J. L. Vivas, I. Agudo, and J. Lopez, "A methodology for security assurance-driven system development", *Requirements Engineering*, vol. 16, pp. 55-73, 2011.
http://doi.org/10.1007/s00766-010-0114-8
NICS Lab. Publications: https://www.nics.uma.es/publications

A Methodology for Security Assurance Driven Development

José Luis Vivas, Isaac Agudo, Javier Lopez

NICS Lab (www.nics.uma.es)

University of Malaga

29071 –Malaga (Spain)

e-mail:{ jlvivas, isaac, jlm }@lcc.uma.es

Abstract: In this work we introduce an assurance methodology that integrates assurance case creation with system development. It has been developed in order to provide trust and privacy assurance to the evolving European project PICOS (Privacy and Identity Management for Community Services), an international research project focused on mobile communities and community-supporting services, with special emphasis on aspects such as privacy, trust, and identity management. The leading force behind the approach is the ambition to develop a methodology for building and maintaining security cases throughout the system development life cycle in a typical system engineering effort, when much of the information relevant for assurance is produced and feedback can be provided to system developers. The first results of the application of the methodology to the development of the PICOS platform are presented.

Key words: *Security Engineering; Security Assurance; Assurance Cases; Built-in Assurance*

Introduction

In this work we present an assurance methodology developed with the aim of providing trust and privacy assurance to the evolving European project PICOS (Privacy and Identity Management for Community Services) [1], and inspired by the Assurance Based Development (ABD) approach to critical systems [2]. The objective of PICOS is to develop and build a state-of-the-art platform for providing the trust, privacy, and identity management aspects of social community services and applications on the Internet and in mobile communication networks. A requirement of PICOS is that assurance should be an integral constituent of the PICOS solution and be pursued in a holistic manner. Assurance should address the requirements, design, architecture, and implementation phases of the development of the PICOS platform. To this end, we are developing a methodology that facilitates an integration of security engineering and security assurance with the aid of the notion of assurance case [3].

Security engineering as a discipline is concerned with building dependable and secure systems that resist not only error or mischance, but also malicious behaviour. Our view is that assurance can best be achieved by documenting the security engineering procedures, and extracting from them the evidence and argumentation needed to build the assurance case. As a result, assurance could also have an impact on the decisions and choices made during system development, becoming in this way more than a simple documentation exercise.

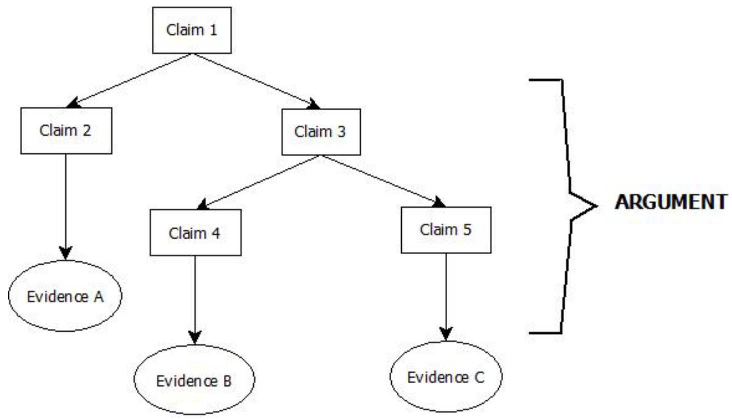
A security assurance case is a structured collection of security-related claims, arguments, and evidence. It presents an argument showing how a top-level claim is supported by objective evidence. Assurance cases typically consist of at least three parts:

Claims: embody what is to be shown.

Arguments: show how a top-level claim is supported by subclaims and ultimately by evidence.

Evidence: can be regarded as claims that do not require further argument; may include testing, code review, formal mathematical proofs, arguments about the nature of the development process, the reputation of the development organisation, and the trustworthiness of the developers, among others.

The structure of an assurance case may be illustrated as in Figure 1.



if Evidence A **then** Claim 2
if Evidence B **then** Claim 4
if Evidence C **then** Claim 5
if Claim 4 & Claim 5 **then** Claim 3
if Claim 2 & Claim 3 **then** Claim 1
Hence

if Evidence A **&** Evidence B **&** Evidence C **then** Claim 1

Figure 1 Structure of an Assurance Case

The starting point of our approach is the goal of creating an assurance case whose structure reflects in some sense the structure of system development itself. Since at each determined stage of development the only available entities are those belonging to the models specified at the current or earlier stages of development, but not to models created later, we see a natural correspondence between the two following structures. The first one is the structure consisting of the different system models and artefacts produced at different stages of development, with their relationships and dependencies. The second one is the structure of an assurance case in which the higher level nodes would consist of claims about the more abstract system models and entities, and the lower level ones to claims about lower level system models and entities. Since at each stage of development the entities belonging to later stage models are assumed to be absent, no claims should be made about them at this phase. Although system development usually happens in an iterative way, and a given model does not follows from another in a pure chronological way, it is nevertheless desirable that a model at a determined level of abstraction does not make references to elements in models that logically belong to a later stage of development or are refinements of the previous one. For instance, a requirements model should not make assumptions about architectural elements. As a result, if the assurance case is developed during system development, and is moreover integrated with it, claims made at a determined stage of development should refer only to elements of models belonging to the current or previous stages of development, but not to later ones.

As an added value, this scheme would facilitate linking an entity belonging to a model at certain development stage, to specific claims at the corresponding level in the assurance case in which this entity is mentioned, and thereafter to the higher level parent claims in the assurance case tree. Changes in the entity may thus be shown to have an impact on each one of those dependent claims. This traceability property is an important feature of the proposed methodology.

In our methodology, an assurance case shows how a top-level claim is supported by lower level claims, which recursively are shown to be supported by other claims. In this way, a hierarchy of goals arises encompassing different levels of abstraction and different phases of system development, and facilitating linking and tracing. Assurance links are established between assurance arguments and development artefacts at each phase of development.

Figure 2 illustrates the relation between the structure of an assurance case, as we envisage it, and the system model structure. A system development strategy might involve the phases shown in the right side of the figure: a requirements phase, a design phase, an implementation phase, and a deployment phase. In the figure, the distinct phases are separated by curved lines, and relations between entities at different levels are shown by arrows crossing this line. Typically, at each phase there is a set of entities used to define the system model at the corresponding level of abstraction. Hence, using a notation similar to UML and a use case driven development methodology [4], the requirements phase might consist of actors, documents, use cases, etc. Three use cases were included in the figure: *UC 1*, *UC 2*, and *UC 3*. The design phase might on its turn consist of system components. Three components are shown: *CP 1*, *CP 2*, and *CP 3*. Each one may take part in the realization of the functionality described in one or several use cases. In the figure, for instance, *CP 1* is shown to take part in the development of *UC 1* and *UC 2*. The implementation phase might consist of classes and packages, each one realizing the functionality of one or more components. Finally, at the most concrete level, the system model might consist of devices like mobile phones, PDAs, servers, code, etc.

On the left side of the figure we see the corresponding assurance case. The structure consists of a main claim (e.g. "the system is acceptably secure"), four subclaims (*Subclaim 1* to *Subclaim 4*, e.g. "confidentiality is enforced", "integrity is ensured", "authentication is guaranteed", etc). These claims are related to high level requirements. More specific high level claims could also be included here as subclaims to any of Subclaims 1 to 4, for instance "the authentication of *Actor 1* in *UC 2* is guaranteed," or "the integrity of *Doc 1* in *UC 3* is enforced." Claims are thus predicates about entities in the corresponding system model, and eventually in higher level system models, but not in lower level ones.

The lowest level claims at the requirements phase are thereafter decomposed into subclaims that belong to the design phase of development, and accordingly refer to entities that belong to the design model, e.g. *CP 1*, *CP 2*, and *CP 3*. For instance, *CP 2* could be a PKI-module, *Subclaim 1* could be "authentication is enforced," and *SubC 1.2* could be "PKI is used for authentication." *CP 2*, in this case, could be a PKI module used in order to provide authentication in *UC 2*. Relationships between claims and system model entities are denoted by links in the figure.

This procedure is repeated until we reach the lower level claims, corresponding to the deployment phase, with entities such as mobile phones and servers. For instance, on the system model part of the figure we see a mobile phone *MP*, which is related to *Class 1* and *Class 2* above (e.g. by being an instance of these classes). We show in green all the entities at higher level system models that are directly or indirectly related to the *MP*. Likewise, we show in the assurance case all the claims that are directly or indirectly related to this entity. Claim *SubC 1.2.1.2.1* is immediately related to *MP*, which means that this claim predicates something about *MP*, for instance that it stores public keys or is able to encrypt any outgoing messages. Going up through the claim hierarchy, we see that claim *SubC 1.2.1.2.1* is a subclaim to *SubC 1.2.1.2* and *SubC 4.1.1.1*, which on their turn are subclaims to *SubC 1.2.1* and *SubC 4.1.1*, and so on. Changes in *MP* could in principle have an impact on all these claims in the assurance case. With the aid of a suitable metrics giving a specific weight to each subclaim of a given claim, it might become possible to establish the level of impact of a determined change in any entity of a system model on the validity of any dependent higher level claims. Moreover, the links between the claims in green on the left side of the figure, and the entities on the right, also in green, would allow us to cross-check the relationships between the assurance case and the system models in case of any changes. This picture shows that a rich and complex set of dependencies is obtained that could be of great help in the analysis of the impact of any choices related to the system entities, and whose complexity might become manageable with the aid of dedicated tools.

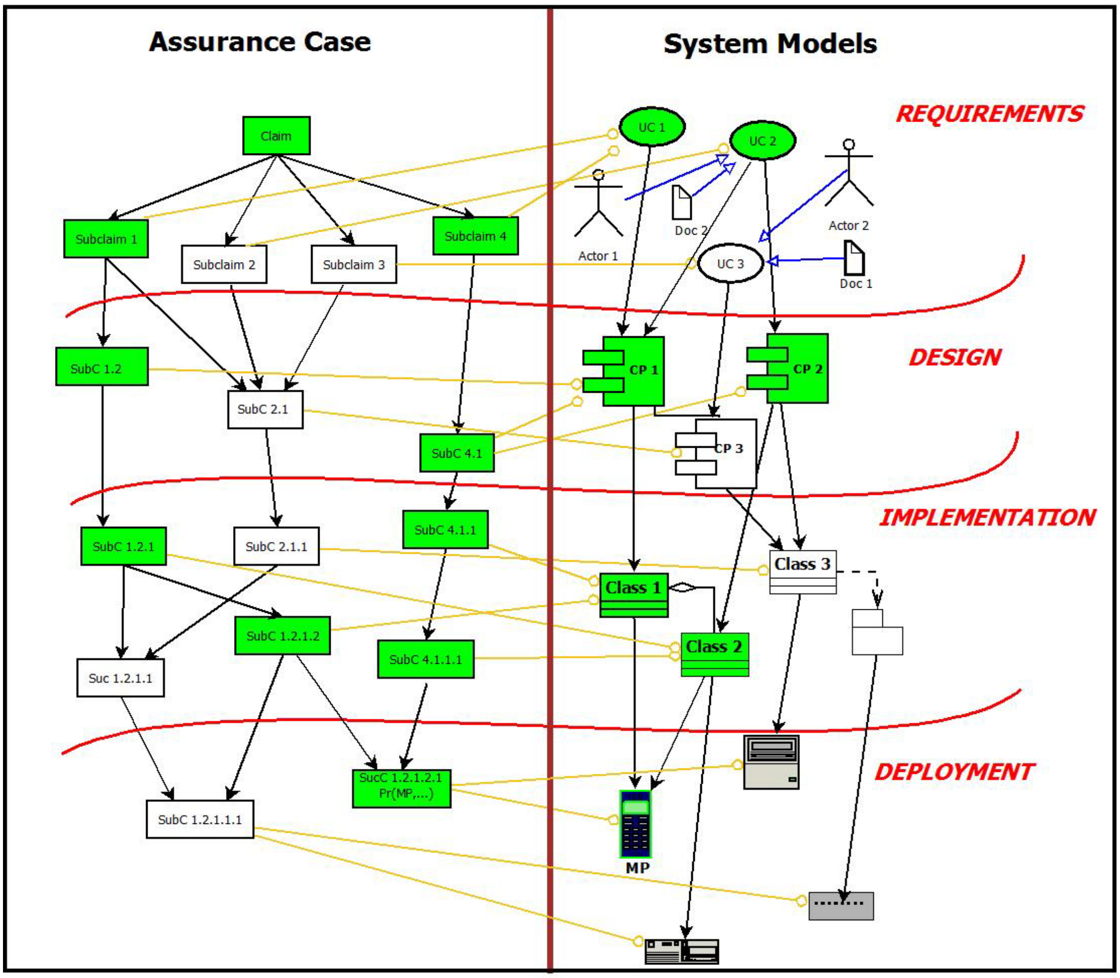


Figure 2 Assurance Cases and System Models

In [5], it is stated that in a typical assurance effort "determination of whether the high-level security objectives have been satisfied is left to the judgment of experts. Typically, the reasoning process these experts employ to arrive at their evaluations is entirely ephemeral, which has a number of disadvantages. The reasoning is non-reviewable. No one else can check the reasoning for gaps or errors. The reasoning process is non-repeatable. Even if another expert reaches the same conclusion, there is no way of determining whether or not he reached it by the same route. Thus the reasoning process itself can be validated only indirectly. As a result, the reasoning process is non-improvable. There is no way to determine whether a given pattern of reasoning can be relied upon generally to provide correct results, since the details of the expert's reasoning are unknown. And, most important, the reasoning is non-maintainable. Systems evolve after deployment and there is no way of determining whether or how any given change to the system should influence the belief that it is adequately secure."

As might be deduced from the previous example, our methodology is intended to target all these difficulties. With the aid of the assurance case structure, it offers a reviewable, repeatable, maintainable and improvable process to determine whether and how high-level security objectives (the high level claims in the assurance case) have been satisfied. The process can be validated and may offer patterns of reasoning that can be shown to be useful. Finally, it offers a way of determining the impact of changes in the system or its components on the high-level security requirements and goals.

The assurance methodology we present here is intended to be agnostic to current security engineering methodologies, but some of these might certainly fit better than others. We have developed this methodology in the context of a use case based development approach, but it could in principle be applied to any methodology which includes a consistent functional specification of the system. We assume only that there is a such a functional description of the system, which is often the case in system development startegies. Even a goal or problem oriented development methodology must contain such a description.

A methodology called PriS [6], similar to ours, has been proposed as an assurance-driven approach to privacy requirements, but in the context of problems frames. Other related methods include requirements engineering methodologies for integrating security and software development at the design level, e.g. Tropos [7], i* [8], NFR [9], KAOS [10], among others. These methodologies do not focus mainly on assurance, and do not address how to translate requirements into functional components, the starting point for our approach. However, these approaches are not in conflict with ours, which is not intended to be an alternative to security engineering methodologies. Our view is that most of these approaches could be easily integrated into our methodology, e.g. by letting security requirements elicited from business goals become the high level claims in the assurance case.

The rest of this paper is organised as follows. The next section gives an overview of the subject of assurance. Thereafter a section is dedicated to a presentation of the methodology. Following this, a section is dedicated for presenting the application of the methodology to the development of the PICOS platform. Finally, in the last section we sum up our experience so far in the development and application of the proposed methodology, and put forward some proposals for future work.

Assurance: an Overview

Assurance has been defined as the level of confidence that an entity meets its initial requirements based on evidence provided by the application of assurance techniques. Security assurance refers to security requirements. In this work we deal only with security assurance. Assurance must provide evidence that the number of vulnerabilities in a software product, including the presence of features that may be intentionally exploited by malicious agents, are reduced to such a degree that it justifies a certain amount of confidence that the security properties of the software meet the established security requirements, and that the degree of uncertainty involved has been reduced. The focus here is on the minimisation of vulnerabilities, since absolute certainty that vulnerabilities have been fully eliminated is in practice unattainable.

Many researchers hold the view that security and security assurance should be an integral part of the development process [11,9,12], and that security is best assured if it is addressed holistically, systematically, and from the very beginning in the software's development process. Hence, good system engineering methods, techniques and practices might be seen in themselves as factors that increase confidence, and should therefore be treated as an integral part of an assurance process.

Requirements can be functional and non-functional. Functional requirements describe interactions between the system and its environment, whereas non-functional requirements are often constraints or restrictions limiting design and implementation choices. Security is commonly seen as a non-functional requirement. However, whereas other non-functional requirements are usually more or less unrelated, at least directly, to the functionality of a system, this is clearly not always the case with security requirements, which are often closely related to the functional aspects of a system. Some security requirements, for instance authentication, are clearly functional in character. Others often imply constraints on the functionality of the system, either by design or as a side-effect, and may therefore be described as anti-functional rather than non-functional. Some approaches do justice to this feature, e.g. the so called misuse or abuse cases [13], which implicitly assumes a close relationship between security and functionality. Use cases are intended to express the functional requirements of a system, whereas many security vulnerabilities arise exactly from the possibility of misusing this functionality or using it unexpected or unforeseen ways. Hence, eliminating those vulnerabilities often amounts to eliminating the intended or non intended functionality of the system in order to prevent attacks.

The challenge here, from the assurance point of view, is that it can never be assured that all the vulnerabilities of a system have been targeted or eliminated, since it is not possible to list every possible sequence of actions by brute force, nor are there efficient methods that can formally prove that a system is secure. Testing is not enough in this context. Security testing methods are immature, and testing by itself cannot gauge security adequately, being better suited to target the functional properties of a system as well as random errors, rather than malicious behaviour. In fact, evaluations methods such as the Common Criteria have been often criticised exactly for focusing on assuring only the functional security requirements of a system or product [14], not the absence of exploitable vulnerabilities. Hence, the adoption of sound system engineering practices may in itself be more confidence building than testing or evaluation. Likewise, allowing assurance to be an integral part of system development will help in bringing on the adoption of sound development practices, evidencing the synergies between both activities.

It must be noted here that security assurance requirements are currently almost always ignored during system development [15], which typically focuses on functional requirements. We have thus very little to fall back upon when it comes to best practices in the field. However, software assurance has nevertheless been a very active area of research, and some advances have been reported in the latest 10 years, with the emergence of a high number of standards, techniques, methodologies.

Security in the Software Development Life Cycle (SDLC)

Security is seldom considered adequately, if at all, during the software development life cycle (SDLC), since security is rarely viewed as a driving factor. Security considerations are typically left to the latest stages of system development, when the architecture and the design have already been cemented. There is a clear lack of experience and best practices promoting security assurance, and current high-assurance development tools do not scale. For large scale systems there is currently no engineering methodology to ensure security during the SDLC. Notwithstanding, it is largely recognized today that security is best assured if it is specified very early in the SDLC process, since security weaknesses often have their origin in inadequate architecture and design choices. Software engineering is usually oriented toward functionality, not security. However, the impact of security on the functionality of the system is usually very strong, to the point that a project may fail because of the late integration into the system of the required security mechanisms.

Security and functionality are certainly different in character; however, they are closely related. Basically, functionality tells what a system should do, whereas security imposes constraints on this functionality by telling what the system shall not do. Often, security will impose restrictions not on the required functionality of a system, but only on undesired sequences of actions, which any complex system will exhibit. A system is comprised of several parts or components with complex behaviour, interacting with each other in ways that cannot be completely foreseen, and that a malicious actor may try to exploit. It is the task of security engineering to try to prevent this, usually by restricting the functionality of the system in a way that eliminates these undesired and exploitable sequences of actions, or by reducing the scope of implementation options. Moreover, any design decision may have a big impact on security. Functionality and security go thus hand in hand, even if they are in principle separate concerns. They should be integrated with each other during system development from the earlier stages, and their steps should be interleaved.

This approach requires that developers constantly document and inform the assurance team about design and implementation choices, and that these choices are made with due consideration to the security requirements at each stage of development. Design choices imply often a new decomposition or refinement of the design model of a system, a decomposition that may have security implications. It is the task of the assurance team to analyse these choices and establish a set of requirements concerning the new components or functionalities, which will further drive the work of the developer. In this way, new security requirements will always be surfacing from the initial set of system requirements, which might assume a variety of forms depending on the level of abstraction of the system in which they are defined. Security analysts should thus take an active role, both proactive and reactive, towards the development process.

It is also important that this approach does not impose new demands on the skill of the system developers, and does not replace common development practices. The only extra demands that should be made are that design decisions and system models are more carefully documented and updated, and that security and assurance requirements are duly taken into consideration when design choices are made. Documentation, apart from being in itself a valuable artefact that enhances the software engineering process, is essential for assurance, and should cover all stages of development, from requirements to architecture, design and implementation.

Assurance Based Development (ABD) and Assurance Cases

Assurance Based Development (ABD) has recently been proposed as an approach by which assurance is created throughout and as a driver to a system's development process. ABD was introduced in the context of critical systems [16] through the combined use of GN [17] and problem frames [18]. The basic idea was the co-development of the system and its assurance case in order to enable developers to make technology choices that address the dependability goals of each component [16]. A similar approach is Assurance Driven Design (ADD) [19], proposed to build together the software and the assurance argument within the context of Problem Oriented Engineering (POE) [20]. All these approaches are based on the notion of problem frames, which was considered to enhance assurance cases by providing them with additional structure [16]. Our approach is similar, but based on the more conventional requirements engineering approach based on use cases, which we believe is also suitable for providing structure and rigour to assurance cases.

Assurance case is the central concept in our approach. A security assurance case is a document that changes as the system that it is documenting changes. Assurance cases has been defined as “a documented body of evidence that provides a convincing and valid argument that a specified set of critical claims regarding a system’s properties are adequately justified for a given application in a given environment.” [21] Assurance cases should be developed alongside the software component or system itself. The amount and type of information contained in an assurance case should be a function of the level of assurance required at each phase of development. This information can be extended, changed, rearranged, and even eliminated during the whole process.

A security assurance case presents arguments showing how a top-level claim is supported by objective evidence. Assurance cases typically consist of at least three parts, as shown in Section 1, namely *claims*, *arguments*, and *evidence*. An assurance case shows how a top-level claim is supported by lower level claims, which recursively are shown to be supported by other claims.

The consequences of a security breach tell how much effort should put into developing arguments and claims, and some cases may therefore require a higher standard of evidence and argumentation than others. Evidence may consist of any confidence enhancing element: programmer training credentials, results of the code review, testing results. Confidence in the argument depends on how convincing the argument and the evidence are in our eyes. Further evidence might be required whenever the current one is considered insufficient.

Claims should be stated succinctly and unambiguously; further information can be provided in an optional element called *context*. Optionally, the *strategy* used to develop a sub-claim from a claim can be explicitly stated, thus providing an additional clue to understand the form that an argument is going to take, as well as information on how to substantiate a stated claim. Other optional elements are *justifications* and *assumptions*.

Many problems associated with assurance cases were highlighted in [22]. These include: the volume and nature of the required evidence; the lack of explicit relationships between assurance claims, arguments, and the supporting evidence; the lack of support for structuring the information; the lack of a standard set of rules of evidence; the lack of guidance on how to gather, merge and review arguments and evidence; the lack of guidance for weighing conflicting or inconsistent evidence; and the difficulty in comprehending the impact of changes because of the huge volume of information. There is hope, however, that these problems may be mitigated by new assurance methodologies and the development of supporting tools.

A Model Based Methodology for Assurance

In [5] it is stated that “establishing a desired high-level security property from the available evidence is bound to require a complex argument”, which is “at best incompletely recorded during the design process.” Furthermore, “the conclusions purportedly established are typically much lower-level than the real security objectives, which have to do with system availability and integrity, information confidentiality, and other high-level properties, rather than failure of some class of attacks.”

These observations point at two important facts. First, it is during the design process itself that the main facts and evidence for the assurance argument are created. Second, the conclusions usually established are typically lower-level and do not reach up to the high level properties, typically system properties. However, security is basically a system property, and assurance should thus ultimately address system properties.

System development might be viewed as a process of continuous refinement and decomposition, from a more abstract system model to more detailed and concrete ones. This procedure is akin to Larsen's view of program development [23], which is regarded to consist of a series of refinement steps in which one component is refined into a combination of a number of subcomponents whose combined properties must satisfy the specification of the refined component.

If the main facts for the assurance argument are produced during the design process itself, then this process of decomposition and refinement should be at the heart of the assurance case. Our methodology is intended to follow this approach, and to provide a means to enable going from low level to high level properties, thus facilitating traceability and maintainability.

The basic intuition behind our approach may be described more formally as follows. The top-down development process of any system design can be viewed as a series of decompositions from a more abstract to more concrete models. At each step a model M of the system is refined into a model M_1 that ideally preserves in some sense the behaviour of M. In terms of process algebra, we might for instance require that both models, which we may also call *agents*, are related by some precise notion of behaviour equivalence, for instance the weak observational equivalence in Milner's Calculus of Communicating Systems (CCS) [24], denoted $\approx$, such that $M \approx M_1$. These models or agents are typically composed of other agents or subagents, and exhibit a particular behaviour towards its environment. Some actions that these subagents may perform are internal and unobservable, whereas others are observable interactions with the environment along their external interfaces or channels. The most abstract model may consist of one single component, i.e. the system itself, which interacts with the users of the system. Use cases, often used in the elicitation of the functional requirements of a system, usually regards the system as a single component. Later, this black box may be decomposed into several components. Each one of these components performs a determined role in the overall system, and can be seen recursively as a black box communicating with its environment, which in this case includes the other components resulting from the decomposition. These components may also be further decomposed and so on, until we reach the smallest elements of the system.

Figure 3 gives an illustration of this process. The box named M may denote the system or a component of the system, also called agents, seen as a black box, with an interface consisting of several channels represented as small black rectangles. The channels might be access points for human actors or for other systems. In a typical use case, the interaction of the agents, usually human actors, with the system is specified, yielding a high level view of the functionality of the system as a whole.

In the subsequent development stages, the system is iteratively decomposed into increasingly greater detail, and more concrete models are defined. This procedure need not be performed explicitly, but it is our belief that it takes place at least implicitly in most cases of system development. Those responsible for assurance should proactively urge developers to capture and record this process in as much detail as possible, and to give a justification for any design decisions taken during the whole system development life cycle.

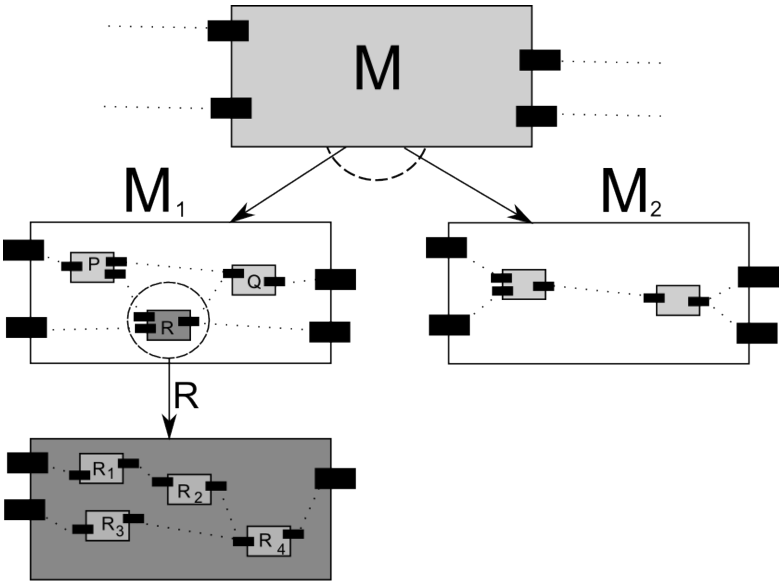


Figure 3 Component Refinement

During this development, different options might be considered, each one yielding a different decomposition or behaviour. We illustrate this in the figure by showing how agent M is decomposed into two alternative agents, denoted M_1 and M_2. Ideally, each one of these agents should exhibit the same external behaviour as agent M. However, this would be too strict. Moreover, the behaviour of agent M itself is seldom specified completely or in sufficient detail to allow us to establish this kind of strong behavioural relation between models. The behaviour of agents M_1 and M_2 may show many discrepancies with respect to the behaviour of agent M, often due to the complexity of the behaviour of the individual components. These discrepancies are usually a source of vulnerabilities.

Consider now agent M_1. It is composed of three agents or subprocesses, namely P, Q, and R. In the language of process algebra, for instance CCS [24], we could specify this fact as $M_1 = P \mid Q \mid R$. Some of the channels associated with these subprocesses are identified with the external channels of agent M, whereas others are internal channels only, used for internal communication among the subprocesses. This decomposition process is recursive. For instance, we might take the agent or subprocess R and see it in isolation, as shown in the figure. It is an agent endowed with several channels for external communication, and with a behaviour that may be described in the same terms as the behaviour of agent M. Hence, it might itself be decomposed into several agents, denoted R_1, R_2, R_3 and R_4 in the figure. This decomposition process will go on until we obtain the entities that are intended to be implemented directly in software.

Asking that $M \approx M_1$ would nevertheless be an extreme requirement. A more realistic requirement can be expressed as follows: if the model M satisfies a certain property Φ (e.g. a property in Hennessy-Milner modal logic [25], in case M is specified as a labelled transition system), denoted $M \models \Phi$, then also $M_1 \models \Phi$. However, it is usually not possible to establish that $M_1 \models \Phi$. A more appropriate requirement would be the following. Assume, for instance, that M_1 is composed of two components C_1 and C_2, hence $M_1 = C_1 | C_2$. Then, we may try to find properties Φ_1 and Φ_2 such that $C_1 \models \Phi_1$ and $C_2 \models \Phi_2$, and such that we may then be able to conclude that $C_1 | C_2 \models \Phi$, i.e. $M_1 \models \Phi$. In order to do this, we must establish some kind of relation between properties Φ_1, Φ_2, and Φ, involving also some argument about how the composition $C_1 | C_2$ relates to Φ_1, Φ_2. This is of course a very complex issue that cannot be carried out formally except in very simple systems. However, it illustrates what any system refinement, implicitly at least, tries to achieve, typically in an informal and ah-hoc way. In an assurance case, Φ_1, Φ_2, and Φ may denote the claims. Claims are however hard to specify in a suitable formal notation, and we may even go so far as saying that assurance is a substitute for formal proof when the latter is not practical or unfeasible.

Therefore, basically what we aim to obtain is an argument establishing a certain link between properties associated with the components of a system, and properties associated with the system seen as a black box. This should be done recursively for each system component, until we reach the level of evidence. In this way, the structure of properties would be related to the structure of the models. Ultimately, the assurance case structure should reflect the structure of the system, and also be related to the distinct development stages of the system, since each stage would represent the system at a certain level of abstraction, i.e. at a certain level of decomposition or refinement. This would also facilitate traceability, since there would be a link between a low level component, the low level claims about properties associated with this component, and high level system claims about the properties of the system. We could in principle see how changes in the design of this component would affect system claims.

The starting point for our assurance methodology is thus the thesis that if the properties of a system are related to the properties of its components, then the structure of an assurance can be made to reflect the structure of the system. A claim about a system in an assurance case is often an assertion about certain properties of the system, and a property associated with a component of this system might constitute as subclaim to the initial system claim. In this way, a hierarchy of goals arise encompassing different levels of abstraction and different phases of system development, facilitating linking and tracing. Assurance links are established and documented between assurance arguments and development artefacts at every phase of development.

Assurance and the SDLC

A typical SDLC process may include the following phases:

1. Requirements
2. Architecture and design
3. Development
4. Testing
5. Deployment

A security risk management process within the SDLC, on the other hand, may include the following:

1. Security requirements specification and risk assessment

2. Security architecture and design
3. Secure implementation
4. Security testing
5. Secure deployment and assurance

Ideally, each phase follows from the previous one according to choices made with regard to what was established in the latter. In a well structured development process, each phase would yield a certain representation of the system. Thus, our starting point is the conjecture that, if a determined phase Φ_n in the development of a system follows from the precious, more abstract, phase Φ_{n-1}, then a claim made concerning features and entities belonging to the system description in phase Φ_{n-1} must be underpinned in some sense by claims made about features and entities belonging to phase Φ_n. This follows from the fact that the specification of the system in one phase should correspond in some sense, at least if the development process is well structured, to the specification of the system in the previous phase.

At the higher levels of an assurance case, the claims would typically correspond to security goals, privacy principles, and so on, elicited by any form of requirements methodology. These claims should thereafter be refined and decomposed until they are rendered operational for further analysis. At this stage, claims may be further decomposed into subclaims with the aid of a vulnerability or threat analysis. An example will illustrate this procedure.

Figure 4 shows a simplified snapshot of an assurance case concerning the Picos project. The main claim we want to show is that Picos complies with all established privacy principles. In order to refine this claim, we decompose it into two subclaims, the first one concerning compliance with the legislation in force in Europe, and the second one with the privacy requirements that are particular for Picos and not covered by legislation. We focus henceforth on the former.

As it is stated now, this claim would be of little help in the assurance work because it is too general. We have to refine it into a set of more manageable subclaims that are suitable for further assurance work. The strategy for this refinement is not obvious; hence, we have decided it to include it explicitly in the assurance case. As can be observed, we decided to concentrate on 3 legislations: the EU Directive, the OECD Privacy Guidelines, and the UN Guidelines. Moreover, we needed a taxonomy of privacy principles. We decided to use the one given in [26], obtaining a number of 20 privacy principles, three of which are shown in the figure: Notice, Purpose, and Consent. We focus now on the last one of these, Consent.

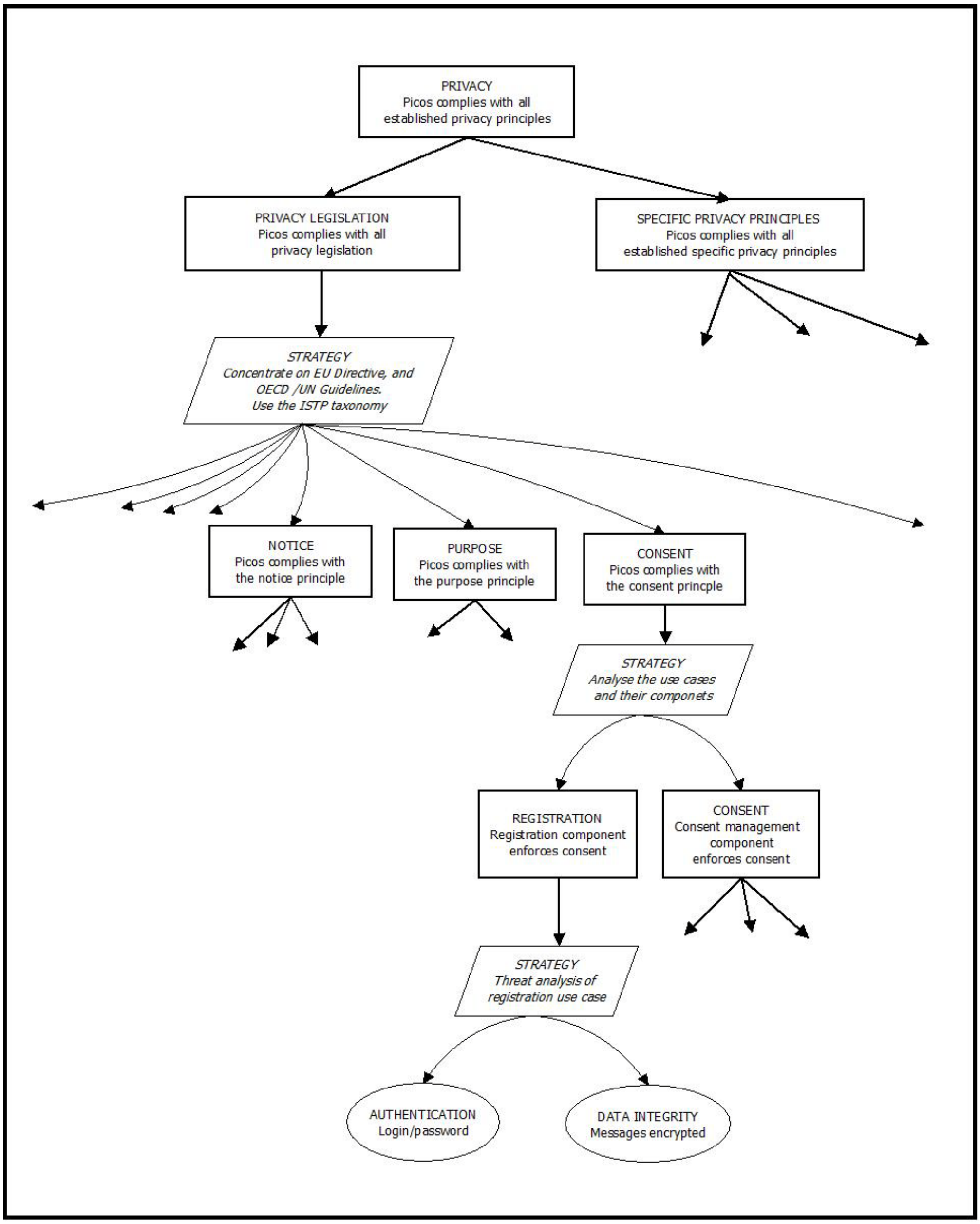


Figure 4 Picos Assurance Case Example

The Consent principle requires "informed consent from the Data Subject to the collection of personal information unless a law or regulation specifically requires otherwise." At this stage it becomes easier to analyse the software with regard to compliance, since this claim is much more concrete than the one saying that the system complies with all legislation.

What follows now only is a simplified picture of the process we followed from this point on. The Consent principle may be regarded as a claim related to the requirements phase, specified in terms of use cases. The strategy now is to look at these use cases to see if the principle is enforced in those use cases or might otherwise be contradicted. Two use cases were considered to be relevant for the Consent principle, i.e. to require collection of personal information: the Registration use case and the Data Management use case. Whenever personal data is collected, informed consent must be required and given.

We go now one step further to the next development stage, the design phase. After having analysed the design, we concluded that there were two components involved in the collection of personal data: the Registration component, and the Consent Management component. The functionalities of these components have now to be analysed in order to ensure that consent is enforced by the system with the aid of these components. Focusing now on the Registration component, we observed that the functionality for achieving consent was included in the behaviour of the component. Hence, we could establish that, preliminarily at least, the registration component enforced consent.

However, we must make sure also that this functionality cannot be subverted by malicious actors. Hence, we decided that both secure authentication and message integrity was required. Going now over to the implementation phase, we realized that a login/password authentication mechanism and cryptography (e.eg. SSL) would be included in the implementation. We regarded these facts as evidence that did not need any further argument or evidence. Of course this is a simplification, the nature of any kind of evidence is always relative, and in a more realistic setting we would also analyse the code, carry out tests, etc, which would become the required evidence instead. However, for the purpose of illustrating the main ideas this example might suffice.

What we are proposing amounts thus to an integration of security engineering and assurance with the help of assurance cases. Assurance cases are in this way turned into a system development tool. In the requirements phase the high-level security or privacy goals and principles are established. These principles and goals are further refined throughout the whole development process. Following the requirements phase, during the design phase the assurance techniques are used to provide evidence that the design meets the high-level security requirements, principles and goals established in the requirements phase. Claims should be made about the correctness of the design with respect to the high-level requirements. Later, during the ensuing implementation phase, assurance should provide evidence that the implementation is consistent with the security requirements, which usually is done by showing that the implementation conforms to the design, and that claims about the entities of the implementation model supports the claims about the entities of the design model. Testing and proof of correctness techniques may be used in this assessment. Later phases, such as deployment, are treated similarly.

Derivation of subclaims

We allow for multiple inheritance in assurance case trees. The reason is that a claim about a low level entity, for instance a software module or component, may have an impact on several higher level claims. For instance, a certain cryptographic mechanism may be used to ensure that diverse goals such as confidentiality and integrity are achieved.

We consider at least three main strategies for deriving subclaims from claims.

The first one we call here *conceptual AND-decomposition*, meaning that the subclaims follow conceptually from the claim and taken together suffice to prove the parent claim.

Hence, if we have a claim C and subclaims, C_1, C_2, …,C_n we would have the following relation:

$$C_1 \wedge C_2 \wedge \ldots \wedge C_n \Rightarrow C$$

This relation is adequate when for instance a claim of type "the system complies with all privacy legislation in force" is decomposed into "the system complies with all national privacy legislation in force" and "the system complies with all international privacy legislation in force."

The second decomposition strategy is *OR-decomposition*:

$$C_1 \vee C_2 \vee \ldots \vee C_n \Rightarrow C$$

This kind of derivation or decomposition might be useful for showing alternatives in the development choices of a system. An OR-decomposition might give a clue to e.g. software developers that there may be acceptable alternatives for a certain goal. For instance, there might be a claim that says that "authentication is provided," and a couple of alternative subclaims saying that "password authentication is provided" and "public key authentication is provided". This tells developers that any of these alternatives will be acceptable, though only one will be necessary.

The third strategy, which we call *obstacle AND-decomposition,* is related to the threats and vulnerabilities that may surface after a system or component at any abstraction level or development phase has been submitted to a threat or vulnerability analysis. For instance, we might need to derive subclaims from high-level security claims with the functionality of the system expressed in terms of use cases. In this case, we may try to analyse the threats and vulnerabilities of the system that may yield, for instance, a set of abuse or misuse cases [13]. This analysis can lead to the specification of countermeasures that may become the desired subclaims supporting the high-level security claims. This procedure can be carried out at any stage of development. Hence, at each phase the basic functionality of a system or a component can be studied and analysed with regard to possible threats and vulnerabilities, and new claims can be derived from this analysis. The subclaims would denote basically countermeasures guaranteeing that the parent goal is not contradicted. Security claims require often this type of decomposition, which may be seen as related to the notion of obstacle or obstruction introduced in [27], where requirements are elicited with the help of scenarios that could lead to a goal violation. Obstacles may be thus defined as undesirable properties of a system which must be avoided in some way. The end product of this approach should be an assurance case that provides a high degree of confidence on the robustness of the deployed system against attacks and the absence of exploitable vulnerabilities in the implementation of security functions.

This strategy for deriving subclaims can be explained more formally as follows. Suppose we want to show that a security claim holds, stated as a negative statement, say

$$\sim B$$

where we let B denote a bad thing that should be avoided. Assume further that, after a threat analysis of the functional specification of the system has been carried out, we discover a certain number of vulnerabilities, threats, or obstacles, which are basically properties of a system or component that might cause B to become true. We may call these threats

$$T_1, T_2, \ldots T_n$$

We assume that the realisation of any of these threats might result in B becoming true. Hence we have

$$T_1 \vee T_2 \vee \ldots \vee T_n \Rightarrow B$$

for some positive integer n. Now, to prevent any of these threats, say T_i, to happen, we introduce the following countermeasures

$$C_{i1}, C_{i2}, \ldots, C_{im_i}$$

which can be regarded also as properties of the system in which the countermeasures have been introduced against threat T_i. It is assumed that if these countermeasures are implemented, the vulnerability disappears, i.e.

$$C_{i1} \wedge C_{i2} \wedge \ldots \wedge C_{im_i} \Rightarrow \sim T_i$$

We let therefore

$$C_{11}, C_{12}, \ldots, C_{1m_1}, \ldots, C_{n1}, C_{n2}, \ldots, C_{nm_n}$$

be the subclaims of ~B. Note, however, that in this form of AND-decomposition, by contrast to conceptual AND-decomposition, we cannot assert the following:

$$C_{11} \wedge C_{12} \wedge \ldots \wedge C_{1m_1} \wedge \ldots \wedge C_{n1} \wedge C_{n2} \wedge \ldots \wedge C_{nm_n} \Rightarrow \sim B$$

since this result does not follow logically from the premises. This fits well with the fact that we can never know if a system is completely secure, because we can never know whether $T_1, T_2, \ldots, T_n$ really are ALL the possible threats, in which case we would be able to say that

$$B \Rightarrow T_1 \vee T_2 \vee \ldots \vee T_n$$

In this case it would indeed be true that $\sim B$ follows from

$$C_{11}, C_{12}, \ldots, C_{1m_1}, \ldots, C_{n1}, C_{n2}, \ldots, C_{nm_n}$$

and the premises. However, as we have shown, this is not usually the case.

If a security goal requires constraining the system functionality, this will usually take the form of a negative claim, i.e. the system will not do something which might lead to an undesired state. Thus, a negative claim can be used also to specify functionality restriction. Countermeasures could be made by changing the initial functionality of the system, or by adding some recovery functionality in case it is not desirable to adopt the first option.

As an illustration, assume have a claim $\sim B$ where B stands for "confidential information is revealed." After a threat analysis we might discover that B would become true if some threat T, e.g. a malicious action, puts system in an undesirable state where B is true. However, this might be prevented by some kind of countermeasure, e.g. a cryptographic mechanism M whose implementation allows us to claim C, standing for "confidentiality in the system is enforced by M," which becomes a subclaim to the claim $\sim B$, thus precluding T.

It must be noted also that the introduction of several countermeasures together may also introduce new threats; hence, the procedure sketched here should be iterative.

The Methodology in Practice: The PICOS Project

Here we give an account of the first results of the application of our methodology to a real world example, the project PICOS (Privacy and Identity Management for Community Services). PICOS is an international research project focused on mobile communities and community-supporting services, with special emphasis on aspects such as privacy, trust, and identity management. The objective of PICOS is to research, develop, build, test and evaluate an open, privacy-respecting, trust-enabling identity management platform that supports the provision of community services by mobile communication service providers. Assurance is intended to be an integral constituent of PICOS and to be pursued in a holistic manner through application of state-of-the-art assurance methods.

The methodology was designed with the aim of meeting this set of assurance requirements. Our first task, corresponding to the requirements phase, was to analyse the established privacy and trust requirements, and refine them in a way that rendered them suitable for further assurance work. These requirements became the higher level claims in the assurance case under construction. Claims were iteratively decomposed through a series of conceptual AND-decompositions into lower level claims until we reached a level of abstraction that was considered appropriate for carrying out an analysis, basically a vulnerability or threat analysis, of the functionality and components of the system with regard to privacy and trust requirements.

In the subsequent step, corresponding to the architecture phase, we required basically three elements in order to elaborate a first assurance evaluation of it:

- The trust and privacy requirements of PICOS
- The basic (architecture-free) functionality of the PICOS system
- The PICOS components, agents of functional entities intended to implement this functionality

PICOS did not follow any established software development methodology, which forced the assurance team try to instil in the members of the development team the need to follow a more formal and documented development approach.

The basic steps of our approach were the following:

1. investigate the given requirements with the purpose of establishing a set of well-defined high-level trust and privacy principles, called claims in the context of assurance cases, suitable for further assurance work
2. investigate the functionality of PICOS, as described in use cases, with regard to the privacy and trust PICOS principles established in step 1, searching for eventual vulnerabilities
3. analyse the functional components intended to support the functionality of the system with regard to the results of step 2.
4. build the (partial) assurance case tree

The trust and privacy requirements are also called principles in PICOS. Hence, we use both terms indistinguishably below.

The functionality of PICOS was given in the shape of use cases. The use cases were presented in an informal way, and the current evaluation cannot be more formal than the description itself. Moreover, these use cases referred to the architecture of PICOS, and we needed in fact high level use cases without reference to architectural concepts. The solution was to extract, in a first step, the component-free functionality of the system from the description of these use cases, and then to investigate how the resulting functionality fulfilled or contradicted PICOS requirements concerning trust and privacy. At a later stage, the components, countermeasures, or other functionality needed to meet the requirements were introduced in the form of architecture-related claims, and the functionality of the described architectural components was analysed with regard to their adequacy in meeting these claims.

Some of the initial requirements were too abstract and in need of further refinement, for instance the requirement that PICOS must be compliant with all legislation, regulation and best practices that exist in the geographical regions in which the Community operates. A more concrete claim such as "Notice is provided to the Data Subject of the purpose for collecting personal information and the type of data collected," which is a consequence of the previous claim since it is required by the legislation, is more useful. Altogether we obtained 24 privacy principles for PICOS. Together with 8 principles related to trust, we obtained a set of 32 privacy and trust principles that were turned into claims for the PICOS platform. These claims were intended to guide the assurance work during the whole PICOS development lifecycle.

Once these claims had been established, the following task was to investigate whether they were supported by the specified functionality of PICOS. Since the functionality of the PICOS platform is extensive and cannot be established in a single step, we decided to focus exclusively on nine use cases that were provided, as these were intended to describe the basic functionality of PICOS. The approach was to investigate each one of the 32 privacy and trust principles in relation to each one of the use cases, and to establish for each principle whether and how it was relevant to each use case, either because the use case displayed a functionality that supported the principle, or otherwise because it contradicted it or exhibited some form of vulnerability that could lead to a state in which the principle would be negated. The procedure involved a kind of threat and vulnerability analysis, necessarily informal in this context. The use cases were considered only with regard to their external functionality, disregarding at this first phase the defined components intended to implement this functionality or other architectural elements included in the initial use case description. The result was presented as a matrix where the rows represented the principles, the columns denoted the use cases, and a mark in any of the slots of the matrix indicated that there was a relation between the corresponding use case and the principle. A short argument was also given for each existing mark. This was the starting point for the vulnerability analysis, and in general for all further assurance work.

Once the relation between the use cases and the principles was established, the following step was to focus on the architectural aspects and investigate the functionality of the components included in each use case with regard to the established principles. This was done by taking one principle at a time, then each one of the use cases to which the specific principle was considered to be relevant (i.e. there was a mark in the matrix), and then analysing the provided functionality of each component included in the use case description, searching for properties that would be relevant for the corresponding principle. This work resulted in a series of observations about the components, their functionality and dependencies. Several omissions were observed, as well as discrepancies between the description of the components by themselves, and their behaviour as described in the use cases. These observations constituted at this stage the main input of the assurance work to developers for the subsequent stages of development of the PICOS platform.

PICOS Privacy and Trust Principles

The PICOS trust and privacy principles that were initially provided were analysed one by one by the assurance team. We present them in the following sections.

Trust principles

The trust principles were provided in a series of documents. Our analysis yielded 8 basic trust principles, and each one was given a code **TrP** followed by a distinguishing number. The trust principles that were turned into claims for the assurance case are the following:

- **TrP1 Openness and Transparency:** PICOS offers services that handle personal information in an open and transparent way.
- **TrP2 Trust between communities**: PICOS recognises trust as a common currency when exchanged between PICOS communities.
- **TrP3 Provenance**: PICOS ensures that members can rely on the provenance of information.
- **TrP4 External services**: PICOS ensures that externally hosted services are delivered in a trustworthy way and that members are aware when external services are less trustworthy than internal services.
- **TrP5 Audit**: PICOS allows processes to be fully auditable by a trusted entity.
- **TrP6 Objective/subjective trust***:* PICOS supports both objective and subjective methods for assessing trust.
- **TrP7 Consensus:** PICOS guarantees that no single entity can act in a way that might compromise the trust and privacy of the community.
- **TrP8 Member accountability**: PICOS ensures that Members are accountable for their actions while being members of the Community.

Privacy principles

As explained above, the privacy principles that were initially provided had to be analysed, and some of them also refined or even discarded in case they were not considered to be related to privacy. The privacy principles that resulted from this analysis were given a code of the form **PrP** followed by an identifying number.

The first principle was the following:

> **PP1 Compliance with legislation**: The PICOS Architecture must be compliant with all legislation, regulation and best practices that exist in the geographical regions in which the Community operates.

Stated in this way, this principle is not suitable for assessing if the PICOS architecture complies with it. We had to decompose this principle into more concrete ones before evaluating the privacy aspects of the architecture. We considered three legislations, the *EU Data Protection Directive*, the *OECD Privacy Guidelines*, and the *UN Guidelines Concerning Computerized Personal Data Files*. Thereafter, we decomposed the principle into more detailed ones following the classification of privacy principles given in the ISTPA Analysis of Privacy Principles [26]. In this classification, 11 main privacy principles were specified: *Accountability, Notice, Consent, Collection Limitation, Use*

Limitation, Disclosure, Access and Correction, Security/Safeguards, Data Quality, Enforcement, and Openness.

Most of these principles were further decomposed into more concrete principles as follows:

- **Notice**: Notice of Collection, Policy Notification, Changes in Policy or Data Use, Language and Timing of Notification
- **Consent**: Sensitive Information, Informed Consent, Change of Use Consent, and Consequences of Consent Denial
- **Collection Limitation:** Limitation of Consent, and Fair and Lawful Means
- **Use Limitation:** Acceptable Uses and Data Retention
- **Disclosure:** Third Party Disclosure, Third Party Policy Requirements, and Disclosure for Legal and Health Reasons
- **Access and Correction:** Access to Information, Proof of Identity, Provision of Data, Denial of Access, and Correcting Information
- **Security/Safeguards:** Safeguards and Destruction of Data
- **Enforcement:** Ensuring Compliance, Handling Complaints, and Sanctions
- **Openness:** Public Policies and Establishing Existence of Personal Data

These principles are now in a sufficiently concrete form to be analysed in the context of the functionality described in the use cases. As an example, we show the analysis corresponding to the Notice principle. This principle was decomposed into 5 principles, as shown above:

1. Notice of Collection
2. Policy Notification
3. Changes in the Policy or Data Use
4. Language
5. Timing of Notification

After checking the relevant legislation, the principle Language was discarded, as it is not clearly covered by the relevant legislation. The other principles were maintained, yielding four privacy principles. Here we present the one of them, *Notice of Collection*, which was given the code *PrP1*:

> **PrP 1 Notice of Collection**: *Notice is provided to the Data Subject of the purpose for collecting personal information and the type of data collected.*

The relevant legislation says the following about this principle:

> **EU**: *Member States shall provide that the controller or his representative must provide a data subject from whom data relating to himself are collected with at least one of the following information, except where he already has it:*
> …
> *(b) the purposes of the processing for which the data are intended* (Article 10)
>
> **OECD**: *The purpose for which personal data are collected should be specified not later than at the time of data collection* (Paragraph 9)
>
> **UN**: *The purpose which a file is to serve and its utilization in terms of that purpose should be specified, legitimate and, when it is established, receive a certain amount of publicity or be brought to the attention of the person concerned* (paragraph 3)

In the light of this, it was decided that the principle should be included among the PICOS claims. Further analysis of the use cases showed that at least one PICOS use case, concerned with the registration process and explained below, was relevant for this principle.

After full analysis of the provided privacy requirements, we were able to establish the list of privacy principles for PICOS, presented in Table 1.

Privacy Principle	Name
PrP 1	Notice of Collection
PrP 2	Policy Notification
PrP 3	Changes in Policy or Data Use
PrP 4	Timing of Notification
PrP 5	Sensitive Information
PrP 6	Informed Consent
PrP 7	Change of Use Consent
PrP 8	Consequences of Consent Denial
PrP 9	Limitation of Collection
PrP 10	Fair and Lawful Means
PrP 11	Acceptable Uses
PrP 12	Data Retention
PrP 13	Third-Party Disclosure
PrP 14	Third-Party Policy Requirements
PrP 15	Access to Information
PrP 16	Provision of data
PrP 17	Correcting Information
PrP 18	Safeguards
PrP 19	Data Accuracy
PrP 20	Public Policies
PrP 21	Data Management
PrP 22	End-to-end privacy
PrP 23	Authentication
PrP 24	Multiple Persona

Table 1. List of Privacy Principles

PICOS Use Cases

We investigated thereafter the relation between the functionality of PICOS, as expressed in the use cases, and the privacy and trust principles of PICOS, presented in the previous section. We analysed each one of the 32 privacy and trust principles in the light of each one of the use cases, and established whether the principle in question was relevant to each use case.

Some privacy principles turned out to be unrelated to all the use cases. These principles turned out to be related to the management of the community members' profiles, which suggested that a 10th use case should have been defined, concerned with personal data management. The assurance team recommended that a new *data management* use case be defined in order to fill this important gap in the description of the functionality of PICOS.

The nine PICOS use cases, with a short description of its objectives, are the following:

- **UC1: Registration**: *registering and creating of a new member profile; creating an initial identity, importing reputation, setting policies and respecting different roles.*
- **UC2: Accessing the community**: *identifying, authenticating and granting authorisation to a member; selecting a service.*
- **UC3: Revocation**: *leaving a community, giving due consideration to content contributed while a member.*

- **UC4: Multiple partial identities**: *creating, selecting and managing multiple member identities (pseudonymous or partial identities).*
- **UC5: Reputation**: *establishing the reputation of members within and across communities; providing recommendation and feedback; registering to receive notifications.*
- **UC6: External services**: *exposing partial identity profiles to external services.*
- **UC7: Content sharing**: *importing/exporting and controlling the sharing of content contributed to the community by members, including automatic/manual tagging and notification.*
- **UC8: Presence**: *setting and controlling the sharing of online status information (location, presence, etc.) about members.*
- **UC9: Sub-community**: *creating and managing a sub-community (sub-group) within the overall community.*

Each claim, except two, was shown to be relevant to only a subset of the use cases. Table 2 shows the matrix telling which trust principles were considered, after due analysis, to be relevant for each use case. From the table we can see that **TrP5 Audit** is relevant to all the use cases. We also see that use cases 5, 6 and 7, regarding Reputation, External Services and Content Sharing respectively are the ones that involved more trust principles.

	UC1	UC2	UC3	UC4	UC5	UC6	UC7	UC8	UC9
TrP 1						Y		Y	Y
TrP 2					Y	Y			
TrP 3					Y	Y	Y		
TrP 4						Y			
TrP 5	Y	Y	Y	Y	Y	Y	Y	Y	Y
TrP 6				Y	Y	Y	Y		
TrP 7									Y
TrP 8	Y			Y			Y		

Table 2. Use cases vs. Trust principles

Table 3 shows the same with respect to the privacy principles. We can see here that use cases 1 and 3 are the ones that are more related to personal data. We can also see that principle 24, Multiple Persona, is relevant to all the uses cases.

	UC1	UC2	UC3	UC4	UC5	UC6	UC7	UC8	UC9
PrP 1	Y								
PrP 2	Y								
PrP 3									
PrP 4	Y								
PrP 5	Y								
PrP 6	Y								
PrP 7									
PrP 8	Y								
PrP 9	Y								
PrP10	Y			Y					
PrP11		Y				Y			Y
PrP12			Y						Y
PrP13		Y				Y	Y		Y
PrP14		Y				Y			
PrP15									
PrP16									
PrP17									
PrP18	Y								
PrP19	Y								
PrP20									

	UC1	UC2	UC3	UC4	UC5	UC6	UC7	UC8	UC9
PrP21	Y		Y	Y		Y	Y		Y
PrP22									
PrP23	Y	Y		Y			Y		
PrP24	Y	Y	Y	Y	Y	Y	Y	Y	Y

Table 3. Use Cases vs. Privacy Principles.

Thereafter, a more extensive analysis was provided on the trust and privacy principles relevant for each use case. Below, we present a summary of the analysis that was carried out corresponding to UC1, Registration.

UC 1: Registration

Description of the use case*:* *All members of the community must be registered if they wish to have full access to the facilities on offer. Guest access is also permitted, but the range of services available to a Guest is severely restricted. In order to gain access to the full range of services, registration is necessary. Registration provides the community operator with assurance that a member is suitable to join the community.*

Every member has one root identity identifying him or her, which is assigned when they register with the community. They are immediately allocated a partial identity when they first interact with the community. Partial identities are basically pseudonyms; a member of the community may have several of those. All identities, including root, have a profile. A member can only have one root identity. Roles other than member, e.g. community administrator, are treated as special cases.

Thereafter, the results of the analysis of the privacy and trust principles relevant to this use case were presented as follows:

- **PrP 1 Notice of Collection**: *Notice of the purpose for collecting profile information must be provided to the member.*
- ***PrP 2 Policy Notification**: Member should be notified during registration of the applicable PICOS policies in terms of Consent, Access and Disclosure.*
- **PrP 4 Timing of Notification***: Policy notification should be given to the member before data is collected.*
- **PrP 5 Sensitive Information***: The member must be informed of, and explicitly consent to, the collection, use and disclosure of sensitive information.*
- **PrP 6 Informed Consent***: The member must provide consent on the collection of personal information during registration.*
- **PrP 8 Consequences of Consent Denial***: Where relevant, the member must be made aware of the consequences of denying consent to the provision of certain personal information. For instance, not providing the age may block the user for some functionality but may allow for a limited use of some service, and not giving the phone number may imply not receiving certain kind of information.*
- **PrP 9 Limitation of Collection***: Only personal information relevant to the purpose of the community should be collected.*
- **PrP 10 Fair and Lawful Means***: Collection of information during registration must be made by fair and lawful means.*
- **PrP 18 Safeguards***: It is necessary to provide mechanism to secure the registration process, so that data is not tampered or eavesdropped, the member is not impersonated by a malicious user, etc. For instance, SSL can be used for securing commutation or CAPTCHA for being sure that only "humans" can register to the community.*
- **PrP 19 Data Accuracy***: The registration process must ensure that the personal data collected is accurate.*
- **PrP 21. Data Management***: The community must allow members to set their preferences for the use of their personal data and to establish at least the basic principles for sharing content data during registration.*
- **PrP 23 Authentication***: Authentication information has to be either collected or generated during registration.*
- **PrP 24 Multiple Persona***: During registration it is possible create one or several partial identities.*
- **TrP 24 Audit***: All actions performed during registration are logged.*
- **TrP 8 Member Accountability***: Members must provide accurate personal information and be made accountable for this. If this data is not accurate the community may decide to take actions against the user, like removing them from the community.*

These observations were intended to guide the future development work with regard to the established trust and principle objectives.

PICOS Component Analysis

In this section we focus on the architectural aspects of PICOS and give an overview of the analysis performed concerning the functionality of the components with regard to the established principles. Previously, the system was regarded only as a black box. Now we take into consideration the components involved in the use cases. These are intended to work together in order to provide the external functionality of the system as described in the use cases. The properties associated with each component must reflect in some sense the properties of the initial system, but how this happens is a complex issue which defies formalization. Our analysis must be, at least at this stage, necessarily ad hoc. Later, standard patterns may emerge that could provide some aid in this work.

We proceeded by taking one principle at a time, concentrating on each of the use cases to which the specific principle is considered to be relevant, and finally analysing the functionality of the components of the use case that were considered to be relevant for the principle in question. The result was a series of observations about the description of the corresponding component, especially with regard to its general functionality and dependencies to other components. Several omissions and gaps could be observed, as well as discrepancies between the description of the components and their behaviour within the use case descriptions. These observations constituted an important input of the assurance effort to the developers with regard to the design and implementation of the components.

In order to understand the relevance of each specified component with regard to the established trust and privacy principles, we needed also to analyse, principle by principle, how the functionally defined for the component in a given use case related the each principle. This is the subject of the next section.

How the Components Support the Principles

The principles must now be analysed in the light of each use case and the components involved. Each principle is thus be related to each relevant use case, and the components involved in the use case and regarded to be relevant for the principle are listed and analysed. Sometimes, a component not mentioned in the use case but deemed relevant for the principle is also included. For each one of the listed components a detailed analysis is provided concerning the role it plays in the use case with regard to the given principle. We focus on such questions as whether the described behaviour of the component is consistent with the behaviour required by the use case, whether any functionality is missing, and similar issues.

We illustrate this by providing one relatively simple example, the analysis of the privacy principle PrP12 *Data Retention*. We start by giving the definition of the principle:

> **PrP12: Data Retention**: *Personal Data is retained no longer than necessary to complete the stated purpose.*

Two use cases were judged to be relevant for this principle. One of them was related to sub-community management, UC9, and is concerned with the management of sub-communities created by members of the community. Information concerning the initiator of this community is supposed to be stored, and therefore the principle of data retention should be considered whenever a sub-community is deleted. There is nevertheless no mention about revocation in the description of the Sub-community component, a gap which we recommended should be filled in later descriptions. After an analysis of the components involved in UC9, only one component in the use case, called the *Revocation* component, was regarded to be relevant for this principle.
The analysis concerning other principles could be more complex than this one, often involving several use cases and components.

PICOS Assurance Case

The claims at highest levels of the PICOS assurance case tree correspond to the trust and privacy principles presented above. The strategy for deriving subclaims from claims at this level was basically conceptual AND-decomposition.

The strategy for deriving *architectural claims* from the principles, on the hand, was based on the study of the provided use cases and on an informal but careful threat and vulnerability analysis.

The structure thus obtained was intended to suffer changes, adjustments, refinements and extensions throughout the system development life cycle of PICOS. The tree will only be completed when evidence becomes available at a later stage of development.

We give below a snapshot of the current assurance case tree and the way it was constructed. The highest level claim is the following:

> **Claim 1**: PICOS complies with all established trust and privacy principles.

The second step is a simple conceptual AND-decomposition separating trust from privacy goals:

> **Claim 1.1**: PICOS complies with all established trust principles.

> **Claim 1.2**: PICOS complies with all established privacy principles.

Focusing on privacy, the next step is a simply enumeration of the privacy requirements initially provided. We show below only the first one:

> **Claim 1.2.1**: PICOS complies with the privacy legislation in force.

This claim was decomposed further according to the classification given in [26], as explained above. Again, we show only the first subclaim here:

> **Claim 1.2.1.1**: *PICOS complies with the notice principle.*

This principle was further subdivided in [26] into 5 principles concerning notice of collection, where only 4 of them was considered relevant according to the legislation in force. The first one was the following:

> **Claim 1.2.1.1.1**: *PICOS provides notice to the data subject of the purpose for collecting personal information and the type of data collected.*

This claim is now on a level of abstraction suitable for an analysis against the specified functionality of PICOS. This is relatively easy to do in this case because the claim is basically a functional requirement that does not need an extensive threat and vulnerability analysis, only the introduction of some functionality to implement it. After the analysis we concluded that this principle was very relevant to the registration use case, as well as to a missing use case related to data management, since the principle should be enforced not only during registration, but also whenever any data collection should occur. In the context of registration, we considered that only one of the components engaged in the registration use case, the *Registration* component, needed be involved in providing this functionality. Hence we arrived at two new subclaims as follows:

> **Claim 1.2.1.1.1.1**: *Notice of collection is provided at registration time by the Registration component.*

Claim 1.2.1.1.1.2: *Notice of collection is provided during data management with the aid one or several currently unspecified components.*

As can be seen from this short overview of the assurance case, its complexity renders it almost unmanageable without the aid of still unavailable tools.

Conclusions and Future Work

We have introduced in this paper an assurance methodology inspired by Assurance Based Development approach. The methodology is growing out of our experience in providing trust and privacy assurance to the evolving European project PICOS. It is therefore not a pure theoretical construction. The lack of robust theoretical results and best experiences in the area forced us to adopt this experimental approach. So far, no major practical or theoretical obstacles have been met.

It must be stressed that we regard the postulates of our methodology as tentative so far and in need of being validated by their further application in the development of other real-world systems. The main issue is whether the needed information associated with the different models of a system under construction, as well as their dependencies, can be extracted in the way we envisage. Some development methodologies would certainly make this task easier.

Suggestions for future work include the development of a dedicated tool and of a metrics allowing us to assign different weights to different claims.

The need for a tool was felt from the beginning of the project. We were confronted with three kinds of entities: claims, use cases, and components. There were in total 32 claims or principles, 9 use cases, and a score of components. The number of relations among these elements is therefore very high and renders the analysis work almost unmanageable. Each one of those three entities brings forth a different view of the assurance case. The first view corresponds to claims: given a certain claim, what is its relation to the use cases, or to the components? The second view corresponds to use cases: given a certain use case, which principles are relevant to it? The third view is the component view: given a component, what is its relation to the use cases and to the principles? These are questions that a designer or developer would find appropriate to ask. A tool would facilitate answering this sort of question. It would also facilitate building the assurance case tree and keeping track of the way one entity is related to the others, for instance which principles are relevant to a given component, or the other way around, which components are relevant to a given principle. This information would be especially helpful whenever changes are made to either components or use cases, or even claims, helping us in this way to update the assurance case tree when changes in any of the entities are introduced.

Finally, the introduction of a metrics would greatly enhance the assurance case. Often, a subclaim can be more or less important for the strength of a claim, something which it is not possible to express in our assurance case. Moreover, a claim is seldom simply true or false: what we usually have is more or less evidence or confidence on the validity of a claim. A metrics would allow us to express this, and also to track the impact that a certain change in the validity of some evidence might have on the claims that depends on this evidence. It would become possible also to determine the level of confidence required from the evidence if the amount of confidence on a claim that depends on this evidence is required to higher than a given level. In general, a metrics would greatly enhance the quality of an assurance case, rendering it more realistic and enabling us to better capture the inherently fuzzy nature of real world evidence.

Acknowledgements. The research leading to these results has received funding from the European Community's Seventh Framework Programme (FP7/2007-2011) under grant agreement n° 215056.

References

[1] "Privacy and Identity Management for Community Services" (PICOS) http://www.picos-project.eu

[2] Patrick J. Graydon, John C. Knight, and Elisabeth A. Strunk. Assurance Based Development of Critical Systems. In 37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'07), pp. 347-357, 2007.

[3] John Goodenough, Howard Lipson, and Chuck Weinstock. Arguing Security - Creating Security Assurance Cases, Carnegie Mellon University, 2007. Available at https://buildsecurityin.us-cert.gov/daisy/bsi/articles/knowledge/assurance/643-BSI.html (last accessed September 19th, 2008).

[4] Y. Jacobson, M. Christerson, P. Jonsson, and G. Overgaard. Object-Oriented Software Engineering - A Use Case Driven Approach. MA: Addison Wesley, New York: ACM Press, 1992.

[5] Steve Dawson. The genesys of Cyberscience and its mathematical Models. SRI International, System Design Laboratory. Technical Report, number AFRL-IF-RS-TR-2005-49.

[6] Kalloniatis C., Kavakli E., Gritzalis S., "Addressing Privacy Requirements in System Design: The PriS Method", Requirements Engineering, Vol.13, No.3, pp.241-255, 2008

[7] L. Liu, E. Yu, and J. Mylopoulos. Security and Privacy Requirements Analysis within a Social Setting, in 11th IEEE International Requirements Engineering Conference (RE'03), Monterey Bay, CA, pp. 151-161, 2003.

[8] L. Liu, E. Yu and J. Mylopoulos. Analyzing Security Requirements as Relationships among Strategic Actors, SREIS'02, e-proceedings, Raleigh, NC, 2002.

[9] Lawrence Chung. Dealing with Security Requirements During the Development of Information Systems. In Proceedings of Advanced Information Systems Engineering, LNCS 685, pp. 234-251, 1993.

[10] A. van Lamsweerde and E. Letier. Handling Obstacles in Goal-Oriented Requirements Engineering. In IEEE Transactions on Software Engineering, Special Issue on Exception Handling, Vol. 26 No. 10, October 2000, 978-1005.

[11] John Mylopoulos , Lawrence Chung , and Brian Nixon. Representing and Using Non-Functional Requirements: A Process-Oriented Approach, in IEEE Transactions on Software Engineering, vol. 18, no. 6, June 1992, pp. 483-497.

[12] Christos Kalloniatis, Evangelia Kavakli, and Stefanos Gritzalis. Security Requirements Engineering for eGovernment Applications: Analysis of Current Frameworks. In Proceedings of the DEXA'04 EGOV'04 3rd International Conference on Electronic Government, LNCS 3183, pp. 66-71, September 2004, Zaragoza, Spain.

[13] Paco Hope, Gary McGraw and Annie I. Antón. *Misuse and Abuse Cases: Getting Past the Positive.* IEEE Security and Privacy, 2:3, pp. 90-92, May-June, 2004.

[14] Software Security Assurance: A State-of-the-Art Report (SOAR), July 31, 2007.

[15] John Wilander and Jens Gustavsson . Security requirements – a field study of current practice. In: E-Proceedings of the Symposium on Requirements Engineering for Information Security, 2005.

[16] Elisabeth A. Strunk and John C. Knight. The Essential Synthesis of Problem Frames and Assurance Cases. In Proceedings of 2nd International Workshop on Applications and Advances in Problem Frames, co-located with 29th International Conference on Software Engineering, Shanghai, May 2006.

[17] Kelly To. A Systematic Approach to Safety Case Management. In Proccedings SAE 2004 World Congress, Detroit, US, 2004.

[18] Michael A. Jackson. Problem Frames: Analysing and Structuring Software Development Problem. Addison Wesley Publishing Company, 2001.

[19] Jon G.Hall, Lucia Rapanotti. Assurance-Driven Design. In The Third International Conference on Software Engineering Advances, 2008 (ICSEA '08), pp. 379 – 388, Oct. 2008.

[20] Jon G.Hall, Lucia Rapanotti, and Michael Jackson. Problem Oriented Software Engineering: A design-theoretic framework for software engineering. In Proceedings of the Fifth IEEE International Conference on Software Engineering and Formal Methods, pp. 15-24, 2007.

[21] R E Bloomfield, P G Bishop, C C M Jones, and P K D Froome. ASCAD—Adelard Safety Case Development Manual, Adelard 1998, ISBN 0 953377105.

[22] Ankrum T. Scott Ankrum and Alfred H. Kromholz. Structured Assurance Cases: Three Common Standards' (slides presented at the Association for Software Quality [ASQ] Section 509 Software Special Interest Group meeting, McLean, VA, January 23, 2006.
[23] Kim G. Larsen and Liu Xinxin. Compositionality Through an Operational Semantics of Contexts. Journal of Logic and Computation 1991 1(6):761-795, 1991.
[24] Robin Milner. A Calculus of Communicating Systems, Springer Verlag, ISBN 0-387-10235-3. 1980.
[25] Matthew Hennessy and Robin Milner. On observing nondeterminism and concurrency. In J. W. de Bakker and Jan van Leeuwen, editors, Automata, Languages and Programming, 7th Colloquium, volume 85 of Lecture Notes in Computer Science, pages 299-309, Noordweijkerhout, The Netherland, 14-18 July 1980. Springer-Verlag.
[26] ISTPA International Security Trust and Privacy Association, Analysis of Privacy Principles: Making Privacy Operational, Version 2.0, May 2007.
[27] C. Potts. Using Schematic Scenarios to Understand User Needs. In Proc. DIS'95 - ACM Symposium on Designing interactive Systems: Processes, Practices and Techniques, University of Michigan, August 1995.