difficult to perform, due to the large number of transition rules. Even in the simplest model, validation is still very time-consuming, even for simple properties, which makes developing and correcting the model difficult. For example, increasing the number of sessions/loop traversals (option *--nb* for CL-AtSe) or the search depth (option *--depth* for OFMC) is time-consuming, as the verification time increases significantly.

Some unsafe versions of the model and a file whose purpose is to give an executability trace were created to evaluate the behavior of the tools on them. Note that the complete model (scene 1) was not created for being validated, but rather to provide a complete formalization of the scenario. In fact, this version contains several useless details for validation, and cannot be handled by the backends, as there are too many steps. This is why there is not any family of problem case validated on this scene, as expressed in Table 2.

On the simplified model (the second scene), the use of the *--untyped* option with OFMC, needed with this tool, does not allow one to obtain the correct trace of execution on the original file. As explained in [9], the reason this option has to be used is that OFMC has limitations with subtypes and composed types. To bypass this issue, minor modifications were made on the model, preferring the use of functions rather than composed types and subtypes. Even doing that, we were not able to find neither an execution trace nor to validate the security properties with OFMC. In particular, the modifications brought an increase in the size of the space search and thus in the time of the verification. By using the *--depth* option we tried to limit the search space. Locally, we managed to obtain results for a depth of 29 in approximately 20 hours. However, even in this case, we do not have the expected results.

With CL-AtSe, the expected results were found on scene 2 using the *--nb 2* option, which allows one to increase the number of loops traversals. SATMC also gives the expected results on this scene.

Orchestration The Orchestrator was evaluated on the Public Bidding example by stating as goal the outcome of a successful bid. The starting point was scene 4, which is a variation of scene 2 with some manual loop optimizations, as described further below. The Bidder process was removed from the model and the successful orchestration showed that the Orchestrator can generate a Bidder process that uses the other services provided in the model (notably the Bid Portal) to achieve the desired outcome. The Orchestrator thus effectively manages to synthesize the Bidder process.

For the resulting orchestration, the properties specified for the original model were verified: authenticity, integrity, non-repudiation, secrecy, and executability, thus checking the correctness of the generated orchestration.

In order to achieve both orchestration and validation, several targeted optimizations were performed on the ASLan output of the ASLan++ translator. In particular, the model was rewritten such that all paths through the main message reception loops would be distinct. This makes all transitions reachable from the initial states on paths which need not repeat any tran-

sition, which significantly lowers the state space exploration complexity for the backends. In particular, for CL-AtSe, the default option *--nb 1* can be employed instead of *--nb 2*, and time decreases 100 times, from more than half an hour to less than 20 seconds.

Overall, this case study showcased a successful combination of orchestration and validation, albeit on a simplified model, and pointed out optimizations that can help in handling more complex models. Some of these optimizations could be performed by enhancing the step lumping option of the translator, while others require rewriting the model with validation performance in mind.

2.1.6 Digital Contract Signing

Security requirements coverage The Digital Contract Signing (DCS) case study [6] describes two parties that have secure access to a trusted third party Web site, a Business Portal (BP), in order to digitally sign a contract. First, BP generates an electronic document corresponding to the terms of agreement between the two parties. Then, the first party accesses BP using a Web browser, views the contract and signs it using a digital certificate. BP verifies the generated signature and stores it. The second party, in turn, connects to the BP Web site, checks the status of the existing signature and then co-signs the contract after viewing it. Once the signatures have been completely verified by the business portal, the signers are notified. Then, the contract is archived for long-term conservation. The BP's internal system is Web service enabled. It delegates the processing of proof elements (signatures, signed documents, timestamps) to a Security Server (SS) using SOAP messages. Three available trusted services are at the disposal of SS: a Time Stamper (TS), a Public Key Infrastructure (PKI), that returns information about the validity of a given certificate and an Archiver (ARC), an external storage facility.

We have made different specifications depending on whether we consider analysis of DCS according to security properties or orchestration of DCS. For the analysis, we have written different ASLan++ specifications depending on the number of participants involved and on the security property checked. This scenario focuses on the authorization policies, the accountability, the workflow security, the application data protection and the communication security problem cases.

Modeling limitations

Simplifications made for the design. We have made some assumptions in order to simplify the specification of the DCS case study.

- The phase of certificate generation and distribution has been abstracted away. We have assumed that authenticated communication is done via messages that are encrypted with a shared key between the participants involved. For instance, we have assumed that there is a symmetric pre-shared key between each signer and the business portal.

  Once the first version of our model is validated, we plan to modify the specification in order to add certificates that will be generated and distributed by the *PKI* before starting the signing procedure. Certificates may be modeled by facts of the form *certificate(A,B)* where A denotes the certificate authority and B denotes the owner of the certificate.

- In the real application, when the first signer requests to sign the contract, the business portal provides him with the contract and the signature policy. The signer has to check that the contract corresponds to the terms agreed upon before. If this is the case, the signer extracts some parameters of the signature policy in order to sign the contract.

  We have abstracted away all these details. We have assumed that the contract sent by the business portal is the right contract on which signers and the business portal have already agreed. Thus, if the signer receives the contract from the business portal then he signs it and sends it back to the portal using his private key.

Handling sets. In the DCS case study, we have to handle sets. For example, in our scenario, the *PKI* has to send on demand or periodically the certificate revocation list *CRL*. We have modeled the *CRL* as a set of pairs of the form *(agent, public_key)* associating to each agent its public key.

As we cannot model sending or receiving sets in ASLan++, we have modeled sets as a concatenation of their elements. Receiving or sending the *CRL* is then modeled by receiving or sending a concatenation of pairs of the form *(agent, public_key)*. Since elements of a set have a precise type (different from message), receiving a message that models a set does not lead to false matching as in the case of Public Bidding.

Results Experimenting with CL-AtSe and SATMC for the verification of the DCS scenario gives different results depending on the scene concerned and on the property checked (security property or executability). Note that this scenario is not supported by OFMC due to the presence of universally quantified Horn Clauses. For our two DCS ASLan++ models (**scene 3** and **scene**

2), that translated into 18 and 30 transitions respectively, SATMC gave as result `INCONCLUSIVE`. The safety of a property cannot be really given, as not all states are always reachable. Therefore SATMC gives this result with the remark that no attack was found for the reached states. The result for **scene 2** and **scene 3** only concerns the verification of security properties. Indeed, for executability tests specifying executability as attack state, SATMC was the first backend that gave a valid execution trace. This tool succeeded also to find an execution trace for the complete version of DCS (**scene 1**) that includes 46 transitions. SATMC was able also to check executability for the most of our specifications with the option `--exec`. Checking security properties for **2** and **3** of the DCS specifications is well supported by CL-AtSe. However, most of the results obtained with this backend were found using the special option `--nb 3` which allows for iteration of loops during analysis, as the default number is "1".

For **scene 1**, the two compliant tools (CL-AtSe and SATMC) either took a long time to produce a result, blocked or run out of memory, as this complete model is made of many steps. It would be more interesting to support at least **scene 2** without fixing the number of loops. This scene includes the Security Server, an entity who plays an important role in the management of digital proofs in the signing procedure. The verification of this complex scenario is difficult to perform, due to the large number of transition rules. This makes modeling, testing and updating the model difficult. That is why we have proceeded in an incremental way, in order to verify our specifications. However, validation still does not always succeed (especially for **scene 1**).

Orchestration The orchestration problem we posed is to generate a Mediator that emulates SS: satisfy BP's requests while relying on the community of available services (namely TS, PKI and ARC). Whereas the BP is the most natural candidate for a service to be orchestrated, we chose to generate SS in order to demonstrate the Orchestrator capabilities, since this entity possesses the most complex behavior of all in DCS.

The models for the Orchestrator was directly designed in ASLan independently from ones intended for the validation. The limitations assumed for these models are as follows: we assume the existence of a unique BP that represent signers and the number of represented signers is limited to two. Even in these settings the resulting model (that includes the generated Mediator) contains more than 40 transitions and the validation phase does not finish in a reasonable time, apparently because the specification satisfies the specified security goals.

We list the features used in DCS orchestration:

- *Input style*
  The orchestration problem was presented in two ways: *(i)* by giving BP as a client of SS, and *(ii)* by giving a partial specification of the SS which had be completed. In both cases the Orchestrator successfully found a possible orchestration in few seconds.

- *Security policies*
  The models take into account individual security policies of available services (e.g. every message received by ARC must be encrypted with his public key, etc) and thus the Mediator produces policy-compliant requests.

- *Rely-guarantee method*
  The Mediator (SS) must produce some assertions (e.g. about the validity of a certificate) on which BP relies to continue his execution. Assertions are claims made by some issuer and stating some property for the parameters they transport. In the Web Service standards similar objects are represented by SAML [38] assertions, which we simply model here using first-order terms. The presence of an assertion in some received message by BP represents an additional constraint to the orchestration problem since SS will have to provide it. For example, to produce an assertion about the validity of a signer's certificate, SS has to contact an internal service: the Assertions Provider (AP) which permits to provide a good assertion only if a positive answer about the validity is given by a trusted third-party (here PKI). AP plays a role similar to the *trust engine* in the *rely-guarantee* method introduced in [30].

  We emphasize here the expressiveness of assertions for the considered orchestration problem, since they can describe for example the need to use only certain schema for the timestamps, or only PKI's offering the *Online Certificate Status Protocol (OCSP)* versus those using the classical *Certificate Revocation List (CRL)*. This can be easily done by tuning the AP service behavior to match the expectations.

2.1.7 Electronic Health Records

There are several situations in which a physician wants to collect eHealth records from an external source. For instance, a physician in a clinic needs data about a patient's allergies and medication from that patient's general practitioner. The GP sends the patient's records, which must eventually be stored in the clinic's system and be brought to the notice of the clinician.

Either the system per default stores all such records or the clinician selectively decides to store them or not. We modeled both options.

The specification for the EHR scenario is formalized in ASLan++.

Security requirements coverage To confer requests from the clinic's physician to the general practitioner and sensitive patient data back again, trust must be bridged from the clinic's physician to the general practitioner and vice versa. The patient data need to be securely transferred from the GP to the clinic's system and the clinic's physician. Integrity and authenticity of the documents must be ensured and verifiable, confidentiality of such personal health information is important. Also no party may falsely deny having originated such relevant requests and documents.

Thus the families of problem cases covered by the EHR scenario combines accountability, trust management, workflow security, privacy, application data protection, and communication security.

Modeling limitations The specification is written in ASLan++ and translated automatically to ASLan using the ASLan++ Connector. The model can be found in `ECR.aslan++`. There are 7 ASLan++ entities in the specification, and none of them uses loops, sets, or clauses. A few use some `if`-statements.

EHR sessions can be instantiated with one of two policy models. In the first case the clinic's *system* archives the patient records automatically, as soon as they arrive from the GP, and in the second case the clinic's *doctor* decides, whether to archive the records or not. Simultaneous sessions may comprise different policies.

1. In this documentation, we have simplified the fact that in the definition of a "signed message", there is a "time stamp" parameter `TT`. This parameter aims to help preventing replay attacks, i.e., by checking the time stamp one can recognize if a message is a copy of a message sent in the past. In ASLan++ models, we cannot consider the time stamp as a real parameter but use it only as a place-holder. Replay attacks could also be avoided using nonces.

2. For confidentiality requirements here, as well as in many other cases, one would actually like to write such a goal in the following way: "<data> must be secret for anyone who is not authorized to read it", without having to express directly who is authorized to read the data. For instance, this information may depend on the particular configuration of the system or service, or on the particular agents participating

in a session, or even, this information may be dynamic. The intention is that the definition of "who is authorized" can be inductive: initially the creator is authorized, and assuming no intruder, if A is authorized and sends it to B, then B is authorized. We do not have an explicit generic ASLan++ construct to define the notion "authorized to read this data". Most probably, one could dynamically extend the list of agents that are allowed to know a secret by manually introducing facts like `secret_SecretValue_set(IID)->add(NewAgent)`, but we have not explored this possibility in detail yet.

Results Translation and executability tests of EHR model have been performed successfully. All three backends are capable of finding the expected attacks for the executability checks.

The translator and the backends were run on a machine with a Pentium 4 CPU at 3,20 GHz, and 1,5 GB of RAM. The translator was the ASLan++ Connector (Version 0.6.0, `-gas`).

Cl-AtSe (Version 2.5-7b, `--not_hc`) took about 6 seconds testing 323 transitions and reaching 22 states for the clinic's *system* to store the records per default, and about 8 seconds testing 576 transitions and reaching 37 states for the clinic's *doctor* selectively deciding to store the records (or not). OFMC (wrapper version 0.1.2 with core 2010c, `--classic --untyped`) took about five seconds with the `--exec`-option to verify executability of 23 rules. It took about the same time to find the executability attack when analyzing the model, visiting 24 nodes (both for *system*).

With all three model checkers, we were able to analyze scenarios with one symbolic session and we could verify all security properties, i.e. all channel and security goals. However, the model checkers behave differently:

OFMC and SATMC are both capable to analyze the specification with one symbolic session and we could verify all channel and security goals. OFMC took about two seconds to come to the conclusion of no attack found, visiting 49 nodes.

CL-AtSe is capable to analyze the specification in more depth and give a conclusion that no attack is found. For one or two parallel sessions, the answer is given in a matter of seconds, while for three parallel sessions it takes nearly an hour.

CL-AtSe turned out to be the model checker that harmonized best with the EHR model. It succeeds in analyzing multiple sessions, while the two other model checkers did not.

- In detail, Cl-AtSe for one session and in the *doctor* case took about 6 seconds to test 316 transitions and reach 24 states. In the *system* case it took the same time to test 307 transitions and reach 23 states. In both cases no attack was found.
- For two sessions (one *system*, one *doctor*), CL-AtSe took about 70 seconds to test 34277 transition and reach 4520 states, and no attack was found.
- In fact, we ran the ECR model with three parallel sessions (two *system*, one *doctor*) with CL-AtSe. The model checker took nearly 45 minutes (about 2600 seconds) to test 372013 transitions and reach 52945 states, and no attack was found.

In summary, with the transition from two to three parallel sessions in the EHR model, the model checking time increased by about 36 times, the number of tested transitions increased by about 11 times, and the number of reached states increased by about 12 times.

A comparison with the results obtained by running only one session of EHR is not meaningful because the model checker uses in that case a level for low complexity.

We also ran our model in a completely untyped version (with Cl-AtSe Version 2.5-6e December 8th 2010). In this case, being accepted as the `ReqSys`, the intruder could perform an attack by sending to the `RspSys` a message which the `RspSys` would accept as a `Soreqsys`. In the following, however, the proof of

```
assert Req_domain: domain(Req_Dr, Req_Sys)
```

will fail because no acceptable relationship between `ReqDr` and `ReqSys` (which is now the intruder) can be established. Obviously, the same attack could be performed when we replaced explicitly one of the agents by the intruder.

More details on the EHR model can be found in [11].

2.1.8 Process Task Delegation

Rather often, a patient carries physical documents containing the results of an examination (e.g., radiology images) or diagnoses by one doctor and presents them to a different doctor for analysis or second opinion. Process Task Delegation (PTD) in eHealth applications promotes the secure and privacy-protecting electronic execution of these workflows.

When using PTD, Dr. C (or some other health care professional or patient) logs into his clinic's application. Interacting with his application, Dr.

C requires data from another application in another domain. His own application refers him to that other application. In the background, on another route the trust servers of these applications (and eventually intermediate trust servers) without Dr. C's interaction negotiate and mediate trust and authorization for Dr. C to access the other application. Thus cryptographic key material and authorization information is not carried by the potentially negligent Dr. C and his potentially vulnerable computer.

If explicit non-repudiable patient consent is required before personal medical data may be transferred, Dr. C must obtain a digitally signed consent from the patient. The local application includes this signed consent in Dr. C's request to the other application (sealed and encrypted). The other application verifies the patient consent before disclosing the medical record, and also notifies the patient of the transfer.

We have modeled the generic PTD scenario and the patient consent PTD sub-scenario. The specifications are formalized in ASLan++.

Security requirements coverage Process Task Delegation (PTD) is a method for electronically creating complex workflows by gluing together different sub-flows in a largely dynamic and decentralized, but nevertheless secure way.

PTD allows one to delegate workflow tasks from one active Web Service or application to one or several other Service Providers in a largely dynamic way. PTD for this also allows for Trust and Authorization Negotiation. Between the domains of the different stakeholders (e.g. clinic, patient, general practitioner, insurance company), trust levels and authorization decisions are negotiated.

PTD in eHealth applications can provide Single-Sign-On Capabilities, so doctors and other eHealth personnel do not have to enter their identification credentials repeatedly over time.

Explicit User Consent also is supported by PTD. This is a basic requirement in many common privacy-protecting scenarios. Patient consent is required, before health records may be transferred and disclosed. On the other hand, for the disclosing or receiving party PTD's facilities for Non-Repudiable User Consent are important, as this enables parties acting on such a consent, to prove their entitlement.

So the PTD scenario combines federation, authorization policies, accountability, trust management, workflow security, privacy, application data protection, and communication security.

Modeling limitations The specifications are written in ASLan++ and translate automatically to ASLan using the ASLan++ Connector. The models can be found in `PTD.aslan++` for the generic PTD, and `PTD_PC.aslan++` for the patient consent sub-scenario respectively.

There are 7 ASLan++ entities (including `Environment` and `Session`) in the generic PTD specification. The patient consent PTD specification contains 8 ASLan++ entities (i.e. the patient as additional entity). Both specifications use neither loops nor clauses nor `if`-statements. Both use some `sets`. However especially the number of entities in combination with a rather large workflow definition result in complex specifications which, after translation to ASLan, prove difficult to be analyzed by the validation platform.

TLS Tunnel Properties / Sessions: We cannot directly express in ASLan++ that the channels used between the Client and Application 1, as well as the channels between the Client and Application 2, are always the same. Although it is possible to express this constraint at a lower layer, no corresponding notation in the ASLan++ has been defined yet. For instance, the same channel used to transport `workflow_request` is used — after upgrading it to a secure channel due to the Client authentication — for transporting the `choice_request`, `choice_response`, `task_invocation`, `task_result`, and `workflow_response` messages.

Fraudulent Clients: We cannot directly express in ASLan++ that the Client is "mostly dishonest": either he fully follows the protocol as specified in the `Client`, or his role is played by the intruder, in which case the intruder has full access to the Client's knowledge, including e.g., the `TaskHandle`. In a fully adequate model, we would let the Client's role be played by a manipulating, but not leaking, intruder. Thus, this "partial" intruder is only interested in gaining access to someone else's secrets, in elevating his privileges in order to gain access to services or applications for which he is not authorized, or in creating confusion (say, allowing an agent A to believe he is sending an information to B, but in reality he is sending it to another agent, which may be or not the partial intruder) but he will never disclose his own secrets.

Non-Repudiation as Goal: With digital signatures, there are sufficient means to implement non-repudiation. However we would like to have more abstract formal means to state goals of non-repudiation explicitly.

Semantic Consistency: A2 can in principle check in an automated manner, whether P belongs to the set of Patients known to A2 (as A2 is

expected to have patient data about P) and whether P is mentioned as data subject in the details of the order. A similar issue concerns Application 1. As line of defense, A1 could check the patient consent as far and as early as possible, and only in the face of a consistent and validly signed consent, encrypt, forward, or decrypt the results from A2 for Client Dr. C. It requires further deliberations whether A1 should be required to check the semantics of the client order and the patient consent and in what manner this can be automated. Means for efficient and expressive modeling of such checks in ASLan++ and the respective programming guidelines should be provided.

Validity Time Ranges: ASLan++ offers formal means to state goals of freshness, but does not allow their implementation via timestamps. For instance a check of the time-stamp `t1` on a signed patient consent from the point of view of an Application (`Actor`).

```
?SignedCS = signed_message(?CS.Actor,t1,?P)
```

Current limitations of ASLan++ to deal with time computations also affect the expressiveness and usefulness of time-stamps. Currently, as "place-holder" for real time stamps, we use discrete values that are checked for equality against an expected time stamp by the receiving party. We would like to have facilities to properly model time stamps on signed contents and to model increase in time and validity time ranges for the freshness check.

Results We have successfully performed translation and executability tests of the generic PTD model (`PTD.aslan++`) and the PTD plus Patient Consent model (`PTD_PC.aslan++`) with Cl-AtSe.

The translator and the backends were run on a Celsius W380 with an Intel Core i5-650 Processor (2 Cores / 4 Threads), 3.20 GHz each Core and 3 GB RAM under Windows XP. We used as translator the ASLan++ Connector 0.5.11 with the option `-gas` and CL-AtSe, Version 2.5-7b with the option `--not_hc` (PTD) and `--free` (PTD_PC).

Cl-AtSe was capable of finding expected attacks for the executability checks. For this it took about 32 seconds (PTD) and 87 seconds (PTD_PC), tested 16841 (PTD) and 42291 (PTD_PC) transitions, and reached 4343 (PTD) and 8950 (PTD_PC) states respectively. SATMC and OFMC could not check executability for both of the models properly.

CL-AtSe is the only backend so far that is able to validate the generic PTD scenario and the PTD plus Patient Consent scenario, yet even here, we

could only verify one (symbolic) session. In this context, however, we could verify all channel and security goals. For one session, CL-AtSe took about 1,5 minutes (100 seconds for PTD) and not quite 4 minutes (220 seconds for PTD_PC), tested 54303 (PTD) and 118062 (PTD_PC) transitions, and reached 14288 (PTD) and 25199 (PTD_PC) states respectively.

CL-AtSe is capable to analyze both specifications fully and give a conclusive verdict that no attack is found. We were not able to receive an answer from CL-AtSe for more than one session. OFMC and SATMC did not check either of the models properly at all.

Comparing these results (one session of the PTD generic model) with the results by running two parallel sessions of the ECR model, we conclude that the complexity of the PTD generic model (one session) is approximately 3 to 4 times higher than the complexity of the ECR model (two parallel sessions).

We cannot explain conclusively why this happens but we assume the reasons are the following:

- The PTD generic model includes 5 fully participating agents (C, A1, A2, TS1, TS2) and generates 14 message transmissions (for the patient consent PTD it would be one more entity (P) and three more message transmissions). On the other hand, the ECR model includes 4 fully participating agents (dr1, dr2, sys1, sys2) and two passively participating agents (pat, db) and generates 7 message transmissions.

- The fact that Application 2 first receives an encrypted message and only gets the corresponding decrypting key several steps later increases the complexity of the model.

- The possibility given to the Trust Servers TS1 and TS2 to weaken the attributes ADT and especially the check of the transitivity of the weaker clause may increase the complexity of analyzing the PTD generic model.

Still, even with just a single session, two potential security issues have already been found during the development of the model.

- We noticed that the message tag `workflow_response` is necessary in the ASLan++ model to avoid a kind of "type flaw" attack by re-playing a `task_invocation` message when the Client actually expects his part of the workflow result. This is just a spurious "attack", possible only because the model cannot express the strict message ordering property actually enjoyed by the channel between Application 1 and the Client after the Client has been authenticated.

- If the Trust Server 2 accepts `token_request` messages containing any `TS1` (as part of `TaskHandle`), it may be tricked by the intruder impersonating a Trust Server 1 (=Actor). This leads to a violation of (Authorization Management - TLS Tunnel Property) described in §5.3.3 of [11]

```
secrecy_goal secret_ADT: A1, Actor,
                         A2, trustServer(A2) : ADT;
```

  The problem is avoided by `TS2` accepting only trusted `TS1`, which we model by the condition `genuine_ts(?TS1)` checked by `TS2` on receiving the `token_request` message. To this end, the entity `Session` contains the fact introductions

```
genuine_ts (trustServer(A1));
genuine_ts (trustServer(A2));
```

More details on the generic PTD scenario and the patient consent PTD sub-scenario can be found in [11].

2.1.9 Access Control Management

The purpose of the Access Control Management scenario for eHRs was to *informally* test the *expressivity* of the ASLan++ specification language for the purpose of "factoring out" the access control part of the case study, and formalizing it independently of any particular work flow. Indeed, in real applications, and in particular in complex ones like the ones pertaining to Electronic Health Records, the Access Control Management specification is uncoupled from the work-flow specification: the designers and administrators that program the use cases are different from the ones that define the access control rules.

Thus, without any transition system given by a work flow, this scenario provides no specification that can be used as input for the Platform. The reader may consider the scenario as "incomplete", any sensible work-flow would complete it to produce a valid input for our platform. Nevertheless, we have included Access Control Management into the library and Platform assessment just to document this possibility of "factoring out" the access control part of a complex case study.

2.1.10 SAML Single Sign-On

The SAML Single Sign-On scenario focuses on the most used profile of the established security standard from OASIS, the SAML Web Browser SSO profile (see [37, §4.1] for more information). Federation, privacy and communication security are the main aspects studied in this context.

We have formally specified and validated industrial relevant scenarios where the SAML SSO services are employed according to the SP-initiated and IdP-initiated SSO protocols with and without artifact resolution. Around 30 formal specifications capturing these scenarios and the variety of interactions and configuration options have been formalized in HLPSL++ and then validated with SATMC. This amounts to around 90 problem cases successfully formalized and validated to cover the wide set of options and configurations of this important SAML Web Browser SSO profile.

The AVANTSSAR library includes 5 of these formal specifications (i.e., 15 problem cases) that provide a good understanding and summary of the formalization and validation carried out for SAML SSO (see Deliverable [11]). In addition, we have been working on one ASLan++ specification recasting the one in HLPSL++ that models the SAML-based SSO for Google Apps. This ASLan++ specification is also part of the AVANTSSAR library (i.e., 3 additional problem cases).

The overall 6 specifications, hereafter enumerated, are representive of the most important profiles of SAML SSO as well as of relevant SAML SSO industrial implementations:

Scene 1: SP_init-FC-one_channel.hlpsl++

Scene 2: SP_init-BC-two_channels.hlpsl++

Scene 3: IdP_init-FC.hlpsl++

Scene 4: IdP_init-BC.hlpsl++

Scene 5: SAML-based_SSO_for_GoogleApp.hlpsl++

Scene 6: SAML-based_SSO_for_GoogleApp.aslan++

Scenes 5 and 6 capture the SP-initiated variant of the SAML SSO as developed in the SAML-based SSO for Google Apps before June 2008 [26] in HLPSL++ and ASLan++ respectively. The main difference between these two formal specifications concerns the channel model used: the ACM is used in HLPSL++, while the CCM is used in ASLan++. As described in [3] and in various AVANTSSAR deliverables, in our validation a serious flaw was detected in the SAML-based SSO for Google Apps.

Scene 1 focuses also on the SP-initiated variant of the SAML SSO where front channels are employed (i.e., no artifact resolution) and one single SSL channel is used in the two message exchanges between the client and the service provider. The main difference with respect to scenes 5 and 6 concerns the authentication assertion that includes all the fields prescribed by the SAML standard and does not lack critical fields as in the version developed by Google. No attack were discovered here.

Scene 2 focuses also on the SP-initiated variant of the SAML SSO, but back channels are employed and the assumption on the single SSL channel between the client and the service provider is relaxed to better capture reality. This latter is the key to discover a flaw in the prototypical SAML SSO use case (as described in the SAML Technical Overview) that allows a malicious service provider to hijack a client authentication attempt and force the latter to access a resource without its consent or intention. This enables an attacker to launch Cross-Site Scripting (XSS) and Cross-Site Request Forgery Attacks (XSRF). This last type of attack is even more pernicious than classic XSRF, because XSRF require the client to have an active session with the service provider, whereas in this case, the session is created automatically hijacking the client's authentication attempt. This may have serious consequences, as witnessed by the new XSS attack that we identified in the SAML-based SSO for Google Apps and that could have allowed a malicious web server to impersonate a user on any Google application.

Scenes 3 and 4 focus on the IdP-initiated variant of the SAML SSO without and with the artifact resolution protocol respectively. The scene 3 models a variant in which authentication assertions are not signed contrasting what the standard requires. As proved by the validation, the scene 3 is vulnerable to trivial attacks. However, by adding the artifact resolution protocol (scene 4), the attacks can be resolved.

This scenario is supported by all the back-ends. In more detail, SATMC supports all scenes except 6 where a timeout is reached. CL-AtSe and OFMC do not support the ACM but they are both able to tackle scene 6. Details about results and modeling limitations are available in Deliverable [11].

2.2 Assessment of the tools

As stated in the Description of Work, the tools comprising the AVANTSSAR Platform should fulfill the following criteria:

- Each family of problem cases must be validated for at least 1 scenario.
- Each scenario must be validated for at least 1 family of problem cases.
- Each validator (CL-AtSe, OFMC and SATMC) must support at least 5 problem cases, each of them coming from different scenarios.

Table 6 shows which tools support at least one family of problem cases for each scenario. For example, the first line tells that, with respect to Loan Origination, the HLPSL++ connector, SATMC and CL-AtSe support at least one family of problem cases.

Even though specific criteria about coverage were not defined for connectors and for the orchestrator, we have decided to include in the table also their support status. This choice was driven by the fact that while all specification files have to pass through validators, they may not require to be translated via connectors. Every "yes" in the columns about connectors and the orchestrator means that all files are supported by the appropriate connector and all files that need to be orchestrated are supported by the orchestrator.

From the point of view of the validators, we consider specification files, where each specification file formalizes a problem case for the scenario it belongs to. So, a cell containing "yes" means that at least one specification file, and thus a family of problem cases for the corresponding scenario, is supported by that tool.

Table 6: Tool support status

Application Scenarios	**# Scenes**	**Connectors**			**Orchestrator**	**Validator**		
		ASLan++	HLPSL++	NW BPM		SATMC	OFMC	CL-AtSe
Loan Origination	2	-	-	yes	-	yes	no	yes
Anonymous Shopping	3	yes	-	-	-	no	yes	yes
Visa Application	1	yes	-	-	-	yes	yes	yes
Car Registration	1	yes	-	-	yes	yes	yes	yes
Public Bidding	4	yes	-	-	yes	yes	yes	yes
Digital Contract Signing	5	yes	-	-	yes	yes	no	yes
Electronic Health Records	1	yes	-	-	-	yes	yes	yes
Process Task Delegation	2	yes	-	-	-	no	no	yes
Access Control Management	1	-	-	-	-	-	-	-
SAML Single Sign-On	6	yes	yes	-	-	yes	yes	yes

3 Efficiency

The results we present in this deliverable were obtained through several testing phases. During each testing phase, we tested all specifications with all the backends and connectors (if required), both to check if the AVANTSSAR Platform fulfilled the validation coverage goals described in § 1 and to collect information about the efficiency of validators. Note that the number of specifications tasted is greater than the one listed in Table 2, as more specifications were used to automatically assess the scenarios to check, e.g., executability or single security properties.

Table 7 shows CPU analysis times of the final testing phase performed. Looking at the time values from the last row (namely "Total"), it is clear that running all tests required many CPU hours. In order to speed up the testing phases, we have parallelized the tasks using up to 8 amazon machines with the same features: Intel(R) Xeon(R) CPU E5430 at 2.66 GHz with 2 GB of RAM. Figure 1 gives an overview of a running testing phase that involves all the 8 amazon machines running in parallel.

All specification files were run against all tools by setting a timeout limit of one hour. When looking through the times from Table 7, consider that each tool performance also depends on the options used and on language features employed in the specification files. For example, if a specification contains composed type declarations, then OFMC has to be invoked with the `-untyped` option and thus the tool will take into account many more reachable states than the ones that would be considered by CL-AtSe and SATMC invoked in typed mode; see [9] for more details on tools and their options.

When we planned the assessment, we had originally envisioned to be able to measure the degree of simplification brought to the validation efficiency by the techniques developed in WP3. The assessment has shown that the automated reasoning techniques, together with the expressiveness provided by our specification languages such as the use of Horn clauses, are indeed fundamental for the validation, up to the point that validation is actually unfeasible without them; hence no comparison times are provided.

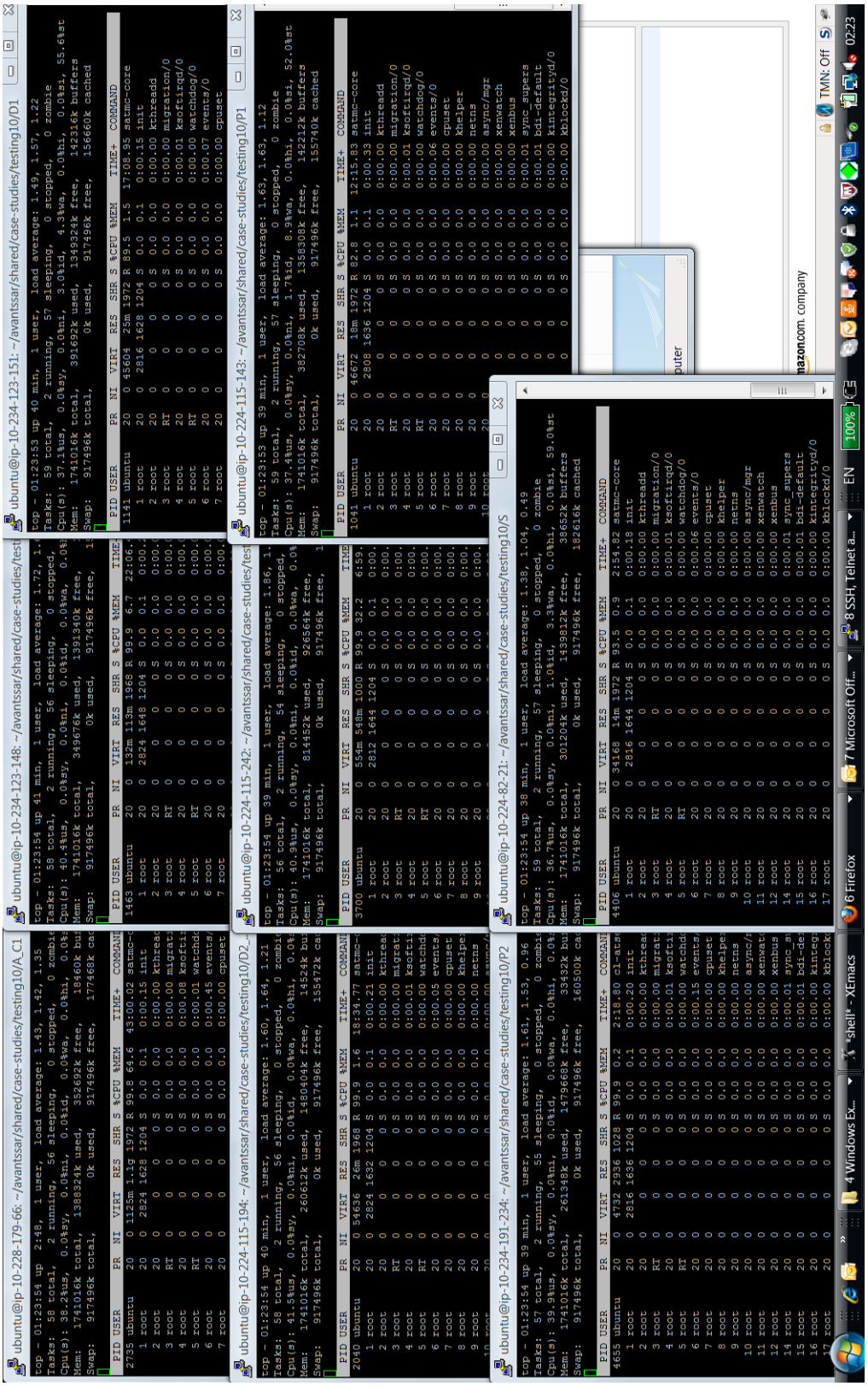

Figure 1: Screenshot of open SSH sessions used to monitor all amazon machines

Table 7: CPU analysis times for each backend on the different application scenarios. Times are in seconds and S/NS/TOUT are abbreviations for Supported/Not Supported/Time OUT. Moreover, for each scanrio the total number of Horn Clauses (HC) and transitions (step) present in the specification files are shown.

	Dimensions		**SATMC**		**OFMC**		**CL-AtSe**	
Application Scenario	**HC**	**Steps**	**Time**	**S/NS/TOUT**	**Time**	**S/NS/TOUT**	**Time**	**S/NS/TOUT**
Anonymous Shopping	180	94	0	0/6/0	14457.83	2/0/4	7205.91	4/0/2
Car Registration	349	258	14460.54	7/1/4	33401.31	2/1/9	7269.35	10/0/2
Digital Contract Signing	238	52	14104.87	9/5/1	7200.00	0/13/2	22506.77	9/0/6
Electronic Health Records	89	48	19.33	1/1/0	3605.08	1/0/1	125.37	1/1/0
Loan Origination	303	418	767.80	9/0/0	0	0/9/0	17928.89	4/0/5
Process Task Delegation	90	39	7201.68	0/0/2	0	0/2/0	1092.43	2/0/0
Public Bidding	117	631	37972.69	10/2/8	50005.97	4/2/14	9642.87	19/1/0
SAML Single Sign-On	21	215	5589.49	15/0/1	22.77	1/15/0	1.85	1/15/0
Visa Application	38	19	44.83	1/0/0	3.12	1/0/0	9.86	1/0/0
Total	1425	1774	80161.23	52/15/16	108696.08	11/42/30	65783.30	51/17/15

4 Remarkable Results

The main result of the project is the AVANTSSAR Validation Platform, shown in Figure 2. The platform includes a *connectors layer*, i.e., a layer of software modules that carry out the translation from application-level specification languages (such as BPMN and BPEL, as well as our own AnB and ASLan++) into ASLan, and vice versa for the platform output. The platform then takes as concrete input a policy stating the functional and security requirements of a goal service and a description of the available services (including a specification of their security-relevant behavior, possibly including the local policies they satisfy) and applies automated reasoning techniques in order to build an orchestration of the available services that meets the security requirements stated in the policy. The platform comprises two main components: the *Orchestrator* tries to build an orchestration, i.e., a composition, of the available services in a way that is expected (but not yet guaranteed) to satisfy the input policy; the *Validator* automatically analyses the validation problem resulting from the Orchestrator output, where a failed validation means the existence of vulnerabilities that need to be fixed; otherwise, the composition of the services is guaranteed to be secure, i.e., to meet the input policy. If the input model already contains a complete service orchestration, the AVANTSSAR platform directly invokes the Validator.

In the following subsections, we summarize the different components of the platform highlighting what we see as the main remarkable results of the project, pointing the reader not only to the assessment tables in the previous sections but also, and more importantly, to the other project deliverables for further details.

4.1 Specification languages

4.1.1 ASLan (and ASLan++)

Modeling and reasoning about trust and security of SOAs is complex due to their main characteristics: they are *heterogeneous*, *distributed* and *dynamic, continuously evolving*. Various modeling languages have been proposed, e.g., BPEL [39], π calculus [31], F# [15], to name a few. Each of them, however, focuses only on some aspects of SOAs, and cannot cover all previously described features, except perhaps in an artificial way. One needs a language fully dedicated to specifying trust and security aspects of services, their composition, the properties that they should satisfy and the policies they manipulate and abide by. Moreover, the language must go beyond static service structure: a key challenge is to integrate policies that are dynamic (e.g.,

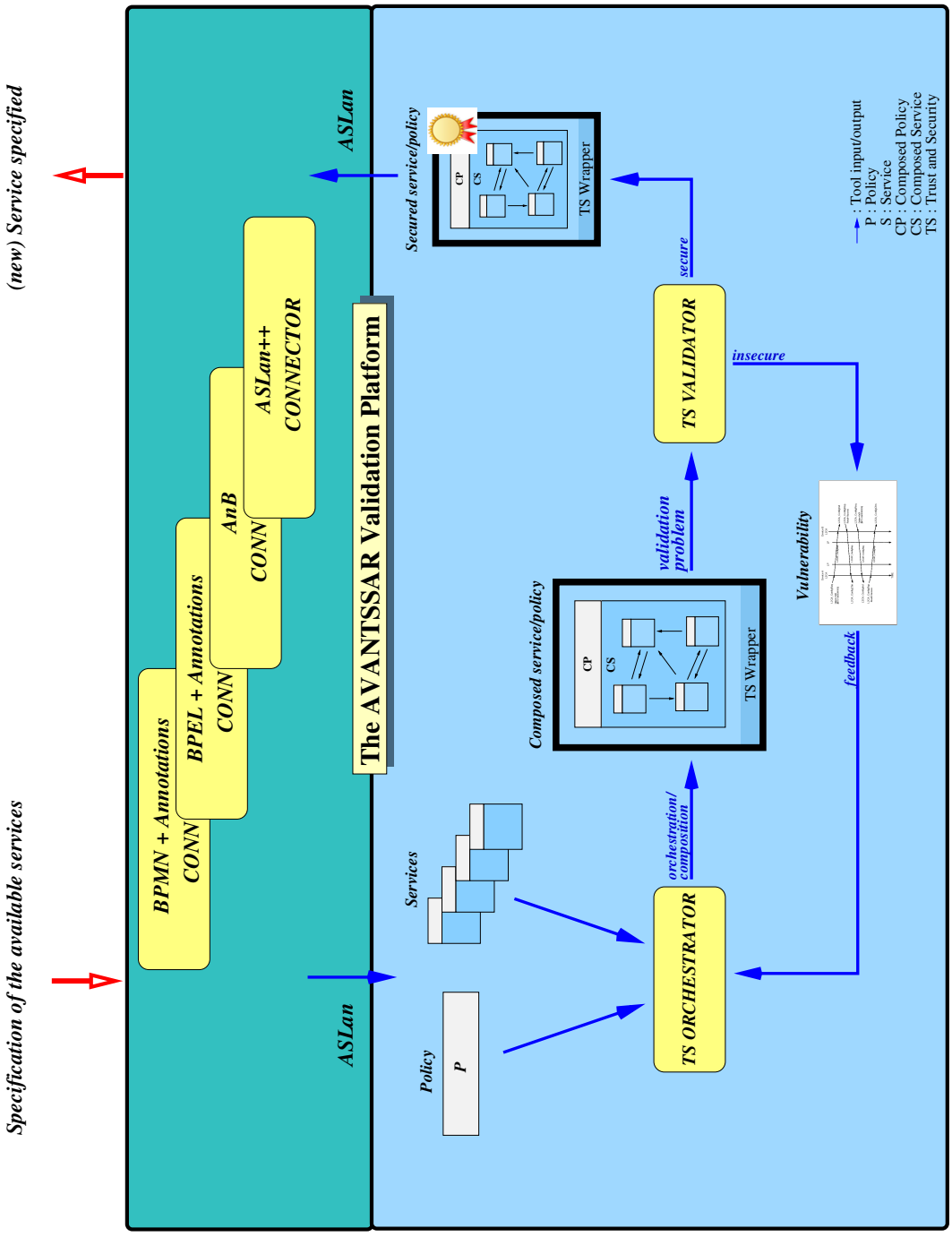

Figure 2: The AVANTSSAR Validation Platform.

changing with the workflow context) with services that can be added and composed dynamically themselves.

As a concrete solution, we have defined a language, ASLan, that is both expressive enough that many high-level languages, such as BPEL, can be translated to it, and amenable to formal analysis. A key feature of ASLan is the integration of *Horn clauses* that are used to describe policies in a clear, logical way, with a *transition system* that expresses the dynamics of the system, e.g., agents can become members of a group or leave it, with immediate consequences for their access rights. *Security properties* can be modeled by using different languages; in ASLan we have chosen to employ a variant of linear temporal logic (LTL), with backwards operators and ASLan facts as propositions. This logic gives us the desired flexibility for the specification of complex goals, as illustrated by the problem cases considered in the assessment, where formalizations in ASLan++, HLPSL++, or annotated BPMN (or even in AnB, although no AnB specification is considered in the assessment) are automatically translated into ASLan.

4.1.2 Industrially suited specification languages

The industry migration activity provides a means to expedite the transferring of the project results into the development process of the selected industrial partners of the consortium. To that end, we have integrated the AVANTSSAR Validation Platform into real industrial environments and described its successful application on some industrial scenarios, with a particular focus on best practices and lessons learned. This has been made possible by the development of *connectors* from and to a number of *industrially suited languages* [9, 12]: BPMN and BPEL, as well as our own AnB and, most notably, ASLan++. As exemplified by the protocols in the library [11], which are specified mostly in high-level languages rather than directly in ASLan, these languages allow engineers and practitioners to specify their case studies in a language close to industrial practice, and then have their specification be automatically translated by the connector(s) into ASLan, and thus fed into the AVANTSSAR Platform. Moreover, all validator back-ends of the AVANTSSAR Platform have the same output format, based on which a connector may be used for representing an attack (textually or graphically) in a more suitable form for the connector users.

4.2 Automated reasoning techniques, orchestration and validation

Due to the inherent complexity (heterogeneity, distribution and dynamicity) of the Internet of Services, the challenge of validating services and service-oriented applications cannot be addressed simply by scaling up the current generation of formal analysis approaches and tools. Rather, novel and different validation techniques are required to automatically reason about services, their composition, their required security properties and associated policies. In particular, one has to consider the various ways in which component services can be coordinated, and develop new techniques, such as model checking, that allow for compositional validation reflecting this modularity, as well as cope with the complexity problem. Moreover, for the practical use and take-up by industry and standardisation organisations, it is essential that any such verification technique provides a high degree of automation.

In the following subsections, we discuss these four points — orchestration, model checking of SOAs, compositional reasoning, and abstraction-based validation techniques — in more detail and describe how they have been implemented in the AVANTSSAR Platform.

4.2.1 Orchestration

Several notions of service orchestration have been advocated (see, e.g., [32]). However, in inter-organizational business processes it is crucial to protect sensitive data of each organization; and our main motivation is to take into account security policies while computing an orchestration. The AVANTSSAR Platform implements an idea presented in [21] to automatically generate a *mediator* that provides a new functionality by composing a set of available services specified in ASLan and possibly tied with security policies.

We present below the important distinguishing features of our approach:

- Several tools have addressed the WS orchestration problem but, to our knowledge, previous works abstract away the security policies attached to the services, while we consider them as an additional constraint.
- Most automatic orchestration approaches work by computing products of (communicating) finite-state automata, where messages are restricted to a finite alphabet. However, by specifying web services in ASLan, we can express a richer set of messages using first-order terms (including symbols for cryptographic functions).
- When there exists a composition of services that meets the client's request, we can always extract an executable ASLan specification of the

Mediator service. This includes the operations for generating messages to be sent by the Mediator and the security checks to be performed by the Mediator on the received messages.

- The orchestration solution can be automatically checked for security against an active attacker by calling the Validator component. If the specification meets the validation goals, i.e., no attack is found, the orchestration solution is then considered safe with respect to the user's requirements, otherwise a verification report including the proof of violation is returned. In the latter case, the Orchestrator is able to backtrack to try an alternative solution.

- The AVANTSSAR Orchestrator has been applied to several case-studies (Digital Contract Signing, Public Bidding and Car Registration Process) that cannot be handled by other tools since the messages exchanged by services are too complex (e.g., they are non atomic and built with cryptographic primitives) and require some automatic adaptation. For example, the Orchestrator has automatically generated a Security Server in the Digital Contract Signing case study (originated from a commercial product of OpenTrust), and was likewise able to generate the Bidder behavior for Public Bidding.

  In the Car Registration Process case study proposed by SIEMENS, the AVANTSSAR Orchestrator has been able to cope with additional constraints imposed by the authorization policies of the available services, specified as a set of Horn clauses.

4.2.2 Model Checking of SOAs

Model checking is a powerful and automatic technique for verifying concurrent systems. It has been applied widely and successfully in practice to verify digital sequential circuit designs, and, more recently, important results have been obtained for the analysis of security protocols. In the context of SOAs, a model-checking problem is the problem of determining whether a given model—representing the execution of the service under scrutiny in a hostile environment—enjoys the security properties specified by a given formula. As mentioned before, these security properties can be complex, requiring an expressive logic. Moreover, in order to take into account the fact that services often rely on transport protocols enjoying some given security properties (e.g., TLS is often used as a unilateral or a bilateral communication authentic and/or confidential channel), it is important to develop model-checking techniques that support reasoning about communication channels

enjoying security-relevant properties, such as authenticity, confidentiality, and resilience.

Among general model-checking techniques, *bounded model checking*, by supporting reasoning about LTL formulae, allows one to reason about complex trace-based security properties. In particular, the AVANTSSAR Platform integrates a bounded model-checking technique for SOAs [1] that allows one to express complex security goals that services are expected to meet as well as assumptions on the security offered by the communication channels.

We have also developed a number of other analysis techniques, along with algorithms and complexity results for service composition and analysis.

4.2.3 Channels and Compositional Reasoning

A common feature of SOAs is an organization in *layers*: we may have a layer that provides a secure communication infrastructure between participants, e.g. a VPN or a TLS channel, and run applications on top of it, as if the participants were directly connected via tamper-proof lines. It is of course undesirable to verify the entire system as a whole: this can easily be too complex for automated methods, and lacks generality and reuse. In fact, an application that requires a secure connection should not depend on the details of the realization of the secure connection and, vice-versa, a system that establishes a secure channel should be able to run arbitrary protocols over it. Thus, a compositionality result is desired: if components are safe in isolation and satisfy certain properties, then they can be composed into a larger system that is also safe.

Recent research has achieved progress in this direction for the parallel and sequential composition of protocols [22, 24, 25], i.e., independently using several protocols over the same communication medium. Adding to these, in the AVANTSSAR project we have obtained first results for the layered compositional reasoning needed for SOAs, namely running an application over a channel [35, 36, 27]. This means verifying that *(i)* a protocol such as TLS indeed provides an authentic and confidential channel, *(ii)* an application system is safe if its communication is routed over a secure channel, and *(iii)* both satisfy certain sufficient conditions (their message formats do not interfere). Here, *(i)* and *(ii)* are verification tasks that the tools can check in isolation, and *(iii)* is a format property that can be checked statically. If *(i)*, *(ii)*, and *(iii)* hold, then we can conclude that running the application over the established channel is also safe.

4.2.4 Abstract Interpretation

The complexity of SOAs is a major challenge for classical model-checking methods. To cope with this, formal validation approaches often strictly bound all aspects of a system, e.g., the number of service runs that honest agents (and a dishonest agent) can perform. However, one would rather verify a system without such limitations, e.g., no matter how many agents use it in parallel. Hence, methods based on abstract interpretation have recently become increasingly popular [16, 17, 18, 19, 23, 40].

The idea between abstract interpretation is to avoid reasoning about a transition system and rather compute an over-approximation of everything that will ever hold true. This allows for the use of classical automated first-order reasoning techniques, in particular resolution or fixed-point computations of static analysis. Thanks to the over-approximation, these systems completely avoid the state-explosion problems of model checking and can analyze systems without bounds on the number of runs. On the downside, the over-approximation can introduce false positives, i.e., attacks introduced by the over-approximation while the actual system is safe.

The ability of abstraction methods to avoid exploration of concrete transition systems has been the reason for their success, but on the other hand also implies a serious limitation for the verification of complex SOAs. Since there is no notion of time, we cannot model that at some time-point, a key, certificate, access right, or membership is revoked. The *set-based abstraction method* [34] can overcome this limitation while preserving the benefits of abstract interpretation. The idea is to organize data by means of sets and to abstract data by set membership: We can then identify all users that belong to the same set of groups as one abstract equivalence class. The difficult part is how to handle the change of the set memberships, e.g., if a user changes from one group to another. Here, the set-abstraction method defines a mechanism to reason about how facts about one class imply facts about a new class, e.g., roughly, a dishonest user would not delete any information that it has learned in the old group, but he cannot read any new information that the old group produces. Thus, revocation of facts can be modeled without the need to directly express sequencing in time.

4.3 Validation results

4.3.1 Guessing and Denial-of-Service Attacks

Two new kinds of attacks, resource exhaustion DoS and guessing, not previously supported by the platform, can now be addressed by adding customized clauses and rules. Guessing attacks are relevant in particular because users

tend to choose weak passwords, and other values such as PIN codes have intrinsically low entropy. These can became the weakest link in more complex protocols, leading further to other attacks. Resource exhaustion is relevant as a common source for denial of service as well as from an economic point of view, to rule out protocol designs that can be exploited to make honest participants spend unreasonable amounts of resources, time or memory.

Modelling of DoS attacks is a recent development, during the last months of the project [29]. It relies on a procedure that augments protocol transitions with costs, as in the framework of Meadows [33]. Costs can be tracked both per principal and per session instance, allowing the detection of excessive use which merely exhausts resources, as well as malicious use, which in addition differs from normal protocol execution. Moreover, we can characterize whether any of these attacks is undetectable by protocol participants. This procedure has allowed us to validate several known protocols (such as STS and JFK) that were previously subject only to manual analysis. Augmenting the protocol is currently manual, but can be implemented as a script.

For guessing attacks, we have developed guessing rules [28], expressed as Horn clauses, which can be easily inserted in any protocol specification. The rules are optimized for the CL-Atse backend, which can evaluate Horn Clauses using a backward strategy both with depth-first and breadth-first search. The first option has proved to be fast in finding attacks on several protocol specifications, requiring only small amounts of memory. Additional intruder abilities can be modeled by adding customized intruder transitions, at an increase in verification cost. Our method can detect both off-line and on-line attacks, identify undetectable attacks, and guess multiple secrets.

Using these rules, we have confirmed previously known attacks on some protocols (MS-CHAP, NTLM) and found new attacks on protocols both from the literature and from practice (a modification of a bank ATM protocol).

4.3.2 SAML SSO

A highlight of the effectiveness of the AVANTSSAR methods and tools is the detection of a serious flaw in the SAML-based SSO solution for Google Apps [4]. Though well specified and thoroughly documented, the OASIS SAML security standard is written in natural language that is often subject to interpretation. Since the many configuration options, profiles, protocols, bindings, exceptions, and recommendations are laid out in different, interconnected documents, it is hard to establish which message fields are mandatory in a given profile and which are not. Moreover, SAML-based solution providers have internal requirements that may result in small deviations from the standard. For instance, internal requirements (or DoS

considerations) may lead the service provider to avoid checking the match between the ID field in the AuthResp and in the previously sent AuthReq. The consequences of such a choice must be examined in detail. The technical overview document provided by OASIS SAML as a non-official addendum increases the clarity in this respect. Still, when Google developed their SAML-based SSO solution for Google Apps they released a flawed product, which allowed a dishonest service provider to impersonate the victim user on Google Apps, granting unauthorized access to private data and services (email, docs, etc.). The vulnerability was detected by the SATMC backend of the AVANTSSAR Platform and the attack was reproduced in an actual deployment of SAML-based SSO for Google Apps. Google and the US Computer Emergency Readiness Team (US-CERT) were informed and the vulnerability was kept confidential until Google developed a new version of the authentication service and Google's customers updated their applications accordingly. The severity of the vulnerability has been rated High in a note issued by the National Institute of Standard and Technology (NIST).

Moreover, as shown in [2], the prototypical SAML SSO use case (as described in the SAML Technical Overview) suffers from an authentication flaw that, under some conditions, allows a malicious service provider to hijack a client authentication attempt and force the latter to access a resource without its consent or intention. It also allows an attacker to launch Cross-Site Scripting (XSS) and Cross-Site Request Forgery Attacks (XSRF). This last type of attack is even more pernicious than classic XSRF, because XSRF require the client to have an active session with the service provider, whereas in this case, the session is created automatically hijacking the client's authentication attempt. This may have serious consequences, as witnessed by the new XSS attack that we identified in the SAML-based SSO for Google Apps and that could have allowed a malicious web server to impersonate a user on any Google application. In our paper, we describe solutions that can be used to mitigate and even solve the problem. These possible solutions are being discussed with OASIS.

4.4 Industry migration

Formal validation of trust and security will become a reality in the Internet of Services only if and when the available technologies will have migrated to industry and to standardization bodies (which are mostly driven by industry and influence the future of industrial development). Such an industry migration has to face the gap between advanced *formal methods (FM)* techniques and their real exploitation within industry and standardisation bodies.

To ease their adoption of formal methods, several obstacles have to be

overcome, e.g.: *(i)* the lack of automated FM technology, *(ii)* the gap between the problem case that needs to be solved in industry and the abstract specification provided by FM, and *(iii)* the differences between formal languages and models and those used in industrial design and development environments (e.g., BPMN, Java, ABAP).

The AVANTSSAR project has addressed these issues, in particular, by devising industrially suited specification languages (model-driven languages), equipped with easy-to-use GUIs and translators to and from the core formal models, and migrating them to the selected development environments. This enables designers and developers from industry and standardization bodies to more rapidly check the correctness of the proposed solutions without having a strong mathematical background.

A concrete example is the industry migration of the AVANTSSAR Platform to the SAP environment. Two valuable migration activities have been carried out by building contacts with core business units. First, in the trail of the successful analysis of Google's SAML-based SSO, the AVANTSSAR Platform has been exploited to formally validate relevant scenarios where the SAP NetWeaver SAML Next Generation Single Sign On services (NW NG SSO) are employed. More than 50 formal specifications capturing these scenarios, the variety of configuration options, and SAP internal design and implementation choices have been specified. Unsafe service compositions and configurations have been detected, and safe compositions and configurations have been put forward for use by SAP in setting up the NW NG SSO services on customer production systems.

The AVANTSSAR technology has been also integrated via a plug-in into the SAP NetWeaver BPM (NW BPM) product to formally validate security-critical aspects of business processes. This provides a push-button technology with accessible user interfaces, bridging the gap between business process modeling languages and formal specifications. Thus, a business process modeler can easily specify the security goals to validate; any violation of the security properties is depicted in a graphical way, enabling the modeler to take counter-measures.

More details on the migration of AVANTSSAR results to industry and standardisation organizations can be found in [12, 13].

References

[1] A. Armando, R. Carbone, and L. Compagna. LTL Model Checking for Security Protocols. In *Journal of Applied Non-Classical Logics*, pages 403–429, 2009.

[2] A. Armando, R. Carbone, L. Compagna, J. Cuéllar, G. Pellegrino, and A. Sorniotti. From multiple credentials to browser-based single sign-on: Are we more secure? In *Proceedings of IFIP SEC 2011*, 2011. To appear.

[3] A. Armando, R. Carbone, L. Compagna, J. Cuéllar, and L. Tobarra Abad. Formal Analysis of SAML 2.0 Web Browser Single Sign-On: Breaking the SAML-based Single Sign-On for Google Apps. In *Proceedings of the 6th ACM Workshop on Formal Methods in Security Engineering (FMSE 2008)*. ACM Press, 2008.

[4] A. Armando, R. Carbone, L. Compagna, J. Cuéllar, and L. Tobarra Abad. Formal Analysis of SAML 2.0 Web Browser Single Sign-On: Breaking the SAML-based Single Sign-On for Google Apps. In *Proceedings of 6th FMSE*. ACM Press, 2008.

[5] V. Atluri, C. Meadows, and A. Juels, editors. *Proceedings of the 12th ACM Conference on Computer and Communications Security, CCS 2005, Alexandria, VA, USA, November 7-11, 2005*. ACM, 2005.

[6] AVANTSSAR. Deliverable 5.1: Problem cases and their trust and security requirements. Available at http://www.avantssar.eu, 2008.

[7] AVANTSSAR. Deliverable 2.2: ASLan v.2 with static service and policy composition. Available at http://www.avantssar.eu, 2009.

[8] AVANTSSAR. Deliverable 3.4: Abstraction and Compositional Reasoning Techniques for Service Analysis. Available at http://www.avantssar.eu, 2010.

[9] AVANTSSAR. Deliverable 4.2: AVANTSSAR Validation Platform v.2. Available at http://www.avantssar.eu, 2010.

[10] AVANTSSAR. Deliverable 5.2: Formalized problem cases. Available at http://www.avantssar.eu, 2010.

[11] AVANTSSAR. Deliverable 5.3: AVANTSSAR Library of validated problem cases. Available at http://www.avantssar.eu, 2010.

[12] AVANTSSAR. Deliverable 6.2.3: Migration to industrial development environments: lessons learned and best practices. Available at http://www.avantssar.eu, 2010.

[13] AVANTSSAR. Deliverable 6.3: Migration to standardisation bodies. Available at http://www.avantssar.eu, 2010.

[14] M. Baudet. Deciding security of protocols against off-line guessing attacks. In Atluri et al. [5], pages 16–25.

[15] K. Bhargavan, C. Fournet, and A. D. Gordon. Verified Reference Implementations of WS-Security Protocols. In *Proceedings of 3rd WS-FM*, LNCS 4184, pages 88–106. Springer, 2006.

[16] K. Bhargavan, C. Fournet, A. D. Gordon, and R. Pucella. Tulafale: A security tool for web services. In *Proceedings of 2nd FMCO*, LNCS 3188, pages 197–222. Springer, 2003.

[17] B. Blanchet. An efficient cryptographic protocol verifier based on prolog rules. In *Proceedings of CSFW'01*, pages 82–96. IEEE Computer Society Press, 2001.

[18] C. Bodei, M. Buchholtz, P. Degano, F. Nielson, and H. R. Nielson. Static validation of security protocols. *Journal of Computer Security*, 13(3):347–390, 2005.

[19] Y. Boichut, P.-C. Héam, and O. Kouchnarenko. TA4SP, 2004.

[20] V. Cheval, H. Comon-Lundh, and S. Delaune. Automating security analysis: symbolic equivalence of constraint systems. In *IJCAR'10*, Lecture Notes in Artificial Intelligence. Springer-Verlag, 2010. To appear.

[21] Y. Chevalier, M. A. Mekki, and M. Rusinowitch. Automatic Composition of Services with Security Policies. In *Proceedings of Web Service Composition and Adaptation Workshop (held in conjunction with SCC/SERVICES-2008)*, pages 529–537. IEEE CS Press, 2008.

[22] S. Ciobâca and V. Cortier. Protocol composition for arbitrary primitives. In *Proceedings of CSF*, pages 322–336, 2010.

[23] H. Comon-Lundh and V. Cortier. New decidability results for fragments of first-order logic and application to cryptographic protocols. Technical Report LSV-03-3, Laboratoire Specification and Verification, ENS de Cachan, France, 2003.

[24] V. Cortier and S. Delaune. Safely composing security protocols. *Formal Methods in System Design*, 34(1):1–36, 2009.

[25] A. Datta, A. Derek, J. Mitchell, and D. Pavlovic. Secure protocol composition. In *Proceedings of 19th MFPS*, ENTCS 83. Elsevier Science, 2004.

[26] Google. Web-based reference implementation of SAML-based SSO for Google Apps. http://code.google.com/apis/apps/sso/saml_reference_implementation_web.html, 2008.

[27] T. Gross and S. Mödersheim. Vertical Composition of Protocols. submitted, 2010.

[28] B. Groza and M. Minea. A formal approach for automated reasoning about off-line and undetectable on-line guessing. In *Proceedings of the* 14^{th} *International Conference on Financial Cryptography and Data Security*, volume 6052 of *LNCS*, pages 391–399. Springer, 2010.

[29] B. Groza and M. Minea. Formal modelling and automatic detection of resource exhaustion attacks. In *Proceedings of the* 6^{th} *ACM Symposium on Information, Computer and Communications Security (ASIACCS)*, 2011.

[30] J. D. Guttman, F. J. Thayer, J. A. Carlson, J. C. Herzog, J. D. Ramsdell, and B. T. Sniffen. Trust management in strand spaces: A rely-guarantee method. In *In Proc. of the European Symposium on Programming (ESOP '04), LNCS*, pages 325–339. Springer-Verlag, 2004.

[31] R. Lucchi and M. Mazzara. A pi-calculus based semantics for WS-BPEL. *J. Log. Algebr. Program.*, 70(1):96–118, 2007.

[32] A. Marconi and M. Pistore. Synthesis and composition of web services. In *Proceedings of Formal Methods for Web Services*, pages 89–157. Springer-Verlag, 2009.

[33] C. Meadows, P. Syverson, and I. Cervesato. Formalizing gdoi group key management requirements in npatrl. In *In Proceedings of the ACM Conference on Computer and Communications Security. ACM*, pages 235–244. ACM Press, 2001.

[34] S. Mödersheim. Abstraction by Set-Membership—Verifying Security Protocols and Web Services with Databases. In *Proceedings of 17th CCS*. ACM Press, 2010.

[35] S. Mödersheim and L. Viganò. Secure Pseudonymous Channels. In *Proceedings of Esorics'09*, LNCS 5789, pages 337–354. Springer-Verlag, 2009.

[36] S. Mödersheim and L. Viganò. Channels as Assumptions, Channels as Goals, 2010. Submitted journal paper.

[37] OASIS. Profiles for the OASIS Security Assertion Markup Language (SAML) V2.0. Available at http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=security, March 2005.

[38] OASIS. Security Assertion Markup Language (SAML) v2.0. Available at http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=security, April 2005.

[39] Oasis Consortium. Web Services Business Process Execution Language Version 2.0. http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.pdf, 11 April, 2007.

[40] C. Weidenbach. Towards an Automatic Analysis of Security Protocols. In *Proceedings of CADE'99*, LNCS 1632, pages 378–382. Springer-Verlag, 1999.