SERENITY
System Engineering for Security & Dependability

Antonio Maña

UNIVERSIDAD DE MÁLAGA

Towards a rigorous IT security engineering

Invited talk

Information Society Technologies

FLACOS'09 - Toledo, Spain, 25-Sep-09

Next 50 minutes

- We will advocate **a change of paradigm** based on
  - the precise and formal specification of security solutions and security properties; and
  - the use of these formally verified properties as the basis for the expression of requirements and the engineering of secure systems.

- We will also introduce two pillars to solve the situation.
  - On the one hand, we present the **SERENITY model** of secure and dependable systems and show how it supports the creation of secure and dependable systems for these new computing paradigms.
  - On the other hand we discuss the concept of **contract** and the role it plays in ensuring a rigorous treatment of security.

- Finally some **conclusions** will be drawn and we'll present a **proposal** for establishing a renewed security engineering discipline.

SERENITY
System Engineering
for Security & Dependability

The current state of affairs

Chapter I – A sad story

Information Society
Technologies

If we need security (and we do)…

- More and more aspects of our daily lives are affected by computing systems.
- Developing secure systems is still an art
- IT Security is treated as an add-on
- The complexity of these systems is becoming larger than the capacity of humans to understand and secure such IT system
- **Security engineering**
  - has been used to denote partial approaches that cover only small parts of the processes that are required in order to create a secure system, like modelling, verification, programming, etc.
- Guidelines, recommendations, best practices, certification and similar approaches lack the necessary rigour and precision
  - Some examples are: Common criteria, traditional security patterns, Sarbanes-Oxley and HIPA Acts, etc

...we need security engineering

- The existence of a rigorous IT security engineering discipline could
  - not only change that situation of lack of guarantees; but also
  - improve the security of IT systems by allowing them to be prepared to operate in unforeseen contexts; and
  - allow us to successfully provide security to the extremely complex and dynamic IT systems that are coming in the near future.

Threat-based security engineering considered harmful

Chapter II – The good one turned villain

FLACOS'09 - Toledo, Spain, 25-Sep-09

Threat-based security engineering

- It once was an appropriate methodology to address the design of secure systems… back when systems were:
  - static;
  - under control of a single entity;
    - That is, designed; developed; deployed and operated by an entity
  - operated in a predictable environment; and
  - having a small to medium complexity level
- But now that circumstances have changed it has turned into a source of problems

Threat-based security enginering
Why it is not appropriate

- **Lack of dynamism and support for evolution**
  - Threats change when systems change even if the protection goals remain the same
  - Full reengineering is required in order to cope with any context change
  - Very dificult to identify changes required in the system, as a result of a context change
- **Poor traceability**
  - Threats are dificult to trace to protection goals or security requirements
  - Systems engineered following a threat- based approach tend to be extremely dificult to maintain and to adapt to new context conditions
- **Expression of user requirements is lacking, not precise or context-dependent**
  - Using threats as the input to design process hides the user requirements →
  - Reduces the longevity and stability of the system specification and the system under development becomes weaker in terms of:
    - maintainability;
    - traceability; and
    - resilience to evolution

Threat-based security enginering
Why it is not appropriate

- **Completeness**
  - The complete set of threats is impossible to identify
  - Future systems will make it harder even for the most experienced and visionary security experts
- **The infamous penetrate-and-patch vicious cycle**
  - New threats and vulnerabilities are discovered during system operation requiring the system to be patched →
  - degradation of the quality and introduction of new vulnerabilities
- **Assurance and certification should not be based on threats**
  - Stating (even proving) that a system can withstand a threat does not say much about what can be guaranteed about the system
- **Poor user communication**
  - "Don't worry, your system will be secure because we'll protect it against cross-site scripting and will use authenticated TLS connections"
  - Customers do not understand if this solves their problem
  - Abuse and misuse cases can help, but still they lack precision and do not provide the guarantees that customers need

SERENITY

Chapter III – Things can improve (I)

FLACOS'09 - Toledo, Spain, 25-Sep-09

What's so new in the new computing paradigms?

- Computing ecosystems (common to many future computing paradigms) will offer
  - **highly distributed,**
  - **dynamic** services,
  - **without a common owner or controller,**
  - where availability and state of elements is **unpredictable**
- in environments that will be
  - **heterogeneous,**
  - **large scale and**
  - **nomadic**,
- where computing nodes will be omnipresent and communications infrastructures will be dynamically assembled,
- and where humans are part of the "system".

Security and Dependability in the new computing paradigms

- Provision of security and dependability for these ecosystems will be **increasingly difficult** using existing security solutions, engineering approaches and tools because of the combination of
  - distributed nature,
  - heterogeneity,
  - scale (size, complexity),
  - dynamism,
  - lack of central control,
  - unpredictability,
  - human presence (with all that it implies)
  - along with the higher demands for privacy, dependability and security.

SERENITY
System Engineering for Security & Dependability

Challenges for S&D

Security and Dependability

- High complexity, large scale and dynamic nature of these computing ecosystems
  - Not possible to foresee all possible situations and interactions which may arise
    - Infrastructure security cannot be flexible enough
  - Not possible to create suitable solutions to address the users' security and dependability requirements
- Lack of control
  - S&D engineers will be faced with pieces of software, communication infrastructures and hardware devices not under their control.
    - Application-level security will not be sufficient
    - Runtime monitoring emerges as an essential element

Information Society Technologies

UNIVERSIDAD DE MÁLAGA

Challenges for S&D

- Unpredictability
  - Traditionally, S&D Engineers have been faced with complex but static and predictable systems
    - existing tools and processes not well-suited for these new environments
  - Again, calls for monitoring and context awareness
- Human presence
  - Affects unpredictability, dynamism, requirements for S&D …
  - Privacy becomes a major concern
  - "Interfacing" to humans becomes a key issue
- Increased needs for S&D
  - Applications not only interact with humans
    - humans are part of the system
  - Systems are intrinsically sensible
  - Applications run on non trusted devices (infrastructure)
  - Systems must survive
    - Evolution of infrastructure / context
    - Evolution of threats
    - …

General challenges

- The concepts of ***system*** and ***application*** as we know them nowadays will disappear,
  - from static architectures with well-defined pieces of hardware, software, communication links, limits and owners,
  - to open architectures that are sensitive, adaptive, context-aware and responsive to users' needs and habits.
  - Precisely, this is what we refer to as ***Computing ecosystems***.

SERENITY in a nutshell

Ambient Intelligence

- **OBJECTIVE**:
  - SERENITY was launched as an initiative for the provision of Security and Dependability (S&D) in AmI ecosystems
- **APPROACH**:
  - SERENITY was based on capturing the knowledge of S&D Engineers and making it available for automated processing
- **FEATURES**:
  - SERENITY supports the provision of S&D both at development time and at runtime
  - SERENITY deals with both static and dynamic aspects
  - SERENITY considers S&D at different abstraction levels
  - SERENITY research has been guided by carefully selected scenarios

How does SERENITY deal with these problems? (I)

- STATEMENT
  - Providing S&D in AmI scenarios requires the ***dynamic application of the expertise of S&D engineers***.
    - SERENITY aims at capturing this expertise and making it available in the above-mentioned scenarios
- LINE
  - ***S&D Patterns*** are the means, complemented by ***Runtime Monitoring*** mechanisms and tools
- COVERAGE
  - In order to cover the complete lifecycle of S&D solutions, SERENITY provides
    - state-of-the-art **techniques and tools for the analysis of S&D Solutions** at different levels; and
    - **Development time and runtime frameworks** to support the automated provision of S&D for AmI applications

How does SERENITY deal with these problems? (II)

- **MAIN POINTS**
  - Separate roles with well-defined responsibilities and techniques
  - Capture precise, machine-understandable knowledge
  - Provide runtime support based on the previously captured knowledge and on context awareness
  - Relate all elements in the system lifecycle

- **MAIN GLOBAL RESULT**
  - SERENITY Lifecycle Model: A model of system lifecycle based on the abovementioned main points

SERENITY Lifecycle Model: actors

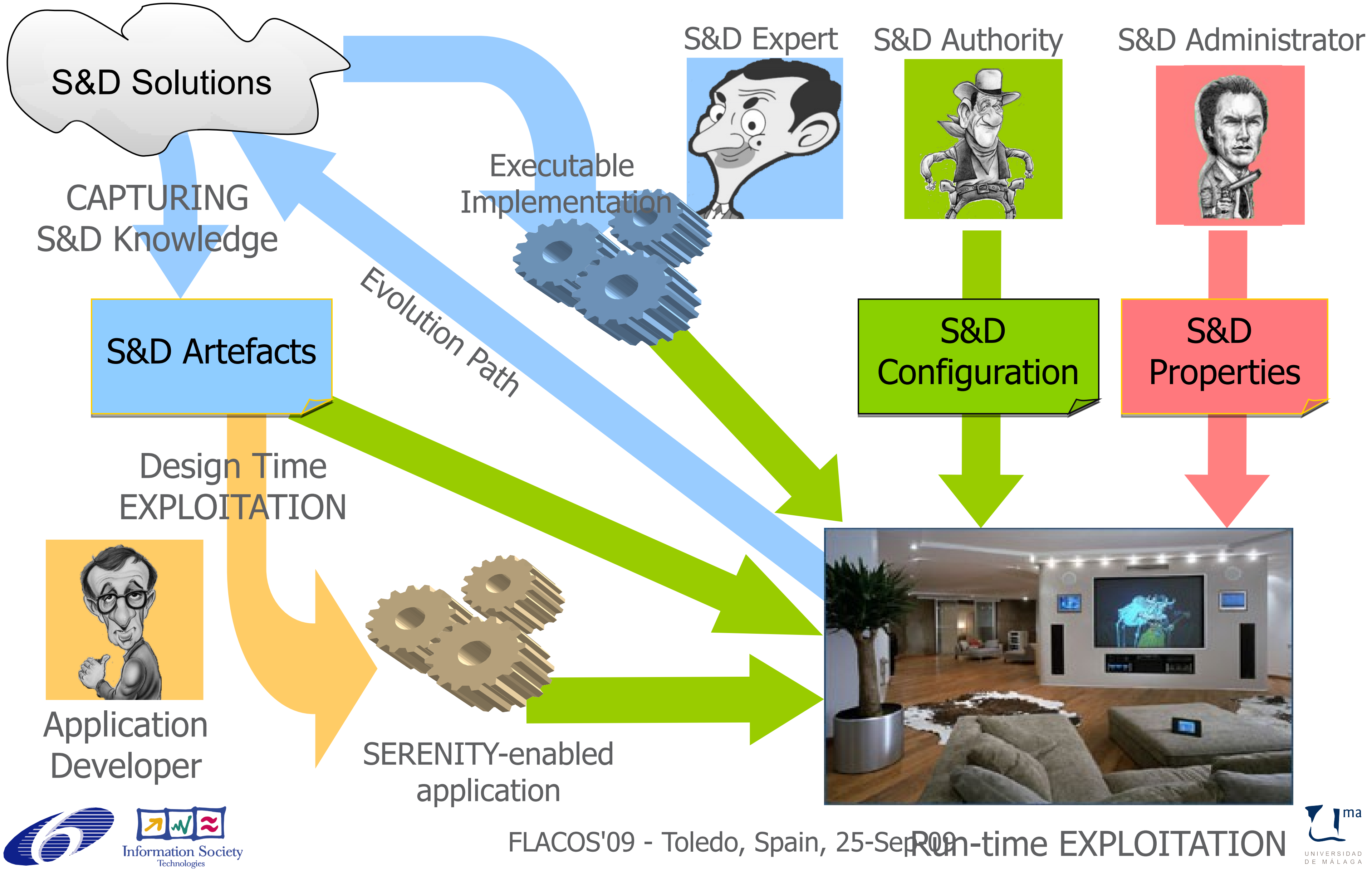
- ACTOR: **S&D Solution Developer**
  - Develops S&D Solutions
  - Creates S&D Artefacts

- ACTOR: **Application Developer**
  - Identifies S&D Requirements
  - Develops applications

- ACTOR: **S&D Authority**
  - Defines S&D Configuration
  - Manages SRF Library

- ACTOR: **Security Officer / S&D Administrator**
  - Defines S&D Properties and S&D Policies

SERENITY Lifecycle Model: Overview



Main individual technologies/results involved

- **Analysis and development of S&D Solutions**
  - At different abstraction levels
- **Modelling S&D**
  - Not only solutions, but also requirements, properties, policies, context, …
- **Development time support**
  - Solution discovery, selection, adaptation and integration
- **Runtime support**
  - Solution selection and dynamic management
- **Runtime monitoring**
  - In open, distributed and uncontrolled scenarios
- **System evolution**
  - Based on the runtime support

Analysing and developing S&D Solutions

- ACTOR: **S&D Solution Developer**
  - S&D Experts: Analyse solutions, create and validate S&D Artefacts
  - S&D Engineers: implement executable components

- Security expert in research and development
- Standardisation bodies for S&D technology
- Implementer for S&D building blocks/services
- Verification, validation and certification
- Expert in legal issues
- ...

Analysing and developing S&D Solutions

S&D Experts

- Mechanisms to promote the reuse of their solutions
- Mechanisms to support interoperability of their solutions
- Tools to analyse and verify their solutions
- Means to precisely describe the solutions (including security properties, context in which the solution is applicable, monitoring rules
- Means for guidelines and generic interfaces for secure implementations
- Mechanisms to relate different solutions and different properties

Scope of support for analysis and verification of S&D solutions

Organisational and legal requirements:

Static trust relations, language for requirements specifications tools for design and validation of patterns.

Workflow and services:

Processes, dynamic behaviour, relies on underlying engines. Needs both, static and process-oriented view. Two different languages. Tool support for BPEL patterns.

Networks and devices:

Large variety of requirements and solutions. Requirements language provides formal semantics for many of them. Tool for validation within AmI scenarios. Patterns range from validated solutions to best practice solutions.

Analysing and developing S&D Solutions

- **S&D Engineers**
  - Support for the development of executable components.
  - Library to make S&D implementations available to the system engineer
  - Provide validated and precisely specified solutions and standards
  - Make implementations available with all information given by the security expert, i.e. provide enough information for a secure deployment

Main individual technologies/results involved

- **Analysis and development of S&D Solutions**
  - At different abstraction levels
- **Modelling S&D**
  - Not only solutions, but also requirements, properties, policies, context, …
- **Development time support**
  - Solution discovery, selection, adaptation and integration
- **Runtime support**
  - Solution selection and dynamic management
- **Runtime monitoring**
  - In open, distributed and uncontrolled scenarios
- **System evolution**
  - Based on the runtime support

Modelling S&D: Basic artefacts

- Modelling S&D Requirements
  - S&D Properties
  - S&D Policies
- Modelling S&D Solutions:
  - S&D Classes
  - S&D Patterns
  - S&D Implementations
  - Executable implementations

Modelling S&D Solutions

- **SERENITY Artefacts**:
  - **S&D Patterns** represent abstract S&D solutions that provide one or more S&D Properties. The popular *Needham-Schroeder public key protocol* is an example of an S&D solution that can be represented as an S&D Pattern.
  - **S&D Classes** represent S&D services (abstractions of a set of S&D Patterns characterized for providing the same S&D Properties and being compatible with a common interface). An example of an S&D Class is the *ConfidentialCommunicationClass*, which defines an interface including among others, an abstract method `SendConfidential(Data, Recipient)`.
  - **S&D Implementations** represent operational S&D solutions, which are in turn called **Executable Components**. It is important to note that the expression “operational solutions” refers here to any final solution (e.g. component, web service, library, etc.) that has been implemented and tested for compliance with the corresponding S&D Pattern.

Main individual technologies/results involved

- **Analysis and development of S&D Solutions**
  - At different abstraction levels
- **Modelling S&D**
  - Not only solutions, but also requirements, properties, policies, context, …
- **Development time support**
  - Solution discovery, selection, adaptation and integration
- **Runtime support**
  - Solution selection and dynamic management
- **Runtime monitoring**
  - In open, distributed and uncontrolled scenarios
- **System evolution**
  - Based on the runtime support

SERENITY supports

- **Identifying and expressing requirements**
  - Mechanisms to express, analyse and relate S&D Properties
  - Mechanisms to comply S&D Policies (both enterprise-wide and others)
- **Developing applications**
- A catalogue of precisely described S&D Solutions available at development time
- Mechanisms to match the application requirements and the solutions that can be used to fulfil these requirements
- Mechanisms to enhance independence from S&D solutions, allowing runtime adaptation to unforeseen context conditions and to support persistence of the application in the future
- Trust mechanisms

Application Development Support API

- The SERENITY Development Time Framework supports developers in finding the solutions they need
- The Serenity Application Support Library gives us access to methods that help us to build SERENITY-aware applications.
- This API encapsulates all the communication with the SRF and the mechanisms used to access the S&D services provided by ECs.

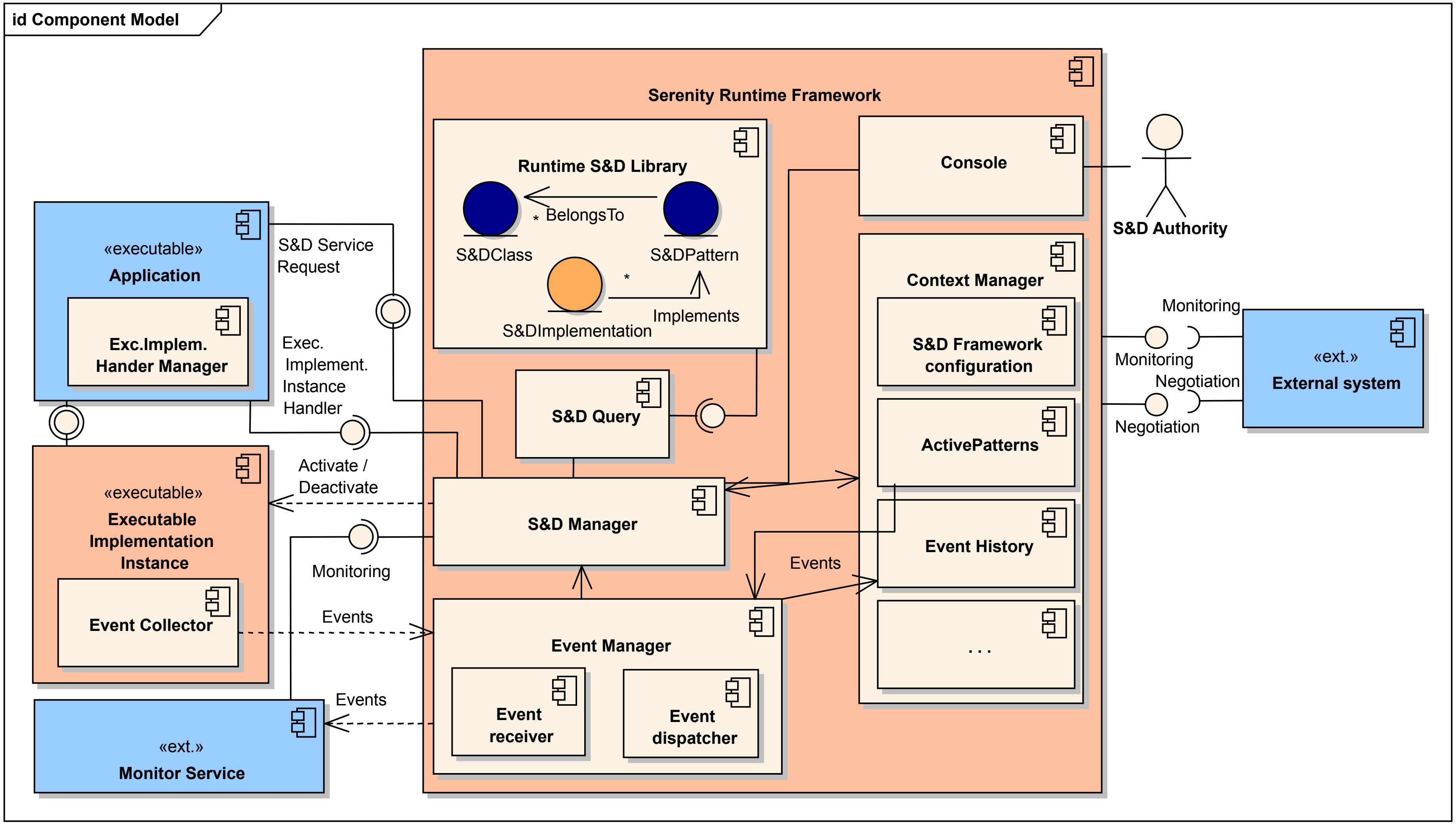
Main individual technologies/results involved

- **Analysis and development of S&D Solutions**
  - At different abstraction levels
- **Modelling S&D**
  - Not only solutions, but also requirements, properties, policies, context, …
- **Development time support**
  - Solution discovery, selection, adaptation and integration
- **Runtime support**
  - Solution selection and dynamic management
- **Runtime monitoring**
  - In open, distributed and uncontrolled scenarios
- **System evolution**
  - Based on the runtime support

Runtime Support

- The SERENITY Runtime Framework (SRF) is responsible for the dynamic provision of S&D Solutions to requesting applications
- It has to deal with the selection, activation, configuration and deactivation of solutions in specific context conditions.
- It controls most of the runtime aspects and plays a central role in system and solution evolution.

The SERENITY S&D model: SERENITY Runtime Framework



Main individual technologies/results involved

- **Analysis and development of S&D Solutions**
  - At different abstraction levels
- **Modelling S&D**
  - Not only solutions, but also requirements, properties, policies, context, …
- **Development time support**
  - Solution discovery, selection, adaptation and integration
- **Runtime support**
  - Solution selection and dynamic management
- **Runtime monitoring**
  - In open, distributed and uncontrolled scenarios
- **System evolution**
  - Based on the runtime support

Monitoring specification

- Monitoring rules are expressed in *EC-Assertion* and have the generic form

$$B \rightarrow H$$

- stating that when B is True, H must also be True. Both B (Body) and H (Head) are defined as conjunctions of Event Calculus predicates.
- The predicates used in monitoring rules express
  - the **occurrence** of an event (*Happens* predicate),
  - the **initiation** or **termination** of a **fluent** (i.e. condition) by the occurrence of an event (*Initiates* and *Terminates* predicates respectively),
  - or the **validity** of **fluent** (*HoldsAt* predicate)
- Predicates are associated with time variables

Monitoring specification

- Time constraints are indicated as time ranges e.g. R(t1,t1+1000)
- The monitoring rules in S&D Patterns must be designed to provide the information required in order to assess the correct functioning of the pattern and executable components that realise it.
- In a running system, the basic building blocks are the **Executable Components** (**ECs**), which are implementations of the S&D Patterns.
- ECs must include appropriate Event Capturers in order to inform their clients about their internal operation.
- All implementations of an S&D Pattern must include code to capture the events used in the monitoring rules of the pattern and to notify the events to the application through the SRF.

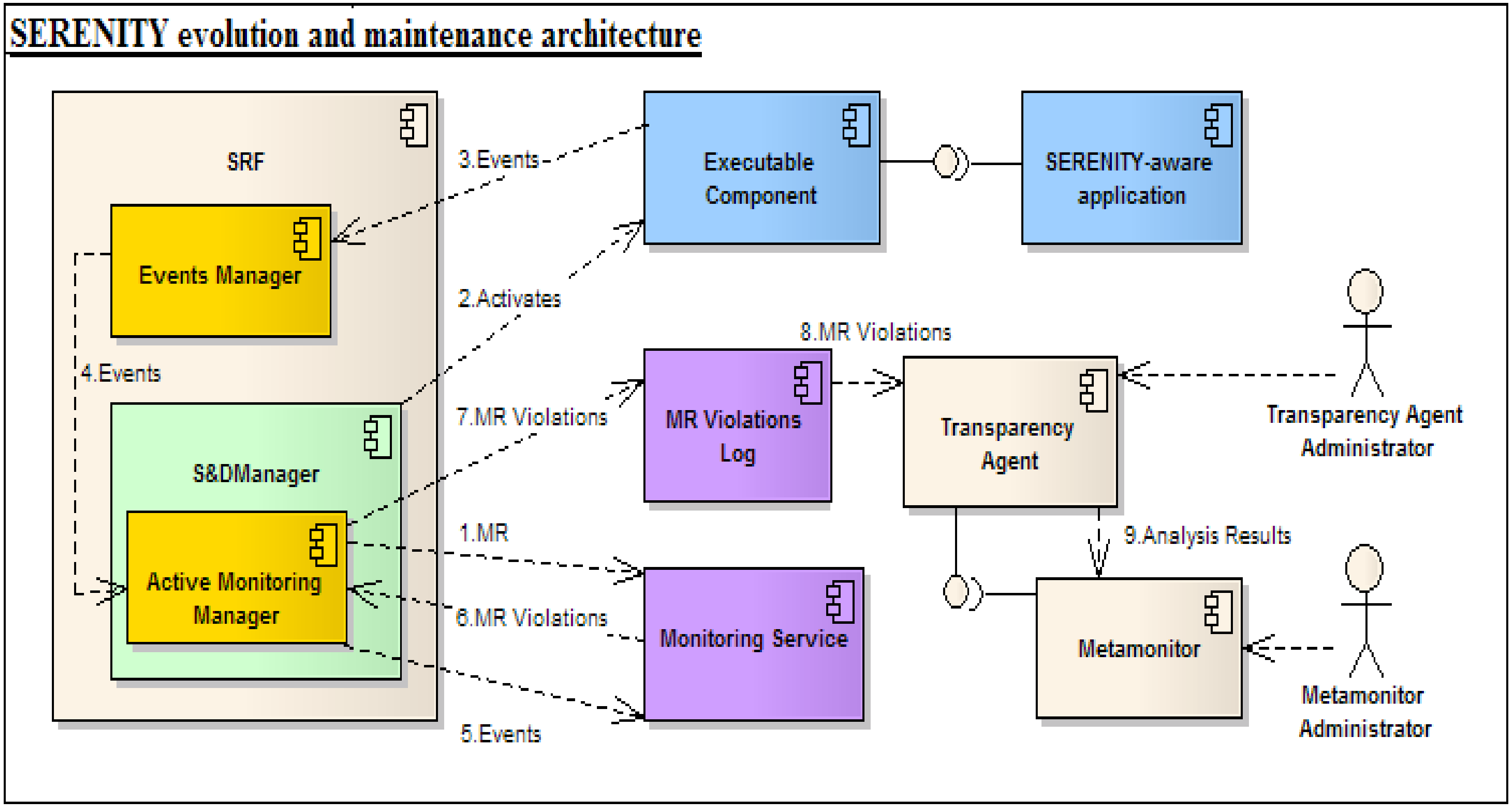
Monitoring operation

- **Monitoring**
  - When an event from an EC is received by the SRF it is managed by the core monitoring mechanism of the framework.
  - In this case, the event is forwarded to the appropriate monitoring service, which evaluates the state of the EC by applying the monitoring rules defined in the corresponding S&D Pattern.
  - If a violation of one of these rules is detected, this violation is reported to the SRF, which registers it and takes appropriate actions (such as deactivate the pattern, pause, reset, etc.)

Main individual technologies/results involved

- **Analysis and development of S&D Solutions**
  - At different abstraction levels
- **Modelling S&D**
  - Not only solutions, but also requirements, properties, policies, context, …
- **Development time support**
  - Solution discovery, selection, adaptation and integration
- **Runtime support**
  - Solution selection and dynamic management
- **Runtime monitoring**
  - In open, distributed and uncontrolled scenarios
- **System evolution**
  - Based on the runtime support

SERENITY maintenance and evolution infrastructure



Monitoring operation

- **Need for additional monitoring layers**
  - To deal with potential **problems caused by the interaction between different ECs**, a second monitoring mechanism is in charge of monitoring at the level of one particular SERENITY framework.
  - To **support maintenance and evolution** of specific S&D solutions and detect problems with non-compliant implementations, as well as problems in the modelling, solution-specific elements called Metamonitors perform vertical analysis.

Evolution Maintenance Elements

- **Transparency Agent**
  - Associated and deployed with a specific SRF
  - **Horizontal analysis**: Analyses data from different S&D Solutions in the same environment.
  - Collects information related to the violations of monitoring rules, analyses it and presents it to the TA administrator.
  - The results may be sent to the Metamonitor depending on certain rules, so the administrator of the Transparency Agent can choose what information to send to the Metamonitor and what to keep as confidential (Controlled Transparency).

Evolution Maintenance Elements

- **Metamonitor**
  - Deployed on an external infrastructure.
  - **Vertical analysis**: Analyses data from different machines about the same S&D Solution.
  - Receives information from several Transparency Agents and performs a new analysis on this.
  - The Metamonitor has a global view of what is the behaviour of the S&D Solutions in different contexts, and therefore is able to deduce proper conclusions that are not possible locally (modification of the description of an S&D Pattern, the deactivation of particular S&D solutions, etc.). This benefits both, the user of these solutions and the solution developer.

What SERENITY does NOT cover

- **Design / Creation of S&D Solutions**
  - Cryptography
  - Hardware
- **S&D Certification**
  - Not only solutions, but also systems, certifiable policies, …
  - Dependencies on context, composability, …
- **Application requirements discovery / refinement**
  - Refinement in application development process (application modelling, MDA, etc.)
- **Domain-specific models**
- **Plus Trust management, Risk, QOS, etc.**

SERENITY
System Engineering
for Security & Dependability

Contracts

Chapter VI – Things can improve (II)

Information Society
Technologies

FLACOS'09 - Toledo, Spain, 25-Sep-09

The role of contracts in a rigorous security engineering

- Informally speaking we could define a contract as an **agreement** between two or more **parties** that establishes **obligations** for these parties and **guarantees** about those obligations.
- More precisely, the *BusinessDictionary.com* states that a contract is a **“Voluntary, deliberate, and legally enforceable (binding) agreement between two or more competent parties.”**
- The main difference between the two is that we do not necessarily assume that a contract has to have any legal meaning (although we do not exclude that possibility).

Uses of contracts

- Contracts can be used in IT security for different purposes. Among these, we highlight the following:
- **Contracts as means for agreeing on security aspects.** In this case the contract establishes the terms (e.g. mechanisms, guarantees, referees and trusted third parties, etc.) that will be used in an interaction between two or more parties. A well-known example of this is constituted by SLAs (service-level agreements) that establish aspects related to the QoS (Quality of Service) like bandwidth, uptime, throughput, etc. Also in this category we find expression of the "terms of use".
- **Contracts as specifications.** A contract can be used to specify aspects of the operation of an entity. For instance, it can be used to specify the means by which the entity ensures the confidentiality of the data processed in an application in cloud computing or service-oriented computing. Another interesting case in this category is that of software contracts, which have been used in component-based development and especially for COTS (components-off-the-shelf). The same concept has recently been applied to the field of secure coding. In fact, some mature development strategies like PCC (proof carrying code) are closely related to this. In PCC, executable code comes with proofs that demonstrate adherence to a contract. These proofs can be verified by the runtime environment prior to code execution.

The role of contracts in a rigorous security engineering

- **Contracts as guarantees.** A contract can be used to state guarantees about the operation of an entity. For instance, it can be used to guarantee that an economic compensation will be available to the user should the confidentiality of the data processed in an application in cloud computing or service-oriented computing be broken.
- **Contracts as disclaimers.** The idea in this case is to make the user aware of the risks that the software introduce and to declare the limitations of the guarantees of a provider.

The role of contracts in a rigorous security engineering

- From the previous descriptions one can easily understand that the concept of contract constitutes a key element in providing precision, control, limitations and rigour into the security engineering discipline.
- Contracts reduce uncertainty and provide support for sound reasoning about dynamic, distributed and composed systems.
- In the SERENITY model,
  - contracts are mainly used to establish agreements between different SRFs,
  - but the contents of the descriptions made using the S&D Artefacts can also be considered as contracts.

SERENITY
System Engineering for Security & Dependability

UNIVERSIDAD DE MÁLAGA

The SERENITY experience.

Towards Information Security as an Engineering Discipline.

Chapter V – The road ahead

Information Society Technologies

FLACOS'09 - Toledo, Spain, 25-Sep-09

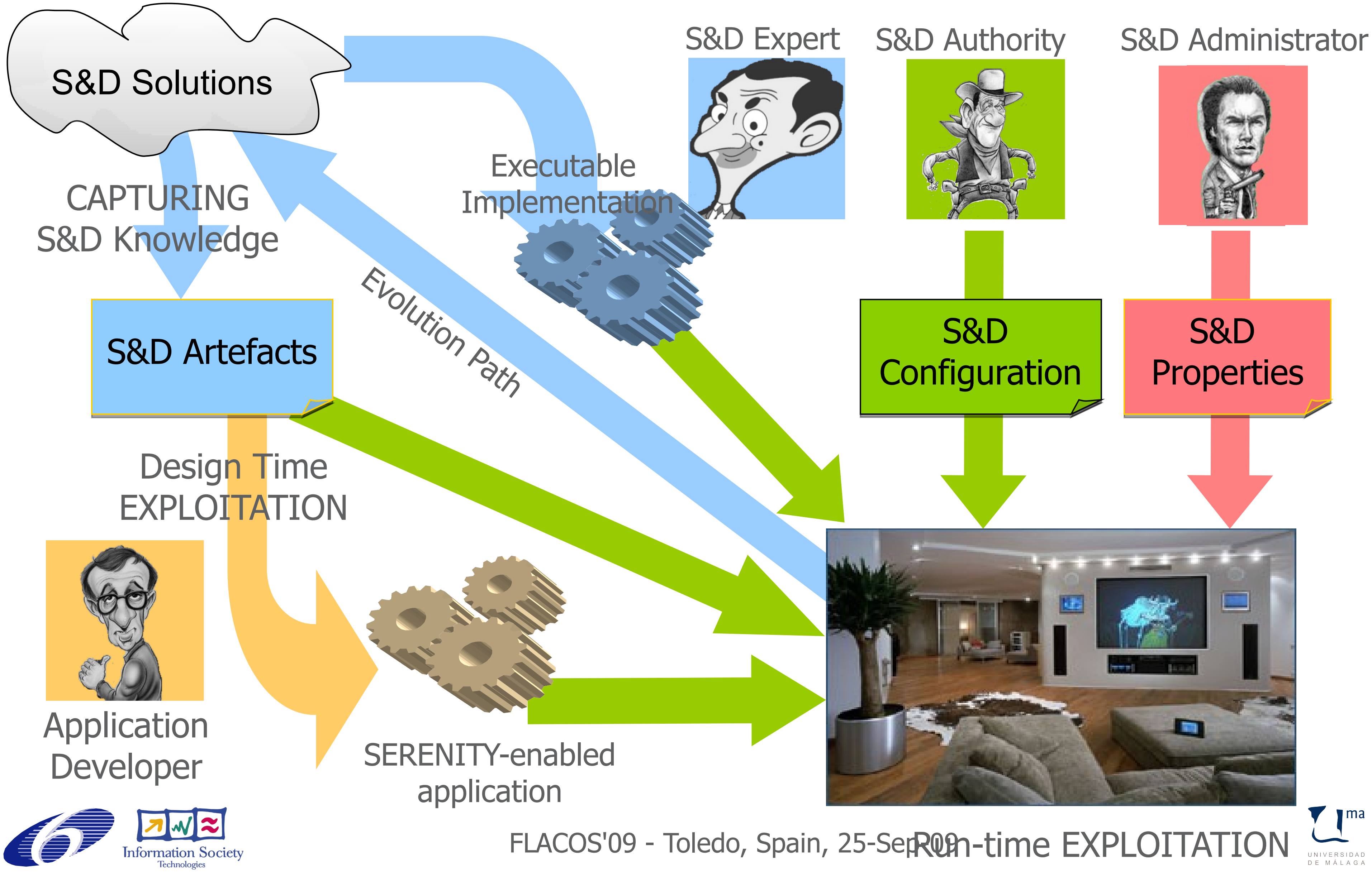
Remember Security Engineering?

- Set of (mostly unrelated) techniques for developing a secure system
  - Processes
  - Requirements engineering
  - Modelling secure systems
  - Code development and Language-based security
  - Verification and Validation
  - Certification and assurance
  - Risk management
  - …

What is the contribution of SERENITY to security engineering?

- MAIN POINTS
  - Separate roles with well-defined responsibilities and techniques
  - Capture precise, machine-understandable knowledge
  - Provide runtime support based on the previously captured knowledge and on context awareness
  - Relate all elements in the system lifecycle
- MAIN GLOBAL RESULT
  - SERENITY Lifecycle Model: A model of system lifecycle based on the abovementioned main points

SERENITY Lifecycle Model: Recall



SERENITY
System Engineering for Security & Dependability

Conclusions

Epilogue – Moving on

Information Society Technologies

Conclusions

- **We have shown that threat-based security engineering is not appropriate anymore due to:**
  - (i) is the origin of penetrate-and-patch situation;
  - (ii) does not result in specifications that can survive evolution; and
  - (iii) does not capture user requirements.
- **We have advocated a change of paradigm** based on the redesign and integration of security engineering mechanisms and tools, grounded in new principles such as
  - the precise and formal specification of security solutions and security properties; and
  - the use of these formally verified properties as the basis for the expression of requirements and the engineering of secure systems; and
  - the use of the concept of contract as a means to increase rigour and trust.

Conclusions

- **We have introduced two pillars to solve the situation**:
  - the SERENITY model of secure and dependable systems; and
  - the concept of contract and the role it plays in ensuring a rigorous treatment of security.
- **Our ultimate goal** is to establish **IT security as a fully fledged engineering discipline**, by means of the definition of integrated processes with well-defined goals and interfaces that combine the different techniques, methodologies and tools to support the engineering of future secure IT systems.

SERENITY
System Engineering
for Security & Dependability

Thank you

Information Society
Technologies