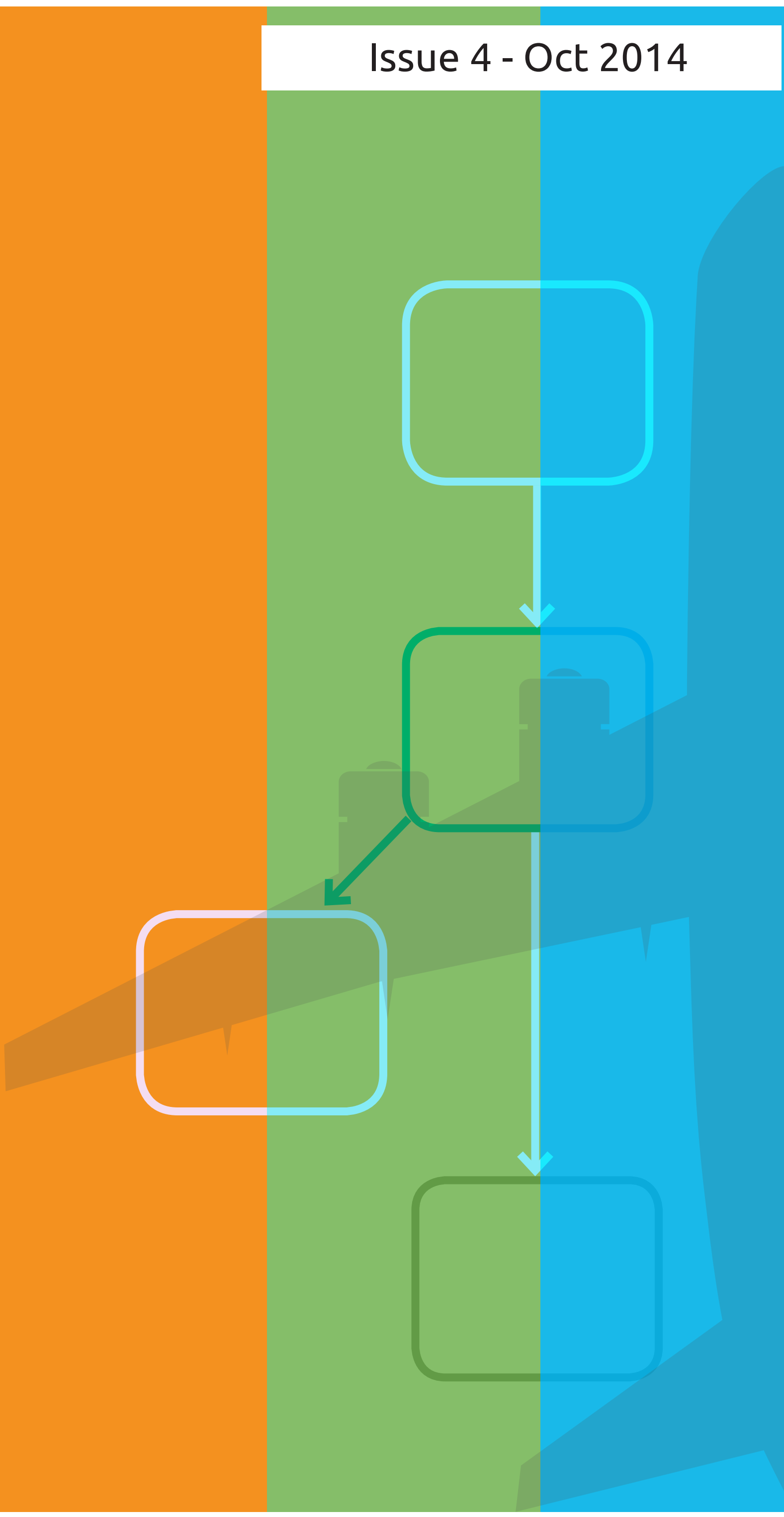
SECONOMICS

Socio-economics meets Security

SECONOMICS synthesizes sociological, economic and security science into a usable, concrete, actionable knowledge for policy makers and social planners responsible for citizen's security

Issue 4 - Oct 2014



SPECIAL ISSUE: SECONOMICS Tools

- SECONOMICS Architecture
- Used tools and Frameworks
- Architecture design
- SECONOMICS components
- Implementation of Airport Security

www.seconomicsproject.eu

SECONOMICS Architecture

The SECONOMICS project creates several models to analyze security aspects and events within its three case studies. Each of these aspects creates an analysis model. The tool helps by creating new instances of these models.

The SECONOMICS architecture defines a component-wise structure which mainly consists of five parts:

- The Analyses Plugin Provider:
  It automatically searches for available analyses and loads their configuration into memory. It provides this type of information to most of the components, like the selector and executor. It is also possible to load custom analyses from a different path later on from the user interface.

- The Selector:
  The Selector enables the user to select an Analysis by double-clicking on it and creating, in this way, a new instance of it.

- The Parameter Interface:
  This is an interface whose functionality is to request the necessary parameters from the user for the current analysis instance. It also displays the result and further information about the model in different tabs.

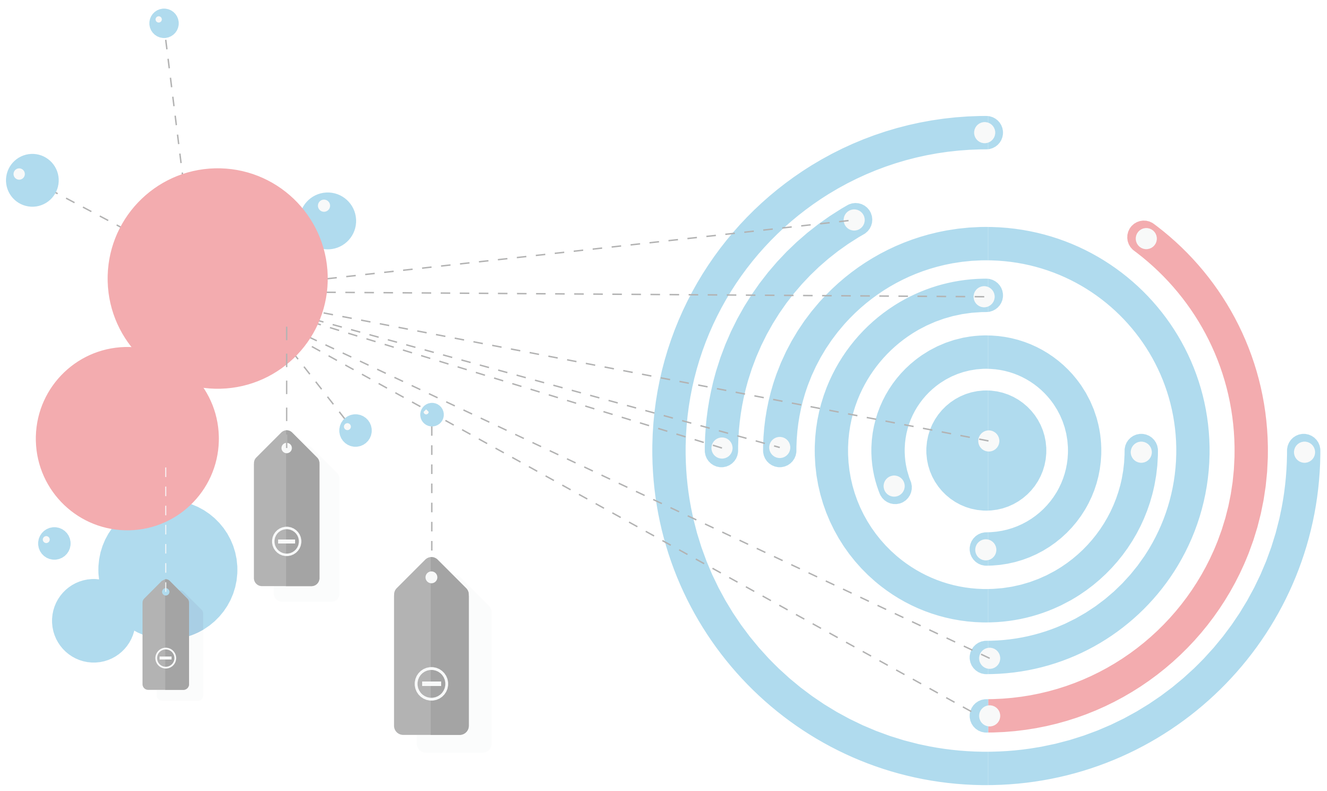
- The Matlab Execution:
  This component runs the Matlab analysis implementation, either compiled or for debugging purposes as a Matlab live session. This module is essential to run analyses using Matlab as the underlying math engine.

- Matlab Models and Analyses:
  The models and analyses provided by this project are provided as Matlab implementations.

The user typically starts the tool, in which all models are included. The user can create a new instance of these models, and then enter all the required parameters step by step and run the analyses. Each analysis implemented in Matlab is running in background with the aid of the Matlab runtime environment, returning the desired results upon completion. These results are displayed as tables, and nicely visualized with specified plots. At the end, a report can be printed.

Used Tools and Frameworks

The tool contains all models developed by the SECONOMICS project in order to give an integrated view and a good user handling. This framework is based on the Eclipse Framework which is released by the Eclipse Foundation. It defines several utility tool boxes which support the implementation process. For instance, it provides the plugin interface and also Graphiti, which is a tool box for visualizing structural diagrams. Eclipse and most of its tool boxes are open source and free of charge. Besides, we have chosen Matlab as the mathematical computation engine. It is a very sophisticated software on complex numerical calculations, comprising an optimized powerful engine. Although Matlab is proprietary, the implementations can be compiled and used license free within Java applications. All files are stored as open and standard XML files. Standard Java implementations are used to parse them. The results arising from Matlab calculations are visualized with the Matlab plotting feature and the Mozilla Webengine displays the outputs in a beautiful and user-friendly way.



Architecture design

The tool is divided into two separated parts: (1) the integrated tool framework that is implemented in Java; (2) its conforming models from the modeling work packages which are implemented in Matlab. The tool framework is separated into self-contained components which were implemented separately. Figure 1 shows the different components and their interaction.

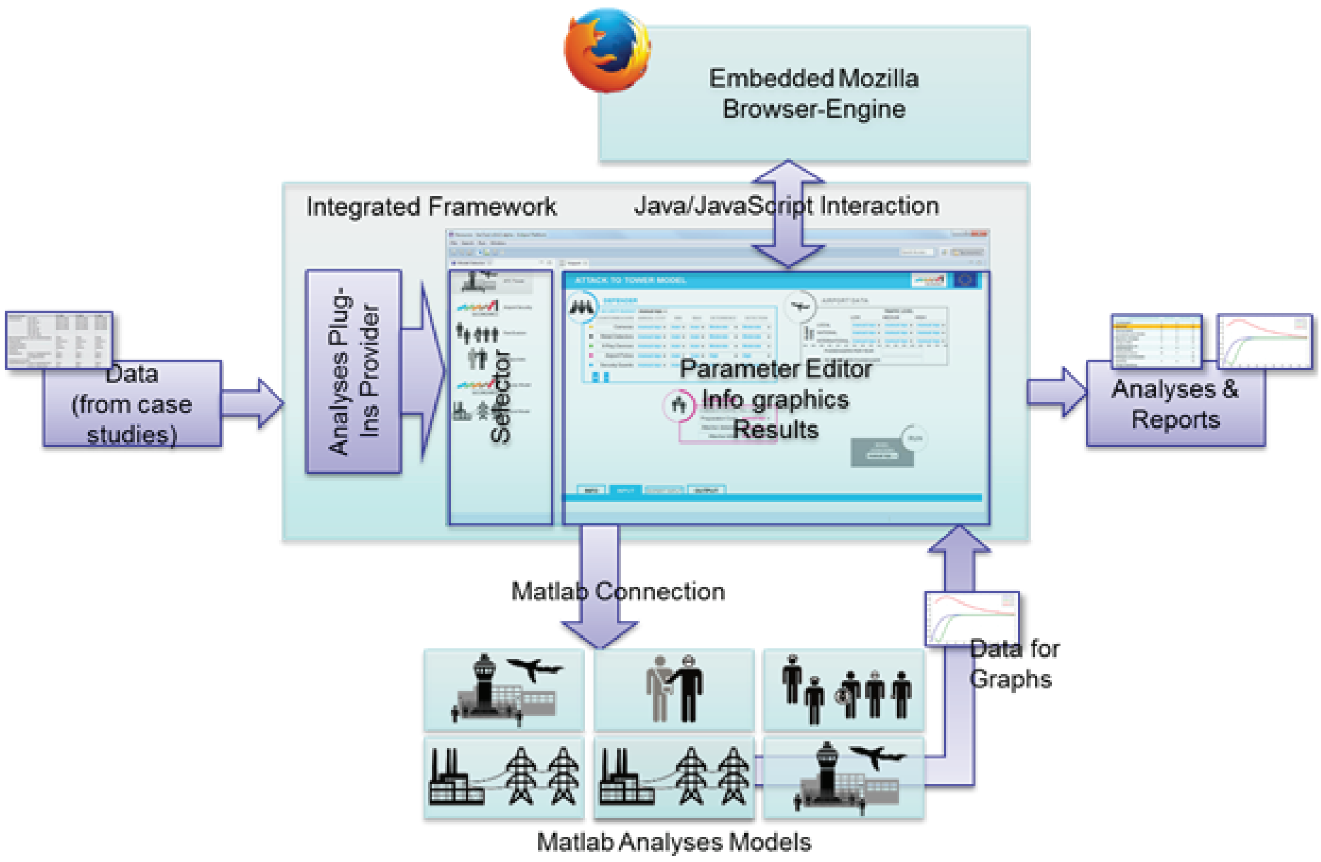


Figure 1 - Complete design of the integrated tool framework

SECONOMICS components

Analyses Plugins Provider

The Analyses Plugin Provider is one of the most important components of this tool. It is responsible for the background loading and provides the analyses that can be performed. It loads the Matlab analyses together with the associated configuration files, stores the information about the analyses and passes them to other components. It reads all data corresponding to a model out of the configuration file, the HTML form files and other associated files. The configuration file contains general information about the model as its name, the icon file name and a short description of the model. It also has information about the parameters and the parameter form. Each parameter has a name and a type, which can be, for example, numeric, character, etc. The file also contains the Matlab connection details as the model file name and the name of the function which should be called.

As the first step, the APP loads all Analysis Configuration Files in the target file system. Therefore, the file system is searched for our XML based configuration files. By the means of the JDOM package from the Apache open source project, the XML structure of the file is parsed to a Java object style. After that process, the data stored in the XML elements of the file has been converted into Java Strings or the numeric Integer values, and a list of Parameters is converted to the required format. With these previous requisites, we can initialize a newly created Analysis object, which can be provided to the internal list of analyses. Public access to this list for other classes of our tool, especially for the Analysis Selector, is provided.

Selector View

The Selector is the starting interface of our tool that the user starts with. It offers the possibility of selecting among different models, which correspond to the analyses that can be performed. The Selector UI is an Eclipse view window and is derived from the Java class ViewPart and is based on a Table Viewer of Eclipse's Standard Widget Toolkit, the Java SWT library. An analysis is visualized by a name and an image, displayed as a table. All the analyses displayed in the Selector are initialized by APP. The loading process is triggered automatically when the tool is started.

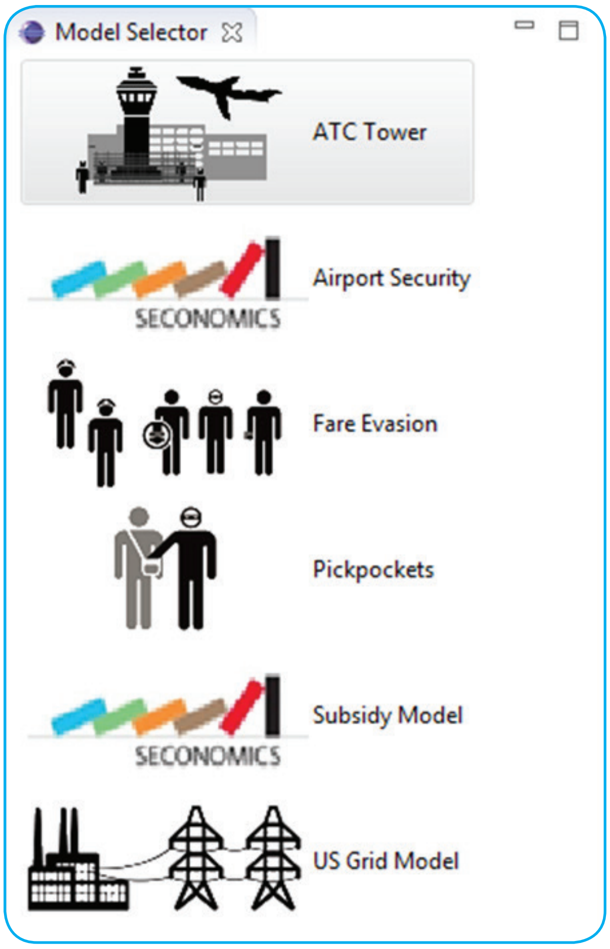


Figure 2 - Selector View enables the user to create a new instance of an analysis

At the initialization, the Selector triggers the Analysis Plugin Provider to import analyses from a certain file path. It gathers the information about the analyses from the Analysis Plugin Provider and presents a list of all models included in the tool. A double click on an analysis opens a new view part, where the parameters for an analysis can now be entered. The Selector is shown in Figure 2.

Parameter Interface

Another component is the Parameter Interface. This interface embeds a Mozilla Firefox render engine to display fully designed parameter interfaces for each model, called XUL-Runner. So these forms are created with HTML and the behavior logic of the form is done with JavaScript. The XUL-Runner provides the connection between both worlds, Java and JavaScript, so the form can interact with the tool and with the underlying Matlab implementations.

The default interaction between the parameter form and Matlab is defined in the model analysis configuration file and it is fully flexible. Several different types of parameters exist like, e.g., text strings, numbers and enumeration values.

Matlab Connection

In several cases, we have to perform highly demanding computational analyses. Therefore we make use of Matlab because of its power and excellent performance as a mathematical engine. As our tool is developed within a Java environment, we have to provide a connection between Java and Matlab. This is encapsulated in the Matlab Connection Component. It has been designed to support several types of parameters and several return values, including numbers, arrays of numbers for plots, and graphics among others. The connection details as, e.g., the name of the compiled Matlab file, the function name and the parameter association are obtained from the Analysis Plugin Loader and read out of the configuration file.

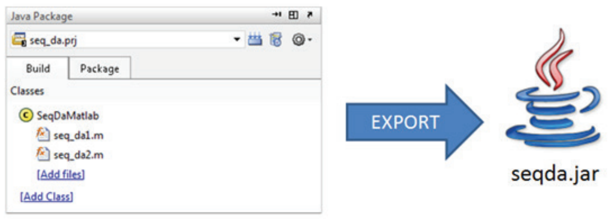


Figure 3 - With the Matlab Builder JA a Java library is created

This component was done with the Matlab Builder JA package. The Builder is shown in Figure 3. With this tool, functions implemented in Matlab code can be deployed as methods of Java classes into automatically generated Java libraries and be exported subsequently as JAR files. We make use of Java Reflection to include Matlab functions at runtime. The execution of a Matlab driven analysis is processed in the Matlab Compiler Runtime, so it is not necessary to own a full Matlab distribution. The Runtime can be downloaded freely from the MathWorks website. So any Matlab function that has been deployed to a Java package or as a standalone application by the Matlab Builder can be run on any machine that has a Matlab Compiler Runtime installed.

For the actual Matlab function call, we need to convert all input data for the analysis to one single Java Object Array. The function is called with that array as an input afterwards, so that an Object Array will be returned. This has to be converted back to ordinary Java data types. In order to get feasible results for the analysis with the policy model, it is necessary to perform a several number of runs in Matlab. The organization of this has to be done inside Matlab, as we want to avoid shifting results from Matlab to Java and back again several times with regard to possible impreciseness. So for the underlying policy model, there is a wrapping function in Matlab, and we only call this one to perform the execution. The whole scheme of how we integrated the Economic and Policy models into our tool is visualized in Figure 4.

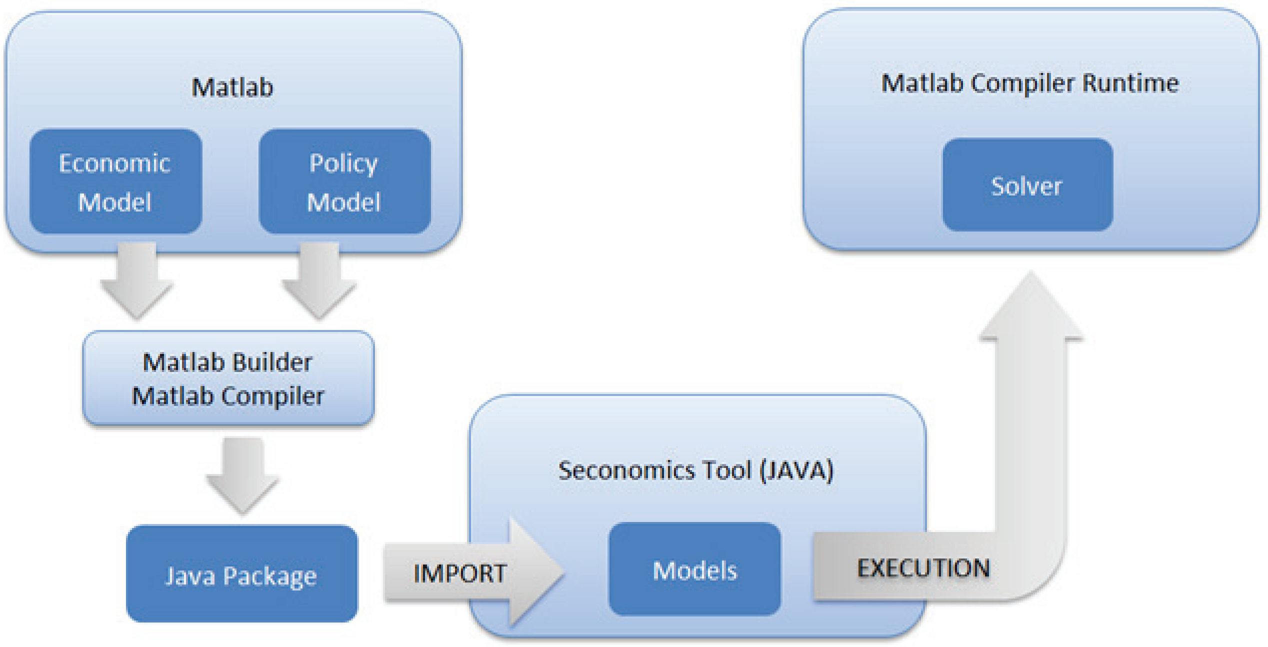
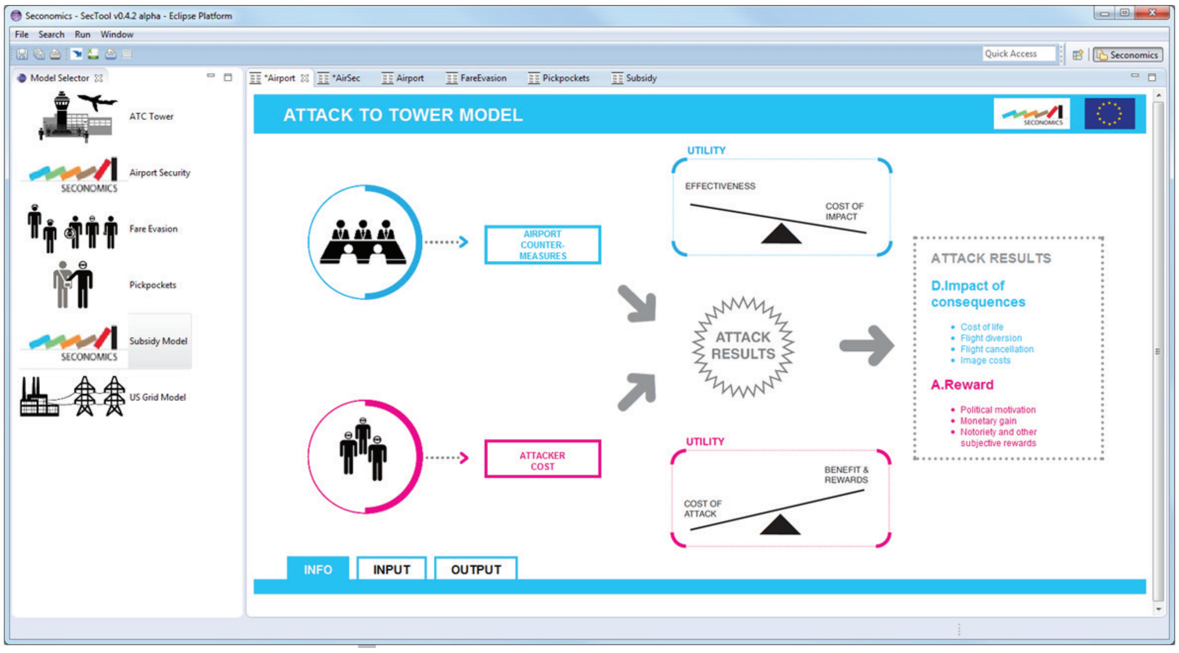


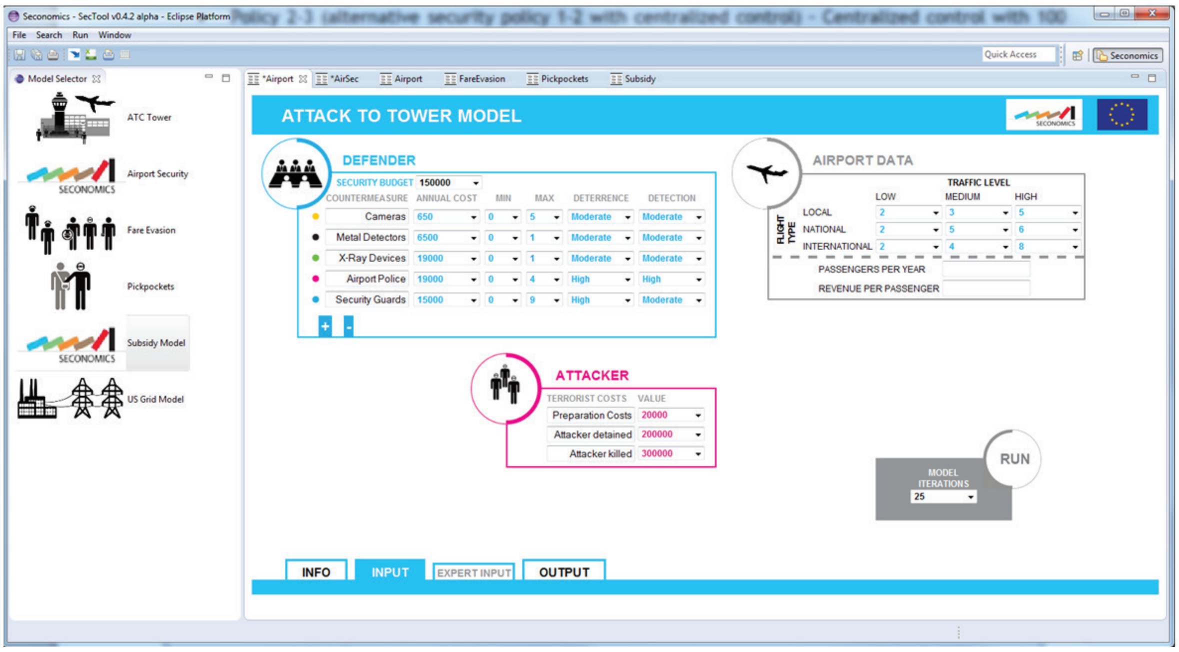
Figure 4 - Scheme of model integration

Implementation of Airport Security

Figure 5 shows the "Airport Attack to the Tower" model as an example of the use of the SECONOMICS tool. In the study of airport security a method that could determine the best alternative strategy for security instrument allocations was being investigated. In particular, we proposed models to be used to assess airport security directed at preventing terrorist attacks.

This study provided a case study for various alternatives for specific security proposals. More specifically, we performed an analysis of implementing current and newly proposed security policies, exploring issues of technological cost and performance. As a result, we are now able to answer the following questions:





DEFENDER

SECURITY BUDGET 150000

	COUNTERMEASURE	ANNUAL COST	MIN	MAX	DETERRENCE	DETECTION
●	Cameras	650	0	5	Moderate	Moderate
●	Metal Detectors	6500	0	1	Moderate	Moderate
●	X-Ray Devices	19000	0	1	Moderate	Moderate
●	Airport Police	19000	0	4	High	High
●	Security Guards	15000	0	9	High	Moderate

AIRPORT DATA

FLIGHT TYPE	TRAFFIC LEVEL LOW	MEDIUM	HIGH
LOCAL	2	3	5
NATIONAL	2	5	6
INTERNATIONAL	2	4	8

PASSENGERS PER YEAR

REVENUE PER PASSENGER

ATTACKER

TERRORIST COSTS	VALUE
Preparation Costs	20000
Attacker detained	200000
Attacker killed	300000

MODEL ITERATIONS 25

Figure 5 - TOP: Tool screenshot showing the info graphics of the model.
BOTTOM: Screenshot showing the parameter input form.

1. Given a current airport security policy, how does change the current security policy alter the cost and/or effectiveness of the airport security? For example, is employing full body scanner or implementing a new training program cost-effective?

2. What are the tradeoffs between alternative security policies?

In each airport, all passengers/items are subject to several checks. They need to pass through various check-points where they are inspected by security personnel and devices (e.g., X-ray, metal detectors and hand search). For passengers' bags, for example, inspection occurs by passing them through a fixed X-ray scanner. Inspectors examine the scanned image for finding any signs of threat. After the inspection, the security personnel/device determines whether the passenger/item is a threat or not. Each security personnel/device therefore produces an alarm or not. In some cases, security personnel perform additional screening which consists of a two-stage inspection process. For suspicious passengers/items, for instance, security staff conducts a hand-search which involves a time and resourcing consuming process.

While careful and prudent settings of security instruments might be able to improve the level of detection of threats, errors in detection of threats cannot be circumvented perfectly. The goal of a passenger/item screening policy is to reduce the risks from terrorist attacks; any newly proposed security policy should be compared with a current security policy. A cost-benefit approach has been widely used to compare and access current and alternative security policies for passenger/item screening.

To conduct a cost-benefit analysis, we need to identify costs and benefits from current and alternative policies. Both costs and benefits have direct and indirect aspects. The direct benefits of the policy are linked with avoiding the damage of a terrorist attack, both to the airport and to the society in general. These benefits come from the deterrent effect of screening policies and the potential to identify and remove a dangerous item before it is used. The indirect benefits of a policy are very difficult, if not impossible, to estimate; for instance X-ray scanning might allow inspectors to better identify unlawful materials such as drugs and smuggled goods. On the other hand, the direct costs commonly include the equipment and personnel related costs associated with a policy, while indirect costs being treated most importantly are the costs caused by delays and congestion. It should be noted that there is high uncertainty and difficulty in measuring parameters of the model for passenger screening. Nevertheless, cost-benefit analysis is a very helpful framework for providing a guideline of this decision context.

In order to conduct cost-benefit analysis, we need to investigate factors consisting of the basis of alternative security policies for screening. Before describing these alternative security policies, we present a simplified version of 'base security policy' for comparison.

- Base security policy for screening processes: The current policy mandates the scanning of 100 per cent of passengers/baggage via deployed screening machines. The costs for this security policy include amortized fixed equipment costs, annual operations and maintenance costs for the scanning equipment and salaries for the operators and inspectors.

To assess the viability of alternative policies, we consider the following scenarios.

- Alternative security policy 1-1 – 100 per cent inspection of passengers/baggage using current screening machines and X per cent additional inspection with a new security measure (e.g., 3D body scanner): This policy requires the scanning of 100 per cent of passenger/baggage with current technology. X per cent need to have additional inspection by a device with a new security technology. New devices and a team of operators and inspectors for these devices are required. In addition to the costs mentioned above, there are additional costs for purchase, operations, maintenance, inspection and delay.

- Alternative security policy 1-2 – 100 per cent inspection of passenger/baggage using current scanning machines with an additional training program: Due to improved training for inspectors, we assume that greater accuracy and a faster inspection rate is possible. While this case improves on 'current security policy' by reducing the false positive rate, it incurs the costs for the training program.

- Alternative security policy 1-3 – 100 per cent inspection of passengers/baggage using a current security technology and an additional inspection with a new security measure. An additional training program is implemented to operators and inspectors.

In addition, the following alternative security policies with centralized control will also be studied.

- Policy 2-1 (base security policy with centralized control) – 100 per cent inspection using current technology with a centralized control system: This policy requires the scanning of 100 per cent of arriving passenger/baggage, but with a remote monitoring system. While this policy is similar to the base security policy, there are some differences: inspectors are located in the monitoring center and the number of inspectors might be less than the number in 'base security policy'; monitoring devices in the center are required; additional infrastructure might needed; and errors in detection of threats and delay would be increased. Therefore, in the control center, the costs include fixed costs for monitoring devices, salaries for inspectors and fixed costs for infrastructure. At an airport, the costs consist of amortized fixed equipment costs, annual operations and maintenance costs for the scanning equipment and salaries for the operators.

- Policy 2-2 (alternative security policy 1-1 with centralized control) – 100 per cent inspection using current technology and 5 per cent additional inspection with new technology with a centralized control system: Similarly with policy 2-1, monitoring devices and inspectors for the current and new technology are located in the control center.

- Policy 2-3 (alternative security policy 1-2 with centralized control) – Centralized control with 100 per cent inspection using current technology with additional training program.

- Policy 2-4 (alternative security policy 1-3 with centralized control) – Centralized control with 100 per cent inspection of passengers/baggage using a current security technology and an additional inspection with a new security measure and an additional training program.

SECONOMICS

Socio-economics
meets Security

Contact Info
Project Coordinator: Fabio Massacci
Università degli Studi di Trento
fabio.massacci@unitn.it

www.seconomicsproject.eu

@seconomics_eu

This project has received funding from the European Union's Seventh Framework Programme for research, technological development and demonstration under grant agreement no 285223