Deliverable

D2.1

Methodologies and tools for data sharing agreement infrastructure

WP2 Data Sharing Agreement Infrastructure

December 2008
Version 1.0

Consequence

Context-aware data-centric information sharing

FP7-ICT-2007-1
ICT-2007.1.4. Secure, dependable and trusted Infrastructures
Grant Agreement 214859

LEGAL NOTICE

Hewlett-Packard Italiana,

Imperial College of Science, Technology and Medicine,

The Science and Technology Facilities Council,

Consiglio Nazionale delle Ricerche,

Centre for Research and Telecommunication for Networked Communities.

Project acronym: Consequence

Project full title: *Context-aware data-centric information sharing*

Work Package: **2**

Document title:	**Methodologies and tools for data sharing agreements infrastructure**
Version:	**1.0**
Official delivery date:	**31 December 2008**
Actual publication date:	19 December 2008
Type of document:	Report
Nature:	Public

Authors: List of authors: Fabio Martinelli, Marinella Petrocchi (CNR), Ilaria Matteucci (CN), Alexey Orlov, Dmitry Starostin (EMIC), Marco Luca Sbodio (HP), Alvaro Arenas, Benjamin Aziz, Shirley Crompton, Michael Wilson (STFC).

Approved by: David Golby (BAES), Vaibhav Gowadia (Imperial)

Version	Date	Sections Affected
0.1	October 2008	Initial draft with first contributions for requirements and state of art sections.
0.2	November 2008	Requirements and state of the art sections are worked out; the proposed approach is pretty ready.
1.0	December 2008	Final delivery of D2.1.

Table of contents

1. Introduction

This document describes the components, the methodologies and the tools underlying the Data Sharing Agreement (DSA) infrastructure used in the Consequence project.

DSAs describe the regulations for managing shared data among multiple participants in several specific domains and contexts. The DSA infrastructure is thus devoted to all the operations underpinning these agreements such as creation, (cooperative) authoring/editing, analysis, manipulation and translation. We envisage in particular that the DSA infrastructure should be at least able to take natural language description of DSAs, provide tools for collaborative editing and then provide mechanisms to translate them into formal descriptions amenable of rigorous analysis and prediction tools. Eventually, we need also components for translating such formal descriptions into directly enforceable policies at system level so as to be appropriately managed in other components being developed for the Consequence framework.

On this basis, we have identified a set of functional components for managing DSAs. These are, in particular:

- DSA Authoring. This component is specifically devoted to the (possibly collaborative) editing of DSA. In particular, it embodies the refinement step from natural language to high level formal specification of DSA. Strictly related to the DSA Authoring, the DSA Vocabulary provides context-specific information.
- DSA Analysis. This component is devoted to the analysis of the DSA agreement when specified in a high level formal language.
- DSA to Policy Mapper. This component performs the mapping from the formal DSA description to the enforceable policies to be deployed on the running systems.
- DSA Lifecycle Manager. This component is developed to manage all the steps involved during the DSA lifecycle.
- DSA Trust Manager. This component deals with the distributed management of the trust relationships among principals operating on DSAs during their lifecycle.

The framework is intended to be modular so that it can be adapted during the project period.

This document describes the requirements for these components (Section 2), surveys the existing methodologies and tools for each of them (Section 3), and presents the modelling approach for the Consequence project in Section 4.

Before illustrating the components requirements, we give the general definition and characterizing features of data sharing agreements.

1.1. Data sharing agreements

A Data Sharing Agreement is an enforceable agreement that documents obligations and permissions of, and prohibitions on, collaborating parties when sharing data. The collection of obligations, permissions and prohibitions is usually captured in the clauses of the DSA. DSAs are agreements among distributed entities, speaking about data and representing data policies being communicated among different parties.

DSAs are widely used within government agencies who share data about individuals to protect the privacy of citizens. With the recent increase in corporate data governance, they are now being adopted more widely between commercial companies who share data in order to both protect the privacy of individuals, but also to protect commercially confidential data. In the past the content of DSA has been included in general collaboration agreements. However with the rise of data governance, many organisations have established general data policies which are either referred to in collaboration agreements or explicit detailed DSAs are entered into for individual projects which operate within the framework of those corporate data policies, but which include more detailed specific terms and conditions.

The structure of data sharing agreements generally indicates the issues addressed by them. The policies are either explicitly stated or included by reference to other organisational data policies. Such policies are drafted in a limited controlled vocabulary, consistently using the constrained natural language syntax preferred by lawyers to avoid ambiguity or misinterpretation.

From an analysis of a set of available agreements, reported in the Appendix, we outline below the general template structure of agreements and the twenty issues addressed – individual agreements may not address all these issues, but only the appropriate subset.

1. Definitions: Terms used in the agreement such as personal data and computing machinery are defined.

2. Parties to the agreement: Definition of the parties making the agreement.

3. Purpose of the agreement: The purpose of the required data sharing in layman's language.

4. Period of agreement: The time period during which the data sharing agreement shall apply.

5. Justification for access: Details of why general data sharing or data confidentiality principles (depending on the culture of the organizations involved) are being breached by this agreement.

6. Description of the data: Detailed information on which data sets will be cover by the DSA.

7. Data Quality: It should make a statement on the degree of commitment to data quality. It could say, for example how the parties are to maintain the quality of the data. Alternatively, it may make a statement that limits liability on the data being accurate, up to date, or reliable.

8. Description of the data users: Statement of who can access and use the shared data. This may be stated in terms of general roles, or specifically by naming individuals.

9. Method of data access or transfer: This section may contains specific details such as, information on the type of encryption to be used, a description of the trusted computing infrastructures that may be used, a description of the trust domains, requirements for physical security, strength of passwords and other security concerns related to data access and transfer.

10. Location of data and custodial responsibility: Description of who is responsible for the data and therefore responsible for the confidentiality requirement stated below.

11. Trustworthiness: This section constrains how much of the infrastructure (computing machinery, hardware, software, network channel etc.) or platform is trusted, e.g., whether a trusted computing platform must be used, users are not permitted to have administrative rights, whether certain hardware and software are to be used. It exports controls on the data.

12. Restrictions on data use: There are often restrictions stating whether the data is for non-commercial or research purposes only.

13. The position of the agreement with respect to derived data. It may describe who is responsible for the derived data and what policies are to be applied to them

14. Dissemination to third parties: These policies may be included in other sections, but address different classes of data and how data can be disseminated to third parties.

15. Confidentiality: The user agrees to establish appropriate administrative, technical and physical safeguards to ensure the confidentiality of the data and to prevent unauthorized use or access to it. The details in this section can be as general as the previous statement, or as detailed as possible depending on the style of agreement.

16. Disposition of data: The policy for disposing of data at the termination, or expiration, of the agreement.

17. Administration of the Agreement: Process for reviewing and updating the agreement – who, how often etc. Procedure for terminating, and perhaps renewing the agreement.

18. Breaches to the agreement: Define the procedure to record breaches and the action to be taken on breaches.

19. Applicable law: The law which applies to the agreement, or a clear statement that the agreement does not constitute a contract which is enforceable in any jurisdiction.

20. Signature: Authorized parties confirming the agreement.

In the Appendix, sample agreements are reproduced, highlighting parts where these aspects are illustrated.

2. Subsystem infrastructure requirements

Here, we give an overview of the DSA infrastructure components, and of the requirements each component should satisfy.

2.1. DSA Authoring

We distinguish three levels of agreements: (1) a *business level DSA*—a human-readable representation of the agreement that addresses main risks related to sharing data; (2) a *formal high-level DSA*— a representation of the agreement using a formal language, which is suitable for analysis and reasoning about the agreement; and (3) an *operational level DSA* – a representation of the agreement in an executable policy language, which enforces data access and usage according to the DSA clauses.

The *DSA Authoring* is a GUI tool to support the drafting of the business level DSA and to transform the business level representation into the formal one. It acts as a bridge between the non-technical business people in charge of writing agreements (lawyers, managers, etc.) and the technical personnel in charge of analysing and implementing the operational policies associated with the agreement clauses.

The DSA Authoring tool must support the following operations, which constitute the functional requirements of the tool.

Id	Requirement Description
Auth-R1	The tool must provide editing capabilities for building business-level DSAs. The editor will act as a User Interface (UI) for the tool, which must be intuitive enough for the end users
Auth-R2	The tool must provide a repository for common business-level DSA templates
Auth-R3	The tool must provide a domain-relevant DSA vocabulary browser, which allows the users to select specific terms and conditions when editing their DSAs
Auth-R4	The tool must provide a storage for all the working and final versions of the business-level DSAs. Basic persistence storage operations (creation, retrieve, update and delete) are supported
Auth-R5	The tool must be able to translate final DSAs from the business level to their representation in the formal language

In addition to these functional requirements, the tool must also be able to satisfy other, possibly non-functional, requirements shown in the table below.

Id	Requirement Description
Auth-R6	A party can have a different number of different DSAs with different parties at the same time
Auth-R7	DSAs are secure from authorised access; in particular they are not revealed to business competitors. So access to a DSA must be secure
Auth-R8	It is desired that the translation from business level DSA to the formal level DSA to be as automatic as possible. In any case, it should be transparent for the DSA

	creators
Auth-R9	Problems discovered during the formal analysis of a DSA are reported to the human actor responsible for creating the DSA
Auth-R10	When required, important decisions can be escalated to a human decision maker for ratification

2.2. DSA Analysis

This component is responsible for analysing a data sharing agreement with respect to some properties that it should satisfy. Properties may range from security properties to conflict detection. This component takes as input the high level description of the agreement, specified by a formal language specifically developed for distributed and concurrent systems, and the property to be analysed, possibly expressed as a logical formula. As output, the component says if the property holds, or not. If the property does not hold, the component may return the reason of the inconsistency of the property.

The analysis process will consist of reasoning about several properties of a DSA, and to make deductions on the state of one, or more, DSAs. E.g.: is there a reachable state where an agent can access a resource (simple safety)? In any reachable state, are there agents who can do a certain action bounded by a specific set of actions (containment)? In any reachable state, are two given properties mutually exclusive (mutual exclusion)? Are two agreements in conflict with each other (conflict detection)? As a particular case of analysis with respect to property of an agreement, evaluation will consist in determining if it is possible to determine if, with a certain DSA, in a certain state, one entity is authorized to perform a certain action.

As defined before, DSAs are agreements among distributed entities, speaking about data and representing data policies being communicated among different parties. Very frequently, DSAs include notions about temporal and location constraint. Several factors, both external, e.g., from the environment, and internal, e.g., related to the kind of data being shared, can dictate conditions whose fulfilment leads to permissions and obligations for the data management. Furthermore, DSAs could be based on trust relationships established between the entities at stake, as well as on the roles and attributes assumed by those entities. Also, concurrency plays a central role, since different agents communicate with each other on which events are allowed to happen, based on which conditions.

Thus, it appears natural, as input to the component, to choose a concurrent language that is able to formally express the agreement. The language will be based on formal semantics, it will be able to describe the communication of data in a distributed environment, and it will have the capability of dealing with notions of environmental context (e.g., time and location). It is advisable that the language will also be able to express (chains of) attributes (and constraints about them) of the object and the subject, like metadata related to a document (e.g., the kind of data) and the roles covered by the principals involved in the agreement. In particular, given the formal foundations of the language, this will have a formal methodology and a formal tool associated, for the evaluation and the analysis of the DSAs. The analysis will address questions possibly involving conditions on 1) environmental, 2) subject and 3) object factors. It will also define equivalences between the policies derived from the agreements, in terms of formal equivalences of processes. Outcomes resulting in breaches and conflicts with respect to a desired property (or group of properties) will guide the process for sanitizing the investigated DSA(s).

The DSA analysis component aims at guaranteeing the following main requisites.

Id	Requirement Description
An-R1	Input/output - The component will take as input a formal description of the agreement, in a high level language, and a property to be verified (expressed, e.g., as a logical formula). As output, the component will say if the property holds, possibly returning a counterexample for its inconsistency.
An-R2	Affecting factors - Both at modelling and analysis level, the component must be able to manage peculiarities of DSAs like, e.g., their concurrent nature.
An-R3	Formal semantics – The language adopted for the model, and the associated analysis methodology and tool, will be based on a rigorous, formal semantics.
An-R4	Relationships with other components. The component must be able to communicate with the *DSA Authoring* component, e.g., for signalling conflicts between DSAs, or the inconsistency of an expected property, in order for the authoring component to correct the DSA. It must also interact with the *DSA to Policy Mapper*, to ensure a consistent translation between the high level language and the enforceable language.
An-R5	Property patterns - In order to provide a user-friendly experience using the analysis tool, it is advisable that properties corresponding to specific analysis are classifiable through natural language patterns.

The above list of requirements is not intended to be exhaustive, as it can be adapted during the project period.

In the last few years several formal and semi-formal analysis methodologies have been developed, e.g., model checking, theorem proving and static analysis. We refer to the DSA Analysis State of the Art section for a short review of these methodologies, since they are suitable for the Consequence intents.

2.3. DSA to Policy Mapper

*DSA*s can be specified at many different levels of abstraction from high-level business goals (*high level* policies) to policies for individual resources (*low level (enforceable)* policies). This passage from the high level to the low level policies is referred as *policy refinement*.

Several definition of refinement can be found in literature. According to [JJ01], it is the process that decomposes the high level policy relevant to a composite system, in our case the global DSA policy that is the formal high level specification of the DSA, denoted by P_{DSA}, into a set of policies that are executed in its constituent parts to implement the behavior intended by the high level policy. Policy refinement is thus capable of mapping high level goals of a composite system automatically down to low level policies. A policy language satisfying refinement is a language that can be refined to some low level details.

The *DSA to Policy Mapper* implements the *refinement* of the global *DSA* policy specified at high-level of abstraction, into a combination of a number of enforceable local DSAs policies. Indeed the *DSA to Policy Mapper* works by taking as input the global *DSA* policy, P_{DSA}, that is the policy that describes *DSA* existing between the several parties involved into the considered system, and giving as output the *local DSA policies,* P^{i}_{DSA} for i=1,…,n where n is

the number of involved parties. A local *DSA* policy is associated to a party of the agreement and it is the policy that has to be respected locally by that party in order to guarantee the global DSA policy is satisfied. Each P^{i}_{DSA} is given at enforceable level.

We can distinguish the work of the *DSA to Policy Mapper* in two phases. In the first phase, the *projection phase*, we have a refinement from Multi-party to Party. It consists of a refinement of the global specification into a combination of a number of components. To guarantee the correctness of this step of refinement we have to identify what properties each component must satisfy in order to ensure that their combination satisfies the specification of the global policy being refined. In this second phase we have a *refinement* from Party to Party. Once a correct specification for each component is found, it is permitted to pass to the implementation phase. This is another refinement steps since we have to pass from a high level of abstraction (the specification level) to a lower one (that can be eventually enforced).

The crucial point is the identification of the particular local policies the components must satisfy in order that their combination satisfies the global specification.

The solution should preferably have the following qualities:

Id	Requirement Description
Map-R1	The Local *DSA* policies, *i.e.,* the local DSA policy of each agent of the system, should be as weak as possible in order to not restrict unnecessarily the choices of sub-implementations.
Map-R2	The Local *DSA* policy must be enforceable. This means that the global policy is translated into operational policies that the system can enforce.
Map-R3	The *DSA* policies, expressed at the enforceable level, have to satisfy the same security properties, once refined, that hold at the specification level.

In general we expect there to be several different possible solutions with the above qualities, as the properties required of one subcomponent clearly will influence what must be required of the other subcomponents.

2.4. DSA Lifecycle Manager

The DSA Lifecycle manager component supports the processes underpinning the lifecycle of Data Sharing Agreements (DSA).

A DSA is an agreement among contracting parties that regulate how they share data. A DSA has a complex lifecycle, which in the most general case comprehends the stages shown in the following diagram, Figure 1.

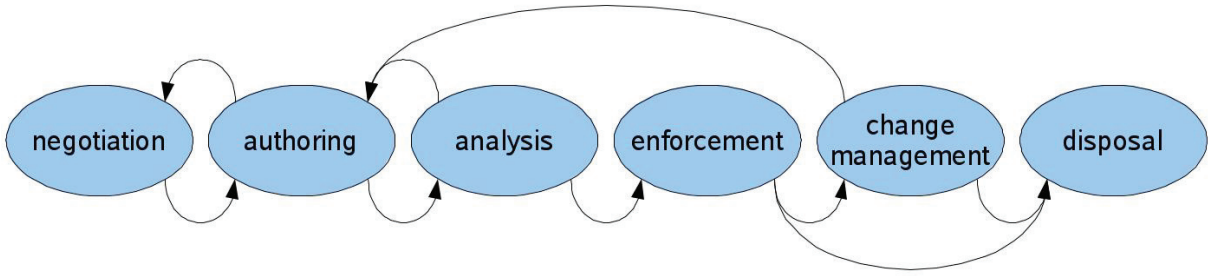

Figure 1. Stages of the DSA lifecycle.

The *negotiation* stage is a preliminary phase during which the contracting parties establish a relationship among them, and negotiate over their respective roles and the scope of the forthcoming DSA among them. The *authoring* stage is an editing phase during which the stakeholders prepare the DSA itself. The authoring is usually done using a *Template DSA*, which the stakeholders fill in with appropriate information. The result of the authoring stage is a human readable document, which is then translated into a formal language suitable for verification and formal check, which happens in the *analysis* stage. The first three stages of the lifecycle (namely negotiation, authoring and analysis) may be iterated several times:

- the iteration between negotiation and authoring is repeated until the contracting parties reach a shared understanding and a common agreement on the DSA;
- the iteration between the authoring and analysis stage is repeated until the formal representation of the DSA satisfy all the required properties being checked in the analysis stage.

The *enforcement* stage is the phase in which the DSA is enacted. Note that the transition from the analysis stage to the enforcement stage involves the successful translation of the DSA to a set of enforceable policies. The enforcement stage is the longest phase, potentially of the stated duration of the agreement, unless the DSA needs to be modified or disposed of. A DSA enters the *change management* stage of its lifecycle when the contracting parties need to change some parts of it. Changes to a DSA require a reiteration of the authoring and analysis stages until a new version of the DSA is ready for enforcement. Finally a DSA enters the *disposal* stage when the contracting parties agree that such DSA is no longer necessary or must be dropped. The disposal stage must also take care of removing all the enforceable policies that have been originated by the DSA being disposed of.

The DSA Lifecycle Manager acts as a document repository for DSAs, and it also supports the various stages of the DSA lifecycle. The Consequence project will not deal with all the stages of the complete lifecycle of a DSA. In particular, the DSA Lifecycle Manager will handle the evolution of the DSA lifecycle through the authoring, analysis and enforcement stages, by interacting respectively with the *DSA Authoring*, *DSA Analysis*, and *DSA to Policy Mapper* components. The following diagram shows the workflow underlying the supported stages of the DSA lifecycle, Figure 2.

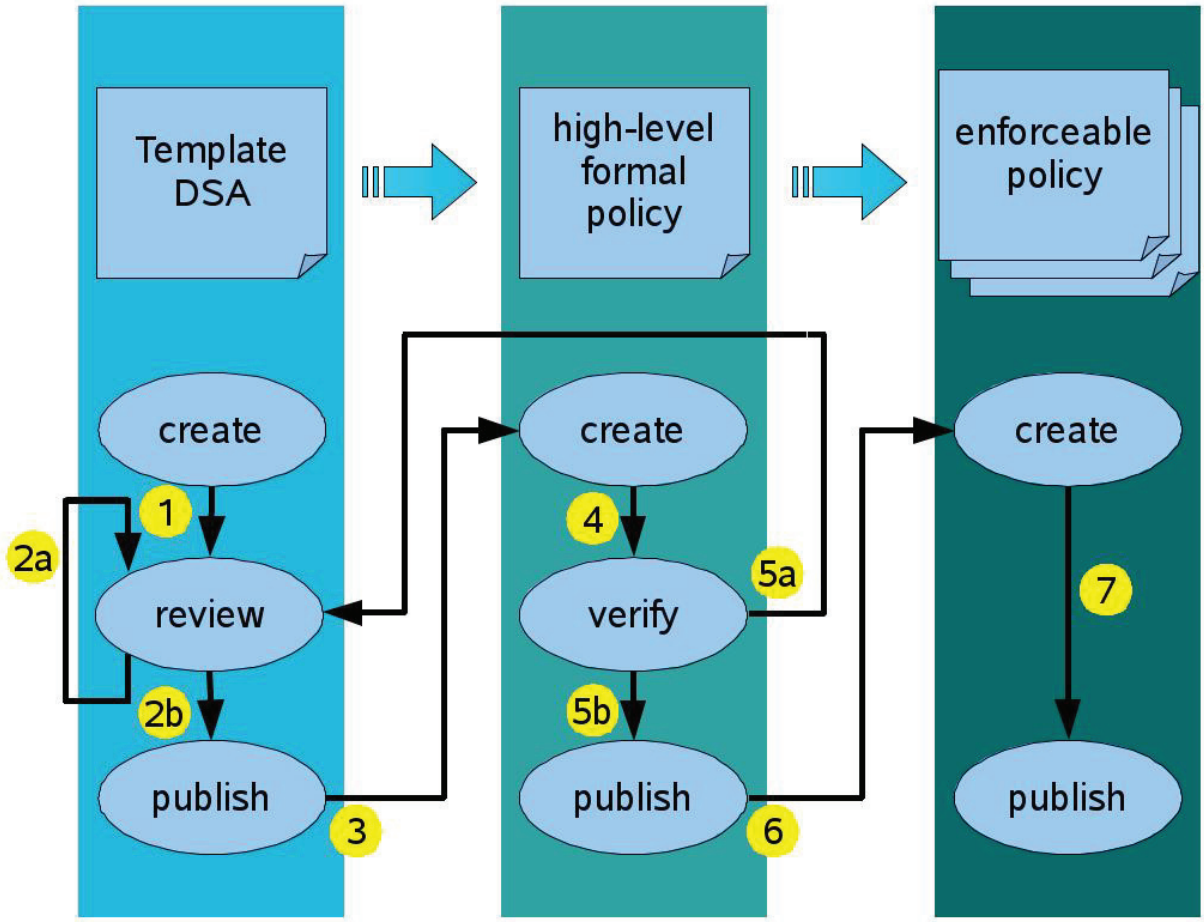

Figure 2. Workflow.

The authoring stage of the DSA lifecycle comprehends the steps from 1 to 3: the stakeholders edit and review a Template DSA; the review step is repeated (step 2a) until the Template DSA is fully filled in, and then it is published (step 2b). The publishing step triggers the translation of the DSA into a formal policy written in a high-level language.

The analysis stage of the DSA lifecycle comprehends the step from 4 to 6. The high-level formal policy is verified by the DSA Analysis component: if the verification step identifies some problems, then the process goes back (5a) to the review step, otherwise it continues the publishing of the high-level policy (step 5b). Finally the high-level policy is refined into a set of enforceable policies (step 6), which are then dispatched for enactment (step 7).

The DSA Lifecycle component supports the collaborative and iterative process that goes from the creation of the natural language DSA to its decomposition into enforceable policies. The DSA Lifecycle is essentially a document management system integrated with the other components of the Consequence platform, and customized in order to effectively enact the workflow underpinning the creation of a DSA and its subsequent translations in formal and enforceable languages.

The following diagrams show the interaction among the DSA Lifecycle Manager and the other components (Figure 3); for each stage of the lifecycle the diagram shows also the actors involved.

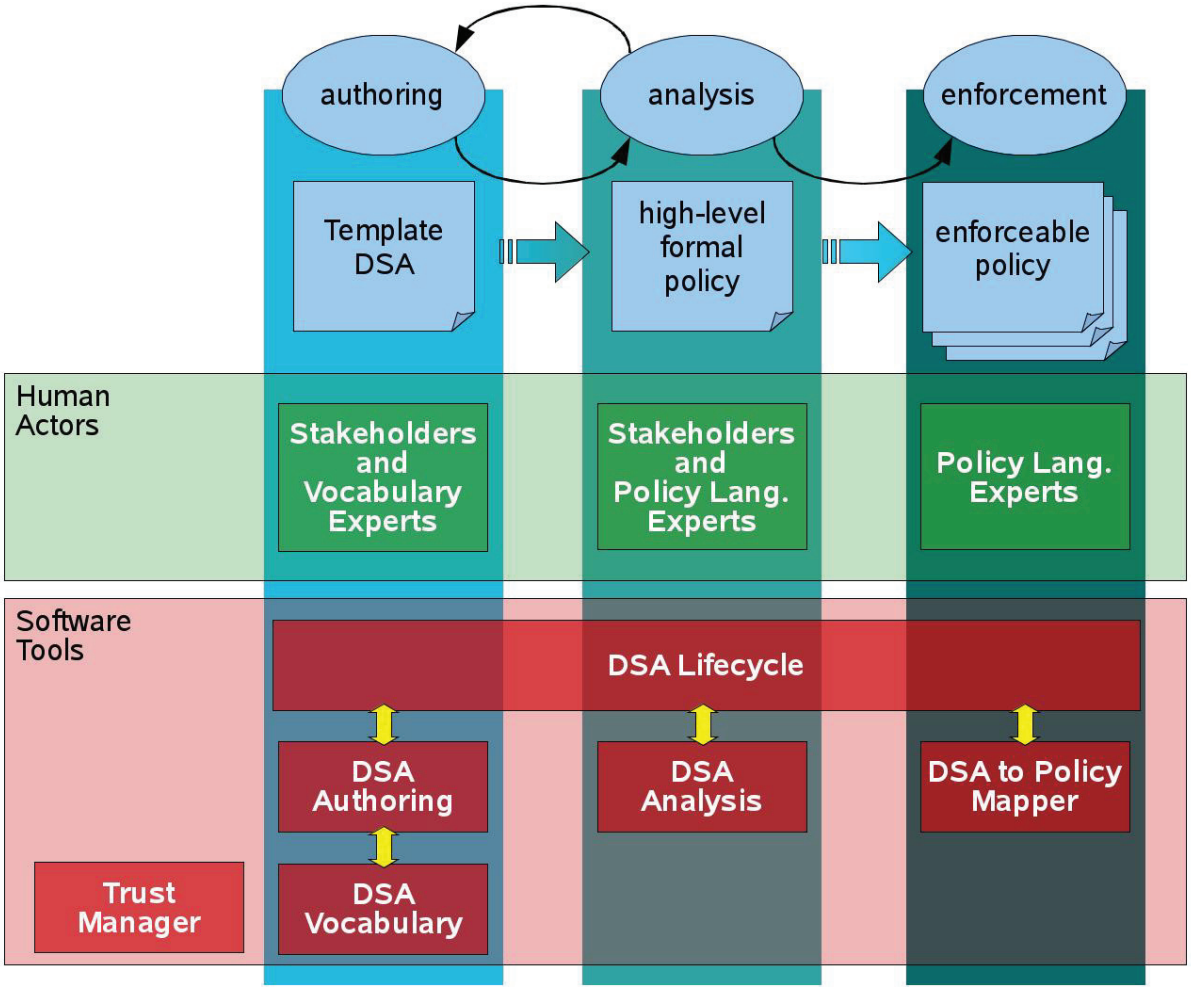

Figure 3. The interaction among the DSA Lifecycle Manager and the other components.

The DSA Lifecycle must satisfy the following high-level requirements:

Requirement ID	Requirement Description
ReqDSAL01	Document repository. The DSA Lifecycle must act as a document repository, and must provide all traditional features of such software. A (non exhaustive) list of features comprehends: access control based on user roles, managing of users' and shared workspaces, possibility of organizing workspaces in hierarchical structures, versioning of documents, handling of documents metadata.

Requirement ID	Requirement Description
ReqDSAL02	Web based interface. Users are able to perform all operations from a web interface. No software installation is required on user's clients. The web interface is also required in order to potentially allow users belonging to different organization to cooperate in the creation of a DSA (for example attorneys from the various contracting parties may need to access the same document repository in order to cooperate in the creation of a DSA).
ReqDSAL03	Document workflow capabilities. The DSA Lifecycle manager must enable the definition and enactment of document workflows, where a document evolves through a number of steps. At each step human intervention may be required (for example, editing or approving the document status). The workflow capabilities must be customizable, in order to meet the exact DSA lifecycle that the Consequence project will define.
ReqDSAL04	Extensibility. The DSA Lifecycle manager must integrate with external tools, such as the DSA Analysis component, or tools that may support the (semi)-automatic conversion from a natural language DSA to a controlled language DSA to a high-level formal policy and to the embeddable policies.

The above list of requirements is not intended to be exhaustive: the high-level requirements will be progressively refined taking into account also the inputs coming from the Consequence test-beds.

2.5. DSA Trust Manager

This component is responsible for managing, at a high level, the trust relationships among the various parties throughout the lifecycle of the agreement. We are mainly interested, at this stage, on the trust relationships among entities creating, exchanging, operating on, and managing DSAs. On the contrary, we do not dwell on the definition and management of roles and attributes that can appear in an agreement.

Actors (individuals and organizations) that create, manage, and operate on DSAs constitute a so called virtual community. This is a collection of entities (digital or physical), interacting in a cooperative manner on the DSAs. The community may be dynamic, as its members can leave and join, and mobile, as its members may change their business goals.

The main goal of our DSA Trust Manager is to define a distributed trust management defining credentials through roles and attributes, possibly parameterised (ReqTrust01). As an example of application in this context, such a credential could tell who, within the virtual community, has the correct rights for editing, modifying, signing, making operational, etc., the agreement.

We intend to rely on a role-based framework, allowing for the establishment of trust in distributed systems. The starting point will be RT, the role-based trust management framework [LMW02, WM03]. This framework allows for the definition of simple and compound credentials. An example is: A.r(p) ← D meaning that A assigns to D the role (or attribute) r, with parameter p. More concretely, in the following credential, organization CNR assigns the role of CNR researcher to Marinella, with ID number 1096: CNR.researcher ("ID=1096") ← Marinella. In Section 4.5, an application example will be given, showing how to infer permissions for operations on DSAs.

3. State of the Art

We describe here the state of the art of available models and technologies, with respect to all the planned components of the DSA Infrastructure.

3.1. DSA Authoring

DSA authoring may benefit from the use of ontology editors and browsers in that these can assist in developing the DSA ontology underlying a DSA and thus be part of the DSA authoring process satisfying the Auth-R3 requirement. One of the most popular ontology editors is Protégé [Protégé],which is free and open source software. Protégé supports both frame-based domain ontologies in which concepts are defined as a subsumption hierarchy of classes with properties and relationships, as well as Web ontology languages such as OWL. Figure 4 illustrates a screenshot from Protégé.

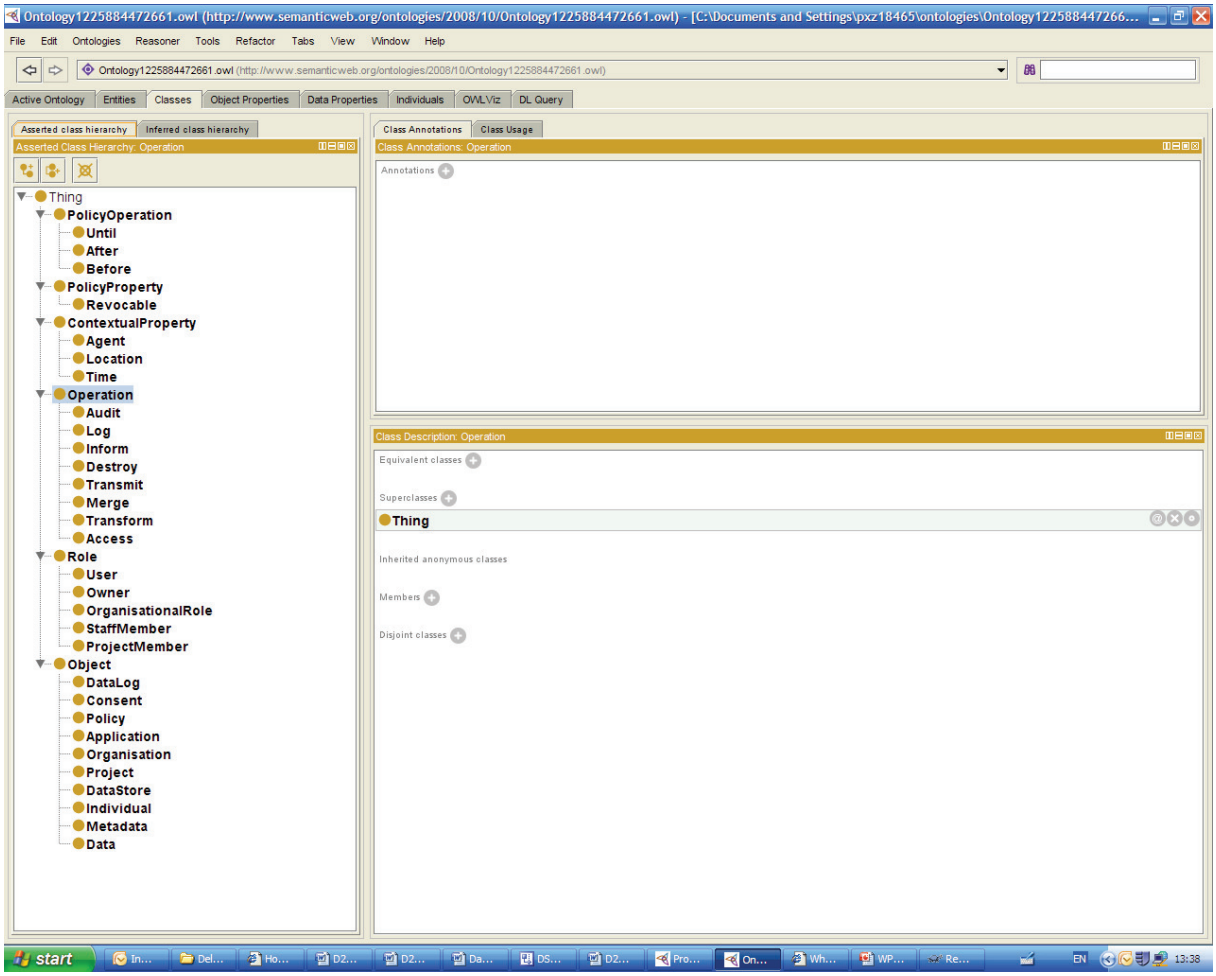

Figure 4. A screenshot from Protégé.

In the area of controlled natural language translation into formal languages, there are two relevant projects. First, there is the Nordic project COSoDIS [COSoDIS], which adopts the strategy of writing contracts directly in a formal specification language called *CL*. The modelling of natural English-based DSA clauses into CL is done based on patterns. Once this is done, a translation function is defined that maps the contract into a variant of the mu-

calculus after which formal reasoning can be applied by way of the NuSMV model checking tool. The second approach is based on project SecPAL [SecPAL]. Here, a grammar of the controlled language of DSA clauses is created and then translated into the formal language using lex/yacc parsing/compiling tools [Parsing].

3.2. DSA Analysis

There has been an evolution in the areas of policy languages, policy analysis and policy evaluation research areas prompted by recent technological trends. From intuitive reasoning alone, it is obvious that basic identity-based access control mechanisms within decentralized environment cannot be completely effective since, *e.g.*, entities may be unknown to each other.

Thus, from traditional access control mechanisms, security research has progressed to arrive at such novel concepts such as, e.g., trust negotiation mechanisms. Along the way the concept of role-based access control was proposed where a peer is assigned to one or more roles and which in turn is exploited in order to take authorization decisions.

Therefore, in recent years many frameworks have been proposed, targeting different application scenarios and provided with different features and expressiveness.

Here, we review languages suitable for representing data sharing agreements, and analysis methodologies built around some of these languages.

From a number of particular concerns, we focus on the ability to deal with specific constraints, like duration validity, limited number of accesses, location, and inheritance of permissions, reputation and trust levels, all within a concurrent environment.

Before introducing specific models and technologies, we briefly review popular analysis methodologies that have been proved to be successful over the last years, and widely used within both academia and industry.

Model Checking

Checking whether a given model satisfies a given logical formula is possible through so called *model checking* techniques. At conceptual level, these techniques apply to several kinds of models and logics. Thus, a very general model checking problem can be formulated as follows: model checking consists in testing if a given formula, expressed in a given (temporal) logic, is satisfied by a given model, specified in some formal language.

Model checking tools are faced with the state explosion problem. Indeed, the logic formula is checked by an exhaustive search of the state space of the model. Since the size of the state space grows exponentially with the number of processes in the model, model checking techniques based on explicit state enumeration can only handle relatively small examples. However, there are several approaches to cope with the state explosion problem, such as *abstraction*, that tests properties on a sort of simplified system (*e.g.*, in a program, not to consider any sort of variables, but only variables of some kind), or *partial order reduction*, that is used to reduce the number of independent interleaving of concurrent processes that need to be considered. For example, given the processes A and B, it is unnecessary to consider both AB and BA interleaving when this is not necessary for testing a certain goal.

Initially developed to reason about correctness of discrete state systems, model checking techniques have been extensively used also for the verification of real time and hybrid systems. Also, particular attention has been paid to the analysis of security protocols and security policies using model checking techniques. The reader is referred to e.g., AVISPA, [AVISPA], and [FG97, FGM00, Low96, Mar05, MCJ97, Mea95, RRA02, ZRG05] for examples of this area of research.

We will focus on model checking for our proposed DSA analysis approach.

Theorem Proving

Broadly speaking, automated theorem proving (ATP) deals with the development of computer programs that show that some statement, the so called conjecture, is a logical consequence of a set of axioms or hypotheses. ATP systems are used in a wide variety of domains, e.g., mathematics, management, electronics. Usually the language of first order logic is used to write conjecture, axioms and hypotheses; however, ATP may also use higher order logics.
The underlying strength of an ATP system is its formality, which prevents any ambiguity; such strength comes from the use of formal methods.
An ATP system provides the proofs showing how a conjecture follows by the axioms and hypotheses, in a way that can be understood and agreed upon by everyone. For instance, considering the famous Rubik's cube problem and how to arrive to the solution state, the proof would describe the sequence of moves needed to solve the puzzle. ATP may also be applied to manage infinite state systems. ATP is a technique widely used among the formal methods community for the verification of security properties and the amount of literature dedicated to this topic is broad.

Well-known and widely used theorem provers are the Isabelle/HOL toolkit [Isabelle], Coq [Coq], ProVerif [Bla02, Bla04], Athena [SBP01], and Prover9 [Prover9]
To cite some of the work related to policies, [Una06] uses Coq for checking that an authorization security policy is conflict-free, both initially and as it evolves by the addition and removal of authorisation rules. In [Dro04], Drouineaud et al. use first order linear temporal logic embedded within the Isabelle/HOL theorem prover to formalise and verify Role Based Access Control (RBAC) authorisations constraints.

Static Analysis

Static analysers are tools that are used at compilation stage to verify certain properties of a model of the system under analysis. These tools are usually fully automatic and therefore, unlike theorem provers, they require little human interaction. Their efficiency is due to the fact that the analysis has polynomial complexity with respect to the specification. On the other hand, the set of properties that can be analyzed is generally smaller compared to other techniques. Static analysis tools statically examine the text of a program, without attempting to execute it. Theoretically, they can examine either a program's source code or a compiled form of the program [CMG04]. Static analysers may discover a great many of security flaws, like buffer overruns, access problems, exception handling, string format problems, and many others, [CW07]. Available tools are either commercial, e.g., Coverity [Coverity], PreFast [PreFast], and PolySpace [PolySpace], or academic, e.g., [BBD05, BDP05, Bla03].

Models

In this subsection, we mainly review research-oriented models and methodologies for describing and analysing DSAs.

Specifying Data Sharing Agreements

Work in [SSR06a, SSR06b] is in the area of specifying data sharing agreements. This work provides models for DSAs that encode data sharing obligations (such as "you must do something if certain conditions are verified"). The obligations are expressed as formulas of Distributed Temporal Logic DTL [EC00].

In the proposed model, DSAs are represented as predicates expressed over a graph where the nodes are principals (individuals or organizations) with local stores, and the edges are channels over which data is communicated.

As simple examples, let us consider *A*, *B*, and *C* range over principals, let c_i be a communication channel, then the logic expresses agreements including, but not limited to, the following:

- *Responsive forwarding*: *B* will send to *C* on channel c_2 each object it receives from *A* on channel c_1 within 24 hours of receiving it;
- *Nondisclosure agreements*: If *B* receives object *o* from *A* on channel c_1, then *B* will not thereafter send *o* to any other principal for a year.
- *Privacy*: *A* must delete all personal information about *B* from local store within one year of *A*'s storing it.

The language does not have a precise formal semantics. It is the authors' opinion, however, that the language can be enriched with formal semantics. This means that a DSA specified with this language may be subject to a variety of automated analysis, whose tractability will depend on the properties of the underlying data model and Distributed Temporal Logic.

Klaim and μKlaim

The Klaim family of process calculi (Kernel Language for agents Interaction and Mobility), see, *e.g.,* [BBDF03, DFP00, NFP98] provide a high-level model for distributed systems, and it serves as a good foundation for studying different notions of security in such systems, see, *e.g.*, [DFPV00, HPN06].

The calculus is designed around the tuple space paradigm [Gel85]. A system consists of a network of distributed, addressable nodes that contain processes and data repositories (tuples). The nodes interact and communicate by asynchronously sending and receiving tuples. Remote evaluation of processes is used to model mobility.

Klaim consists of three layers: nets, processes, and actions. A net consists of processes or tuples located at a location *l*, or a composition of two nets. Processes can be composed from three base elements: action prefixing, parallel composition, or the invocation of a process declaration. Actions **out** (**in** and **read**) allow outputting (inputting) tuples to (from) another location's tuple space; while the **in** action deletes the read tuple, the **read** action is non-destructive. The **eval** action models mobility by allowing to send a process to another location, where it will be evaluated, and the **newloc** action allows to create new locations.

Here, we consider this language for its capability to provide mechanisms for specifying and enforcing of policies controlling the usage of resources, and authorizing migration and execution of processes to different locations. Klaim exploits a capability-based type system for programming access control [DFP00, DFPV00]. In the type system, types define permissions of processes (*e.g.*, dowloading/consuming tuples, creating new nodes, *etc.*).

In particular, with Klaim as a starting point, another calculus has been developed, called μKlaim, that is at the core of Klaim, with a simpler syntax, with process distribution and mobility, and correlated with a type system that enables dynamic acquisition and modification of security policies and process privileges [GP03].

These calculi deal with distributed systems that share policies, each specifying access control to documents. In particular, the μKlaim framework can model the dynamic acquisition of privileges. Some examples are, *e.g.*, a process may decide to share privileges with other processes running at the same node; privileges may have an expiration date: timing information can easily be accommodated in the framework by assigning privileges a validity

duration and by updating these information taking into account time passing; it is possible to model that a right is acquired only for a limited number of times; the granter of a privilege could decide to revoke the privilege previously granted, [GP04].

Subscribing Online Publications. The example is inherited from [GP03]. Suppose that a user U wants to subscribe a license to enable accessing on-line publications of a given publisher P. The scenario is modelled by using three locations l_U, l_P and l_S, respectively associated to *U*, *P* and to the repository containing *P*'s on- line accessible publications. First of all, *U* sends a subscription request to *P* including its address (together with the access right o) and credit card number; then, *U* waits for a tuple that will deliver it the access right r needed to access *P*'s publications and proceeds with the rest of its activity. The following process implements that behaviour.

$$U = out(\text{“}Subscr,\ l_U : [l_P\ !o],\ CrCard)@l_P\ .in(\text{“}Acc,\ l_S :\{r\})@l_U\ .R$$

In practice, user *U* outputs to location l_P a tuple containing the string *Subscr*, the capability type testifying that from location l_U are authorized output actions to l_P, plus the opportune credit card data. Then, it expects to receive a tuple that grants the account, with an opportune label in a string field of the tuple and the permission to perform read actions to l_S. R is the continuation R that may contain operation like *read(. . .)*@l_S. Behaviour of P is described as:

$$P = in(\text{“}Subscr,\ !x : o,\ !y)@l_P\ .$$

check credit card *y* of *x* and require the payment.

$$out(\text{“}Acc,\ l_S : [\ x\ !\{o\},\ !y])@x \mid P$$

Once *P* has received the subscription request and checked its validity, it gives *U* an access right r over l_S.

Policy Conformance through static validation

In [HNNP08], the authors use a variant of the Klaim language and static analysis techniques as a step towards solving the problem of licence *conformance* of processes, consisting of checking if a process can be trusted to obey restrictions imposed by a policy. In particular, the authors consider policies that restrict the use and replication of information, *e.g.*, imposing that a certain information may only be used or copied a certain number of times.

The analysis tool is a static analyser for a variant of Klaim. Basically, static analysers are tools used at compilation stage, offering a high degree of automation and therefore they may require less human intervention. Often, performing static analysis requires availability of the whole system at compile time. However, when considering policies analysis, and above all enforcement, in a dynamic context privileges may change, *i.e.*, can be acquired at run time, *e.g.*, through plug-ins, or dynamic downloads. The approach presented in [HNNP08] dispenses with the requirement of having availability of the *whole system* in advance, by modularizing the system and signalling possible misuses to be checked *a posteriori*.

In particular, the analysis reveals if there exists, at any point in time, a number of copies of the managed information (*i.e.*, information subject to a certain policy) that exceeds what specified in the related policy. Also, it is possible to check that the client software will never attempt to copy or use any managed information in an unauthorized manner.

This analysis methodology has been developed specifically for dealing with policies restricting the use of documents (to the standard Klaim actions (read/in/out/eval/newloc), [HNNP08] introduces the **use** action, to model the capability of using documents), and in particular it examines their copies. The case study is quite restrictive, in the sense that the policy does not admit copies at all for the managed information. However, it could be worth

investigating this framework further, and, in particular, address issues such as: analyse if there exists a copy of a document, and to report if this copy has been manipulated against the policy.

Other Frameworks

There exist other interesting frameworks that should be helpful in the DSA agreements specification and analysis. We recall *LicenseScript* and the *Audit-based compliance control*.

LicenseScript (LS) [CCDE06] is a language based on multiset rewriting [CDMS99] and programming logic [SSKK86]. It expresses licenses that define the term and conditions of use for data, in particular for music, videos, etc.

LicenseScript is able to regulate distribution of data not only to the single device, but also access and use of data can be managed by licenses that are bound to so called authorized domain, *i.e.*, the collection of devices that belongs to an user, or a householder. Licenses are then bound to a domain, rather than to an individual device. Now, to implement correctly the concept of authorized domain, the language provides the possibility to duplicate licenses. On the contrary, we would have a system dependent from availability of the domain server. Duplicating licenses allows somehow solving issues in a distributed, also unknown environments. Also, distribution and use of data are managed by licenses that can be dynamically updated, defining constraints such as time validity, use no more than n times, *etc.*
In [CCDE05, CCDE07, and CDM04] the authors introduce a policy framework modelling ownership of data and administrative policies, actuating an audit based compliance control. Users can create documents, and authorize others to process the documents, *e.g.*, read/write them.
The novelty in this approach is that it is audit-based, which means that policies are not enforced *a priori*, but checked *a posteriori*. Basically, it assumes the presence of an auditing authority with the task and the ability to observe the critical actions of the users. These, in their turn, keep a secure log of their actions and circumstances, to prove favourable facts to the auditors. In practice, users, instead of having to check whether the policy they have received is actually the right policy for a given piece of data, only have to check that the source is accountable. As an example, Bob receives from Alice the authorization to read a document d. Before reading, Bob checks that Alice is an accountable user (*e.g.*, by using Alice's authorization). Thus, there is no need to check that the policy was produced by an authority on that document, before enacting the policy. If it turns out that Alice was not allowed to authorize Bob, then Bob would be able to "put the blame on Alice".

Technologies products

In this subsection, we review standard-oriented models and methodologies for describing and analysing DSAs.

Analysis of XACML policies

The eXtensible Access Control Markup Language (XACML) [Ris07] is a standardized, expressive and increasingly popular language for writing access control policies for distributed resources. XACML enables the use of arbitrary attributes in policies, allows for expressing negative authorization, conflict resolution algorithms and hierarchical Role Based Access Control, to cite some of the most relevant features.

Every XACML policy has an associated Policy Decision Point (PDP) and a Policy Enforcement Point (PEP). Requests are formed in the decision language by the PEP and sent to the PDP. The PDP implements the policy under consideration, requests are either permitted or denied (unless some error occurs), and the decision is returned to the PEP. The PDP may

query the context to learn the value of environmental attributes, and make its decision based on these values. Summarizing, the *subject* builds a request to perform an *action*, *e.g.*, write copy read print.., on a *resource*, *e.g.*, a file. Requests may be evaluated according to environmental factors specifying further constraints, like "doing something during working hours", or "reading only from a secure machine".

In [Bry05], the author shows how to use the CSP process algebra [RSG01] for formalising and analysing XACML access control policies [LPLK03].

The CSP model proposed in [Bry05] considers the interactions between PEP and PDP, modelled as CSP processes trying to synchronize each other on events representing requests.

The core concepts of XACML are represented in CSP, *i.e.*, modelling requests, modelling roles, modelling single rules, capturing environmental factors, and combining rules into policies. Some properties specified in XACML are translated into CSP, and model checking is used to verify that a policy meets a certain property.

The case study analyzed in [Bry05] is the one of an access control model of a university database which contains student grades. The scenario includes *n* users of the database, each of which may take the role *faculty* or *student*. The database contains *internal* and *external* grades, and the relevant actions are *receive*, *assign* and *view*. The requirements of a correct access to the database are translated from XACML into CSP, and then properties to be verified are checked with the model checker FDR, www.fsel.com. Examples of analyzed properties are, *e.g.*, "There do not exist members of Student who can assign external grades", or "No user who has previously assumed the role of Student can assign external grades".

3.3. DSA to Policy Mapper

Several works have been done in developing process refinement theories. For that reason there are several definitions of refinement according to the field it is applied to.

Models

According to [KL04], refinement is a projection of a big policy into small one whose composition gives the original. Indeed they consider different levels of abstraction from high-level business goals to policies for individual resources. Ideally, it should be possible for an organization to derive their system policies (low level policies) from their overall business goals (high level policies). From this point of view, policy refinement is the process that decomposes the high level policy relevant to a composite system into a set of policies that are executed in its constituent parts to implement the behavior intended by the high level policy. Policy refinement is thus capable of mapping high level goals of a composite system automatically down to low level policies. A policy language satisfying refinement is a language that can be refined to some low level details.

Also in [LX91] the authors refer to the notion of refinement as decomposition. They say that, for large systems, the implementation is not expected to be extracted directly from the initial specification. Each intermediate specification may be viewed as a refinement step that simply consists of a refinement of one component into a combination of a number of subcomponents that are not yet implementable.

The notion of refinement can be seen as the passage from a coarser level of specification to a finer one.

In [ST07] the authors present a case study of Event-B applied for the refinement of a controller for a security property along the different network layer of the TCP/IP stack. They

also prove that the refinement is valid. Formal verification can be performed on the resulting model. This approach is applied to a network security policy. The policy is refined through the different network layers and the relation between the layers are modeled in order to keep a trace and to ensure that the refinement is valid. Formal verification can be processed on the resulting model. One of the main advantages of this approach is the use of formal methods. Indeed policies are formally defined and the refinements can be checked. It is very suitable for critical systems and those which need traceability link between models. However, the approach is difficult to use in a large problem. Indeed, the refinement definitions can become very complex and no patterns are available to help in this approach.

The term refinement is also used in the development of software components. Indeed it is quite often required to relate systems belonging to different abstraction levels. In particular, a lot of work has been done to study the passage from the specification level (high level of abstraction) to the implementation level (low level of abstraction).

In formal methods, program refinement is the transformation of an abstract high-level formal specification into a concrete low-level (executable) program. *Action refinement* (see [GRZ01, RG97]) converts a specification of an action on a system into an implementable program (e.g., a procedure). The basic idea of the *process refinement* is that, given two processes P and R(P), R(P) is the *implementation* of P if it is described at a deeper level of details then P.

In [BL04] the authors refer to the policy refinement problem as the passage from the high specification level of a system to the implementation of the system itself guaranteeing that the goals are achieved. As a matter of fact, in this paper, the authors have presented an approach by which a formal representation of a system, based on the Event Calculus, can be used in conjunction with adductive reasoning techniques to derive the sequence of operations that will allow a given system to achieve a desired goal.

In [JJ01] the authors have considered the notion of *stepwise refinements* a useful paradigm of system development. Indeed, also in [JJ01], the refinement is the passage from the abstract specification to a concrete specification which is implemented through several step of refinement. The paper is focused on the study of relevant properties, in particular security properties, that have to be preserved also to concrete specification level.

The secrecy property considered in [JJ01] relies on the idea that a process specification preserves the secrecy of a piece of data *d* if the process never sends out any information from which *d* could be derived, even in interaction with an adversary. In general, it is slightly coarser than the first kind in that it may not prevent implicit information flow, but both kinds of security properties seem to be roughly equivalent in practice. Also, more fine-grained security properties may be hard to ensure in practice.

For that reason they have proposed a secrecy-preserving refinement, one can also address situation where implementations of formally verified security protocols turn out be insecure.

The specification of the system is viewed into the specification Framework FOCUS that, in addition to executable specifications, also allows the use of non-executable specifications. Executable specifications allow a rather straightforward modeling of cryptographic aspects as encryption.

After presenting their notion of secrecy, that is refinement-preserving, several notions of refinements are presented:

- Property refinement: it is a trace inclusion between the two policies;
- Interface refinement: it is a process refinement of a process that manages the communication between two other processes;

- Conditional refinement: it is a process refinement with respect to a total relation on the input and output streams of the abstract process that has to valid also after the refinement.

Speaking about the relation between the secrecy property and refining, in [Bru94] is preserved a case study showing that application of modal process logic to formal refinement of a failure-recovery protocol used in a safety-relevant air transport application. Intuitively, modal process logic allows specifications to express that certain action must be performed by an implementation, and that other actions may be performed by an implementation.

Technologies products

In [USS07] the authors have proposed an automated and domain independent approach to derive policy bounds on the low level metrics such that the high level goals are respected. The refinement procedure is made by the identification of relevant low-level attributes as well as policies that define bounds on them based on high level policies. The proposed refinement procedure is made up of several phases and it is carried out using a combination of data classification and test & development approach.

Another approach for refining policies is using KAOS, [DL93]. KAOS is a goal oriented methodology to analyze and refine requirements, especially, security requirements. KAOS is based on several models which give a different point of view about the situation analyzed. There are four models:

• *The goal model* represents the goals that have to be met by the system. Each goal is refined into sub-goals until the sub-goals responsibility can be assigned to an agent. The refinement process is an "and/or" relationship between sub-goals and parent goal.

• *The responsibility model* defines the responsibility assignments between terminal goal (requirement or expectation) and the agents.

• *The object model* is use to define the domain concepts that are relevant in accordance with the goal model. It is quite similar to UML class diagram. Objects in KAOS correspond to entities in UML.

• *The operation model* describes the agent behaviors in term of operations. Each terminal goal is rendered as a set of operations which are performed by an agent. Operations can have object as inputs or outputs.

A policy template [NAM06] can be used with KAOS in order to refine policies, since KAOS manages requirements. The template is filled for each security requirement and refined according to the goal refinements until a specific implementation detail level is reached. KAOS is suitable for the refinement process because of its library of refinement patterns that have been proved and can be reused in concrete situations. Refinement is formally defined in KAOS and can be easily checked using model checking tools like, e.g., Spin [Spin]. Refinement properties such as completeness, correctness and minimality are clearly defined and have been used to verify the refinement patterns. In addition to refinement suitability, KAOS offers other interesting mechanisms to reason on security requirements. Indeed, threats that present risks for organizations can be identified and an obstacle analysis can be done in order to find new security requirements. KAOS uses temporal logic to formally define the goals. KAOS seems to be very adequate for security analysis, and it offers mechanisms for verification through Event Calculus and implementation through Ponder [Ponder].

It is possible to refine policies by using the process algebra Communicating Sequential Processes (CSP) to model security policies at different levels of abstraction and check their refinement and correctness (see [Bry05]). This method considers XACML as a language to

specify access control policies and give a semantic mapping of a subset of XACML to CSP. The CSP mapping is given for a basic access control policy. The case chosen is role based access control (RBAC) in a simple organization. This approach is sufficiently powerful to check properties of policies using the model checker associated with CSP. The meaning of refinement in this approach is quite similar to the refinement definition in KAOS. But the correctness check is missing. Indeed, the informal level is not covered by this approach and no traceability between refinements is explicitly represented as in the KAOS methodology. Although the CSP approach can be adapted to the refinement process, it is not particularly suited for it, and other approaches seem to be more appropriate.

3.4. DSA Lifecycle Manager

In the area of document management and workflow several tools are available. We review in the following sections Alfresco and Sharepoint, giving a brief overview of their capabilities that may be customized for implementing the DSA Lifecycle Manager.

Alfresco

Based on the DSA Lifecycle manager high-level requirements we have assessed some document management and workflow systems, and we have identified Alfresco [Alfresco] as a candidate solution to implement the DSA Lifecycle Manager.

Alfresco is an open source solution for enterprise content management. It offers several advanced capabilities: Document and Knowledge Management, Collaboration, Web Content Management, Imaging, Workflow, etc. Alfresco is available in two version (both based on the same open source technologies): a free edition (Alfresco Labs), and an enterprise edition (Alfresco Enterprise Edition); the former is supported by Community Forums, whereas the latter is provided with commercial support. For a comparison between the Alfresco Enterprise Edition and Alfresco Labs Edition see [AlfrescoEE] and [AlfrescoLE].

Alfresco is a J2EE cross-platform application. It offers a very complete set of document management capabilities, which facilitates collaborative document creation and editing. All functionalities are available through a rich web client interfaces. Simple workflow functionalities are built-in. The Web Services API and the open architecture ensure very broad possibilities of customization and integration with external software/tools (for example, tools for the verification of formal DSA documents). The interested reader may find an overview of Alfresco's capabilities in SubAppendix 2.

Microsoft Office SharePoint Server

Microsoft Office SharePoint Server 2007 is a server program that is part of the 2007 Microsoft Office system. Generally it is an integrated portal solution that may be used to facilitate collaboration, provide content management features, implement business processes, and supply access to information that is essential to organizational goals and processes. Extensive product information may be found at the Microsoft Office SharePoint Server website [SharePoint]. Microsoft Office SharePoint Server 2007 has six major feature areas:

1. **Collaboration**. The enabling technologies that allow teams to work together effectively, providing intuitive, flexible, and secure mechanisms for sharing information through the use of wikis and blogs, collaborating on and publishing documents, maintaining task lists, conducting surveys, developing and maintaining site templates customized for specific business uses, and implementing workflows.

2. **Portal**. The facilities that provide the capabilities to personalize the user experience of an enterprise Web site, to target content to various audiences based on sets of rules, to automatically facilitate intuitive navigation through the Web site while tailoring the navigation to the individual rights of the user, to deliver comprehensive site content management and structural facilities, and more.
3. **Enterprise Search**. The critical ability to quickly and easily locate relevant content distributed across a wide range of sites, document libraries, business application data repositories, and other sources, including files shares, various Web sites, Microsoft Exchange public folders, and Lotus Notes Databases — and to find the appropriate people who can help answer questions or be involved in projects.
4. **Content Management**. The facilities for the creation, publication, and management of content, regardless of whether that content exists in discrete documents or is published as Web pages. Content management scenarios include document management, records management, and Web content management.
5. **Business Forms and Integration**. The ability to rapidly and effectively implement forms-based business processes, from design to publication to user access, by using standard Web browsers or a rich client application such as Microsoft Office InfoPath 2007. Also includes the ability to connect with structured systems such as databases and line-of-business applications, and the ability to access that information in a number of ways.
6. **Business Intelligence**. The ability to deliver information critical to business objectives through a wide range of mechanisms, from server-based spreadsheets accessing business data in real time and performing sophisticated analyses to the presentation of key performance indicators (KPIs) through enterprise Web sites.

Microsoft Office SharePoint Server 2007 is a mature enterprise solution with rich workflow support. Because of this it can easily serve as a solid backbone for Consequence DSA Lifecycle Manager. Further information of Microsoft Office SharePoint Server 2007 is illustrated in SubAppendix 3.

3.5. DSA Trust Manager

This section reviews the state of the art in trust management systems. The structure of the section reflects three areas of research: trust models and metrics, reputation systems and trust-related security. The notion of trust takes different meaning depending on the context of its usage. In fact, within computer science research, the notion of *e-trust* (electronic trust) is more important than the generic notion of trust. However, for the sake of brevity we use the terms "trust" and "e-trust" interchangeably in the following state-of-art description.

Trust Models

In the work of Kini and Choobineh [Kin98], trust is given a definition based on the perspectives of personality theorists, sociologists, economists and social psychologists. In fact, trust in [Kin98] is defined as being "a belief that is influenced by the individual's opinion about certain critical system features". This definition covers various aspects of human trust in computer-dependent systems, but it does not address the notion of electronic trust, i.e. trust among different computer processes and among processes and humans.
In [Gam88], Gambetta proposes the following definition: "trust (or, symmetrically, distrust) is a particular level of the subjective probability with which an agent will perform a particular action, both before (the trustor) can monitor such action (or independently of his capacity of ever to be able to monitor it) and in a context in which it affects (the trustor's) own action".

This implies that trust is a belief or an estimation of the trustor towards his/her trustee. Castelfranchi and Falcone [Cas98] extend the definition of [Gam88] to include the notion of *competence* along with *predictability*.

The Trust-EC project [Trust-EC] of the European Commission Joint Research Centre (ECJRC) defines trust as being: "… the property of a business relationship, such that reliance can be placed on the business partners and the business transactions developed with them" [Jon99]. These "properties of business relationships" include the confidentiality of sensitive information, the integrity of valuable information, the prevention of unauthorised copying and use of information, the guaranteed quality of digital goods, the availability of critical information, the management of risks to critical information and finally, the dependability of computer services and systems.

Grandison and Sloman [Gra00] review several definitions of trust in literature and propose, following a brief analysis of such definitions, their own definition of trust as being: "… the firm belief in the competence of an entity to act dependably, securely and reliably within a specified context", which in some sense, is a summary of the definition given by [Jon99] stressing the aspect of context.

Dimitrakos [Dim01] defines the trust of a service requestor, A, to a service provider, B, in relation to some service, X, as follows: "Trust of a party A in a party B for a service X is the measurable belief of A in B behaving dependably for a specified period within a specified context in relation to X. Where:

- A party can be an individual entity, a collective of humans or processes, or a system.
- The term service is used in a deliberately broad sense to include transactions, recommendations, issuing certificates, underwriting, etc.
- The above mentioned period may be in the past, the duration of the service, future or always.
- Dependability is used broadly to include security, safety, reliability, timeliness, and maintainability.
- The term context refers to the relevant service agreements, service history, technology infrastructure, legislative and regulatory frameworks that may apply.
- Trust may combine objective information with subjective opinion formed on the basis of factual evidence and recommendation by a mediating authority.
- Trust allows one agent to reasonably rely for a critical period on behaviour or on information communicated by another agent. Its value relates to the subjective probability that an agent will perform a particular action (which the trustor may not be able to monitor) within a context that affects the trustor's own actions."

Dimitrakos also distinguishes between the notions of *distrust* and *the lack of trust* as follows: "Distrust of a party A to a party B for a service X is A's measurable belief in that B behaves non-dependably for a specified period within a specified context in relation to service X". Compared to the lack of trust, in this sense, distrust becomes a useful measure to revoke previously agreed trust, obstruct the propagation of trust, ignore recommendations and communicate a party's trust value and whether this has reached the level of a *blacklist* for a class of potential business transactions.

In [Yah93], Yahalom et al. question the classification of entities as being *trusted* or *untrusted* as being an oversimplification of real world situations. Their view is that trustworthiness should only be measured with respect to a *business task* that an entity is assigned. Such tasks have different levels of significance, characteristics and verifiability. The performance of an entity in all tasks is not subject to aggregation; an entity may be good at some tasks but bad at others. Hence, trust is refined as being single-task-based rather than being all-tasks-based.

In [Her06], Hermoso et al. adopt a model of trust that is based on the notions of *confidence* and *reputation* of an agent within a role-based organisation and that is measurable by values

within [0, 1]. The reputation of an agent is measured according to how well the agent performed in some role in the past. This will impact the confidence other agent's have on this particular agent.

Trust Metrics

In order to be able to measure the concept of trust, one needs to talk about a trust *metric*. Toone et al. [Too03] define a trust metric as a metric that: "specifies the trustee properties that are relevant to the trustors' decision of whether or not to place trust in a trustee. Furthermore, a trust metric defines either qualitatively or quantitatively how to assign a rating to a trustee based on observed values of these properties". In a distributed computing environment, a trust metric is needed in order to establish security and privacy properties among participating entities.

There are several metrics for measuring trust.

According to [Bin04], trust metrics can be classified into *quantitative* (e.g. numerical value) and *qualitative* (e.g. elements of a total-order structure built from the set {minimal, average, good}). In any trust model, a combination of both classes of trust metrics may be used depending on the roles/tasks of the trustee and trusted entities. For example, an agent could be assigned to a number of roles and the level of trust in its success in each individual role is measured by an integer. However, the overall aggregation of the trust levels may be abstracted to a member of the set {minimal, average, good}.

On the other hand, a global trust metric is one in which the view of all entities in a system are taken into account while a local metric is biased towards an individual entity's view of the system. A famous example of a global trust metric is the PageRank [Pag98] metric used by Google™, whereas examples of local trust metrics include Levien's Advogato trust metric [Advocado], Golbeck and Hendler's metric for Semantic Web trust [Gol04], and Sun Microsystem's Poblano model [Che01]. The second dimension classifies trust metrics as *distributed* vs. *centralised*. This indicates whether the trust information and computation are distributed across the network or whether centralised on a single machine. Most of the above metrics are centralised. The EigenTrust [Kam03] metric constitutes an example of distributed metrics.

Finally, the third dimension classifies metrics as either *scalar* or *group*. In the former, metrics analyze trust assertions independently, whereas in the latter, group trust metrics evaluate groups of assertions "in tandem". PageRank can be classified as a global group trust metrics since the reputation of one page depends on the ranks of referring pages, whereas Advogato is an example of local group trust metrics.

In [MP07, Ris07] the authors enriched the RT language [LMW02, WM03] with notions of trust (and trust levels) towards performing some functionality, and for giving recommendations.

The language defines roles in terms of (chains of) credentials. In the following credential, e.g., organization IIT assigns the role of IIT researcher to Marinella, whose distinguished name adopted on the Consequence project is "CN=Marinella, OU=IIT, O=CNR, L=Pisa, C=IT".

IIT.researcher('CN=Marinella, OU=IIT, O=CNR, L=Pisa, C=IT') ← Marinella

A trust credential of the form A.r(p,v) ←D says that A trusts D for performing r with parameter p and level v. Compound credentials can be defined too, according, for example, to the notion of transitivity of trust, and combining trust levels with opportune mathematical operators, [MP07, Ris07].

The motivation for defining a language of trust credentials is because policies may take into account not only behavioural aspects, but also reputation and recommendations factors as well. Generally speaking, the reputation of a user can be calculated based on past experiences

of other services with respect to that user. A service will recommend interactions with a user based on a positive record of 'good behaviour'.

In this context, RT could be easily extended to the management of data sharing agreements. RT with trust is attribute-based, *i.e.*, it links principals (both individuals and groups) to the more general notion of attributes and properties; the trust version of the language encodes notions of trust and recommendations into policies.

Reputation Management

In this section we review existing reputation systems employed in different areas of computing science.

The authors of [Jos07] say that reputation is what is generally said or believed about a person or a thing. They argue that reputation is a mean of building trust, as one can trust another based on a good reputation. According to Abdul-Rahman and Hailes [Abd00], a reputation is *an expectation about an agent behaviour based on information about or observations of its past behaviour*. There are two main sources for building the reputation of an entity: the past experience and the collected referral information.

Zacharia and Maes [Zac00] proposed SPORAS & HISTOS. In SPORAS, reputation is based only on direct transaction ratings between users. Users rate each other after a transaction with continuous values from 0.1 to 1. Each user reputation aggregates in a recursive fashion the received ratings: users with very high reputation will experience much smaller rating changes after each update and ratings are discounted over time. In HISTOS they take into account also the social network created between users through performing transactions.

Sabater and Sierra [Sab01] propose a model, named REGRET that considers the following dimensions of the reputation: *the individual dimension*, which is the direct trust obtained by previous experience with another agent, *the social dimension,* which refers to the trust of an agent in relation with a group, and *the ontological dimension,* which reflects the subjective peculiarities of an individual. Their model focuses on SLAs between two parties, with several variables under interest. Ramchurn et al. [Ram04] further refine the system by detailing more on the terms of a SLA.

Another approach is that of Huynh et al. [Huy06], who propose that a trust model must (1) take into consideration a wide variety of information sources, (2) the agents should be able to evaluate the trust for themselves (distribution and local control) and (3) the trust model should be robust to lying. They address the first two requirements thus building a trust model based on: (1) interaction trust, (2) role-based trust, (3) witness reputation and (4) certified reputation. To obtain the trust value for an agent, one has to aggregate each piece of reputation mentioned above. They propose to weigh each component in order to reflect the emphasis the model assigns for each of the information sources above.

Gupta et al. [Gup03] propose a reputation system for a Gnutella-like file sharing system to track back the past behaviour of users and to allow drawing up decisions like who serves content to and who requests content from. In this model, the reputation of a peer depends on its behaviour assessed in accordance with the contribution of the peer to content search and download and its capability expressed in terms of processing power, bandwidth, storage capability and memory. Each peer in the network gets credit for (1) processing query response messages, (2) serving content and (3) sharing hard-to-find content in the network. Content serving and sharing hard-to-find content are assessed based on the quality of the service provided (in terms of the bandwidth and file size). For each download, a peer reputation is debited with a similar amount as for serving the same content. The reputation score is simply a summation of the receiving credits with or without deducing the debits.

TrustMe [Sin03] is a user-based approach, adopting the principle of obtaining references about a peer, before engaging in a transaction with that peer. Broadly speaking, TrustMe

functions in the following manner: each peer is equipped with a couple of public-private key pairs. Trust values of a peer (B) are randomly stored at another peer (THA) in the network. Any peer, A, interested in the trust value of another peer, B, broadcasts the query on the network and the THA peers reply to this query. Based on the received trust value, A then decides to enter or not in interaction with B. After interaction, A files a report for B indicating the new trust value for B and therefore, THA can modify the trust value of B accordingly. It is assumed that somehow A updates the trust information for B and broadcasts back this information to its storage located at THA.

PeerTrust [Xio04] is based on five important parameters contributing to a general trust metric: (1) feedback a peer obtains from other peers, (2) feedback scope counted as the number of total transactions that a peer had with other peers, (3) the credibility factor for the feedback source, (4) the transaction context factor discriminating between mission-critical and non-critical transactions and (5) the community context factor for addressing community-related characteristics. In fact, as revealed by their general trust metric formula, the trust metric for a peer is composed by the community context factor metric and the weighted satisfaction received for previous transactions.

Finally, in NICE [She06], the authors adapted the idea of the social network described in the agent-based approaches to the structure and the security requirements of a fully decentralized P2P network, equipped with a PKI infrastructure. Each agent comes to the system with a pair of public and private keys and the messages are signed by the peers who are creating them. Therefore, after each transaction between a peer client A and a servant B, the peer A generates a cookie with its perceived feedback (trust value) for the transaction. The servant peer B can store the cookie received from A as a reference of its effectiveness in other transactions. When a peer A deliberates to enter a transaction with peer B, a cookie might exist between A and B and in this case, this cookie contains the trust peer A has for B. If such a cookie does not exists, A will ask its partners whether they have cookies for B and the partners will continue to spread the request into the network till a path between A and B is established. On this graph structures paths between A and B are evaluated, giving the required trust value.

Technologies products

In this section we concentrate on the *public-key infrastructure* support for trust models along with some of its variants.

Public-Key Infrastructure

Public-Key Infrastructure (PKI) is usually described as a community of *principals*, a community of *verifiers* and a hierarchy of authentication authorities known as the *certification authorities.* The purpose of a PKI is to allow the verifiers to authenticate any attribute in general (but specifically the identity) of principals using the certificates issued by certification authorities in a distributed manner. Such authentication is usually necessary whenever interactions among different entities (e.g. a client and a server) take place over public insecure networks. There are many references in literature on PKI and the way it works, for example [Ada02] [Vac04].

In [Bin04], the following three trust models are identified based on the mechanisms of the previous section.

1. The *mesh* model: In this model, a verifier only trusts its own certification authority (the certification authority that issued its certificate) or certification authorities that cross-certify with its own certification authority.
2. The *hybrid* model: this model is similar to the mesh model except that the requirement of cross-certification of authorities is relaxed to the trusted list managers instead.

3. The *bridge* model: in this model, several mesh networks of trust are related via a hub known as the *bridge certification authority*, which is used to establish across multiple hierarchies of authorities.

Trust Mechanisms

Tim Moses ([PKITM] *PKI Trust Models)* identifies four PKI mechanisms that can be used to establish trust by a verifier in the authenticity of a principal in some distributed environment:

1. Out of bound mechanisms, which include mechanisms for publishing of fingerprints of authentication keys in a trustworthy location, protocols for secret-key sharing and techniques for embedding keys in trusted software.
2. Certificate trust lists, which are essentially the same as the out-of-bound mechanisms but where a trusted list manager secures, using its digital signature, the published fingerprint instead of storing the fingerprints in trustworthy locations.
3. Certificate request messages, which are used by *registration authorities* responsible for forwarding requests from principals (for obtaining certificates) to the certification authorities. In this case, a registration authority acts as an intermediate step between the principal and the certification authority.
4. Cross-certification, which refers to the mechanism used in verifying certificates that are not necessarily directly trusted by the verifier, but that are trusted by a certification authority trusted by the verifier (indirect trust). These mechanisms give a rise to a number of trust models as discussed next.

Simple Public-Key Infrastructures/ Simple Distributed Security Infrastructures

The Simple Public-Key Infrastructure SPKI is an authorization model born as an effort to overcome the over complication and scalability problems of PKIs. It is specified in [Ell99a, Ell99b]. The authorization certificate defined in the SPKI specification identifies principals only as public keys, and binds them to a set of privileges, rights and authorizations. SPKI removes some of the overhead associated with certification authorities and allows for more flexible delegation of authority. The risk with this approach is that it places greater responsibility of trust on the verifier's shoulders.

The Simple Distributed Security Infrastructure SDSI [SDSI] is another design for public key infrastructures that bound local names (of individuals or groups) to public keys, but carried authorization only in Access Control Lists (ACLs) and does not allow for delegation of subsets of a principal's authorization.

Finally, the combined SPKI/SDSI allows the naming of principals, creation of named groups of principals and the delegation of rights or other attributes from one principal to another. It includes a language for expression of authorization and the notion of threshold subject, a construct granting authorizations or delegations only when a certain number of subjects allow for them.

4. Proposed model/approach

Here, we present one proposed approach for each component of the DSA infrastructure pointing out how that model implements the requirements listed in Section 2.

4.1. DSA Authoring

The different components of the DSA Authoring tool are shown in Figure 5 below.

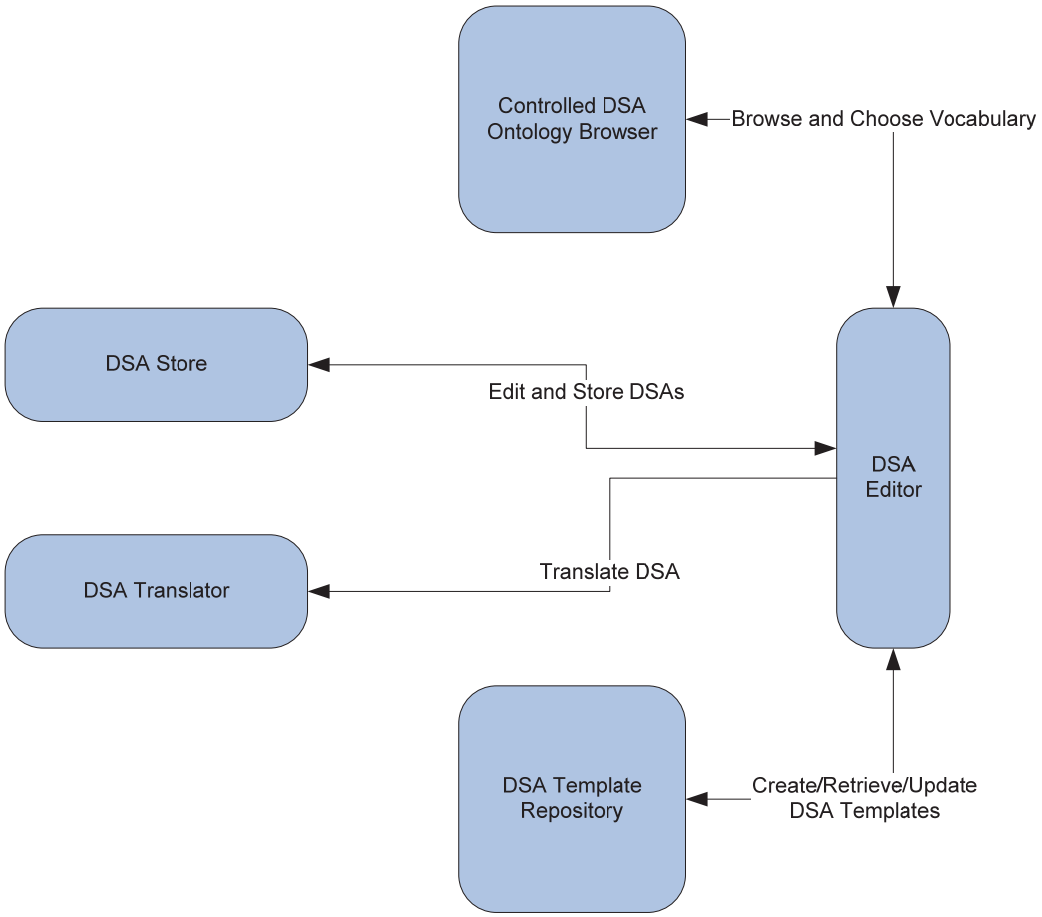

Figure 5. The components of the DSA Authoring tool.

The tool comprises the following components: a *DSA Editor*, a *DSA Template Repository*, a *DSA Store,* a *DSA Vocabulary Browser* and a *DSA Translator*.

The *DSA Editor* provides a front-end to the users, through which they will be able to initiate the different functionalities of the DSA authoring tool. The DSA Editor extends the functionality of a basic editor by "marking" (tagging) natural languages clauses (obligations, permissions, prohibitions) that will need to be translated into policies. Note that an agreement may contain additional information that would not need to be translated into a policy. The DSA Editor will also interact with the other components as discussed below.

The *DSA Template Repository* is a repository of model agreements as well as clause templates. For instance, there will be business-level DSA templates in the domains of

scientific experiments and crisis managements, including full agreement templates as well as example of potential domain specific clauses. The DSA Template Repository will facilitate the creation, retrieval and update of templates when one is using the DSA Editor.

The *DSA Vocabulary* is an ontology browser which allows the users to search and include terms of particular thesaurus/ontology in a DSA document when using the DSA Authoring tool. At this stage in the project, it is envisaged to build on existing software providing these functionalities. The browser will be used to create an initial model vocabulary for DSAs based on the various DSAs dealt with in the Consequence test-bed scenarios. One such vocabulary is shown in the figure below (Figure 6).

Figure 6. Example vocabulary.

The elements of the vocabulary are described as follows.

Objects

Data – files, documents, images and other forms of digitised data (raw data, digitised data, quality controlled data, derived data). Further terms such as data collection and data set also need to be defined.

Metadata – they provide a user with any necessary details on the origin or manipulation of the data so that they then can use it – to enrich the data to maximise value to current and future users.

Individual – human or agent

Data Store – entity at a *location* at which *data* is stored

Project – an entity that has a role of *project member* that can be fulfilled by one or more *Users*

Organisation – an entity that has a role of *staff member* that can be fulfilled by one or more *Users*

Application – an entity that performs *operations* on *data*

Policy – statement of policy, subtypes include obligation, prohibition

Consent – consent of an agent required as a condition in a policy

Datalog – a record written by a log operation and read during an audit operation

Roles

Project Member – a role of Projects that can be filled by Individual

Staff Member – a role of Organisation that can be filled by Individual

Organisational Role – role in an organisation filled by an Individual

Owner – role of an individual - the human or agent that owns data, and whose consent may be required to permit access/usage by an operation

User – role of an individual - human or agent that can fill user roles – user as *project member*, or staff member of *organisation*

Operations

Access – subtypes, read, write, modify, delete, copy

Transform- transform data (D1) into derived data (D2)

Merge – merge data (D1) with data (D2) to produce data (D3)

Transmit – operation to transmit Data from one *User* to another *User*

Destruction – deletion of *Data* from *Data Store*

Inform – to send a message to a Role or individual

Log – to store information, usually about access to data, including a timestamp

Audit – to analyse a datalog

Contextual Properties

Time – subtypes start date/time, end date/time, metrics unspecified

Location – place where data can have operations performed on it, or where, metrics unspecified

Agent – user or user role who can perform operations on data, metrics unspecified

Policy Properties
Revocable – can a policy be revoked or not

Policy Operations
Before – conditional operator on a Time property.

After - conditional operator on a Time property.

Until - conditional operator on a Time property.

The *DSA Store* is a repository of completed (actual) agreements. The actual DSA can be seen as instances of DSA templates. The DSA Store provides functionalities for the management of the agreement, such as search, retrieval, versioning and update of agreements.

Finally, the *DSA Translator* is a tool generating the formal level representation of a DSA business representation. We envisage the DSA Translator initially to be based on pattern translation, i.e. there will be a template of clause agreements in controlled natural language and their corresponding translation into the formal policy language. However, the DSA Translator must also allow for natural language translation tools to be plugged into it, such that it can support a wider range of DSAs.

How the DSA authoring component meets the requirements

The definition of the DSA Authoring tool into its current subcomponents is clearly aligned to fulfil the functional requirements, Auth-R1 to Auth-R5, of Section 2.1. Each of the proposed subcomponents corresponds to one such requirement. Requirements Auth-R6 to Auth-R10 represent extra functionalities or non-functional properties that must be implemented or respected. For example, Auth-R9 requires that the DSA Authoring tool is capable of accepting feedback from the DSA Analysis and then highlighting the feedback to the user. The DSA Analysis will have functionalities also to give feedback to the DSA Authoring tool.

4.2. DSA Analysis

In this section, we first present a concurrent language, namely POLPA, the POLicy Process Algebra, based on a formal semantics, through which we intend to model the significant part of data sharing agreements. Then, we illustrate the methodology that we aim at exploiting for the analysis and evaluation of DSAs. We consider a simplified fragment of a real data sharing agreement, in order to show its formal model, and the property samples to be investigated on.

POLPA. The language allows for distributed communication, and it is decorated by predicates expressing conditions over contextual factors, like time/location/attributes/trust.

We see the sequence of possible events specified by a data sharing agreement as a process P that is the result of the composition of actions, belonging to the set Act and ranging over α β γ …, and predicates, belonging to the set $Pred$ and ranging over p, r, .. Predicates and actions may be parameterized. Composition is through parallel and choice operators, as described by the following grammar:

$$P ::= stop \;||\; all \;||\; \alpha(\vec{x}).P \;||\; p(\vec{x}).P \;||\; x := e.P \;||\; P_1 \; or \; P_2 \;||\; P_1 \; par_{\{Alfa\}} \; P_2 \;||\; Z$$

where $\vec{x}$ is a vector of expressions, built over a set of variables, and including variables and constant values. Z is a constant process definition $Z \doteq P$. Throughout the section, referring to

actions and predicates, we will omit making parameters $\vec{x}$ explicit when this is not strictly necessary.

The informal semantics of the language is the following:

- *stop* is the process that cannot do nothing;
- *all* is the process that allows all;
- $\alpha(\vec{x})$ *.P* is the process that performs action $\alpha(\vec{x})$ and then behaves as *P;*
- $p(\vec{x})$*.P* is the process that behaves as *P* when the predicate $p(\vec{x})$ is true*;* when the predicate is false, the process gets stuck.
- *x := e.P* assigns to variable *x* the value of expression *e* and then behaves as *P*, in which occurrences of *x* are replaced by *e*;
- P_1 *or* P_2 represents the choice between P_1 and P_2. The future evolution of a policy is defined as a choice between two component policies P_1 and P_2.
- P_1 $par_{\{Alfa\}}$ P_2 represents concurrent activity requiring synchronization between the component policies. In particular, any action belonging to the set of actions *Alfa* can only occur when both the component policies permit that action. When P_1 and P_2 engage in actions not belonging to *Alfa*, the process represents independent concurrent activity and the events from both the component processes are arbitrarily interleaved in time.
- *Z* is the constant process. We assume that there is a specification for the policy $Z \doteq P$ and *Z* behaves as *P*.

The "*if then else"* construct can be specified in the language as follows: $p(\vec{x}).P_1$ *or (not* $p(\vec{x})).P_2$; this represents the process behaving as P_1 when *p* is true, and behaving as P_2 when *p* is false.

It is worth noticing that specifying predicates $p(\vec{x})$ that must be true *while* P executes is not naturally captured by the language. However, the language can be enriched by inserting a pre-emptive operator ▷ whose informal semantics is "execute P while p is true, else abort P".

For modeling the behavior of our systems, we adopt a *labeled transition system*, *i.e.*, a structure (*P, Act,* $\xrightarrow{\alpha}$), where *P* is the set of processes, *Act* is the set of actions including the internal invisible action τ , and $\xrightarrow{\alpha}$ is a ternary relation, which is a subset of *P x Act x P*, representing a transition relation between processes through an action. In particular, $\xrightarrow{\alpha}$ is the least relation between processes induced by the axioms and the rules of Figure 7, describing the operational semantics of the terms of our language in a standard way, as in [Plo81], where rules are expressed in terms of a set of premises, possibly empty (above the line) and a conclusion (below the line), plus a set of side conditions, possibly empty.

$$(prefix)\ \frac{}{\alpha.P \xrightarrow{\alpha} P} \qquad (predicate)\ \frac{P(\vec{e}/\vec{x}) \xrightarrow{\alpha} P'}{p(\vec{x}).P(\vec{x}) \xrightarrow{\alpha} P'}[p(\vec{e}/\vec{x}) = \text{true}]$$

$$(or)\ \frac{P \xrightarrow{\alpha} P'}{PorQ \xrightarrow{\alpha} P'} \qquad (par1)\ \frac{P \xrightarrow{\beta} P'}{Ppar_{\{\alpha 1,\ldots,\alpha n\}}Q \xrightarrow{\beta} P'\ par_{\{\alpha 1,\ldots,\alpha n\}}Q}[\beta \notin \{\alpha 1,\ldots,\alpha n\}]$$

$$(par2)\ \frac{P \xrightarrow{\beta} P'\ Q \xrightarrow{\beta} Q'}{Ppar_{\{\alpha 1,\ldots,\alpha n\}}Q \xrightarrow{\beta} P'\ par_{\{\alpha 1,\ldots,\alpha n\}}Q'}[\beta \in \{\alpha 1,\ldots,\alpha n\}] \qquad (const)\ \frac{P \xrightarrow{\alpha} P'}{Z \xrightarrow{\alpha} P'}[Z =_{def} P]$$

Figure 7. Operational semantics. Symmetric rules for or, par1 are omitted.

In the figure, the symmetric rules for *or* and *par1* are omitted. The expression $P \xrightarrow{\alpha} P'$ means that the system can move from P to P' through the action α. Predicate $p(\vec{e}/\vec{x})$ is evaluated by substituting to variables in vector $\vec{x}$, if any, values in vector of expressions $\vec{e}$.

Actions and predicates. Within a DSA-oriented scenario, actions refer to communications among principals involved in the DSA, or to scenario-specific actions, like reading, writing, printing, copying, etc., some data or document.

Predicates refer to conditions that must be satisfied in order for some action to take place. Evaluation of a predicate results in a boolean expression *true/false*. Evaluation occurs when all the variables in $\vec{x}$ are substituted with values in $\vec{e}$. We consider only decidable predicates.

The set *Pred* of predicates include time-based predicates $p_{time}(\vec{x})$, location-based predicates $p_{loc}(\vec{x})$, role-based predicates $p_{rol}(\vec{x})$, and trust-based predicates $p_{tru}(\vec{x})$.

An example for a time-based predicate is *IsBefore(y)* returning *true* if current time is before *y*. An example for a location-based predicate is *IsAtDistance(s,o,min d,max d)*, returning *true* if the distance between subject *s* and object *o* is within interval [*min d,max d*]. A role-based predicate is, e.g., *HasRole(s,rol)*, returning true if subject *s* has role *rol.* Broadly speaking, one can consider a role-based predicate within the more general context of attributes. From this point of view, the predicate could speak not only about attributes of the subject, but also about attributes of the object. For trust-based predicates, we have, e.g., *RepMaxOf(s,r)*, returning *true* if the reputation of the subject *s* is greater than a threshold *r*.

Predicates evaluation involves retrieving context data about location and time and attribute data about trust and roles. As usual in policies evaluation scenarios, these data are made available by third parties through service interfaces called, e.g., Location Services, Trust Services, *etc*. Thus, at modelling level, we do not expect to manage direct access to location, time, trust, role information. Rather, we assume a global architecture relying on sending location, time, and trust and role requests to external services that give the corresponding answers.

Notation $\vec{x} = \vec{e}$ (similarly >, <, etc.) is a shortcut to denote that, with $\vec{x} = [x_1 \ldots x_n]$, and $\vec{e} = [e_1 \ldots e_n]$ then $x_1 = e_1 \ldots x_n = e_n$. Vector $\vec{e}$ contains no variables. Logical connectives *not*, *and*, *or* represent standard negation, conjunction and disjunction, respectively.

$$p(\vec{x}) ::= p_{time}(\vec{x}) \;||\; p_{loc}(\vec{x}) \;||\; p_{rol}(\vec{x}) \;||\; p_{tru}(\vec{x}) \;||\; not\ p(\vec{x}) \;||\; p(\vec{x})\ and\ p(\vec{x}) \;||$$

$$p(\vec{x})\ or\ p(\vec{x}) \;||\; [\vec{x} = \vec{e}]; [\vec{x} \leq \vec{e}] \ldots$$

Intuition. The basic idea is that a policy specifies the actions one can perform in a given moment of time. As an example, the policy $\alpha(\vec{x}).all$ means that one is allowed to perform the action α and then everything. With this mechanism one can describe several authorization frameworks, as the ones based on action history. For instance, we can say that the action *write* can be performed only after the security relevant action *open*, by using the policy *open.write.all*.

The formal semantics of a DSA policy thus shows which the allowed sequences of actions are. Thus, it is basically able to express authorizations. On the other hand, DSAs often refer to the notion of *obligation*, whereby a party is required to perform a particular action. Indeed, while authorizing someone to do something does not necessarily imply that the right will be actually exercised. On the contrary, an obligation dictates duties, generally guarded by precedent events, namely actions or conditions. Obligations refer to something that is obliged to happen, whereas authorizations speak about something that may happen. For example,

"After 6pm, you may write" is an authorization, while "After 6pm, you must write" represents an obligation.

There would be several mechanisms to handle obligations. We decided to use a simple one as follows. At the modeling level, we may express obligations as POLPA processes of the form $q.start_{ob}.\alpha.end_{ob}$, whose meaning is "*after q, α must start and end*". In the obligation construct, q may range either over actions or predicates, and $q \neq \alpha$. In a more general way, α can be a POLPA process P, and q does not occur in P.

Some DSA patterns. The language is able to express often used patterns of DSAs. In the following, we show some examples. Sometimes, we avoid making variable assignments explicit, for the sake of readability. We also make use of constructor functions for messages. The examples range from authorizations to obligations constructs.

- A *Non disclosure* statement, like: "If B receives object o from A, then B may send o to any principal not before than 1 year" can be expressed in POLPA as
  - p(a) = [a=A and After(ini,365d)]
  - P_B = recv(a,b,o).ini:=get(ini_time).(p(a).send(B,r,o).P_B or not(p(a)).P'_B) where P'_B does not contain sending actions.
- A *Privacy* statement, like: "A will delete all personal information about B within one year of A's storing it" is an obligation and it can be expressed in POLPA as
  - P_A = ini:=get_time(ini_time). $start_{ob1}$. [Innext(ini,365)].cancel(A,$data_B$,rep_A).end_{ob1}.P'_A
- *Responsive forwarding* includes forwarding behaviors like: "B will send C each objects it receives from A within 24 hours of receiving it". This statement is an obligation.
  - P_B = recv(a,b,o).ini:=get(ini_time). [a=A]. $start_{ob1}$.[Innext(ini,24h)]. send(B,C,o).end_{ob1}.P'_B
- *Recurrence* denoting the performance of some event with regularity, like: "A will send B the latest update to object o every 24 hours". This statement is an obligation.
  - P_A = send(A,B,o).P'_A
  - P'_A = ini:=get(ini_time).$start_{ob1}$.[Eq(ini,24h)].send(A,B,update(o)).end_{ob1}.P'_A

The language can also combine similar patterns to formalize more complex agreements, and it can capture a variety of data sharing policies. Given the parameterised nature of the predicate construct, the temporal conditions in the above examples can be easily replaced by location, trust, attribute-based constraints.

Multiple rules can be expressed by combining the single rules through the parallel and the choice operators of the language. The use of the parallel operator may require the specification of a set of actions over which the single rules must synchronize.

Specifying multiple rules may raise conflicts. As an example, the *Non disclosure* and the *Responsive forwarding* statements listed above are in conflict with respect to the sending capability of B. Part of the DSA analyser component will be devoted to detect such conflicts.

Analysis methodology. Broadly speaking, the goal of the analysis is to check if two POLPA processes are equivalent, or, more often, if one process *refines*[1] the other, according to some

[1] Even if the notion of process *refinement* is commonly used in formal analysis to express relations between processes, we prefer to avoid this terminology here, since it is used also in the DSA to Policy Mapper component

notions of process equivalence and process comparison. In the literature, many definitions for those notions have been given. The choice of using one notion rather than others generally depends on the kind of property (safety / liveness) one is willing to analyse.

So called *safety* properties are those properties stating that "something bad does never happen". On the other hand, a *liveness* property expresses that "eventually, something good must happen".

The "single lane bridge problem" helps in illustrating the differences in defining the two properties. Suppose that there is a bridge that is only wide enough to permit a single lane of traffic. Consequently, cars can only move concurrently if they are moving in the same direction. A safety violation occurs if two cars moving in different directions enter the bridge at the same time. If cars arriving from the left are labelled red, and cars from the right are labelled blue, the corresponding safety property is expressed by a process stating that while red cars are on the bridge, only red cars can enter; while blue cars are on the bridge, only blue cars can enter; if the bridge is empty, either a red or a blue car may enter. On the contrary, asking a question like: "does every car eventually get an opportunity to cross the bridge?" represents an example of liveness. Thus, one is asking if it is always the case that an action is eventually executed.

It is worth noticing that safety is commonly investigated in security analysis, e.g., one may want to investigate if "the secret message will never be disclosed to the malicious user".

Within DSA analysis, checking safety properties would mean answering questions such as:

- Is it possible, for a principal covering a certain role (admin, employee, secretary...) (or enjoying some attributes, such as being trusted with a certain level) performs a certain action (e.g., write/read/print) on document data of type t: file/socket/...etc..., within a certain temporal interval, from a certain location?

It is also possible to compare agreements, and possibly revealing conflicts between them, for example when trying to combine them. This may be the case, e.g., when independent organizations want to work together, and each has an internal agreement they wish to satisfy on joint assets.

For the analysis of safety properties, it is sufficient to consider trace equivalence and trace refinement relation, defined as:

- *Traces Equivalence*: a process Q is identified with the set of traces (sequences of events) that it can perform *tr(Q)*, and Q and P are equivalent iff *tr(Q) = tr(P)*;
- *Traces Inclusion*: Q is traces included in P, denoted as $Q \leq_{tr} P$, if all the possible sequences of events that Q can perform are also possible in P. Alternatively, one may write $tr(Q) \subseteq tr(P)$, i.e., the set of traces that Q can perform is a subset of those of P.
- *Simulation relation:* Q simulates P, denoted as $P \preceq Q$, if Q matches all the movements of P.

As a simple example, we consider two processes P and Q defined as follows:

$P = \alpha . \beta .stop$ or $\alpha . \gamma .stop$

$Q = \alpha .Q'$ with $Q' = \beta .stop$ or $\gamma .stop$

---

with another meaning. Thus, to compare processes, in the following we will use other terms, like comparison, or inclusion.

The traces of P and Q are the same: α, $\alpha\ \beta$, and $\alpha\ \gamma$, thus Q and P are traces equivalent. On the contrary, Q is not similar to P. Indeed, upon consuming action α, either β or γ can still be consumed in Q, while the same choice is resolved at the beginning of the computation in P.

Traces inclusion and simulation relation are sufficient for checking safety properties, and for comparing agreements. Indeed, if P represents the correct, expected behavior as one's goal, Q, whose traces are included in P's traces, will only have safe behaviors. Thus, the analysis consists in checking a traces inclusion relation, and we need tools devoted to this aim.

At the analysis level, checking obligation properties formally requirew consideration of liveness (does the event represented by the obligation end eventually occur?), and analyzing liveness can be done by, e.g., checking the so called Failures-Divergences relation. A failure of Q is defined as a pair (s,X), where s is a trace of Q, and X is the set of events that Q can refuse to perform at that point. A divergence of Q is traces after which Q may livelock, i.e., perform an infinite sequence of silent, internal actions).

- *Failures-Divergences Relation*: Q is a failure-divergence inclusion of P, denoted as P $\leq_{fd}$ Q, if failures(Q) $\subseteq$ failures(P) and divergences(Q) $\subseteq$ divergences(P).

With this notion, one can analyse if it is possible for the specification under investigation to end up in a state where it is not possible to execute events in X. Also, detecting divergences is useful to describe "don't care situations", as a livelock.

Below, we briefly list some tools suitable for performing the kind of DSA analysis we will need in Consequence.

Tools suitable for DSA analysis.

FDR, [FDR], (Failures-Divergence Refinement) is a model-checking tool for state machines, with foundations in the theory of concurrency based around CSP, the language on *Communicating Sequential Processes* [Hoare85]. The tool allows both the verification of safety and liveness properties.

Comparisons of processes in the style of FDR are conducted also by **CIRCUS** [CW03, CGO09, OC08, CIRCUS]. Circus is a combination of Z [DW96], CSP [Hoare85], the refinement calculus [Mor94], and guarded commands [HH98]. It captures specifications, design, and programs for concurrent systems. It is very convenient for reasoning about dynamic and reactive aspects of distributed systems, and captures both safety and liveness properties.

PaMoChSA [PaMoChSA] is a model checker for computer aided verification of security protocols, based on the idea of analysing security protocols as open systems, i.e., as systems with some unspecified component representing an adversary trying to interfere with the normal execution of the protocol. The underlying theory of the checker extends the partial model checking techniques of [And95] to Crypto-CCS [Mar03, GMP08, Mar05], a process algebra that specifies cryptographic primitives and trust and reputation constructs.

The Rodin Platform [Butler07, EventB] is a platform for the development of systems in the formal Event-B [Abrial07] language in a stepwise refinement style [Back98]. Event-B is an extension of the B-method [Abrial96], which incorporates ideas from Action systems [Back89]. The semantics of Event-B are formalized in terms of *proof obligations*, which can be used both to show that the model is sound with respect to some behavioral semantics as well as prove properties of the system.

The implementation of Rodin is based on Eclipse, [Eclipse], and it provides the following facilities for the programmer:

- Design-time feedback based on type verification
- Proof obligation generation and automatic discharging of simple proofs, as well as providing an interactive environment for the programmer to manually discharge more complicated proofs.
- Since Rodin is based on Eclipse, it is possible to plug with it any other tools, such as model checkers. Pro-B [Pro-B], provides one such plug-in for the verification of linear temporal logic properties of Event-B systems.

Example case study. We consider here a simplified fragment inherited from a real University Framework Agreement, enriched with constraints about location and trust. The agreement is between the Duckburg University DUNIV and the Duckburg County University Educational Council for grants and scholarships DUEC. The agreement is stipulated with the intent of actuating an administrative cooperation between DUNIV and DUEC, by the sharing of students data, contained in the database D_BASE. For the sake of readability, in the following we deal with authorizations constructs.

We consider the following setting. Users of the database may take the role administrator and employee, both for the university and for the council, i.e., UserRole = (DuecAdmin, DuecEmpl, DunivAdmin, DunivEmpl). Student's data may assume type of: DataType = (Financial, Personal, Finalmarks). The relevant actions are Actions = (Read,Write,Comm).

Thus, the set of all the possible events is Allevents = (Read(u,d) , Write(u,d), Comm(a,b,d)) with u, a, b are variables indicating users, and d is a variable to indicate student data.

The unconstrained policy for the database is a policy that allows any of the events in the set Allevents. Formalizing this policy in POLPA, we obtain the process D_BASE defined as follows:

$$D\_BASE = read(u,d).D\_BASE \;\; OR \;\; write(u,d).D\_BASE \;\; OR \;\; comm(a,b,d).D\_BASE$$

Now, we add constraints according to the Framework Agreement, saying that:

- Any user in the role of DuecAdmin may either write financial data or read personal data;
- Any user in the role of DunivAdmin may either write final marks data or read personal data;
- Any user in the role of DuecEmpl may read personal data if inside DUEC building;
- Any user in the role of DunivEmpl may read personal data if his/her reputation > 10.

According to these constraints, the process describing the DUEC specification is:

```
DUEC =
HasRole(u,DuecAdmin). u:=duecAdm.
                (HasRole(d,Financial). d:=fnclData.Write(duecAdm, fnclData).DUEC
                                or
                 HasRole(d,Personal). d:=prslData.Read(duecAdm, prslData).DUEC
                )
                                or
HasRole(u,DuecEmpl). u:=duecEmp.
```

(HasRole(d,Personal) and IsInArea(duecEmp,Area)).
d:= prslData.Read(duecEmp, prslData).DUEC

First, the specification evaluates the role of the user. If the user is a DUEC administrator, then the expression *duecAdm* is assigned to variable *u.* Then, the data type of the student data is evaluated. If the data are of type *Financial, then* the expression *fnclData* is assigned to variable *d,* otherwise, if the data are of type *Personal,* then the expression *prslData* is assigned to variable *d.* According to the data type, either action *Write* or *Read* can be performed by the DUEC administrator.

If the user is a DUEC employee, then the expression *duecEmp* is assigned to variable *u*. Then, if the type of the student data is *Personal*, and the user is in the neighborhood of area *Area*, then the expression *prslData* is assigned to variable *d* and the user may perform action *Read.*

The specification for the DUNIV process is similar.*DUNIV =*
HasRole(u,DunivAdmin). u:=dnvAdm
(HasRole(d,Finalmarks). d:=FMarks.Write(dnvAdm,FMarks).DUNIV
or
HasRole(d,Personal). d:=prslData.Read(dnvAdm,prslData).DUNIV
) or
HasRole(u,DunivEmpl).u:=dnvEmp.
(HasRole(d,Personal) and RepMaxOf(dnvEmp,10)).
d:=prslData.Read(dnvEmp,prslData).DUNIV

The user policy ALL_USER is defined as the parallel composition between DUEC and DUNIV.

ALL_USER = DUEC par DUNIV

The user policy and the unconstrained database policy shares the all_events interface:

$SYSTEM = ALL\_USER\ par_{\{all\_events\}}\ D\_BASE$

To validate a model, we can define events that describe violations to properties that we aim at assuring. Then, one may check if the traces of the system with no possibility of performing those events are included in the system under evaluation. Alternatively, we may check the negation of the trace inclusion relation, as in the following example. Let the reader suppose that the investigated property is

"There do not exist members of Duniv.Empl who can write Final Marks data"

To prove this, we first define the bad event Test:

Test = Write(dnvEmp,Fmarks)

Then, we check if the event Test is included, or not, into the specification SYSTEM:

$not\ (Test \subseteq SYSTEM)$

The outcome of such an analysis can be obtained in few seconds by a model checker like FDR, in its traces refinement model.

How the DSA analyser meets the requirements

The DSA analyzer satisfies requirement An-R1, since it takes as input a formal specification, and a formal property, and it returns an answer about the fulfillment of the property, according to some notion of equivalence. Clearly, as usual for complex properties and models, the verification/analysis problem has theoretical limitations that would not allow us to give an answer in all the cases. Thus, we will also investigate suitable abstraction that can provide useful feedbacks to the DSA analyzer user. For requirement AN-R2: both at a modeling and at a verification level, the proposed solution will be able to manage concurrent systems. Also, the language is flexible enough to deal with DSAs peculiarities, such as time, location, trust…etc., through the introduction of predicates. In the analysis phase, we will consider the tools that are able to address those requirements. The proposed solution deals with formal models and methodologies, thus fulfilling requirement An-R3. In our opinion the framework should be flexible enough to define, classify and parameterize *patterns* of properties, creating in this way a kind of taxonomy of the DSAs relevant properties (An-R5). Satisfaction of the dialogue requirement, An-R4, is not explicitly resolved here since it concerns relationships with a number of other components within the system. Further work on this requirement will be addressed during the next phase of the project.

4.3. DSA to Policy Mapper

The DSA to Policy Mapper is an architectural component. It has a dual function: it permits the system to map a global DSA policy into several local ones, both specified at high level of abstraction, and to refine each local policy from high level to a lower (enforceable) one.

A graphical representation of the structure of DSA to Policy Mapper component is given in Figure 8.

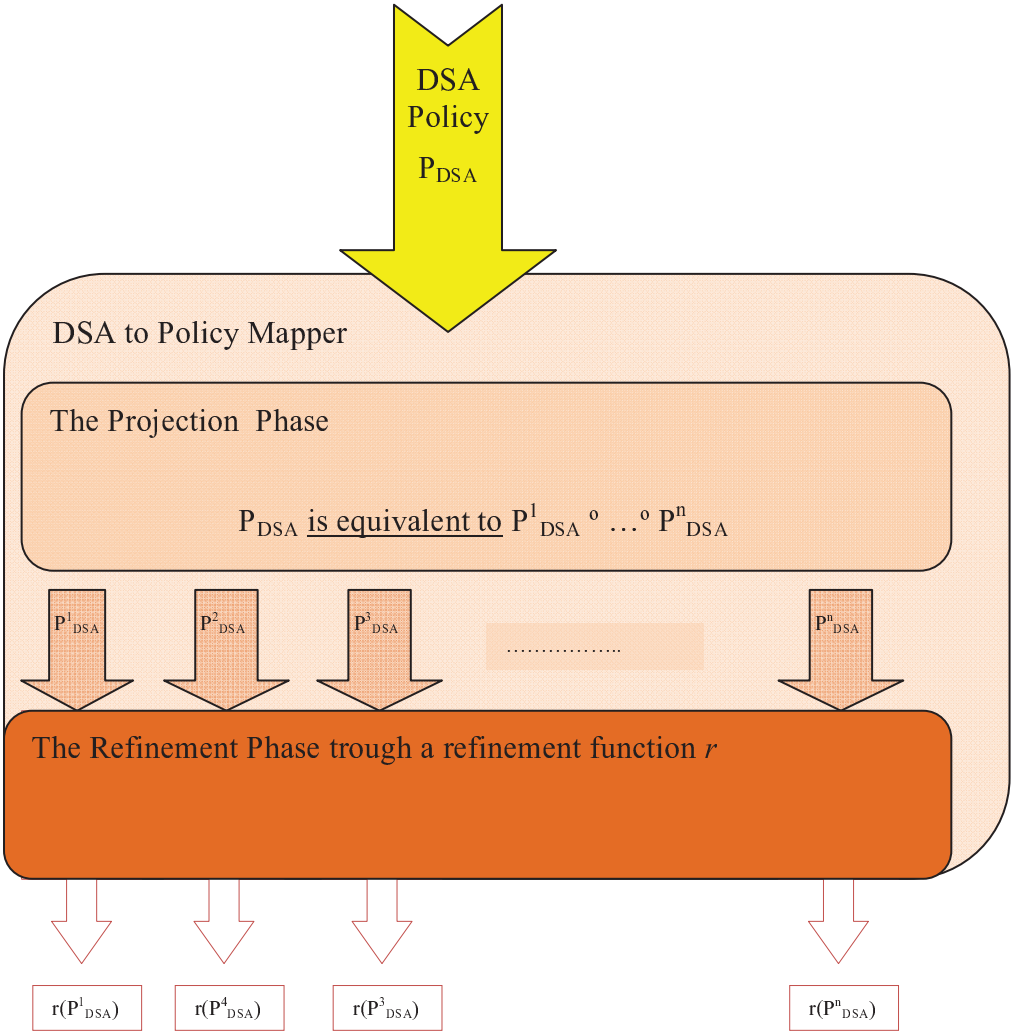

Figure 8. How the DSA to Policy Mapper works.

The Projection phase

The first step performed by the DSA to Policy Mapper is the decomposition of a given DSA global policy into several local policies. Each local policy expresses the DSA policy that each agent involved in the agreements has to satisfy.

According to the choice of the language for expressing DSA policies, the projection derives directly from how the global specification of the data sharing agreement is given. As a matter of fact, in the POLPA language the Data Sharing Agreements of a distributed system is specified as a parallel composition of the Data Sharing Agreements of each component.

As a simple example, let us consider the following DSA policy:

"B will send C each object it receives from A within 24 hours of receiving it"

Using POLPA, this global DSA policy between the agents A, B, and C, is expressed as follows:

$$P_{ABC} \doteq P_A \; par \; P_B \; par \; P_C$$

where

$P_A \doteq send(A,B,o).stop$

$P_B \doteq recv(a,b,o).[innex(24h)\ and\ a=A\ and\ b=B].send(B,C,o).stop$

$P_C \doteq recv(b,c,o).stop$

It is worth noticing that we already have the description of the policy in term of local behavior of each component directly from the description of the global policies thank to the usage of the process algebra based language we have chosen.

Hence, the DSA to Policy Mapper firstly works by taking a global DSA policy expressed in the POLPA language, P_{DSA} for short, and then, knowing the number of agents involved in the system, here denoted as $U_1,\ldots, U_n$, it obtains P^i_{DSA} for i=1,…,n s.t.

$$P_{DSA} \text{ is equivalent to } P^1_{DSA} \circ \ldots \circ P^n_{DSA}$$

and each U_i is compliant with the referred P^i_{DSA}. In the above notation, $\circ$ represents a composition operator among policies.

Now "is equivalent to" and "is compliant with" have to be further explored in this context.

For that reason we recall the notion of behavioral equivalences. In particular we refer to the notion of *weak simulation* and *weak bisimulation.*

The basic one-step transitions are extended to τ - abstracting transitions in the usual way:

$$P \overset{\sigma}{\Rightarrow} P',\ P \overset{\alpha 1 \ldots \alpha n}{\Rightarrow} P',\ P \xrightarrow{\tau} * \xrightarrow{\alpha 1} \xrightarrow{\tau} * \ldots \xrightarrow{\tau} * \xrightarrow{\alpha n} \xrightarrow{\tau} * P'$$

where σ is a sequence of actions in *Act*, *i.e.,* σ in *Act**.

We can give the following definition.

Definition. *Let* $(E, Act, \rightarrow)$ *be an LTS over the set of actions Act, and let R be a binary relation over E. Then R is called* weak simulation*, denoted by* $\preceq$ *over* $(E, Act, \rightarrow)$ *if and only if, whenever* $(P,Q) \in R$ *we have:*

$$if\ P \xrightarrow{\alpha} P'\ then\ \exists Q'\ such\ that\ Q \overset{\alpha}{\Rightarrow} Q'\ and\ (P',Q') \in R$$

Recalling that the converse R^{-1} of any binary relation R is the set of pairs (Q, P) such that (P,Q) are in R, we give the following definition.

Definition. *A* weak bisimulation *is a relation R such that both R and* R^{-1} *are weak simulations, i.e., if for each* $(P,Q) \in R$ *and for each a* $\in Act$*:*

$$if\ P \xrightarrow{\alpha} P'\ then\ \exists Q'\ such\ that\ Q \overset{\alpha}{\Rightarrow} Q'\ and\ (P',Q') \in R$$

$$if\ Q \xrightarrow{\alpha} Q'\ then\ \exists P'\ such\ that\ P \overset{\alpha}{\Rightarrow} P'\ and\ (P',Q') \in R$$

Two processes P and Q are weakly bisimilar if there exists a bisimulation R such that $(P,Q) \in R$. The maximal weak bisimulation is $\approx$ which is the union of every weak bisimulation. It is easy to check that this relation is still a weak bisimulation and moreover is reflexive, symmetric and transitive.

Hence, in our approach, the notion of "is equivalent to" is modeled by the weak bisimulation relation while the notion of "is compliant to" by the notion of weak simulation. The choice of this behavioral equivalences for comparing the behavior of an agent and its policies, fulfils the requirement Map-R1 of the DSA to Policy Mapper. As a matter of fact, the decomposition is made in such a way it is as weaker as possibly to be compliant with the DSA policies.

Although when we have a global description without components of the DSA, by appropriately extending the theory of **decomposition of policies** (see [GMM08, LX91, MM08]), we plan to be able to obtain local predicate $\mathbf{p_{Ui}(x)}$ as projection of the global ones.

The Refinement phase

In the development of software components, it is quite often required to relate systems belonging to different abstraction levels. In particular, a good deal of work has been done to study the steps that proceed from the specification level (*high level* of abstraction) to the implementation level (*low level* of abstraction). This transition is referred as *refinement*.

In formal methods, *program refinement* is the transformation of an abstract high level formal specification into a concrete low-level (executable) program. *Action refinement* converts a specification of an action on a system into an implementable program (*e.g.*, a procedure).

Here we recall some notions introduced in [GRZ01, RG97] on action refinements and some of its properties. Moreover we show our result that allows the passage from a DSA policy expressed at high specification level to DSA policy expressed at enforceable level.

The main idea is that, given a *refinement function r* mapping abstract actions to concrete processes, where the notion of *abstract* and *concrete* are accompanied by a change of alphabet, the implementation of a specification *S* is given by the *syntactic substitution* of a concrete process $r(\alpha)$ for actions α in *S* or by the *semantic substitution* of the model of concrete processes $r(\alpha)$ for actions a in the semantics of *S*.

For our purpose, in order to avoid unnecessary complications, we single out the fragment of refinement terms, **R**, that can be used as the refinement of abstract actions as follows:

$$P ::= stop \;||\; all \;||\; \alpha(\vec{x}).P \;||\; P_1 \; or \; P_2$$

Moreover, as derivate process, we consider $P::= P;P$, i.e., the sequence of two processes.

Let Act_A and Act_C be respectively the set of abstract and concrete actions, then the refinement function *r* is of the form $r : Act_A \rightarrow \mathbf{R}_C$ where $\mathbf{R}_C$ is the set of term on concrete actions and it is ranged over by $P_C, Q_C, \ldots$. In order to distinguish, we denote by $P_A, Q_A, \ldots$: the term on the abstract actions.

It is important to note that we overload notation by using $r(E)$ to denote a refined process *E*, *i.e.*, given a process *E*, the process $r(E)$ is the result of the application of the function *r* to each action in $Sort(E)$.

In order to preserve the atomicity the images of *r* are constrained to be:

– Non-empty, *i.e.*, a visible abstract action cannot simply disappear during refinement.

– Eventually terminating, *i.e.*, the refinement of a given action cannot "get stuck" during execution.

– Visible, *i.e.*, no τ actions are performed.

It is important to note that some refinement functions does not contain a degree of *confusion* since we are considering Act_A and Act_C as disjoint sets.

The Vertical Bisimulation relation

One of the central concepts of the refinement procedure is the notion of *vertical implementation relation* parameterized w.r.t. a refinement function *r*. It describes the relation that exists between a process expresses at high specification level and its refinement at a lower specification level obtained through the refinement function *r*. The notion of vertical

relation is associated with the notion of horizontal, or basis, relation. It represents the relation that exists between two terms at the same level of abstraction.

The notion of vertical implementation was introduced in [GRZ01, RG97]. Here we are interested to a particular vertical implementation: The *vertical bisimulation*. In particular, in [MM09], we have study the vertical bisimulation relation by considering the weak simulation as horizontal. According to [GRZ01, RG97], the vertical bisimulation is a congruence w.r.t. the operators of the considered language.

An important extension with respect to the standard notion of bisimulation is that into the vertical bisimulation we have to take in account that in any given state the implementation, there may be associated refined actions whose execution has not yet terminated. These actions are collected in a set of *residual refinements*, that is a multiset of non terminated refinements terms. Hence, terminated processes do not contribute to the residual set.

The vertical bisimulation is denoted by $\preceq^r$. It is formed of three components:

- a strict *down-simulation*, in which each abstract move must be matched by a sequence in the implementation. It regards only runs of the refinement in which the residual *R* is empty. The intermediate states of the implementation are not taken into account;
- an *up-simulation* in which each move of the implementation should find a justification either as the initial action of a new refined action or as a continuation of a pending refinement. Indeed, it is a dual notion w.r.t. the strict down simulation and it investigates what happens in any given state of the implementation, *i.e.*, there may be associated refined actions whose execution has not yet terminated. It has to maintain the multiset of residual refinement: either the implementation's move corresponds to the initial concrete action of a refined abstract action, or it is an action of a residual refinement;
- a *residual simulation* requiring that each move of the pending refinements must be matched by the implementation, without the specification moving at all. This implies that the pending refinements can be "worked off" in any possible order or indeed in parallel by the implementation.

The interested reader may find a complete definition and construction of residual sets in [GRZ01]. It is not a matter that is addressed further in this deliverable.

Some interesting rules. According to [RG97], some interesting relation between the operators of the language and the vertical bisimulation relation holds. Here we recall some of them that are useful for our framework:

$$\frac{\alpha \preceq^r r(\alpha)P_A \preceq^r P_C}{\alpha.P_A \preceq^r r(\alpha);P_C} \qquad (2)$$

that guarantees the vertical bisimulation relation is a congruence w.r.t. the prefix operator.

A similar rule holds also for the synchronization operator:

$$\frac{P_A \preceq^r P_C Q_A \preceq^r Q_C}{P_A \|_A Q_A \preceq^r P_C \|_{r(A)} Q_C} \text{ r preserves and is distinct on A} \qquad (3)$$

where $\tilde{r}$ (A) is the set of refined actions obtained by applying r to each action in A.

How the DSA to Policy Mapper meets the requirements

The projection phase of the DSA to Policy Mapper meets the requirement Map-R1. According to POLPA, a DSA Policy is given, at specification level, as a composition of local DS policies; each of them refers to a particular agent of the agreement satisfying that policy.

Requirements Map-R2 and Map-R3 are refinement functions; we are able to map a DSA policy from the high level to the enforceable level in a formal way. Moreover, under some additional assumptions on the refinement function r, e.g., that it is allowable, and on the horizontal relation we chose (in [MM09] we have considered the weak simulation as the basis of the considered vertical bisimulation), the requirement Map-R3 is satisfied, *i.e.*, the security properties satisfied at the high level are preserved also at the low one.

Finally, it is worth noticing that further investigation may be needed for the definition of a tool suitable for implementing the action refinement theory described above. Indeed, to the best of our knowledge, such tools are not currently available.

4.4. DSA Lifecycle

The DSA Lifecycle Manager will be implemented via a SharePoint site and developing a set of customized workflows using its built-in workflow capabilities.

Finally a necessary integration development will be performed to integrate the SharePoint portal and other related tools such as the DSA authoring component.

How the DSA Lifecycle meets the requirements

Here is the reminder of the major Lifecycle requirements from Section 2.4 together with the description of how it will be matched.

Requirement ID	Requirement Description and how SharePoint meets it
ReqDSAL01	Document repository. The DSA Workflow must act as a document repository, and must provide all traditional features of such software. A (non exhaustive) list of features includes: access control based on user roles, managing of users' and shared workspaces, possibility of organizing workspaces in hierarchical structures, versioning of documents, handling of documents metadata. *Portal is one of the major functionality groups for SharePoint. It includes content management system for documents and web content; full security solution; and extensive search capabilities*

Requirement ID	Requirement Description and how SharePoint meets it
ReqDSAL02	Web based interface. Users are able to perform all operations from a web interface. No software installation is required on user's clients. The web interface is also required in order to potentially allow users belonging to different organization to cooperate in the creation of a DSA (for example attorneys from the various contracting parties may need to access the same document repository in order to cooperate in the creation of a DSA). *SharePoint Server delivers complete web-based experience; and web-based interface is the major way of communicating with the product. In other words all SharePoint features are accessible through the web-interface, and this interface can be customized by built-in or external tools.*
ReqDSAL03	Document workflow capabilities. The DSA Lifecycle manager must enable the definition and enactment of document workflows, where a document evolves through a number of steps. At each step human intervention may be required (for example, editing or approving the document status). The workflow capabilities must be customizable, in order to meet the exact DSA lifecycle that the Consequence project will define. *SharePoint workflow capabilities were described in the section 3.4.*
ReqDSAL04	Extensibility. The DSA Lifecycle manager must integrate with external tools, such as the DSA Analysis component, or tools that may support the (semi)-automatic conversion from a natural language DSA to a controlled language DSA to a high-level formal policy and to the embeddable policies. *SharePoint has an extensive Application Programm Interface (API) set, that allows tight and seamless integration with different software packages, Such integration will be a part of the development process during the further stages of WP2 and WP1.*

4.5. DSA Trust Manager

Here, we first present the RT language in its basic version, RT0, and then we show through a simple example how we can manage trust relationships between principals (both individuals and groups) operating on DSAs.

RT0 supports 4 kinds of credentials.

- Simple member credentials, like: ***Epub.discount*** ← ***Alice*** stating that the electronic shop of on line publications *Epub* offers to *Alice* the special price *discount.*
- Simple containment credentials, like: ***Epub.discount*** ← ***IIT.researcher*** stating that on-line shop *Epub* offers to all the *researchers* of organization *IIT* the special price *discount.*
- Linking containment credentials, like **Epub.discount** ← **CNR.institute.researcher** stating that on line shop *Epub* grants to everybody who is a researcher of some institute of the National Research Council *CNR* the special price *discount.*
- Intersection credentials, like: **Epub.discount** ← **IIT.researcher** ∩ **IIT.adminstaff** stating that *Epub* offers to both *researchers* and *administrative* staff of *IIT* the special price *discount.*

Combining use of more than one credential may serve for proving access control properties, like, e.g., *authorization*, as in the following:

1. Epub.discount ← CNR.institute.researcher
2. CNR.institute ← IIT
3. IIT.researcher ← Alice

The first credential states that the e-shop offers discounts to every researchers of CNR institutes. The second credentials states that IIT is a CNR institute. The third credentials states that Alice is a researcher of IIT. The opportune combination of the three credentials results in the fact that Alice will be authorized to obtain a special price.

Now, consider the following scenario, illustrated in Figure 9. There is a machine representing the DSA authoring tool. We want to answer the following question: who may use the DSA authoring tool?

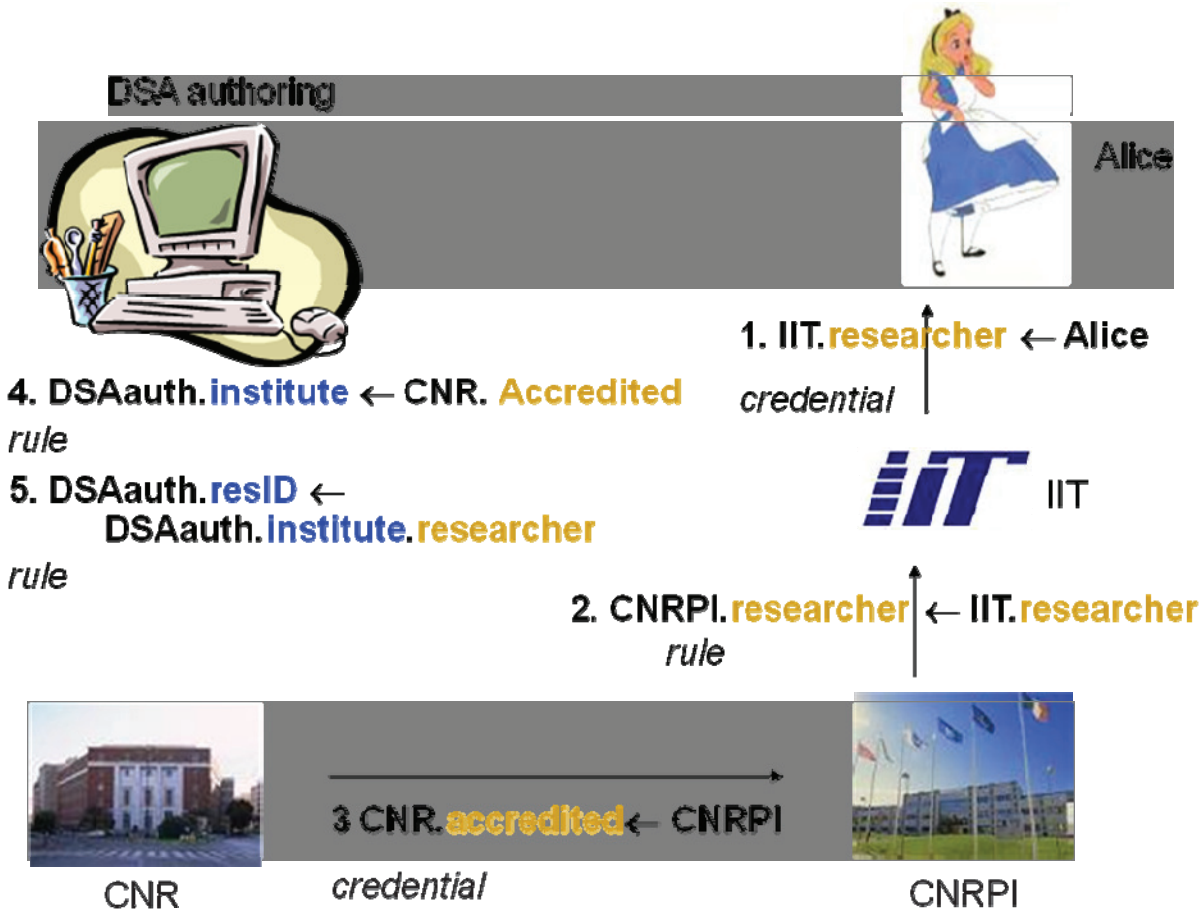

Figure 9. Credentials and access rules in RT.

The access policy at the "DSA authoring" machine is "you can access, if you have role *resID* granted by *DSA-Auth*.

It could be easily checked that Alice's access is granted. Indeed, from credential 1 and access rule 2, we obtain the fact that Alice is a researcher of CNR in Pisa, expressed by the newly formed credential *a*: CNRPI.researcher ← Alice.

From credential 3 and access rule 4, CNR in Pisa is considered an institute by the DSA authoring machine, and this is expressed by the newly formed credential *b*: DSAauth.institute ← CNRPI.

Finally, we can combine *a* and *b* through rule 5, by obtaining that Alice is granted with identifier resID by the DSA Authoring.

How the DSA Trust Manager meets the requirements

The proposed approach allows for a distributed trust management in a DSAs context. The process of chain discovering and permissions solving can be made automatic exploiting the implementation of an algorithm that calculates the minimal set of simple members credentials, starting from two set of available credentials, i.e., simple and not simple credentials, [CMMPV07].

Conclusions

This document describes the work done in WP2 on methodologies and tools for the DSA Infrastructure produced after the 1st year of the project.

The work has been carried on as planned in accordance with the other WPs and overall Consequence project architecture.

We expect that DSA Infrastructure will be further elaborated and expanded during the project lifetime. Indeed, we also expect changes in accordance to the test beds scenario development.

As a final remark, the work has been carried on without significant deviations from the expectations.

References

[AAMM08] B. Aziz, A. Arenas, F. Martinelli, I. Matteucci, and P. Mori. Controlling usage in business process workflows through fine-grained security policies. In *TrustBus*, pages 100 117, 2008.

[Abrial96] J.R. Abrial. *The B-Book: Assigning Programs to Meanings*. Cambridge University Press, 1996.

[Abrial07] J.R. Abrial and S. Hallerstede. *Refinement, Decomposition and Instantiation of Discrete Models: Application to Event-B*. Fundamentae Informatica, 77(1-2), 2007.

[Advocado] http://www.advogato.org/.

[Alfresco] http://www.alfresco.com/.

[AlfrescoEE] http://www.alfresco.com/products/networks/compare/ .

[AlfrescoLE] http://www.alfresco.com/products/networks/compare/files/alfresco_enterprise-community_comparison_0508.pdf.

[And95] H. R. Andersen, *Partial model checking*, in: *LICS '95: Proceedings of the 10th Annual IEEE Symposium on Logic in Computer Science* (1995), p. 398.

[AVISPA] http://www.avispa-project.org/.

[Back89] R.J. Back. *Refinement Calculus II: Parallel and Reactive Programs*. In J. W. deBakker, W. P. deRoever, and G. Rozenberg, editors, Stepwise Refinement of Distributed Systems, volume 430 of Lecture Notes in Computer Science, pages 67–93, Mook, The Netherlands, May 1989. Springer-Verlag.

[Back98] R.J. Back and J. von Wright. *Refinement Calculus: A Systematic Introduction*. Graduate Texts in Computer Science. Springer-Verlag, 1998.

[BBD05] 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.

[BBDF03] L. Bettini, V. Bono, R. D. Nicola, G. L. Ferrari, D. Gorla, M. Loreti, E. Moggi, R. Pugliese, E. Tuosto, and B. Venneri. The klaim project: Theory and practice. In *Global Computing*, pages 88–150, 2003.

[BDP05] C. Bodei, P. Degano, and C. Priami. Checking security policies through an enhanced control flow analysis. *Journal of Computer Security*, 13(1):48–85, 2005.

[Bin04] J. Bing, G. Brændeland, D. Chadwick, J. Claessens, T. Dimitrakos, D. Golby, J. Haller, A. Jones, C. Keser, M. S. Lund, E. Lupu, T. Mahler, L. Martino, B. Matthews, X. Parent, C. Geuer-Pollmann, B. Sadhighi, J. Sairamesh, L. Schubert, K. Stølen, N. Tuptuk, E. Weizenböck, S. Wesner, K. Wulf, Y. Karabulut, T. G. Zaragoza. TrustCom: State of the Art Evaluation – phase 1, 2004.

[BL04] A. K. Bandara, E. Lupu, J. D. Moffett, and A. Russo. *A goal-based approach to policy refinement.* In POLICY, pages 229–239. IEEE Computer Society, 2004.

[Bla02] B. Blanchet. From secrecy to authenticity in security protocols. In *SAS*, pages 342–359, 2002.

[Bla03] B. Blanchet, P. Cousot, R. Cousot, J. Feret, L. Mauborgne, A. Min, D. Monniaux, and X. Rival. A static analyzer for large safety-critical software. In *Proc. SIGPLAN*, pages 196–207, 2003.

[Bla04] B. Blanchet. Automatic proof of strong secrecy for security protocols. In *IEEE Symposium on Security and Privacy*, pages 86–, 2004.

[Bry05] J. Bryans. *Reasoning about XACML Policies using CSP*. In Proceedings of the 2005 Workshop on Secure Web Services (SWS'05), Fairfax, USA, November 2005, ACM Press, 28-35.

[Bru94] G. Bruns. *Applying process refinement to a safety-relevant system*. Technical Report Tech. Report ECS-LFCS-94-287, Laboratory for the Foundations of Computer Science, University of Edinburgh, April 1994.

[Butler07] M. Butler and S. Hallerstede. *The Rodin Formal Modelling Tool.* BCS-FACS Christmas 2007 Meeting - Formal Methods in Industry, London, December 2007.

[Cas98] C. Castelfranchi, R. Falcone. Principles of Trust for MAS: Cognitive Anatomy, Social Importance and Quantification. In Proceedings of the Third International Conference on Multi-Agent Systems (ICMAS'98), Paris, France, July 1998, IEEE Computer Society,72-79.

[CCDE05] J. G. Cederquist, R. Corin, M. A. C. Dekker, S. Etalle, and J. I. den Hartog. An audit logic for accountability. In *POLICY*, pages 34–43, 2005.

[CCDE06] C. N. Chong, R. Corin, J. Doumen, S. Etalle, P. H. Hartel, Y.W. Law, and A. Tokmakoff. Licensescript: a logical language for digital rights management. *Annales des T´el´ecommunications*, 61(3-4):284–331, 2006.

[CCDE07] J. G. Cederquist, R. Corin, M. A. C. Dekker, S. Etalle, J. I. den Hartog, and G. Lenzini. Audit-based compliance control. *Int. J. Inf. Sec.*, 6(2-3):133–151, 2007.

[CDMS99] I. Cervesato, N. Durgin, P. D. L. J. C. Mitchell, and A. Scedrov. A Meta-Notation for Protocol Analysis. In *Proc. CSFW-12*, pages 55–69. IEEE Computer Society Press, 1999.

[CDH04] R. Corin, S. Etalle, J. I. den Hartog, G. Lenzini, and I. Staicu. A logic for auditing accountability in decentralized systems. In *Formal Aspects in Security and Trust*, pages 187–201, 2004.

[CGO09]C. G. de Castro A. C. Gurgel and M. V. M. Oliveira. Tool Support for the ***Circus*** Refinement Calculus. In *ABZ Conference*, Lecture Notes in Computer Science. Springer-Verlag, 2008. To appear.

[Che01] R. Chen, W. Yeager. Poblano: A Distributed Trust Model for Peer-to-Peer Networks. Technical Report, Sun Microsystems., 2001.

[CIRCUS] http://www.cs.york.ac.ok/circus.

[CK05] S. Cattani and M. Kwiatkowska. *A refinement-based process algebra for timed automata*. Form. Asp. Comput., 17(2):138–159, 2005.

[CMG04] B. Chess and G.McGraw. Static analysis for security. *IEEE Security and Privacy*, 2(6):76–79, 2004.

[COSoDIS] http://www.ifi.uio.no/cosodis/.

[Coverity] http://coverity.com.

[Coq] http://coq.inria.fr/.

[CW07] B. Chess and J.West. *Secure Programming with Static Analysis*. Addison-Wesley Software Security Series, 2007.

[DFP00] R. D. Nicola, G. L. Ferrari, and R. Pugliese. Programming access control: The klaim experience. In *CONCUR*, pages 48–65, 2000.

[DFPV00] R. D. Nicola, G. L. Ferrari, R. Pugliese, and B. Venneri. Types for access control. *Theor. Comput. Sci.*, 240(1):215–254, 2000.

[DL93] A. Dardenne, A. V. Lamsweerde, and S. Fickas. *Goal-directed requirements acquisition*. In Science of Computer Programming, pages 3–50, 1993.

[Dim01] T. Dimitrakos. System Models, e-Risk and e-Trust. In Proceedings of the First IFIP Conference on E-Commerce, E-Business, E-Government: Towards The ESociety: E - Commerce, E-Business, and E-Government (I3E 2001), Zurich, Switzerland, October 2001, Kluwer, 45-58.

[Dro04] M. Drouineaud, M. Bortin, P. Torrini and K. Sohr. A First Step Towards Formal Verification of Security Policy Properties for RBAC. In Proceedings of the Fourth International Conference on Quality Software (QSIC'04), Braunschweig, Germany, September 2004, IEEE Computer Society, 60-67.

[DW96] J. Davies and J. Woodcock . *Using Z: Specification, Refinement and Proof*. Prentice Hall International Series in Computer Science. 1996.

[EC00] H.-D. Ehrich and C. Caleiro. Specifying communication in distributed information systems. *Acta Inf.*, 36(8):591–616, 2000.

[Eclipse] http://www.eclipse.org/.

[Ell99a] C. Ellison. SPKI Requirements. Internet Engineering Task Force (IETF) Request For Comments. http://tools.ietf.org/html/rfc2692. 1999.

[Ell99b] C. Ellison, B. Frantz, B. Lampson, R.Rivest, B.Thomas, and T. Ylonen. SPKI Certificate Theory. Internet Engineering Task Force (IETF) Request For Comments. http://tools.ietf.org/html/rfc2693. 1999.

[EventB] http://www.event-b.org/platform.html.

[FDR] www.fsel.com.

[FG97] R. Focardi and R. Gorrieri. The compositional security checker. A tool for the verification of information flow security Properties. *IEEE TSE*, 27(3):550–571, 1997.

[FGM00] R. Focardi, R. Gorrieri, and F. Martinelli. Non interference for the analysis of cryptographic protocols. In *Proc. ICALP'00*, volume LNCS 1853, pages 354–372. Springer, 2000.

[Gam88] D. Gambetta. Can We Trust Trust?. In Trust: Making and Breaking of Cooperative Relations, Blackwell Publishers, Oxford, 1988, 213-237.

[Gel85] D. Gelernter. Generative communication in linda. *ACM Trans. Program. Lang. Syst.*, 7(1):80–112, 1985.

[GMM08]R. Gorrieri, F. Martinelli, and I. Matteucci. Towards information flow properties for distributed systems. In *In Proceedings of the 3rd VODCA Views On Designing Complex Architectures*, 2008. To appear.

[GMP08] R. Gorrieri, F. Martinelli and M. Petrocchi. Formal models and analysis of secure multicast in wired and wireless networks. In Journal of Automated Reasoning. To appear.

[Gol04] Jennifer Golbeck, James A. Hendler. Accuracy of Metrics for Inferring Trust and Reputation in Semantic Web-Based Social Networks, In Proceedings of the Fourteenth International Conference on Engineering Knowledge in the Age of the Semantic Web (EKAW 2004), Whittlebury Hall, UK, October 2004, Springer Lecture Notes in Computer Science 3257, 116-131.

[GP03] D. Gorla and R. Pugliese. Resource access and mobility control with dynamic privileges acquisition. In *ICALP*, pages 119–132, 2003.

[GP04] D. Gorla and R. Pugliese. Dynamic management of capabilities in a network aware coordination language. Technical report, 06/2004, Dip Informatica, Univ. di Roma ”La Sapienza, 2004.

[Gra00] T. Grandison and M. Sloman. A Survey of Trust in Internet Applications. In IEEE Communications Surveys and Tutorials, 3(4), September 2000.

[GRZ01] R. Gorrieri, A. Rensink, and M. A. Zamboni. *Action refinement.* In Handbook of Process Algebra, pages 1047–1147. Elsevier, 2001.

[Her06] R. Hermoso, H. Billhardt and S. Ossowski. Integrating Trust in Virtual Organisations. In Proceedings of the AAMAS06 Workshop on Coordination, Organization, Institutions and Norms in agent systems (COIN), Hakodate, Japan, May 2006.

[HNNP08] R. R. Hansen, F. Nielson, H. R. Nielson, and C. W. Probst. Static validation of license conformance policies. In *First International Workshop on Advances in Policy Enforcement (APE'08)*. IEEE Computer Society, 2008.

[HH98] C.A.R. Hoare and J. He. *Unifying Theories of Programming*. Prentice-Hall, 1998.

[Hoare85] C. A. R. Hoare. *Communicating Sequential Processes*. Prentice-Hall, Englewood Cliffs, NJ,1985.

[Hol97] G. Holzmann. The model checker SPIN. *IEEE Transactions on Software Engineering*, 23:279–295, 1997.

[HPN06] R. R. Hansen, C. W. Probst, and F. Nielson. Sandboxing in myklaim. In *ARES'06: Proceedings of the First International Conference on Availability, Reliability and Security*, pages 174–181, Washington, DC, USA, 2006. IEEE Computer Society.

[Isabelle] http://www.cl.cam.ac.uk/research/hvg/Isabelle/index.html

[JJ01] J. Jürjens. *Secrecy-preserving refinement*. In FME '01: Proceedings of the International Symposium of Formal Methods Europe on Formal Methods for Increasing Software Productivity, pages 135–152, London, UK, 2001. Springer-Verlag.

[Jon99] S. Jones. TRUST-EC: Requirements for Trust and Confidence in ECommerce. European Commission, Joint Research Centre, 1999.

[Kam03] S.D. Kamvar, M.T. Schlosser and H. Garcia-Molina. The Eigentrust Algorithm for Reputation Management in P2P Networks. In Proceedings of the Twelfth international Conference on World Wide Web (WWW '03), Budapest, Hungary, May 2003, ACM Press, 640-651.

[Kin98] A. Kini, J. Choobineh. Trust in Electronic Commerce: Definition and Theoretical Consideration. In Proceedings of the Thirty-first International Conference on System Sciences (HICSS 1998), Hawaii, USA, January 1998, IEEE Computer Society, 51-61.

[KL04] V. W. Kevin Carey, David Lewis. *Automated policy-refinement for managing composite services*. In M-Zones White Paper June 04, whitepaper 06/04, Ireland, June 2004.

[LMW07] N. Li, J. Mitchell, and W. H. Winsborough. Design of a role-based trust management framework. In *S&P*, pages 114–130. IEEE, 2002.

[Low96] G. Lowe. Breaking and fixing the Needham-Schroeder public-key protocol using FDR. In *Proc. TACAS'96*, volume LNCS 1055, pages 147–166. Springer, 1996.

[LPLK03] M. Lorch, S. Proctor, R. Lepro, D. G. Kafura, and S. Shah. First experiences using xacml for access control in distributed systems. In *XML Security*, pages 25–37, 2003.

[LX91] K. G. Larsen and L. Xinxin. *Compositionality through an operational semantics of contexts.* Journal of Logic and Computation, 1(6):761–795, Dec. 1991.

[Mar03] F. Martinelli. Analysis of security protocols as open systems. *TCS*, 290(1):1057–1106, 2003.

[Mar05] F. Martinelli. Towards an integrated formal analysis for security and trust. In FMOODS, volume LNCS 3535, pages 115--130. Springer, 2005.

[MCJ97] W. Marrero, E. Clarke, and S. Jha. Model checking for security protocols. Technical Report CMU-CS-97-139, Carnegie Mellon University, 1997.

[Mea95] C. Meadows. Formal verification of cryptographic protocols: a survey. In *Proc. ASIACRYPT'94 – Advances in Cryptology*, volume LNCS 917, pages 135–150. Springer, 1995.

[MM08] F. Martinelli and I. Matteucci. Synthesis of local controller programs for enforcing global security properties. In *ARES*, pages 1120–1127, 2008.

[MM09] F. Martinelli and I. Matteucci. Action Refinement for Security Properties Enforcement. In *Proceedings of the International Symposium on Engineering Secure Software and Systems*, 2009. To appear.

[MP07] F. Martinelli and M. Petrocchi. On relating and integrating two trust management frameworks. *Electr. Notes Theor. Comput. Sci.*, 168:191–205, 2007.

[NAM06] S. Naqvi, A. Arenas and P. Massonet. *Deriving Policies from Grid Security Requirements Model*. In Proceedings of the CoreGRID Integration Workshop, Krakow, Poland, 2006.

[NFP98] R. D. Nicola, G. L. Ferrari, and R. Pugliese. Klaim: A kernel language for agents interaction and mobility. *IEEE Trans. Software Eng.*, 24(5):315–330, 1998.

[OC08] M. V. M. Oliveira and A. L. C. Cavalcanti. ArcAngelC: a Refinement Tactic Language for Circus. *Entcs*, **214**:203 - 229, 2008.

[Pag98] L. Page, S. Brin, R. Motwani and T. Winograd. The PageRank Citation Ranking: Bring order to the Web. Technical Report, Stanford Digital Library Technologies Project.

[PaMoChSA] http://www.iit.cnr.it/fabio.martinelli/pamochsa.htm.

[Parsing] http://dinosaur.compilertools.net/.

[Plo81] G. Plotkin. A Structural Approach to Operational Semantics. Technical Report DAIMI-FN-19, Aarhus University, 1981.

[PKITM] http://www.itu.dk/courses/DSK/F2003/PKITrust models.pdf.

[PolySpace] http://polyspace.com.

[Ponder] http://ponder2.net/.

[PreFast] http://www.microsoft.com/whdc/devtools/tools/prefast.mspx.
[Pro-B] http://www.stups.uni-duesseldorf.de/ProB/overview.php.

[Protégé] http://protege.stanford.edu/,

[Prover9] http://www.cs.unm.edu/~mccune/mace4/.

[RG97] A. Rensink and R. Gorrieri. *Action refinement as an implementation relations*. In TAPSOFT '97: Proceedings of the 7th International Joint Conference CAAP/FASE on Theory and Practice of Software Development, pages 772–786, London, UK, 1997. Springer-Verlag.

[Ris07] E. Rissanen. extensible access control markup language (xacml) version 3.0 (core specification and schemas), 2007.

[RRA02] Ryan P.Y.A and Arnesen R.R. A process algebraic approach to security policies. DBsec, volume 256 of IFIP Conference proceedings, pages 301-312, Kluwer, 2002.

[RSG01] Ryan P.Y.A. and Schneider S.A. and Goldsmith M.H. and Lowe G. and Roscoe A.W.*The Modelling and Analysis of Security Protocols: the CSP Approach*. Addison-Wesley, 2001.

[SecPAL] http://research.microsoft.com/projects/SecPAL/.

[SBP01] D. Song, S. Berezin, and A. Perrig. Athena: A novel approach to efficient automatic security protocol analysis. *JCS*, 9(1,2):47–74, 2001.

[SDSI] http://theory.lcs.mit.edu/~cis/sdsi.html.

[SharePoint] http://www.microsoft.com/sharepoint/.

[Spin] http://spinroot.com/spin/whatispin.html.

[SSR06a] V. Swarup, L. Seligman, and A. Rosenthal. A data sharing agreement framework. LNCS 4332, pages 22-36, 2006.

[SSR06b] V. Swarup, L. Seligman, and A. Rosenthal. Specifying data sharing agreements. In POLICY 2006, pages 157-162, 2006.

[SSKK86] M. J. Sergot, F. Sadri, R. A. Kowalski, F. Kriwaczek, P. Hammond, and H. T. Cory. The british nationality act as a logic program. *Commun. ACM*, 29(5):370–386, 1986.

[ST07] N. Stouls and M. L. Potet. *Security policy enforcement through refinement process*. In J. Julliand and O. Kouchnarenko, editors, The 7th International B Conference, volume 4355 of Lecture Notes in Computer Science, pages 216–231. Springer, 2007.

[Too03] B. Toone, M. Gertz and P. Devanbu. Trust Mediation for Distributed Information System. In Proceedings of IFIP TC11 SEC2003, Athens, Greece, May 2003, Kluwer, 1-12.

[Trust-EC] http://dsa-isis.jrc.it/TrustEC/.

[Una06] D. Unal, M. U. Caglayan. Theorem proving for Modeling and Conflict Checking of Authorization Policies. In Proceedings of the Seventh International Symposium on Computer Networks (ISCN'06), Istanbul, Turkey, June 2006, IEEE, 146-151.

[USS07] Y. B. Udupi, A. Sahai, and S. Singhal. *A classification-based approach to policy refinement.* In Integrated Network Management, pages 785–788. IEEE, 2007.
[US94] A. C. Uselton and S. A. Smolka. *A process algebraic semantics for statecharts via state refinement*. In E.-R. Olderog, editor, PROCOMET, volume A-56 of IFIP Transactions, pages 267–286. North-Holland, 1994.

[vGG00] R. van Glabbeek and U. Goltz. *Refinement of actions and equivalence notions for concurrent systems*. Acta Inf.,37(4-5):229–327, 2000.

[WM03] W. H. Winsborough and J. Mitchell. Distributed credential chain discovery in trust management. *JCS*, 11(1):35–86, 2003.

[Yah93] R. Yahalom, B. Klein and T. Beth. Trust Relationships in Secure Systems –A Distributed Authentication Perspective. In Proceedings of the 1993 IEEE Symposium on Research in Security and Privacy, California, USA, May 1993, IEEE Computer Society, 150-164.

[ZRG05] N. Zhang, M. Ryan, and D. P. Guelev. Evaluating access control policies through model checking. In J. Zhou, J. Lopez, R. H. Deng, and F. Bao, editors, *ISC*, volume 3650 of *Lecture Notes in Computer Science*, pages 446–460. Springer, 2005.

Appendix

In this Appendix, we first show samples of policies from real data sharing agreements, indicating which issues identified in the introduction of this document they are examples of. Next, we give an overview of Alfresco and Microsoft Office SharePoint Server 2007's capabilities.

Sub-Appendix 1 : Real DSAs samples

US DataTrust Terms of Use (2008, USA)[2]

Restrictions on Data Use. The data is for your personal and non-commercial use.

[2] http://www.usdatatrust.com/disclaimer.htm

Dissemination to third parties. You may not modify, copy, distribute, transmit, display, perform, reproduce, publish, license, create derivative works from, transfer, or sell any information, data, software, products or services obtained from this Web site.

Location of data and custodial responsibility. You agree not export or re-export the data, directly or indirectly, to any countries that are subject to USA export restrictions.

NASA Planetary Data archive user license (USA, 2005)

Restrictions on Data Use. Unless permitted by appropriate governmental export authorization, you agree that the data will not be used in the design, development, production, or use of nuclear, missile, chemical or biological weaponry.

Location of data and custodial responsibility. All technical data delivered under this Agreement are subject to US export control laws and may be subject to export or import regulations in other countries. You agree to comply strictly with all such laws and regulations and acknowledge that you have the responsibility to obtain such licenses to export, re-export, or import as may be required after delivery to you.

Economic and Social Data Service End User License (2006, UK)[3]

Restrictions on Data Use. To obtain permission prior to using part or all of the data collections for commercial purposes by contacting the registrar and/or relevant data service provider, where relevant, in order to obtain an appropriate licence from the rights holder(s) in question or their permitted licensee if one is available.

Dissemination to third parties. To give access to the data collections, in whole or in part, or any material derived from the data collections, only to registered End Users who have entered into an End User Licence and accepted the relevant Special Conditions necessary to access and use the data collections.

Location of data and custodial responsibility. To ensure that the means of access to the data (such as passwords) are kept secure and not disclosed to a third party except by special written permission or licence obtained from the original data service provider.

Confidentiality. To preserve at all times the confidentiality of information pertaining to individuals and/or households in the data collections where the information is not in the public domain. Not to use the data to attempt to obtain or derive information relating specifically to an identifiable individual or household, nor to claim to have obtained or derived such information. In addition, to preserve the confidentiality of information about, or supplied by, organisations recorded in the data collections. This includes the use or attempt to use the data collections to compromise or otherwise infringe the confidentiality of individuals, households or organisations.

Derived Data. At the conclusion of my research (or if earlier at any time at the request of a member of the Data Team), to offer for deposit in the data collection(s) on a suitable medium and at my own expense any new data collections which have been derived from the materials supplied or which have been created by the combination of the data supplied with other data. The deposit of the derived data collection(s) will include sufficient explanatory documentation to enable the new data collection(s) to be accessible to others.

[3] http://www.esds.ac.uk/aandp/create/eul.asp

Breaches to the agreement. I understand that breach of any of the provisions of this Agreement will lead to immediate termination of my access to all services provided by the Data Team either permanently or temporarily, at the discretion of a member of the Data Team, and may result in legal action being taken against me. I understand that where there is no breach of this Licence, it may be terminated, or its terms altered, by a member of the Data Team either after 30 days notice; or, if a service charge has been paid in advance, at the end of the period for which payment has been made, whichever is the longer. The failure to exercise or delay in exercising a right or remedy provided by this Agreement or by law does not constitute a waiver of the right or remedy or a waiver of other rights or remedies.

Data Quality. The members of the Data Team bear no legal responsibility for the accuracy or comprehensiveness of the data supplied.

Crime Reduction WebSite Data Sharing Protocol (2006, UK)[4]

Location of data and custodial policy. The following named individuals are designated to assume responsibility for data protection (including notification where appropriate), security and confidentiality, and compliance with relevant legislation: Name, Post, Organisation.

Method of data access. We agree that information disclosed must:

- Not be emailed over internet links, where that it practical
- Be protected by back-up rules
- When stored in a computer system it must be password protected and we agree that this password will be revised regularly
- Be located in a geographically secure environment
- Not be inputted / accessed without industry standard security devices as defined by BS7666.

Disposition of data. All data held by us is subject to a specified shelf-life of X. All data disclosed to us will be held until Y.

Inter-Agency Data Sharing – Model Agreement (2004, USA)[5]

Description of data users. Access to the data shall be limited to the minimum number of individuals necessary to achieve the purpose stated in this section and to those individuals on a need-to-know basis only.

Custodial Responsibility. The User agrees to establish appropriate administrative, technical, and physical safeguards to protect the confidentiality of the data and to prevent unauthorized use or access to it.

---

[4] Data Sharing Protocol for Crime Reduction produced by the UK Home Office Policing and Crime Reduction Group. http://www.crimereduction.homeoffice.gov.uk/infosharing.doc

[5] Inter-Agency Data-Sharing Agreement, Centers for Medicaid and Medicare Services, http://www.cms.hhs.gov/states/letters/smd10228.asp

Disposition of Data**.** The requestor and its agents will destroy all confidential information associated with actual records as soon as the purposes of the project have been accomplished and notify the providing agency to this effect in writing.

WLAP Data Sharing Agreement (2004, Canada)[6]

Description of the data. The parties agree in principle to the open and free sharing between the parties of meteorological, air, water and terrestrial data gathered from monitoring locations;

Dissemination to third parties.

- The parties agree to seek opportunities to provide the public with seamless Web access to federal and provincial environmental data and information;
- Data or information acquired by one party from the other under the Agreement shall not be disseminated commercially or otherwise disclosed, transmitted or sold to any organizations other than the parties to the Agreement without the written consent of the party that owns the information. The parties thus recognize that the organization that produces a particular set of data or information item is the only one, at its option, entitled to receive payment for subsequent distribution or use;
- Quality controlled data or information acquired by one party from the other under the Agreement may be disseminated or otherwise disclosed or transmitted without cost to any individual or organization without the need for written consent from the party that owns the information providing that (1) the data or information are not modified, (2) the jurisdictional ownership is clearly acknowledged and (3) the release does not compromise the privacy or commercial competitiveness of the data source;
- Raw data or draft information acquired by one party from the other under the Agreement may not be disseminated or otherwise disclosed to any individual or organization other than the parties to the Agreement without the written consent of the party that owns the data or produced the information;

Derived data. The Agreement authorizes the production, dissemination and sale of derivatives by the parties. Where appropriate (e.g., in scientific reports and articles), the party that owns the data shall be explicitly acknowledged as the owner of the data used;

Confidentiality. The parties agree to acquire and use data and information in a manner that respects the scientific value of the data, legislation and intellectual property rights of the parties;

Ontario Model Data Sharing Agreement (1995, Canada)[7]

Quality of Data. Describe what steps will be taken to verify the accuracy and completeness of the personal information before it is used. Identify the steps that will be taken to ensure that the personal information is up-to-date.

Method of data access. For any personal information stored on a computer:

---

[6] Environmental Data Sharing AGREEMENT Between Environment Canada, Pacific and Yukon Region and British Columbia Ministry of Water Land and Air Protection, April 29, 2004. http://www.env.gov.bc.ca/air/airquality/pdfs/ec_wlapagreement.pdf

[7] http://www.ipc.on.ca/english/our_role/guidelin/shardoc.htm

- identify the method of transfer;
- identify the controls in place to ensure the security and completeness of transmission (encryption);
- identify the controls in place to ensure that only the required personal information will be transferred; and describe the types of audit trails and/or management reports produced to ensure that personal information will be processed in a complete and accurate manner.

Health and retirement study conditions of use (2006, USA)[8]

Confidentiality. Make no attempts to identify study participants.

Method of data access. Not to allow others to use your username and password to access this site.

Dissemination to third parties. Not to transfer HRS Public Release data to any third party other than staff or students for whom you are directly responsible except as indicated.

Disposition of Data. To certify the destruction of any downloaded Public Release data file as well as any data files derived from the downloaded file when requested to do so by the Health and Retirement Study.

Breaches to the agreement. Report immediately to the Health and Retirement Study at hrsquest@isr.umich.edu any disclosure of study participant identity.

Sub-Appendix 2 : Alfresco's capabilities

The following sections give a short overview of Alfresco's capabilities.

Document management capabilities

Alfresco offers a sophisticated and rich web client interface. Each user has its own customizable dashboard, which collects all relevant information for the user, and provides access to documents' spaces and repositories.

Alfresco web client offers a complete set of document management functionalities: documents can be organized in spaces (folders); attributes and metadata are automatically extracted from common document types (for ex(ample Microsoft Office documents); documents previews are available for most document formats; search functionalities are built-in. Alfresco handles document's versions, and allow users to perform check-in and check-out operations. The web client also provides interesting collaborative functionalities: for example, users can start a discussion over a document: users' posts are shown in a threaded forum-like layout, where users can easily exchange messages.

Alfresco offers also a smooth integration with Microsoft Windows through CIFS (Common Internet File System), which makes content available via a shared drive interface.

[8] https://ssl.isr.umich.edu/hrs/reg_cou.html

Workflow capabilities

Alfresco offers built-in workflow capabilities, which enable the construction of simple approval workflow. The following picture shows an example (Figure 10).

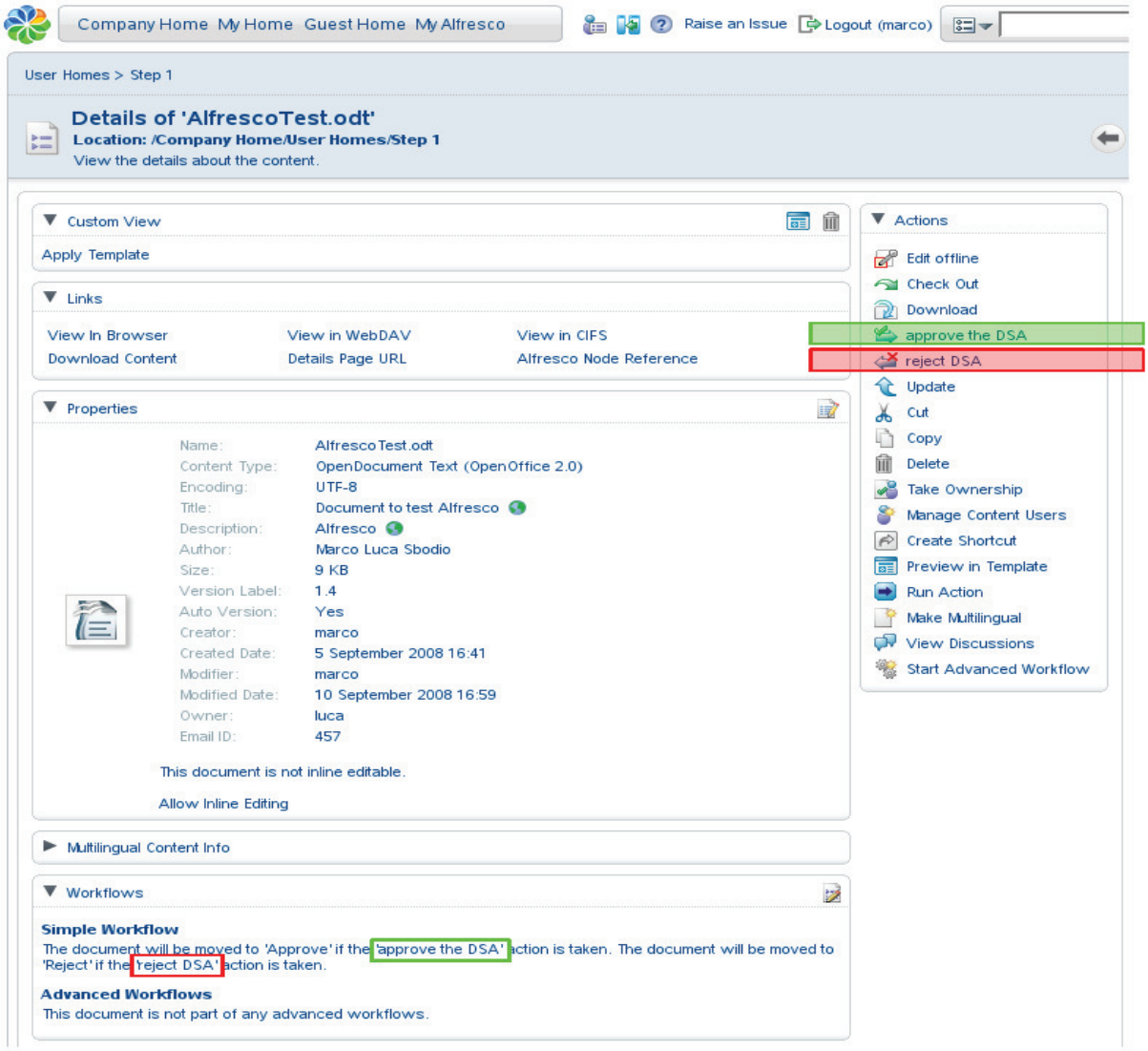

Figure 10. A simple example of a workflow in Alfresco.

The simple workflow illustrated above shows the possibility of approving or rejecting a document (in this case a fictitious DSA). The simple workflow is implemented through rules that are added to document's spaces, and that can specify a number of predefined actions (for example moving the document from one space to another, as shown in the example above).

More advanced workflow capabilities are available. Alfresco integrates with JBoss jBPM (http://www.jboss.com/products/jbpm), which delivers workflow, business process management (BPM), and process orchestration in a scalable and flexible way.

Architecture and extensibility

Alfresco is a J2EE application: it is shipped with Tomcat, but is potentially deployable also in other J2EE compliant containers. It is supported on Linux, Solaris, and Windows operating systems. Supported databases include MySQL and Oracle 10. Authentication is supported with Active Directory, and OpenLDAP. Examples of tested software stacks are available at http://www.alfresco.com/services/support/stacks/ .

Alfresco offers a Content Management Web Services API, which provides integration end points for authentication, query and model manipulation, content manipulation, collaborative content creation, content classification, management of actions and rules.

Sub-Appendix 3 : Microsoft Office Sharepoint Server 2007's capabilities

The following sections give a short overview of Office SharePoint Server 2007's capabilities.

Workflow capabilities

Workflows are available in Office SharePoint Server 2007 for all libraries and lists, to control the publication of documents and other list items (Figure 11 and Figure 12). These workflows can be defined to start automatically or manually, and multiple workflows can be assigned to a library or list according to the organization's business needs. Workflows can trigger e-mail messages to workflow participants, and a user's workflow tasks can be displayed on pages within the enterprise portal.

The workflow templates included with Office SharePoint Server 2007 can be applied and configured without any need for development. These pre-built workflow templates will satisfy common workflow needs in most organizations. They include options to configure the workflows in a number of ways, including the ability to specify whether a workflow is to be parallel or serial, and the ability to define the workflow participants, along with other settings.

Of course, it is anticipated that many organizations will have their own specific workflow requirements that may not be satisfied by the pre-built workflow templates. In these instances, site administrators have two options for creating their own custom workflows:

1. **Office SharePoint Designer 2007:** The Office SharePoint Designer 2007 application can be used to visually design workflows with the assistance of the Workflow Design Wizard. This tool presents a code-free approach to designing workflows and is most suitable for site- and list-specific workflows.
2. **Visual Studio and Workflow SDK:** Developers can implement custom-built workflows through the use of the Workflow Software Development Kit (SDK) and the visual workflow designer in Microsoft Visual Studio 2005. This is the most general environment for building organization-specific workflows that may need to be deployed across the portal.

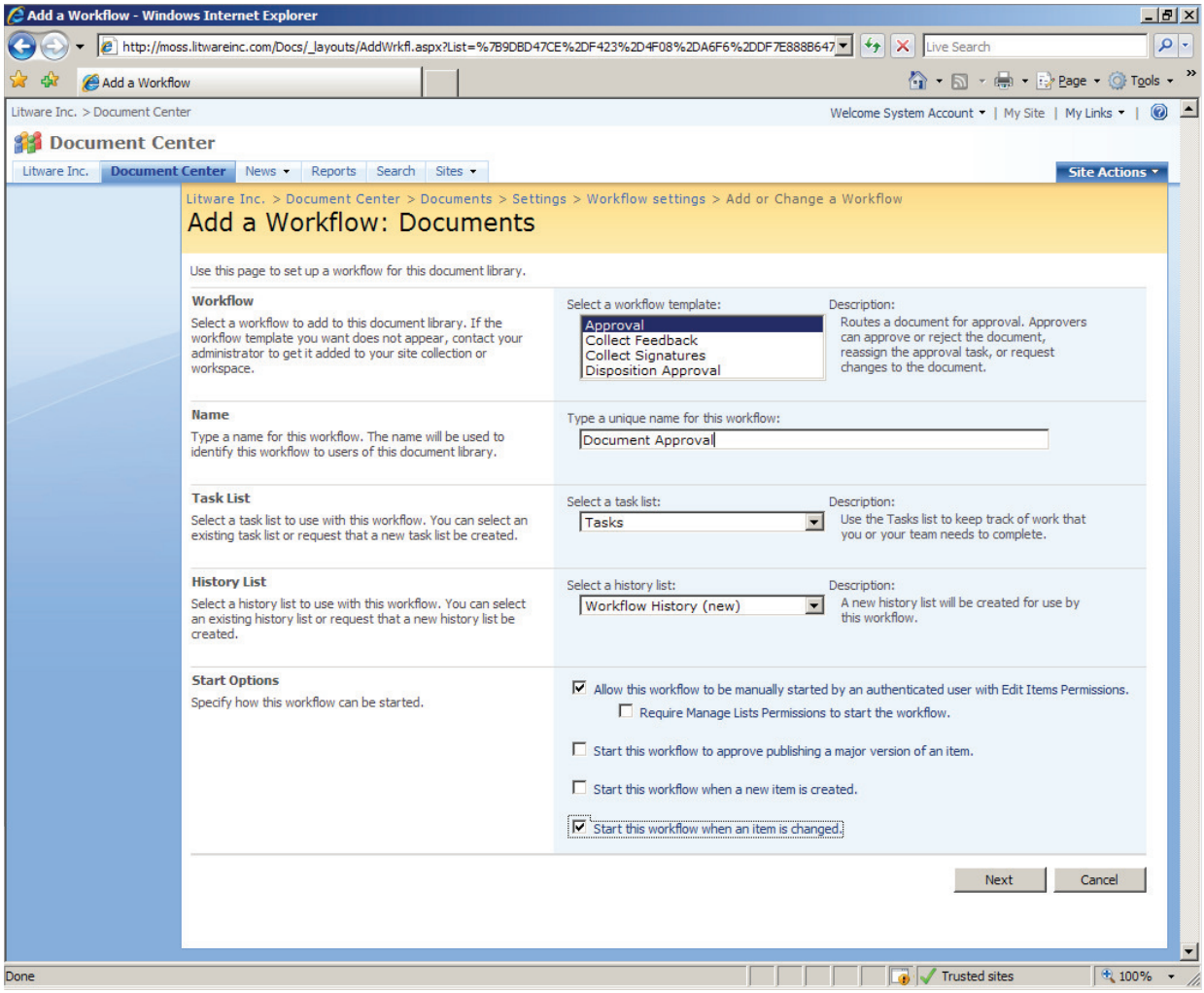

Figure 11. Adding a workflow to a document library.

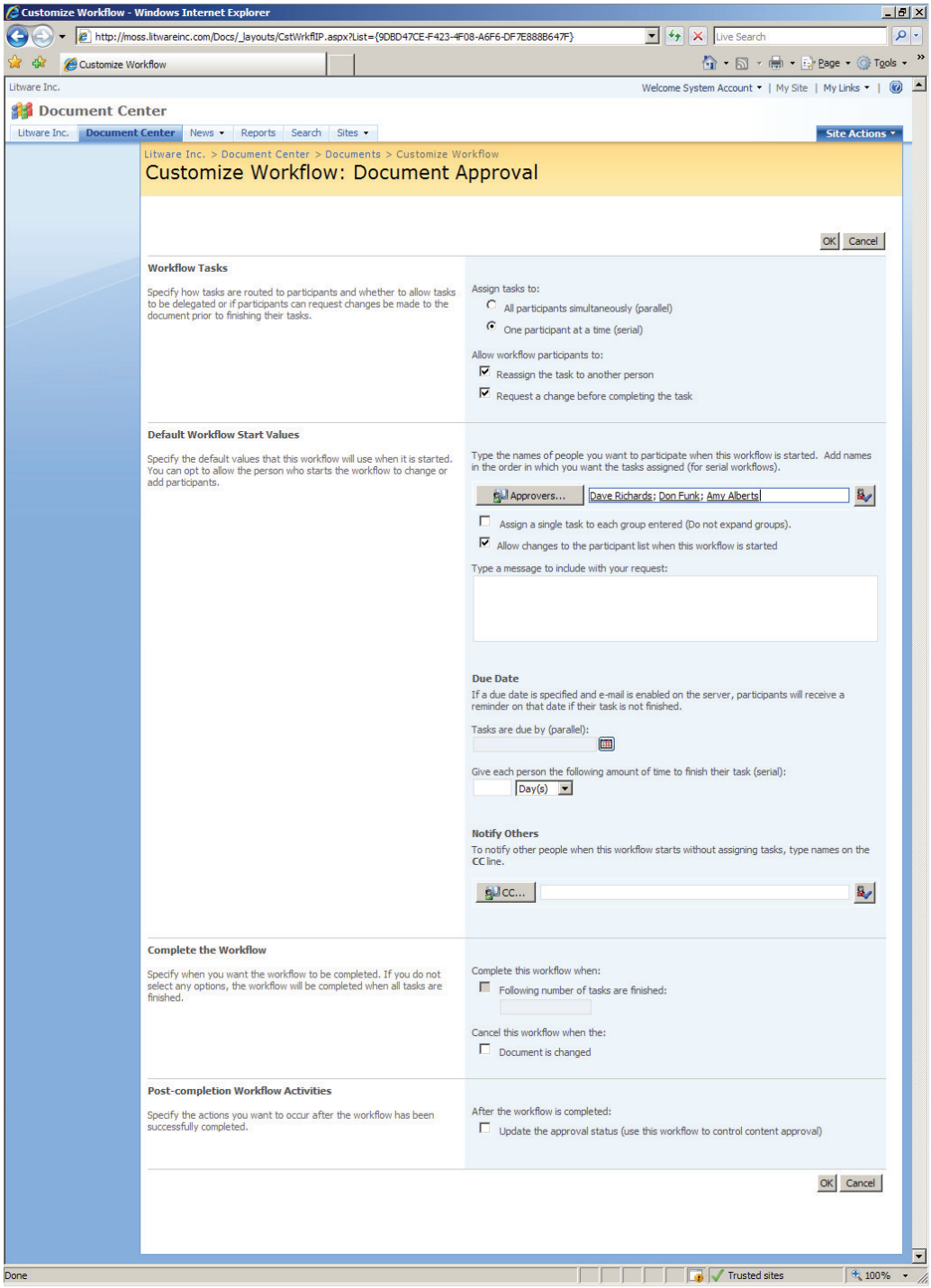

Figure 12. Customizing a workflow.