



## City Research Online

### City, University of London Institutional Repository

---

**Citation:** Bloomfield, R. E., Chozos, N., Popov, P. T., Stankovic, V., Wright, D. & Howell-Morris, R. (2010). Preliminary Interdependency Analysis (PIA): Method and tool support (D/501/12102/2 v2.0). London: Adelard LLP and City University London.

This is the unspecified version of the paper.

This version of the publication may differ from the final published version.

---

**Permanent repository link:** <https://openaccess.city.ac.uk/id/eprint/3091/>

**Link to published version:**

**Copyright:** City Research Online aims to make research outputs of City, University of London available to a wider audience. Copyright and Moral Rights remain with the author(s) and/or copyright holders. URLs from City Research Online may be freely distributed and linked to.

**Reuse:** Copies of full items can be used for personal research or study, educational, or not-for-profit purposes without prior permission or charge. Provided that the authors, title and full bibliographic details are credited, a hyperlink and/or URL is given for the original metadata page and the content is not changed in any way.

---

---

---

City Research Online:

<http://openaccess.city.ac.uk/>

[publications@city.ac.uk](mailto:publications@city.ac.uk)

---

**Preliminary Interdependency  
Analysis (PIA): Method and tool  
support**

**© Adelard LLP and City University London, 2010**

Produced for Technology Strategy Board, under contract number BK030F





## **Summary**

This report is the second deliverable of the TSB-funded PIA:FARA project. The report presents a method called Preliminary Interdependency Analysis (PIA) which can be used to conduct interdependency analyses in complex systems such as Critical Infrastructures. The method is supported by a toolkit which can be used to conduct qualitative and quantitative analyses of interdependencies between complex systems.

## **Authors**

R Bloomfield  
N Chozos  
P Popov  
V Stankovic  
D Wright  
R Howell-Morris

**Document control**

Reference: D/501/12102/2  
Status: FINAL

VERSION	REVIEW NO./ISSUED	DATE
v0.1D	issued in draft	19 May, 2010
v0.1J	R/1890/12102/4	21 June, 2010
v1.0	issued	23 June, 2010
v1.1A	R/1969/12102/5	11 November, 2010
v2.0	issued	15 November, 2010

Approved: Robin Bloomfield  
Verified: Fan Ye

**Distribution**

Alan Bennett, Technology Strategy Board Monitoring Officer  
Paul Lewis, Technology Strategy Board  
Rhys Williams, Centre for the Protection of National Infrastructure

**Contents**

1 Introduction..... 7

2 Method: Preliminary Interdependency Analysis (PIA) ..... 8

    2.1 PIA model architecture: two levels of abstraction ..... 9

    2.2 PIA stages ..... 10

        2.2.1 Stage 1: Critical infrastructure / Service description ..... 13

        2.2.2 Stage 2: Service-model development ..... 16

        2.2.3 Stage 3: DBMS model development ..... 17

        2.2.4 Stage 4: Identification of dependencies between the services ..... 17

        2.2.5 Stage 5: Probabilistic parameterisation ..... 18

        2.2.6 Stage 6: (optional) Adding deterministic models of behaviour ..... 19

        2.2.7 Stage 7: Exploratory interdependency analysis ..... 20

3 Tool support: PIA Toolkit ..... 21

    3.1 PIA Toolkit Usage Patterns ..... 22

    3.2 Overview of PIA workflow ..... 22

    3.3 PIA Toolkit Execution Engine..... 25

    3.4 PIA Toolkit Architecture ..... 26

    3.5 Tool support for the service-level model ..... 29

        3.5.1 Interconnectivity matrix plug-in ..... 29

        3.5.2 Google Earth plug-in ..... 30

4 Conclusions ..... 30

5 Glossary ..... 32

6 Bibliography ..... 33

Appendix A An overview of the ASCE and Möbius tools ..... 34

    A.1 Assurance and Safety Case Environment (ASCE) ..... 34

    A.2 Möbius ..... 34

Appendix B Rome case-study – an example of applying the PIA method..... 36

Appendix C PIA Designer User Manual ..... 40

    C.1 Part 1 – Create PIA Project ASCE network..... 41

    C.2 Part 2 – Populate the State Transitions network view ..... 45

    C.3 Part 3 – Populate Physical Network (PN) views ..... 47

        C.3.1 Populate Intra PN views ..... 47

        C.3.2 Populate Inter PN view ..... 48

    C.4 Part 4 – Populate Stochastic Association (SA) network views..... 50

        C.4.1 Populate Intra SA views ..... 50

        C.4.2 Populate Inter SA view ..... 51

    C.5 Part 5 – Configuration of the Simulation Plug-ins..... 52

    C.6 Part 6 – Creation and configuration of the PIA simulation study ..... 55

## Figures

Figure 1: Overview of PIA method and toolkit.....	9
Figure 2: PIA method stages .....	12
Figure 3: Example views used in PIA .....	14
Figure 4: A Coupling point as a link between services in PIA .....	16
Figure 5: UML component diagram of the PIA toolkit.....	28
Figure 6: Interconnectivity matrix plug-in .....	29
Figure 7: Google Earth plugin.....	30
Figure 8: A state machine diagram for a local telephone exchange and the links status-fields editor plug-in .....	37
Figure 9: Part of the intra-service PN view of the Power Transmission service.....	37
Figure 10: A part of the SAN of the Telco SDH service.....	38
Figure 11: The “Node search” plug-in .....	38
Figure 12: The Network status-field editor plug-in.....	42
Figure 13: The HTML form used for the validation when creating a new PIA project.....	42
Figure 14: The Link status-field editor plug-in .....	45
Figure 15: The Generate State Space values plug-in.....	46
Figure 16: The status-fields for the node MVC1 of an Intra_PN ASCE network.....	47
Figure 17: The Inter-service links (coupling points) plug-in .....	49
Figure 18: The Physical Network views import plug-in .....	51
Figure 19: The plug-in used for configuration of the Möbius-based simulation study .....	53
Figure 20: The plug-in used for specifying the particular execution order of Deterministic plug-ins in a PIA project.....	53
Figure 21: The plug-in for mapping entities defined in a Deterministic plug-in to entities from the PIA project.....	54
Figure 22: The plug-in for generation of the Möbius-based simulation project .....	56

## Tables

Table 1: CI/service definitions for an information infrastructure .....	14
Table 2: PIA elements .....	15

## 1 Introduction

One of the greatest challenges in enhancing the protection of Critical Infrastructures (CIs) against accidents, natural disasters, and acts of terrorism is establishing and maintaining an understanding of the interdependencies between infrastructures and the dynamic nature of these interdependencies. Interdependency can be a source of “unforeseen” threat when failure in one infrastructure may cascade to other infrastructures, or it may be a source of resilience in times of crisis, e.g., by re-allocating resources from one infrastructure to another [1].

Understanding interdependencies is a challenge both for governments and for infrastructure owners/operators. Both, to a different extent, have an interest in services and tools that can enhance their risk assessment and management to mitigate large failures that may propagate across infrastructures. However, cost of investment in infrastructure modelling and interdependency analysis tools and methods, including the supporting technology, may reach millions of pounds, depending on the size of the system to be modelled, on the level of detail and on the mode of modelling (real-time or off-line). These factors will determine the software, hardware, data and personnel requirements.

It is therefore very important to understand what the scope and the overall requirements of an interdependency analysis service are going to be, before proceeding with such an investment. However, the decision on what modelling and visualisation capabilities are needed is far from simple. Detailed requirements may not be understood until some modelling and simulation has been conducted already, in order to identify critical dependencies and decide what level of fidelity is required to investigate them further.

This report presents an approach to interdependency analysis that attempts to address these challenges. The approach—*Preliminary Interdependency Analysis (PIA)*—starts off at a high-level of abstraction, supporting a cyclic, systematic thought process that can direct the analysis towards identifying lower-level dependencies between components of CIs. Dependencies can then be analysed with probabilistic models, which would allow one to conduct studies focussed on identifying different measures of interests, e.g. to establish the likelihood of cascade failure for a given set of assumptions, the weakest link in the modelled system, etc. If a high-fidelity analysis is required, PIA can assist in making an informed decision of what to model in more detail. The method is applicable as both i) a lightweight method and accessible to Small-to-Medium Enterprises (SMEs) in support of their business continuity planning (e.g., to model information infrastructure dependencies, or dependencies on external services such as postal services, couriers, and subcontractors); and ii) a heavyweight method of studying with an increasing level of detail the complex regional and nationwide CIs combining probabilistic and deterministic models of CIs.

This deliverable illustrates the use of PIA on a case study (Appendix B): a regional system of two CIs namely the power grid and telecommunication network around Rome, Italy (i.e. *Rome case-study*).

PIA is supported by a toolkit. The *PIA Toolkit* consists of two software applications:

- The *PIA Designer*, which allows a modeller to define a model of interdependent CIs and define the parameters needed for any quantitative study. For visual representation the tool uses a proprietary tool *ASCE* [4].

- The **Execution engine**, which allows for executing a model developed with the PIA Designer, i.e. a simulation study based on the model to be conducted and the measures of interest to be collected. The Execution engine uses *Möbius* [12], customised extensively with a bespoke proprietary development.

The current version of the toolkit allows for two main categories of models:

- Model of interdependent CIs at a fairly high level of abstraction (i.e. without detailed modelling of the networks used by the respective services). The model can be parameterised and then the simulation executable can be deployed on the Execution Engine.
- As above but adding any degree of detail that the modeller may consider necessary including high fidelity deterministic models available as 3<sup>rd</sup> party software modules.

This report presents the PIA method and offers a detailed description of how the PIA Toolkit can be used.

## 2 Method: Preliminary Interdependency Analysis (PIA)

Preliminary Interdependency Analysis (PIA) is an analysis activity that seeks to understand the range of possible interdependencies and provide a justified basis for further modelling and analysis. Given a collection of CIs, the objectives of PIA are to develop, through a continuous, cyclical process of refinement, an appropriate *service model* for the infrastructures, and to document assumptions about resources, environmental impact, threats and other factors.

PIA has several benefits. In particular, PIA can

- help one to discover and better understand dependencies which may be considered as “obvious” and as such are often overlooked (e.g. telecommunications need power)
- support the need for agile and time-efficient analyses (cannot always wait for the high fidelity simulation)
- be also used by Small-to-Medium Enterprises (SMEs) and not just infrastructure owners and government

PIA allows for the creation and refinement of interdependency models, in a focused manner, by revisiting earlier stages in the PIA process in the light of the outcomes of latter stages. For example, an initial application of PIA should result in a sufficiently concrete and clearly defined model of CIs (and their dependencies). However, following the first design iteration, an analysis of the model could cause us to question the assumptions made earlier on in the design process. As a consequence, the model may be revised and refined; as we shall see later on, revisiting previous phases of the development process is a key aspect of the PIA method and philosophy overall.

PIA consists of two parts:

- *Qualitative analysis*. The modelling exercise begins with a definition of the boundaries of the system to be studied and its components. Starting off at a high level, the analyst may go through a cyclical process of definitions, but may also be focused on a particular service, so the level of detail may vary between the different parts of the overall model. The identification of dependencies (service-based or geographical) will start at this point.
- *Quantitative analysis*. The models created during the qualitative PIA are now used to construct an *executable*, i.e. a simulator of the model behaviour in the presence of failures

of the modelled entities for the chosen model parameterisation. The model parameterisation may be based either on expert judgement or on analysis of incident data. Examples of such data analyses and fitting the available data to plausible probabilistic data models was presented in the recent WP1 deliverable [2].

The PIA Toolkit provides support for both the qualitative and quantitative analyses. Figure 1 illustrates an overview of the method and the toolkit.

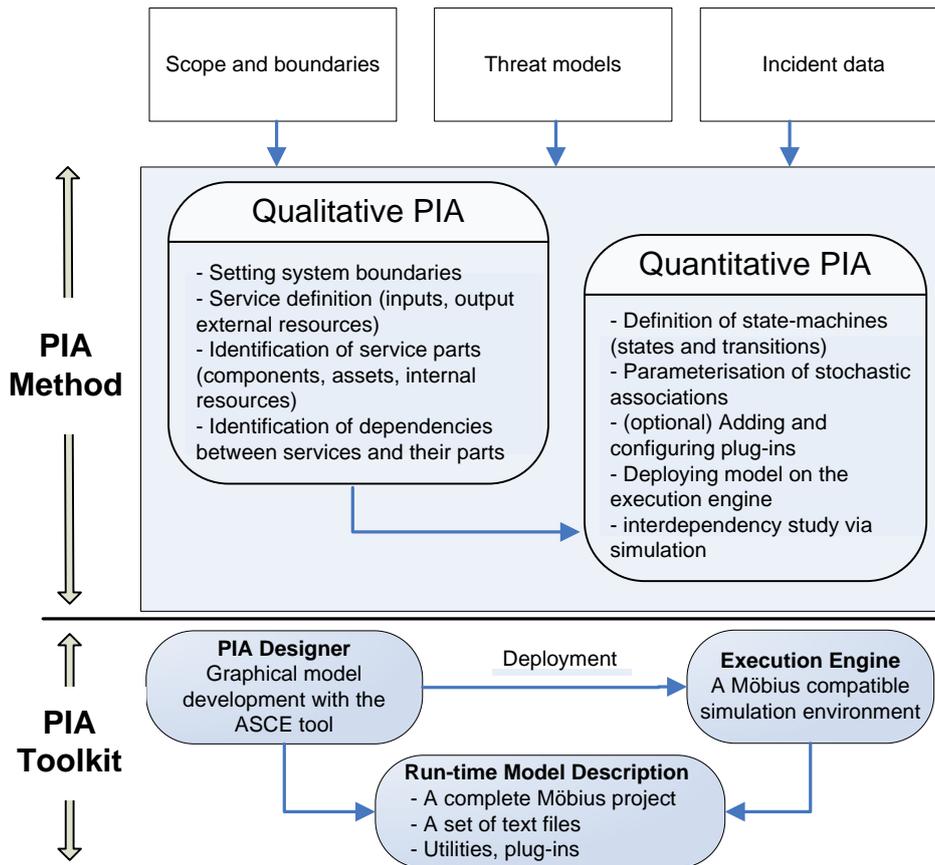


Figure 1: Overview of PIA method and toolkit

The interdependency models, of course, have to be related to a purpose and this should be captured in terms of a scenario and related requirements. The narrative aspect of the scenario is enormously important as it provides the basis for asking questions and discovering interdependencies as the starting point for more formal models.

Typically the systems of interdependent CIs of interest are complex: include many services which in turn consist of many parts. Given the complexity and size of the analysed systems tool support is essential. The aim of WP2 is therefore to produce a toolkit that supports PIA (including both qualitative and quantitative stages).

### 2.1 PIA model architecture: two levels of abstraction

PIA models broadly operate at two distinct levels of abstraction.

- *Model of interacting services (service-level model)*. The modelled CIs are represented by a set of interdependent services. Here, the view is purposefully abstract, so that we can reason about dependencies among the services (i.e. data centre X depends on power plant Y). Service-level dependencies are elicited by the defined lower-level dependencies among each service's constituent entities (physical components, resources etc.). These associations among components are referred to within PIA as *coupling points*. The coupling points *incoming* to a service can be associated with the resources that the service requires (e.g., a telecommunication service consumes "commodities" supplied by a power service). The resources consumed by a service can be obtained from the organisation's reserves (*internal resources*) or provided by another organisation (*external resources*). The *outgoing* coupling points instead define how the outputs from a service get consumed by other services (as either inputs or resources).
- *Detailed service behaviour model (DSBM)*. Implementation details are provided for an *individual service*, e.g. the networks upon which a particular service relies. For instance a Global System for Mobile (GSM) telecommunication operator typically relies on a network of devices deployed to cover a particular area (e.g. masts, etc.). Via DSBM we can choose the level of detail used to model these networks. In the example above DSBM may range from a connectivity graph – which cells of the network are connected with each other to a high fidelity model of the protocols used in the GSM network. We tend to think of DSBM as the networks owned (at least partially and/or maintained) by the respective service operator, i.e. an organisation. Although such a view is not necessary, it allows one to model several important aspects via DSBM. For instance the level of investment and the culture (strong emphasis on engineering vs. outsourcing the maintenance) within the organisation will affect how well the network is maintained (i.e. frequency of outages and speed of recovery). Thus, the process of recovery (a parameter used in DSBM) can be a useful proxy of the level of investment. Thus, through DSBM one can study scenarios which at first may seem outside the scope of PIA. An example of such a scenario would be comparing the deregulation with tight regulation in critical CIs.

## 2.2 PIA stages

PIA is carried out in seven stages (Figure 2):

- Stage 1.** *CI description and scenario context* (Section 2.2.1). A CI description provides a concrete context and concept of operation. This is the first level of scoping for the analysis task; the CI description gives the first indications of analysis boundaries. DSBM entities are identified and recorded.
- Stage 2.** *Model development* (Section 2.2.2). A model of the services (resources, inputs, outputs, system states) and the operational environment and system boundaries are developed, based on the CI description. Model boundary definitions are used at this stage to further restrict the scope of the analysis. Dependencies between the services are identified and the *coupling points* are defined: these refer on the one hand to the inputs and resources required by each of the services and on the other hand to the outputs that each of the services produces.
- Stage 3.** DSBM model development (Section 2.2.3). DSBMs are defined by selecting the right level of abstraction for the services: some of the services may be treated as black-boxes; in this case their representation in the DSBM will require no refinement in comparison with Stage 2. For those services, which are modelled in more detail one starts by defining explicitly their components and the assets including resort to using *existing models* of the underlying *physical* networks used

by the services or use other formalisms, e.g. such as PVS [9]. A level of consistency is achieved between the service model and DSBM: the coupling points appear in both Views.

- Stage 4.** *Initial dependency and interdependency identification* (Section 2.2.4). While some of the service dependencies have already been identified and recorded in Stage 2 (via input/output/resource identification), at this stage the modeller looks for additional sources of dependence (e.g. common components/assets), which may make several services vulnerable to common faults or threats. These can be derived by examining the service-level model, taking into account other contextual information (e.g. scenarios, threat models, attacker profile). The captured dependencies are modelled as *stochastic association* between the services or components thereof. Each stochastic association is seen as a relationship between a parent and a child: the state of the parent affects the modelled behaviour of the child.
- Stage 5.** *Probabilistic model development* (Section 2.2.5). Since we are dealing with risk, we take the view that, given the state space formed by the modelled entities (MEs), a stochastic process must be constructed upon it that captures the unpredictable nature of the states of the MEs, their changes and the interactions between CIs over time. In this stage probabilistic models of the MEs are defined. These are *state-machines*, a well known formalism in software engineering, modelled after the formalism used in the *Stochastic Activity Networks (SANs)*.
- Stage 6.** *(optional) Adding deterministic models of behaviour* (Section 2.2.6). At this stage the modeller may decide to extend the behaviour of the probabilistic model adding deterministic models of behaviour. Such a step may be useful when the modeller is seeking to extend the fidelity of the simulation beyond the standard mechanisms possible with a pure probabilistic model.
- Stage 7.** *Exploratory interdependency analysis* (Section 2.2.7). A Monte Carlo simulation [10] is used to quantify the impact of interdependencies on the behaviour of the system under study and draw more conclusions about the probability of interdependency-related risk.

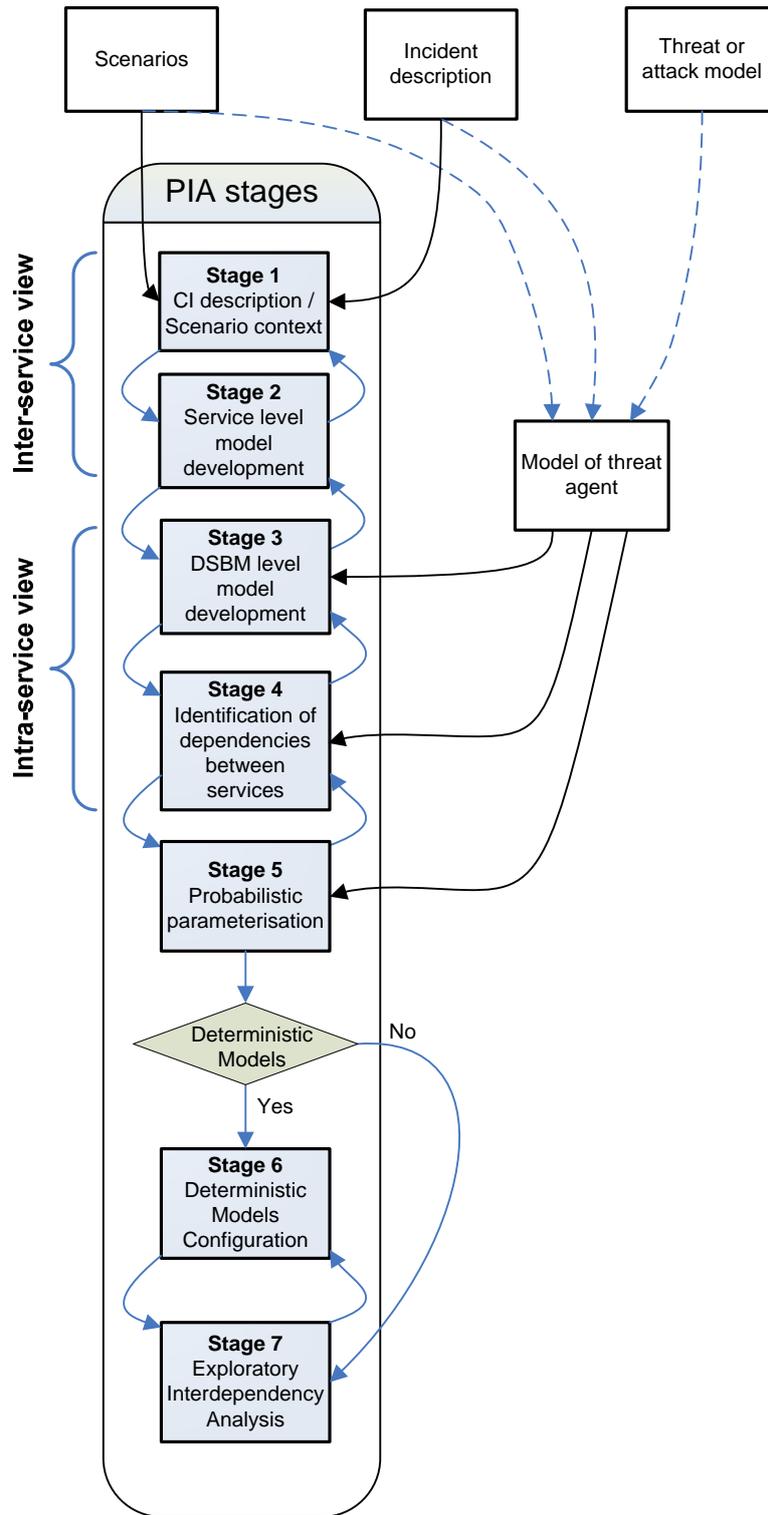


Figure 2: PIA method stages

During these stages we found that the narrative information coming from the following sources was relevant and useful:

- *Scenarios*: PIA is a scenario-driven approach. Once the system has been modelled, “what-if” questions will be used to explore vulnerabilities and failure cascade possibilities. Scenarios can be developed from a variety of assumptions or experiences. For instance, one can begin by asking a question as abstract as “what happens if there is a flood”, or “if power plant X fails”. Such questions form the basis for scenarios, which focus the analysis on particular conditions, exploring potential vulnerabilities.
- *Incident description*: PIA can be used to model an incident that has already occurred. This can be used as a baseline for generating and exploring variations of the same scenario or simply further exploring a system that has been compromised, or has failed, as the incident revealed unpredicted vulnerabilities and failures.
- *Threat or attack model*: Here, we are considering modelling assumptions based on malicious attacks.
- *Model of threat agent*: The above (scenarios, incident description, threat or attack model) are elements that will shape the profile of a threat that is modelled in our system. This can be a malicious agent (e.g. a terrorist) or a source of natural disaster (e.g. flood).

The seven stages are described in more detail in the following sections.

### 2.2.1 Stage 1: Critical infrastructure / Service description

#### 2.2.1.1 Definitions

A service provider (typically an organisation or a company) provides a *service*. Typically the service provider utilises a network, which in turn consists of *components* that use *resources* to provide an *output*. The relationship ‘whole-parts’ between a service and its parts (components, internal resources and assets) is explicitly modelled at this stage.

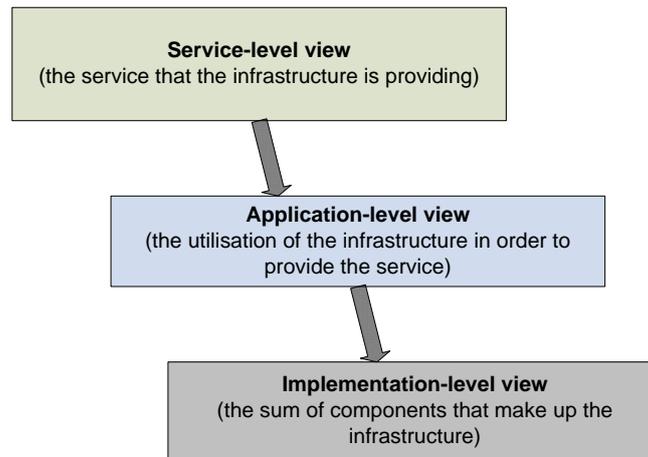
*Loss events* occur when the service is interrupted, either by a component failure or by exhaustion of resources. Measures of interest which will be studied are also identified and recorded at this stage.

The definition of the service and of its parts alone will be a useful process as it will help identify the boundaries of the system to be modelled, and the usually abstract initial understanding of some obvious dependencies will begin to become clearer. However, these definitions need to be coherent, as the subsequent modelling and analyses will be based on them. We would expect that the definitions would be developed by a team of experts, possibly from various levels within the service organisation; this is because during the development of definitions we consider both high-level views (e.g., production of energy) and low-level views (e.g., identification of specific physical components), and most importantly, how they are related.

Similarly to other modelling approaches (most notably UML<sup>1</sup>) PIA uses *different views* which allow the modeller to deal with complexity (i.e. separate concerns) and switch easily the focus of analysis from dealing with the whole to dealing with its parts and from modelling the entities of concrete critical infrastructures (with their concrete engineering meaning) or to the description of the probabilistic behaviour. The figure below gives an example of views which we found to be useful in practice (Figure 3).

---

<sup>1</sup> **Unified Modelling Language (UML)** is a standardised general-purpose language for modelling software engineering artefacts.



**Figure 3: Example views used in PIA**

Below we elaborate further on the use of the views listed in Figure 3, based on the work on an ongoing case study which considers the information infrastructure of an SME.

View	Description
Services	<ul style="list-style-type: none"><li>● Production and delivery of reports</li><li>● Help-desk</li><li>● Licensing</li><li>● Invoicing</li></ul>
Application	<ul style="list-style-type: none"><li>● Access to data and information (read-only)</li><li>● Modification of data and information (create, modify, delete, save)</li><li>● Communication (face-to-face, telephone, email etc)</li></ul>
Implementation	<ul style="list-style-type: none"><li>● Hardware</li><li>● Software</li><li>● Data</li><li>● People</li></ul>

**Table 1: CI/service definitions for an information infrastructure**

What is presented in this table is the first layer of definitions for the three views. Following several iterations this leads to a rather detailed list of individual components, such as servers, hubs, databases and people. As mentioned previously, this process will require the involvement of people from various levels of the service organisation, so that the link between the service-level view and the implementation-level view can be achieved.

Section 2.2.1.2 describes a typical set of types of elements that are to be identified and defined during a PIA. These may be different depending on the project.

### 2.2.1.2 PIA elements

Component	Definition
<b>Service organisation</b>	<p>A service with certain characteristics is provided by a service organisation. The service is essentially a label for the process of transforming the resources into a saleable commodity.</p> <p>This is the highest level of abstraction for defining physical components. Service organisations may be power plants, data centres, airports, etc.</p>
<b>Component</b>	<p>A component is a commodity, part of the service, which is used in the transformation of resources into a product. Components may fail in operation, either as a result of wear and tear (physical hardware elements) or design faults (software, hardware, procedures), in which case the service output may be affected. Whether the component failure will affect the service output depends on the service's internal resilience (its ability to withstand component failures, e.g. as a result of fault-tolerant design).</p> <p>An important aspect to consider when dealing with components is that they may be geographically dispersed, forming a distributed network of components. In addition, in many cases, considering people as components in the model is a sensible strategy.</p>
<b>Asset</b>	<p>An information asset (e.g. technical know-how, data, procedures, algorithms) is used either directly or indirectly to produce a product. The information assets may be stolen, misused or corrupted as a result of accidental failure or malicious behaviour, in which case the supplying organisation may be adversely affected. Assets can be electronic or paper-based records and files, or softer aspects such as trust and reputation.</p>
<b>Resource</b>	<p>Resources are being consumed. They are supplied by internal reserves or external services. External resources are commodities which are normally consumed in the process which leads to the supply of a product at the service output. This consumption is important as it can be a source of hidden dependencies. Electrical power, air conditioning as well as consumables would fall under this category.</p>
<b>Input</b>	<p>The input to a service is the demand for a given product. Demands can come from other services, the public, legislation, governmental directives, etc. The demand is seen here as a request for a service based on an agreement of some sort (such as a contract between the supplying organisation and the consumer).</p>
<b>Output</b>	<p>An output is a product of a modelled component.</p>
<b>Environment</b>	<p>In most cases, under "environment" we would expect to see aspects of weather and other natural phenomena (e.g., earthquakes).</p>

**Table 2: PIA elements**

Producing a set of coherent definitions may require several iterations. We recommend that these are recorded in a systematic, clear manner, as these definitions will determine the outcome of the rest of the modelling and analysis.

### 2.2.1.3 Service state

The state space of a modelled service is defined either *explicitly* or *implicitly*. The explicit definition of the service space is applied when the service is modelled as a black-box, i.e. no

further refinements detailing its parts are used. In this case the modeller would associate the service with a state machine in which the possible service states are spelled out.

We found that in the case of explicit definition of a service it might be useful to consider a minimalistic state space {OK, impaired, failed} so that the modeller can distinguish between the possible degrees of operability (from fully operational (OK), to totally non-operational {failed}, impaired denoting partial operability).

In the case a DSBM is associated with a service, the service state is implicitly defined by the state spaces of its parts: components, internal resources and assets. In this case the modeller is expected to define the state of the parts, but no explicit definition of state machine associated with the entire service is required; the service state space is the Cartesian product of the state spaces of its parts.

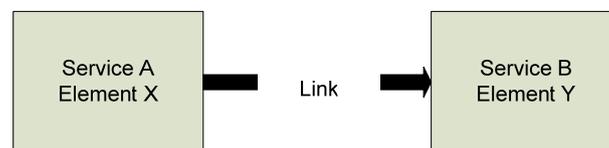
#### 2.2.1.4 Scope and boundaries

Building a model of interdependent CIs typically requires multiple iterations of refinement: starting with the definition of the services and how they are interdependent (input, output, resources) at a higher level of abstraction and then gradually progressing by adding details, e.g. DSBMs for the services judged to require a more detailed description. DSBMs themselves can be refined multiple times – possibly driven by the results obtained from the previous iterations of refinement.

Our practical experience with PIA has been with geographically compact studies – a critical information infrastructure of an SME and a regional study of two interdependent CIs. The particular types of modelled entities are dictated by the context of the study. These may include a set of hardware components, e.g. the ones used in the telecommunication and in the power grid CIs in the case of the regional system case that we started studying in IRRIS [6] and the entities of the information infrastructure of an SME in an ongoing case study.

#### 2.2.2 Stage 2: Service-model development

In this phase, the dependence *between the services* modelled are identified: input/external resource – output relationships, the components (e.g. of the same type such as a PC running the same OS and application software) which may be subject to common mode/cause failure, or stochastic associations are identified and marked as *coupling points*(see Figure 4) between the services.



**Figure 4: A Coupling point as a link between services in PIA**

This stage is primarily concerned with the development of the inter-service models, i.e. dependencies between the services. Some initial steps in defining the intra-service models, however, may also be undertaken in this stage such as identifying the components, the internal resources and assets of some services.

### 2.2.3 Stage 3: DBMS model development

This stage is focussed on developing the DSBM models (i.e. detailed inter-service models) for those services chosen to be modelled in detail. The parts that make up the service (components, assets, internal resources) defined in Stage 2 are now scrutinised and their associations (deterministic and stochastic) are identified and represented in the DSBM.

At this stage we define the state-machines used to model the behaviour of the modelled entities. The services for which no DSBM is defined will be associated with a state-machine which will define the states and the state changes of the entire service.

In case a DSBM is associated with a service, the modeller is expected to define a separate state-machine for each of the service's parts. The state of a service in this case is represented by the Cartesian product of the states of its parts.

### 2.2.4 Stage 4: Identification of dependencies between the services

At this stage the analysts proceeds by asking 'what-if' questions and exploring particular threat models and scenarios which may cross the service boundaries.

Apart from the obvious dependencies (e.g., telecoms need power), which may be observed by considering the functional association between entities, there are some other aspects of CIs that need to be taken into account. The following are some key concepts that should be considered in threat models and scenarios:

- *Geographical dependencies.* Vulnerabilities may lie not only in functional dependencies between components, but also in the risk of, for example, an explosion occurring in a nearby site. Recording the geographical information for each element is done in the stage of Service Description (Section 2.2.1.3). At design time we can identify and record, e.g. using a special link type "near to". However, such an approach will be of limited value. It will identify some and will miss many interdependencies due to geographical proximity. Our experience indicates that a systematic study of the impact that geographical proximity will have on interdependencies between the services will require a combination of static analysis and (simulated) stochastic modelling whereby the anticipated disruptions (earthquakes, flooding, sabotage, etc.) are instantiated at randomly chosen location and with a random severity (e.g. magnitude of the earthquake). Implementing such an approach via static analysis may be less effective than using a simulation with a randomly generated location of the disruption and a randomly generated area affected by the disruption.
- *Competition for resources.* During a crisis, dependencies may become apparent between entities that share the same resource(s). When more than one element reallocate their resources (e.g. maintenance personnel), there may be competition between them. This may lead to starvation of the particular resource, or the accumulation of dependencies to one element, which will then be bearing the risk of causing multiple disruptions, should that fail too.
- *Resilience perspective.* The failure of one element may have an indirect effect on another during a crisis if it compromises a resource, a component or an asset that are critical to its recovery. This is the kind of dependency that may not be visible during normal operation.
- *Common mode/cause models.* The modeller should also look for additional sources of dependence such as common vulnerability between the components of services. An

example of common vulnerability is the use of several PCs of exactly the same type (the same hardware specs, OS and set of applications). In this case one is justified in making a modelling assumption that an *actual* failure or compromise of a component of the particular type in one of the services implies that the other components of the same type in either the same service or in other services become *more likely* to fail or be compromised than before the failure of the first component occurred<sup>2</sup>.

The captured dependencies get modelled probabilistically in the form of *stochastic associations*, which can occur between the services or components thereof depending on the level of abstraction used. Each stochastic association is seen as a relationship between a *parent* and a *child*: the state of the parent via the stochastic association affects the modelled stochastic behaviour of the child. The stochastic association is characterised by its *strength*. For example, using a stochastic association between parent A and child B, the modeller can define that the rate of failure of B will increase 10-fold (in this case we say that the strength of the association is set equal to 10) when A is in a failed state in comparison with the rate of failure B when A is working correctly.

When considering modes of behaviour (i.e., how a failed or impaired element impacts the rest of the model), the temporal aspect needs to be taken into account. Dependencies, especially when considering competition for resources or recovery, may arise over time. Vulnerabilities may start becoming obvious as resources or the capacity of a system reach their limits. Modelling these temporal aspects is important.

Incident history should be taken into account when asking what-if questions, as previous incidents can help unveil similar and/or related scenarios.

### 2.2.5 Stage 5: Probabilistic parameterisation

Our method of quantifying interdependencies is based on a combination of techniques outlined in greater detail in [6]. A key problem is to define mechanisms of association between distinct MEs, whereby the model is given a structure that is realistically different from the rather limited concept of a number of state-holding MEs, embedded in a common simulated time line, behaving independently, as parallel stochastic functions of simulated time. Rather, we require a definition of a multivariate stochastic process of states of the MEs, representing the MEs' interactions with one another: interaction which may be deterministic (discussed in stage 7), or which may be probabilistic in the sense that the states of MEs, and perhaps also transient environmental stresses and perturbations, will influence the risks to which other MEs are subject in precisely defined ways. Every event that an ME may manifest, will take the form of a transition between two states of its assigned state space. As a result, the stochastic process to be modelled is of significant complexity, which makes *Monte Carlo simulation* [10] the preferred model analysis technique. We have built a generic tool, based around the simulation solver of the Möbius SAN tool [3][12] (see Appendix A for an overview of the tool), augmented by additional code of our own design, to conduct such simulations. These take the form of continuous time, discrete event driven simulations of CI behaviour and interaction, represented as sequences of changing states of MEs.

We use the notion of a *type of modelling entity* (TME). A separate state-machine is defined for every TME. The modelled system, however, may include *multiple instances* of the same TME,

---

<sup>2</sup> We note in passing that elsewhere we provided an extensive justification of the plausibility of this assumption. It is common knowledge in safety-critical area to consider common-mode/cause failures, which informed the concept of stochastic association deployed in PIA.

which will share the same state-machine (number of states and transitions between the states). The probabilistic description of the instances, however, may be unique – the SAN model associated with the instances of TME may be different (i.e. the probability distributions associated with the transitions of the state-machines) for the different instances.

At this stage the modeller provides probabilistic parameters related to the transition parameters of the state-machines which model the behaviour of the modelled entities. The modeller can specify a unique set of parameters even if two modelling entities are of the same type (i.e. share the same TME). The following set of parameters must be provided:

- the model of transitions (*competing risks* is the default for SAN) and unless there are good reasons to use a different model we would use the default
- the type of distributions which characterise the transitions from the current state to any possible next state (exponential, Weibull, or other probability distributions)
- the parameters of the specified distribution (e.g. the rate parameter of an exponential distribution).

Also parameterised at this stage are the stochastic associations between the MEs, i.e. the associations' strengths. In the current implementation of the toolkit we assume that the effects of multiple parents of the same child are *independent* of each other. The design of the toolkit, however, allows for implementing non-independent stochastic associations.

Once the parameterisation is completed, the modeller can study the effect of systematic parameter variation on the behaviour of the model as a whole. For each assignment of parameter values, an experiment is undertaken. This allows the effect of the parameter on the variability of defined CIs properties, e.g. such as the frequency of large incident cascades or outages of service, to be obtained.

#### *2.2.6 Stage 6: (optional) Adding deterministic models of behaviour*

At this stage the modeller may decide to extend the model description by adding deterministic models which increase the fidelity of simulation and will be difficult to represent using the stochastic model only. For instance, the use of such models will allow for propagating in detail the consequences of failures or repairs of the modelled entities. Good examples of deterministic models are the flow models (e.g. DC/AC power flow models, various telecommunication traffic models, etc.).

There are many ways of adding deterministic models to a Monte Carlo simulator. In the past we used tight coupling between the probabilistic and the deterministic models [6] – the code implementing the different models was linked in the same simulation application. In PIA:FARA we designed a pluggable architecture of the executables (i.e. simulators which can be deployed on the Execution Engine). The executable only contains the functionality of the probabilistic model, but offers API for adding at run-time any number of pre-existing pluggable modules (available as shared loadable libraries). A deterministic model, developed to be compliant with the pluggable API can be added to the list of modules to be loaded and used by the executable at run time. Details on the architecture and a more detailed description of how the modeller can configure an executable to use a set of plug-ins are given further in this document (see Sections 3.3 and 3.4).

A point worth mentioning here is that although the pluggable architecture was designed with deterministic plug-ins in mind, it can be used to implement any functionality, including probabilistic extensions, which are difficult to define statically. For instance, the model of environmental disturbances (earthquakes, flooding, etc.) discussed in Section 2.2.4 may be modelled using plug-ins which would determine at random the location and the magnitude of the event, thus making it possible to model the disturbance with a simple state-machine with states (active, inactive).

### 2.2.7 Stage 7: *Exploratory interdependency analysis*

In order to carry out the exploratory interdependency analysis, Monte Carlo simulations [10] are used. For each experiment, a number of replications of the evolution of the model, using identical parameters, over a defined interval of simulated time, may be conducted. One replication of an experiment differs from another solely in that the random number generator is seeded at a different starting point. Then statistical sampling theory may be used to study the distribution properties of system measures of particular interest for each set of assigned parameter values.

The focus of the studies is defined in the so called *reward variables*, the values of which are computed at simulation time over the states of the SAN model implemented by a Monte Carlo simulator. The same model can be executed with a number of rewards. There are two groups of simulation activities:

- Calculating statistics on the defined rewards based on multiple repetitions of a simulation, e.g. comparison of distributions of important CI dependability summary statistics between experiments with different parameterisations and/or level of abstraction. Typical examples here are:
  - Sensitivity analysis with respect to a particular model parameter (e.g. how the variation of the strength of stochastic association affects the results)
  - Distribution of *cascades* (i.e. outages that include more than one modelling entity)
- Using the simulation traces obtained from the history of a *single experiment* over a long interval of simulated time may be searched for interesting features. Examples of analyses of this kind include:
  - Searching for cascade failures and scrutinise these to understand better the “mechanisms” of cascades or use these in training.
  - Visualisation of the trace (e.g. with the Google Earth application [8]) so that one can “get a feel” about how the behaviour/operability of the modelled system might vary over time.

### 3 Tool support: PIA Toolkit

The qualitative part of the method described in this document could be applied without the use of tools. However, such an activity would be both more resource intensive and failure prone, since e.g., the application of the method will need to be performed specifically for each new PIA study without a possibility of reuse. In any case, the quantitative part does require the use of tools.

In the past we relied on two separate tools: ASCE for the qualitative part of the analysis and Möbius SAN for the quantitative part (a short description of each tool is given in Appendix A). The main objective for WP2 in the PIA:FARA project was to develop a toolkit which would allow a modeller to progress seamlessly from qualitative to quantitative interdependency analysis. More specifically, in WP2 we set out to achieve the following:

- Development of models of interdependency between CIs in a graphical environment. This is achieved with the *PIA Designer* tool which:
  1. allows the modeller to switch between *different views* dealing with the different stages of modelling as described in Section 2. This includes being able to easily change the modelling assumptions and the model parameters used for the quantitative analysis.
  2. offers support of the pluggable architecture of the run-time engine (i.e., execution engine, see the following paragraph) whereby an executable (to conduct quantitative interdependency analysis by simulation) would be configured to use the simulation plug-ins needed in a study before deployment of the simulation executable on the run-time engine.
- Performance of probabilistic analyses based on the graphical models developed with the PIA Designer. This is achieved with the software tool referred to as *Execution Engine*, which has an architecture that separates the quantitative analysis tasks common to all PIA studies (such as SAN models of the modelled entities and their stochastic associations) from the specifics of the particular study (e.g., optional add-ons needed only in some PIA studies, specific parameterisations of the stochastic associations, etc.). This objective was achieved by adopting a pluggable architecture in which software modules compliant with the architecture can be easily added to a study by configuring the simulation study.

Below are some of the ways in which the PIA Toolkit can enhance the application of the PIA approach:

- *Visualisation*. PIA models are enhanced with graphical representation. Visualisation can assist in the visual exploration of system dependencies and, when combined with geographical information, support the thought process for considering disaster scenarios such as flood or earthquake. ASCE has a powerful and flexible graphical modelling environment; in addition, PIA Designer has the capability to export the modelled system to *Google Earth* [9] application if geographical coordinates have been entered as attributes (*status fields* in ASCE terminology) of the constituent model entities. Visualisation can also be used to facilitate communication among different analysts and other stakeholders as they are looking at the graphical models. Model views created in PIA Designer are hereafter also referred to as *ASCE networks* (adopting the jargon used by the ASCE tool developers and modellers).

- *Analysis automation.* Mathematical modelling is aided, and parts of it are automated through the integration of the tools for qualitative and quantitative analysis in the PIA Toolkit.

A significant part of the PIA Toolkit development has involved the work on integration of the PIA Designer and the Execution Engine. This work aims to deliver a seamless integration between the two parts of the toolkit—the user interacts with PIA Designer as the front-end of the toolkit, with only the minimal need for interaction with the Möbius-based execution engine. After the user finishes with the PIA model development in PIA Designer, the necessary information is communicated to the Execution Engine, so that probabilistic model is initialised and the simulation is executed.

In this section we give a description of how the PIA Toolkit is used to support the PIA method with the benefits of visualisation and enhanced analysis features, as well as present the architecture of the toolkit.

### 3.1 PIA Toolkit Usage Patterns

In general, we envisage two approaches to development of PIA models:

- Interactive analysis performed “from scratch”, whereby the analyst is building the PIA model afresh by going through the PIA stages as described in Section 2.2. In this way, the PIA Toolkit user would generate the model views manually, in a step-by-step manner: creating the necessary physical entities belonging to the CI services under scrutiny, assigning the required parameter values to these entities, creating and parameterising stochastic associations etc.
- Automated analysis, whereby the analyst is using pre-existing data sets about the modelled system to programmatically produce initial graphical representations of it. This approach is suitable when, for example, a description of a real incident is available. In such cases, some software utilities that are part of the PIA Toolkit can help generate ASCE networks belonging to a particular PIA study.

In both cases, the PIA stages as defined in Section 2.2 are followed. However, depending on the amount and the format of the data sets available, the analyst will have to go through a series of steps where interaction with the tools will be required. It should be noted that these two PIA development approaches are not necessarily mutually exclusive, i.e. the Interactive and Automated analysis might be combined inside a single PIA study.

### 3.2 Overview of PIA workflow

This sub-section provides a software-centred description of the steps taken when using the PIA Designer to accomplish a series of tasks necessary for the development of a PIA model. It explains typical usage of the PIA Designer software.

The user of PIA Designer, PIA Analyst, can generate two broad categories of the PIA model representations, i.e. *model views*: *Intra-* and *Inter-CI* service views. The former depicts the associations between the entities of the same CI service, while the latter depicts the associations across CI service boundaries. An orthogonal categorisation of the model views is as follows:

- *Physical Network (PN) view*, which depicts the physical entities (physical nodes or physical links) of the interacting CI services (e.g. when modelling a PN view of an electrical power

transmission service, we would for example include high-voltage cabins as nodes and high-voltage trunks as links among the other kinds of entities the service might consist of).

- *Stochastic Associations (SA) view*, which is a representation of the stochastic associations (see Section 2.2 for an explanation of the term) between the model entities.

The interaction between the PIA Designer software and its user consists of the following sequence of steps:

- Step 1.** *Create Notational Schemas.* By executing this step the PIA Analyst creates the necessary ASCE *schemas* that are used as the basis for developing ASCE networks (see Appendix A) used in the PIA model. For example, a notation for describing the Intra-PN view of each service modelled in the PIA study will need to be developed.
- Step 2.** *Create PIA Project network.* PIA Analyst then creates PIA Project network which is used to maintain the references to all ASCE networks belonging to a particular PIA model. Every PIA Project has a *default* set of ASCE networks. The set of default networks includes: *Inter-PN*, *Inter-SA*, *StateTransitions* and *SimulationPlugins* networks. For more information see sub-section C.1 in Appendix C.
- Step 3.** *Populate State Transitions network.* The user creates the *state machines* for all the entity kinds, i.e., all TMEs (see Stage 5 in Section 2.2) used in the particular PIA model. For example, if the PIA was applied to a Public Switched Telephone Network (PSTN) CI, then the set of TMEs could include the following: Backbone Exchange, Transit Exchange, Local Exchange etc.

A state machine for each TME consists of the set of states and the associated state transitions. Each state transition needs to be parameterised. The state transition parameters include the i) *Function\_type* – the family of functions this state transition belongs to, ii) *Function\_name* – the particular function used for the state transition and iii) *Function\_parameters* – the parameters that the state transition function accepts. Initially, every entity instance of a particular TME takes on the default state transitions values. The PIA Analyst can, however, specify a set of parameters for a particular entity instance which is different than the default one. For more information see sub-section C.2 in Appendix C.

- Step 4.** *Populate Physical Network (PN) networks.* This step consists of the following two activities:

*Populate Intra PN networks.* This is the central point of the development, or refinement, of the PIA model of any of the services: it processes the topology information of physical networks of each modelled CI service to create respective graphical representations (i.e., ASCE networks). Each such network is based on the notation, i.e., ASCE schema, created in Step 1. These networks consist of a possibly large number of entities and thus the visual representation aids in comprehending their complexity.

*Populate Inter PN networks.* This activity is based on the definition of the coupling points – the model entities which represent interfaces between different CI services (e.g., a high-voltage-cabin is a coupling point of a Power Transmission service supplying the power to a base transceiver station of a Telecommunication GSM service). The coupling points are defined for both PN and SA model views. According to the PIA method, there are two interpretations of the term coupling point: when considering PN view the term defines if an entity is physically connected to, or more

precisely causally affects, at least one entity from another service, while in an SA ASCE network the attribute defines if an entity is stochastically associated with at least one entity from another service (see Section 2.1 for further explanation of the coupling points term).

For more information about this step see sub-section C.3 in Appendix C.

**Step 5.** *Populate Stochastic Associations (SA) networks.* This step includes the following:

*Populate Intra SA networks.* During the execution of this step the stochastic associations between the entities of the particular, i.e., “native”, CI service are identified and the values of the parameters of each association are specified. In addition, in the Intra-SA view the stochastic associations affecting entities belonging to other, i.e. “foreign” CI services are identified if an entity, say  $E_N$ , from the native CI stochastically affects an entity, say  $E_F$ , from a foreign service, i.e.,  $E_N$  is stochastic “parent” of  $E_F$  (see Section 2.2). Each service from the underlying PIA model has its own Intra-SA view. The underlying ASCE schema of any Intra-SA view is, however, general-purpose – it is shared among all PIA studies.

*Populate Inter SA networks.* Please see the description of the sub-step Populate Inter PN networks (Step 4) above.

For more information see sub-section C.4 in Appendix C.

**Step 6.** *Configuration of the Simulation Plug-ins.* This step allows for parameterisation of the simulation plug-ins which can be embedded in the probabilistic model execution.

A simulation plug-in<sup>3</sup> is a standalone piece of code, i.e., a dynamically linked library, which extends the functionality of the PIA *simulation model template*, which is the central part of the PIA Toolkit execution engine. The categories of simulation plug-ins are as follows: Initialisation, Deterministic, Trace, Rate and Reward. In each PIA model there must exist one, and only one, Initialisation plug-in, and there are zero to many simulation plug-ins belonging to the other categories. The Initialisation plug-in is used for the initialisation of the PIA simulation model template according to the needs of the particular PIA study.

For more information see sub-section C.5 in Appendix C.

**Step 7.** *Creation and configuration of the PIA simulation study.* The data about the PIA model is gathered from the respective model views, by examining the corresponding ASCE networks, and passed to the Möbius-based execution engine. These data are supplied in a particular format and serve the purpose of inputs to the probabilistic model simulation.

Examination and data gathering from the model views (ASCE networks) is performed using a separate PIA Toolkit component developed in Java programming language.

For more information see sub-section C.6 in Appendix C.

Further explanation of the use of the PIA Designer tool is given in its User Manual document in the Appendix C.

---

<sup>3</sup> The term *plug-in* is used for describing different parts of the PIA Toolkit. Beside the simulation plug-ins there are ASCE-based PIA Designer plug-ins, which are standalone pieces of a scripting language code that enhance the functionality of the ASCE engine. These ASCE-based plug-ins have been used extensively for implementing the functionality of the PIA Designer tool.

### 3.3 PIA Toolkit Execution Engine

The back-end part of the PIA Toolkit consists of the custom-built execution engine which is based on Möbius tool simulation solver (see Appendix A, as well as the deliverables produced as part of the IRRIS project [6], for details about the Möbius tool). The execution engine is based on the concept of continuous time, discrete event driven Monte Carlo simulation.

The central part of the execution engine is the PIA *simulation template*, a general-purpose probabilistic model implemented in the Möbius tool, which is used as the basis for generating an arbitrary PIA probabilistic model. The data obtained from the qualitative part of the PIA model, developed in the PIA Designer, is used to initialise the PIA simulation template in the specific way. These data include information about particular topology, stochastic associations and state transitions of the underlying PIA model.

The main characteristic of the PIA simulation template is its pluggable architecture. Using this kind of architecture the functionality of the template is enhanced and augmented through simulation plug-ins – additional pieces of standalone code distributed in the form of dynamically shared libraries (in the terminology adopted by Microsoft Windows they are referred to as dynamically linked libraries (dlls)). Each simulation plug-in has an associated data file with it, which is used for the configuration of the respective dll. There exist different categories of simulation plug-ins, differentiated based on its purpose. Simulation plug-ins are used for:

- initialising the probabilistic model in the way specific to a particular PIA model. This type of simulation plug-ins belongs to the Initialisation category.
- augmenting and/or enhancing the functionality of the simulation template. There are several categories of this type of simulation plug-ins: *Deterministic*, *Trace* and *Reward*. A short description of each category is provided below.

*Deterministic* plug-ins implement engineering /deterministic models which are embedded in the generic model of stochastic dependence – they can influence the “dynamics” of a subset of model entities in a specified way. For example, a deterministic model describes how a subset of model entities instantaneously change state values of another subset of model entities (see Stage 6 of the PIA method described in Section 2 for further description of the *deterministic* models). Examples of deterministic plug-ins are as follows: Direct Current (DC) approximates power flow model for power flow components, or “flattening” of the electrical battery after a fixed period of time, etc.

*Trace* plug-ins augment the functionality of generating the simulation traces of particular failure scenarios. The simulation traces are computer generated formalised text descriptions of a sequence of discrete events in simulated time. They are occasionally found to contain complex cascading event sequences. The trace generation feature can be enhanced with the complementary visualisation of the simulation event sequences, in order to aid the understanding of model behaviour. Also, the trace and the associated visualisation should help in examining the (large-scale) interdependency effects between CIs. After a careful review, for the current purposes, we decided to use the Google Earth application to visualise these traces. Google Earth [8] offers a rich feature set for displaying 2D and 3D images of varying resolution of the Earth's surface and as such it enables a powerful graphical interface for showing state changes of each of the CI model entities.

*Reward* simulation plug-ins enhance the functionality of the template in regard to Möbius reward variables. The Möbius-based execution engine analyses the constructed models to estimate parameters of so-called *reward function*. Analysts are free to define an infinite variety of *reward functions* (or *rewards*) once a model has been defined and implemented using the Möbius-based execution engine. These rewards are functions on the simulator event sequence for one replication of a Möbius simulation of a model built using this tool. Essentially most such reward functions will usually either count transitions between subsets of the model's state space; or accumulate the total simulated time spent within some subset of the input space; or integrate some step function of time defined as a function of the current model state.

### 3.4 PIA Toolkit Architecture

A schematic representation of the PIA Toolkit architecture is given in Figure 5, as a UML Component diagram. The component diagram displays the software execution environment of the whole PIA toolkit. It models the software with concrete elements in the physical world that are the result of a development process and reveals software configuration issues through dependency relationships.

PIA Designer consists of the following components:

- An instance of the ASCE tool executable,
- 15 PIA ASCE plug-ins which implement the functionally necessary for the qualitative PIA. The plug-ins are implemented using JavaScript programming language, with the exception of one of them, *PIA\_InterServiceLinks.xml*, which is implemented using VBscript language.

The plugins enable the PIA analyst to create the PIA model views, each one of which uses a PIA ASCE schema as its underlying notation. The ASCE plug-in referred to as *PIA\_SimulationPluginsConfigurator.xml* is used for the creation of a subset of the files used as inputs to the PIA execution environment – these files are used for the configuration of the simulation plug-ins used in the simulation model. There are 3 such files: *Plugin paths and names*, *Legacy Initialize Nodes Plugin Data*, and *Plugins ID mappings*.

- Seven PIA ASCE schema files which are used as the underlying notations for the various PIA model views (ASCE networks):
  - *PIA.xml*, *PIA\_InterCIPN.xml*, *PIA\_InterCISA.xml*, *PIA\_IntraSA.xml*, *PIA\_Project.xml*, *PIA\_SimulationPlugins.xml*, *PIA\_StateTransitions.xml*.

These ASCE schemas are general-purpose. They are to be used in an arbitrary PIA study. In addition to these 7 schemas, PIA assumes that a separate schema for Intra-PN view is created for each service modelled in a particular PIA study. In one of the case-studies used throughout the PIA:FARA project, we have created 2 such schemas: *PIA\_PowerPN.xml* and *PIA\_TelcoPN.xml*, which are used for graphically representing Power and Telco CI service, respectively, of the *Rome* case-study.

- The *PIA Simulation Study Generator* utility, which examines the ASCE networks created by the PIA ASCE plug-ins and creates 3 input files necessary for the execution of the PIA simulation: i) *ServiceTimesGen* file – this file contains the data about the entities used in the PIA study and the corresponding topology, the data about the state transitions of each entity instance, and the values of the parent scaling factors for each entity instance; ii) *RomeCMGen* – this file contains the information about the direction of the stochastic associations between the entities of the PIA study – for each model entity (“parent”) the set

of entities it stochastically influences (“children”) is provided; and iii) *ServiceKinds* – this file contains the information about the TMEs (types of the modelling entities), i.e., entity kinds, used in the PIA study.

The PIA Execution Engine uses the files generated by the PIA Designer as its inputs. This part of the PIA toolkit includes the following components:

- PIA simulation model template, which is initialised in the specific way for each PIA study. The initialisation data are obtained from the 3 input files generated by the PIA Designer: *ServiceTimesGen*, *RomeCMGen* and *ServiceKinds*.
- Exactly one simulation plug-in belonging to the Initialisation category. This plug-in implements the initialisation of the PIA simulation model template for a particular PIA study according to the information supplied in the three input files. The data file for the initialisation plug-in contains the absolute path to the directory of the simulation model where the three input files are located.
- (*Optional*) A set of simulation plug-ins belonging to the other categories. Each of these plug-ins has a data file associated to it. Also, there is another file specific to the simulation plug-ins belonging to the Deterministic category. The file, titled *Plugins ID mappings*, contains the mappings between the entities used in the PIA study and the entities used in each of the deterministic plug-ins.

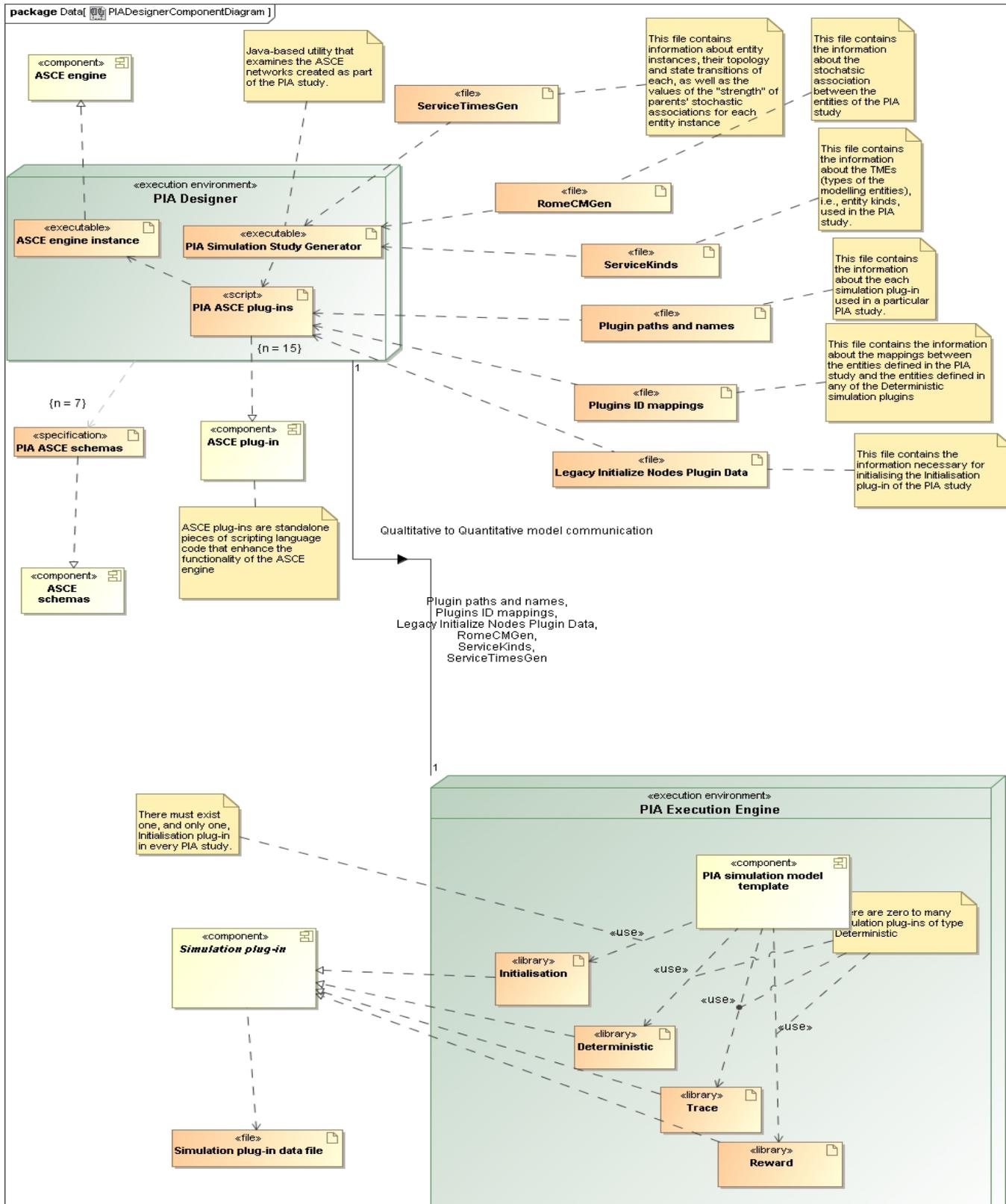


Figure 5: UML component diagram of the PIA toolkit

### 3.5 Tool support for the service-level model

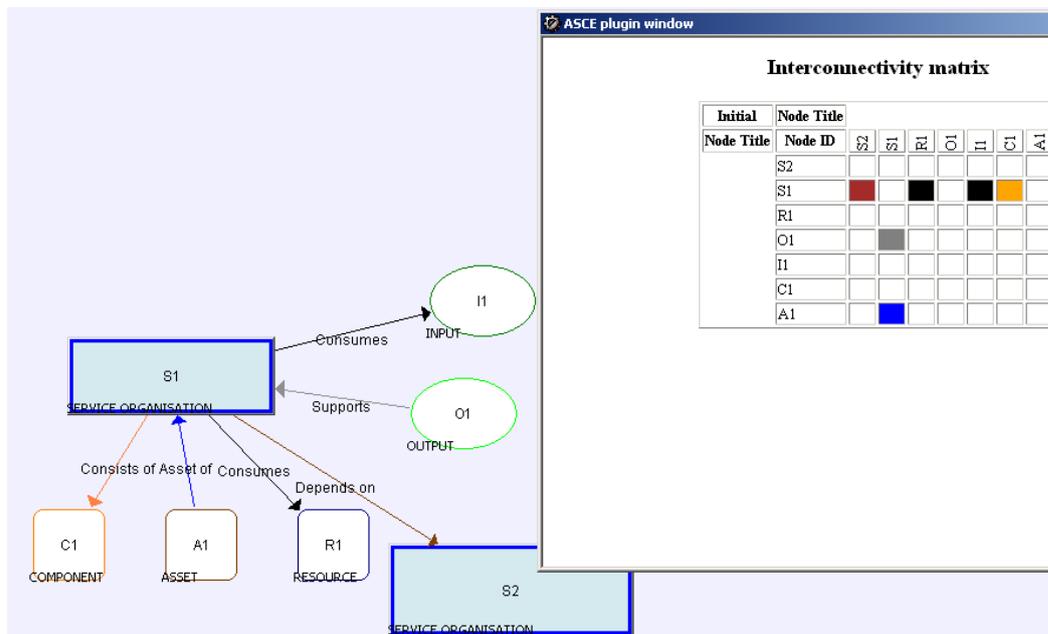
In addition to the ASCE plug-ins used for the development of the PIA Designer tool (see Section 3.4), we have implemented a couple of ASCE plug-ins which aid the creation of the model of interacting services (service-level model).

As discussed in Section 2.2.4, the identification of dependencies between CIs lies upon the careful consideration of the relationships between model entities, and a thoughtful process of asking what-if questions and applying scenarios to the model.

Here, the tool and the effort placed in providing the definitions and the service model begin to produce results: assisted by the visualisation of ASCE's graphical environment, the user can explore dependencies by examining the network and can communicate scenarios and findings with colleagues. In addition, the PIA Toolkit also comes with two ASCE plug-ins that can be used to enhance this investigation. These are discussed in the following sub-sections.

#### 3.5.1 Interconnectivity matrix plug-in

Figure 6 shows a screenshot of the Interconnectivity matrix plugin. The plugin can be used to present in a tabular format the various links between the model entities devised in service-level model. This can be used to improve the visualisation of associations and dependencies between services.



**Figure 6: Interconnectivity matrix plug-in**

Furthermore, the blank cells in the matrix indicate that there is no dependency recorded between the entities in the corresponding pair. The user can comment on the justification for why there is no dependency between the two entities.

The interconnectivity matrix can be exported to a Microsoft Excel spreadsheet.

### 3.5.2 Google Earth plug-in

In order to consider the aspect of geographical dependencies, Geographical Information Systems (GIS) need to be used. Having entered the geographical coordinates for each entity modelled in the ASCE network, invoking the Google Earth plug-in will result in a map being produced which represents nodes in the model and their service status (see Figure 7).

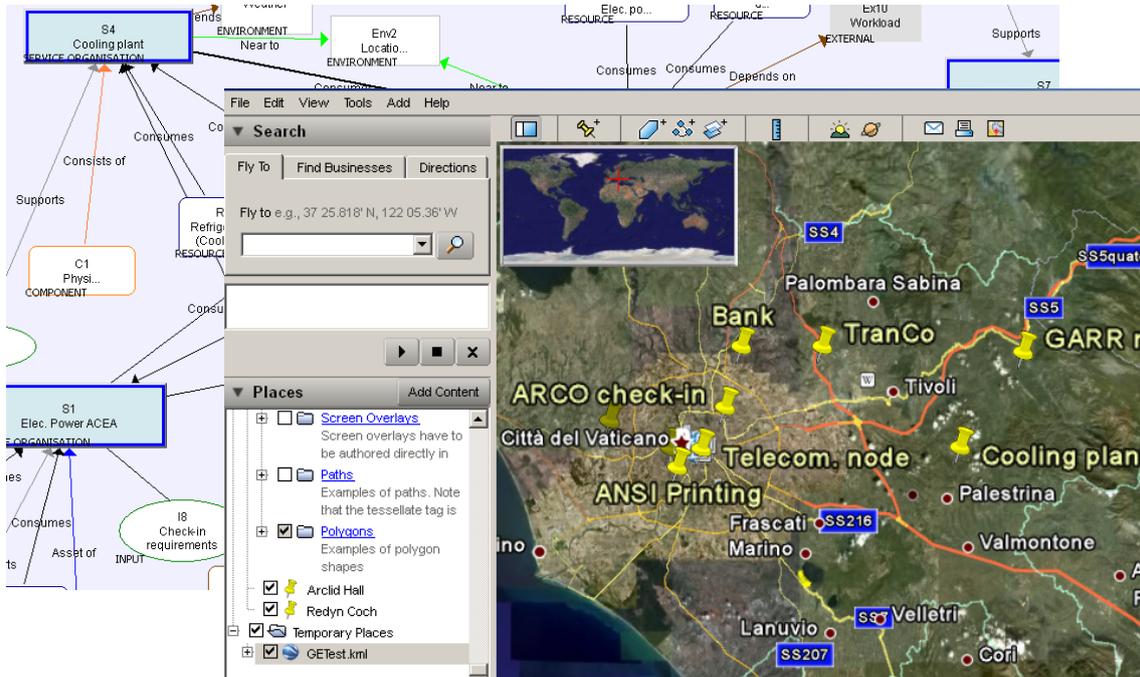


Figure 7: Google Earth plugin

These two plug-ins, along with the ASCE graphical environment, can facilitate the qualitative examination and investigation of dependencies. Once the system has been built, the analyst can begin to ask what-if questions, and reconfigure the model parameters to observe changes in the system behaviour. As discussed in Section 2.2, the information about the system, previous incidents and the threat agents will shape a threat model that can drive this exercise.

## 4 Conclusions

This report presented the PIA approach and described how the associated toolkit is to be used to carry out a PIA study. This section summarises the benefits of using this approach.

PIA is a lightweight, fairly quick and easy, affordable approach to identifying and analysing interdependencies in complex systems. The analyst, with very low start-up costs, can model the entire system under investigation, starting from a high level of abstraction, and taking a modular approach to model development. With PIA, the analyst will be able to identify, analyse and quantify the risk of interdependency down to a physical component level.

Given the complexity and size of CI systems and the high costs required for such analyses, a preliminary, affordable approach such as PIA can become very useful in identifying further modelling and analysis requirements: for instance, a PIA may identify a critical vulnerability in a segment of the infrastructure which will need to be modelled with high fidelity and/or real time. This may not be a requirement, however, for the entire inter-service model. PIA can

identify further requirements for specific elements of the system, whilst maintaining a good understanding of the entire system within the wider scope of the models. Given the costs of such modelling, carefully deciding what to model will support making the right decisions in terms of resource allocation, hardware performance requirements and investment overall.

The WP2 of the PIA:FARA project delivered a tool-supported method, along with supporting guidance for CI interdependency analysis. Besides the use of the method on a case study based on the incident near Rome, Italy (see Appendix B), the method has also been applied internally within the project on another case-study which models an SME's information infrastructure. The case studies were important since they served as a validation for the method and the tools. They also helped the PIA developers to identify the areas of improvement and further development.

## 5 Glossary

Term / abbreviation	Explanation
AFI	Abstract Functional Interface
API	Application Programming Interface
ASCE	Assurance and Safety Case Environment, Adelard <a href="http://www.adelard.com/web/hnav/ASCE/index.html">http://www.adelard.com/web/hnav/ASCE/index.html</a>
CI	Critical Infrastructure
Coupling point	A model entity that represents an interface between different CI services
Dependency	Single direction dependencies of one infrastructure on another
DSBM	Detailed Service Behaviour Models
GIS	Geographical Information Systems
IA	Interdependency analysis
II	Information Infrastructure
IRRIIS	Integrated Risk Reduction of Information-based Infrastructure Systems
LLP	Limited Liability Partnership
Möbius	A software tool for modelling the behaviour of complex systems, developed by the PERFORM research group from the University of Illinois at Urbana-Champaign <a href="http://www.mobius.illinois.edu/">www.mobius.illinois.edu/</a>
PC	Personal Computer
PERFORM	Performability Engineering Research Group
PIA	Preliminary Interdependency Analysis
PIA Toolkit	A set of software tools to support the PIA method
PSTN	Public Switched Telephone Network
PVS	PVS is a specification language integrated with support tools and a theorem prover. See <a href="http://pvs.csl.sri.com">http://pvs.csl.sri.com</a>
SAN	Stochastic association networks are mathematical modelling formalism which provide stochastic extensions to Petri nets and are typically used for performance and dependability evaluation.
SMEs	Small-to-Medium Enterprises
State	Model entities can have a variety of states that i) describe their different phases of operation (e.g. start-up, operating, shutdown, maintenance), ii) describe their level of operability (e.g. OK, impaired, failed), iii) indicate characteristics of the commodities supplied (e.g. power phase angle for electrical power services). The definition of the possible states/modes that a PIA model entity can occupy is necessary for the identification and analysis of (inter)dependencies between model entities.
State machine	A mathematical abstraction which describes the behaviour of model (entity) and is composed of a finite number of states and state transitions.
State transition	The event of a state change of a particular model entity
TSB	Technology Strategy Board, see <a href="http://www.innovateuk.org/">www.innovateuk.org/</a>

---

UML	Unified Modelling Language is a standardised general-purpose modelling language in the field of software engineering. <a href="http://www.uml.org/">http://www.uml.org/</a>
XML	Extensible Mark-up Language

## 6 Bibliography

- [1] Bloomfield R, Chozos N, and Nobles P, “Infrastructure interdependency analysis: Requirements, capabilities and strategy”. Adelard document reference: d418/12101/3, issue 1, 2009, available for download at <http://www.csr.city.ac.uk/projects/cetifs.html>
- [2] PIA: FARA project, WP 1 deliverable, “Industrial Sector-Based Modelling of 1337 Critical Infrastructure Incidents in the European Union”, Adelard document reference D/496/12102/1, Issue 1, April 2010
- [3] Courtney T, Derisavi S, Lam V, and Sanders WH. “The Möbius Modeling Environment”. Tools of the 2003 Illinois International Multiconference on Measurement, Modelling, and Evaluation of Computer-Communication Systems, Universität Dortmund, Fachbereich Informatik, 2003
- [4] Adelard LLP, Assurance and Safety Case Environment (ASCE) tool, <http://www.adelard.com/web/hnav/ASCE/index.html>
- [5] Idaho National Laboratory, “Critical Infrastructure Interdependency Modeling: A Survey of U.S. and International Research”, 2006, [www.inl.gov/technicalpublications/Documents/3489532.pdf](http://www.inl.gov/technicalpublications/Documents/3489532.pdf)
- [6] EU project IRRIS, “Integrated Risk Reduction of Information-based Infrastructure Systems, EU project”, 2006–2009, <http://www.irriis.org>
- [7] Ciancamerla, E. and M. Minichino, A Mini, Telco-blackout Impacting Other Infrastructures, Including ACEA Power Grid, 2007, ENEA. Ciancamerla, E. and M. Minichino 2007
- [8] Google, Google Earth, <http://earth.google.co.uk/>
- [9] Computer Science Laboratory, PVS Specification and Verification System, <http://pvs.csl.sri.com>
- [10] Hammersley, JM, Handscomb, DC Monte Carlo Methods. London: Methuen, 1975
- [11] Object Oriented Group, <http://www.bpmn.org/>
- [12] University of Illinois at Urbana-Champaign, Mobius, <http://www.mobius.illinois.edu/>

## Appendix A An overview of the ASCE and Möbius tools

### A.1 Assurance and Safety Case Environment (ASCE)

ASCE is an information management and representation tool that provides a flexible, dynamic and customisable graphical environment with highly extensible analytical capability. The tool is an industry standard for the UK Ministry of Defence, used widely among nuclear and aviation industries for the creation and management of safety cases.

ASCE is highly customisable software product: on top of its underlying engine lies a flexible graphical environment, which can be used to design or modify graphical notations (referred to as *ASCE schemas*). These schemas allow ASCE to be used in a wide range of business contexts where there is a benefit in explicitly structuring information into meaningful parts according to a particular notation. *ASCE networks* are collections of nodes and links created according to a particular ASCE schema, and visualised using the ASCE display engine.

Creating nodes and links is made simple and intuitive through the use of a drag-and-drop feature. ASCE has a powerful, easy-to-use node editor which allows users to import text from standard word processing applications. It supports text formatting, tables and heading styles to show the logical structure of the document.

In addition, the functionality can be extended with the use of “*plugins*” – pieces of additional code, written in scripting languages such as VBscript or JavaScript, which are configurable by the user: a plugin is either loaded or disabled depending on the needs of the particular user. Examples of what functionality is achievable with the plugins are as follows:

- connecting to third party tools and file formats, interrogate them and importing the data
- exporting data about ASCE networks
- popup windows to present and collect information from the user
- analysing the structure of ASCE networks

A comprehensive description of the ASCE tool can be found on the Adelard website [4].

ASCE-based support in the PIA Designer consist of a set schemas and associated networks which model the various views of interdependent CIs, e.g. physical network view, stochastic association view etc., each one of which is represented with a particular ASCE network. Also a number of ASCE plug-ins have been developed to support the application of the PIA method.

### A.2 Möbius

Möbius™ is a software tool for modelling the behaviour of complex systems. The tool is based on the framework which allows for formal mathematical specification of model construction and execution. The tool is extensible in the sense that new modelling formalisms and model solution methods can be easily integrated in this broad framework. The core of the implementation of this framework is *abstract functional interface (AFI)*, which includes a set of functions that enables both the communication between models, as well as the communication between models and model solvers.

Models can be solved either analytically/numerically or by simulation. Source code in C++ programming language is generated and compiled for each model, and the object files are

packaged to form a library archive. These libraries are linked together along with the tool's base libraries to form the executable for the solver. The executable is run to generate the results. The base libraries implement the components of the particular model formalism, the AFI, and the solver algorithms.

Although the tool was originally developed for studying the reliability, availability, and performance of computer and network systems, its use has expanded significantly. Time- and space-efficient discrete-event simulation and numerical solution, based on Markov processes, are both supported.

The tool was developed, and is maintained, by the Performability Engineering Research Group (PERFORM) in the Centre for Reliable and High-Performance Computing at the University of Illinois at Urbana-Champaign. More information about the tool can be obtained from the webpage at [12].

## Appendix B Rome case-study – an example of applying the PIA method

One of the case-studies used in the PIA:FARA project for further development and validation of the PIA method is based on the real incident that occurred in Rome, Italy. The incident affected telecommunications, and subsequently the power, CI. The case-study is referred to as *Rome scenario*. Further information about the incident can be found in [6][7].

One of the purposes of this analysis is to carry out an in-depth analysis of a major, nationwide, complex incident, and identify other likely failure modes that could have emerged in this case.

There are two critical infrastructures modelled in the Rome scenario: Power and Telco. Power CI consists of 2 CI services: Transmission and Distribution, while the Telco contains 3 services: Public Switched Telephone Network (PSTN), Global System for Mobile Communications (GSM) network and Synchronous Digital Hierarchy (SDH) network. For each of the CI services two ASCE networks are created:

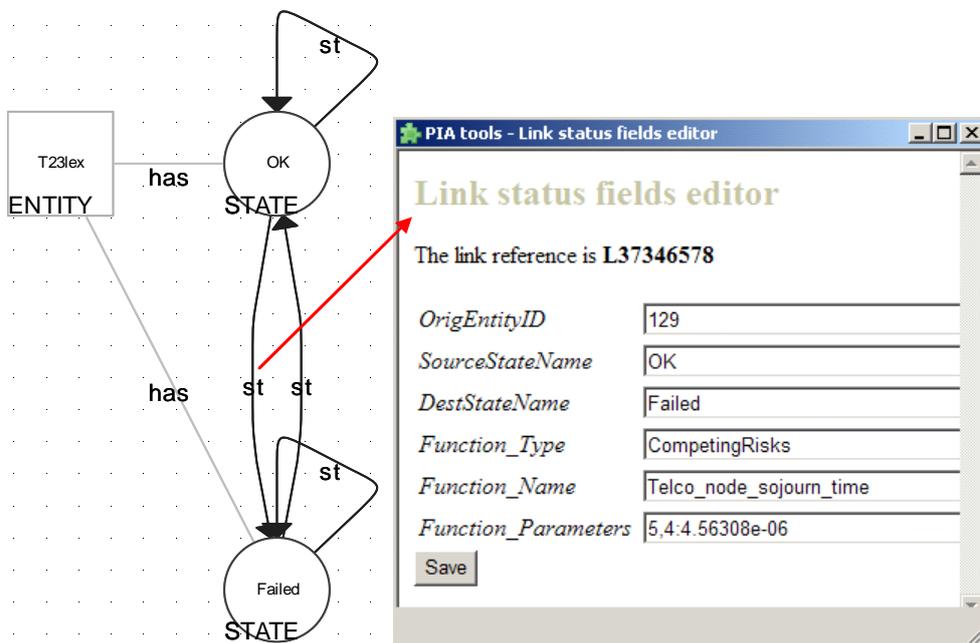
- The first depicts Intra-service Physical Network view; a part of one such network, the Power Transmission PN, is given in Figure 9.
- The second depicts Intra-service Stochastic Association view; one such example, the Telco SDH SA network, is given in Figure 10.

In Figure 9 some of the entity types of Power CI service are shown: *hvc* – high voltage cabin (represented by the blue square), *swt* – power switch (represented by the purple square), high voltage trunk (represented by the blue link), copper cable (represented by the orange link). The window on the right presents the attributes, i.e. ASCE status-fields, of one of the depicted nodes – *P13hvc*.

The entity types of Telco CI service, which are depicted in Figure 10 as ASCE nodes, are as follows: physical nodes (such as *T16adm*, an add-drop multiplexer) and physical links (such as *ring6-T28adm-T30adm*, an add-drop multiplexer ring). The ASCE links indicate the stochastic associations between the entities of the SDH service. The associations are all reciprocal for depicted subset of entities of the particular CI service – this indicates interdependency between the nodes of each node pair. But, surely, there is a possibility of unidirectional stochastic associations. The window on the right presents the attributes, i.e. ASCE status-fields, of one of the depicted nodes – *T16adm*.

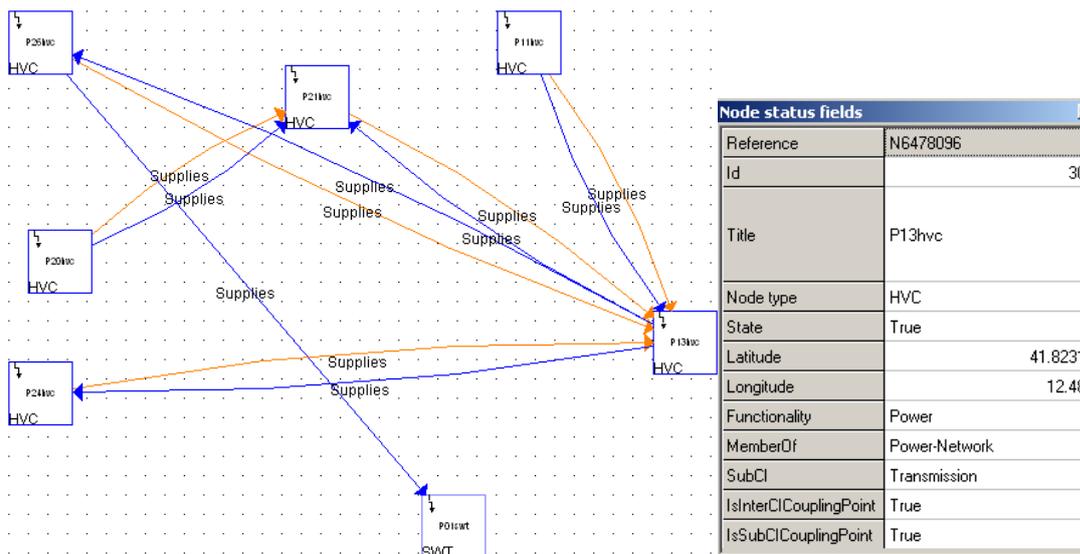
Besides showing a CI service, it is also possible to generate an ASCE network representing a PN model view or an SA model view of the whole CI (Power or Telco).

Each of the model entities has an associated set of *state transitions* based on the corresponding state space definition. State transitions of every model entity are represented with a state machine diagram implemented in ASCE (one such state machine is given in Figure 8), in which the nodes represent state values and links represent state transitions. Each state transition is characterised with the following attributes: a function that calculates the probability of that state transition, the type/family of the particular function, and the parameter values needed for the calculation of the function. In this example, the size of the entity's state space is 2 (the state values are "Failed" and "OK"). The "OK-to-Failed" state transition (pointed from by the red arrow) has the attribute values as follows: the type of the function is: *CompetingRisks*, the function name is: *Telco\_node\_sojourn\_time* and the function parameter values are: *5,4:4.56308e-06*.



**Figure 8: A state machine diagram for a local telephone exchange and the links status-fields editor plug-in**

An ASCE plug-in has been developed to provide the option of editing the state transition parameters (see Figure 8). The plug-in is named *Links status-fields editor* and is an example of use of the powerful ASCE plug-ins, which add functionality beyond that provided by the core ASCE product. This plug-in analyses the structure of the underlying ASCE network and the associated schema, and displays the status field values of the selected link.



**Figure 9: Part of the intra-service PN view of the Power Transmission service**

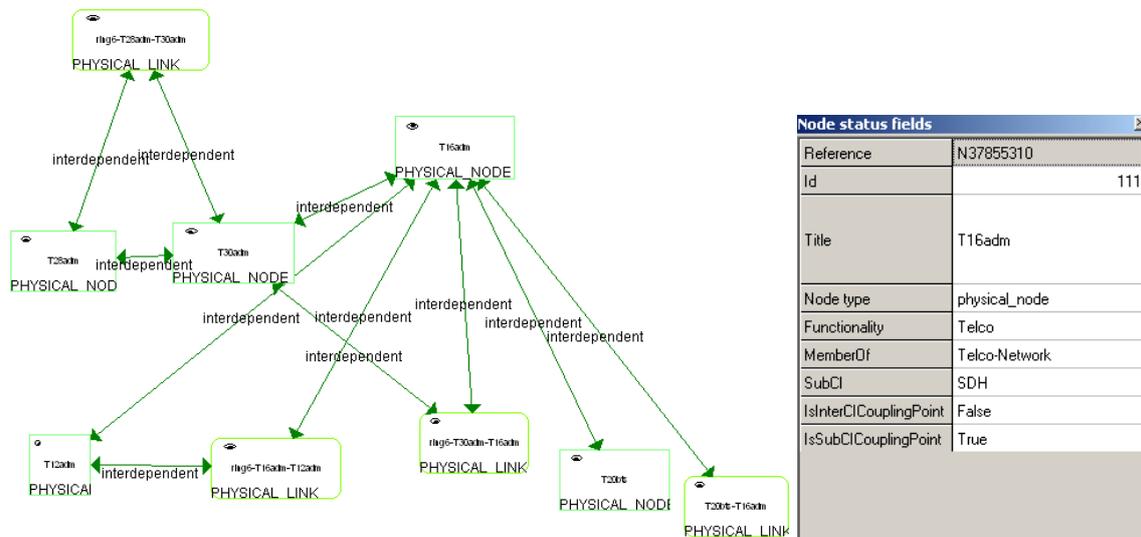


Figure 10: A part of the SAN of the Telco SDH service

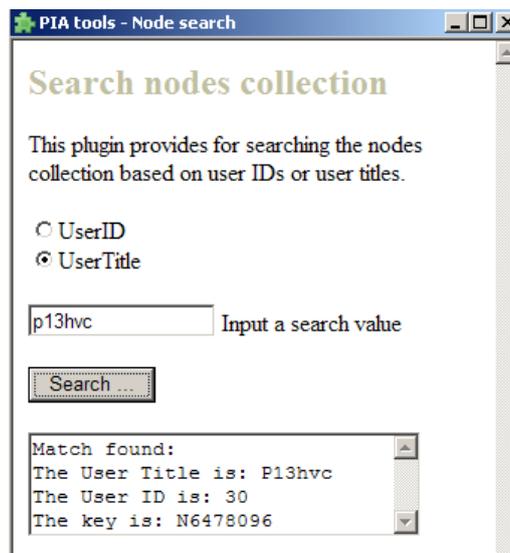


Figure 11: The “Node search” plug-in

Beside the plug-in *Links status-fields editor*, several other ASCE plug-ins have been implemented to facilitate the development and refinement of probabilistic models. Some of the plug-ins are as follows:

- “*Node search*” (see Figure 11) – it allows for the ASCE nodes’ collection to be examined based on a search term of choice, such as *userID*, or *userTitle*. After the PIA analyst enters the value to be searched, the nodes of the underlying network are examined and, if a match is found, the matched node is navigated to and selected. This plug-in, thus, allows for easier interactive analysis of large topologies.

- *“Propagate network change”* – it allows for a change in one ASCE network to be propagated to another. For example, when the PIA analyst creates a new node in an intra-service physical network, this change is automatically reflected in the corresponding state transition network by creating the state machine for the newly introduced model entity.

## Appendix C PIA Designer User Manual

This Appendix describes the necessary steps for creating an arbitrary study in ASCE-based *PIA Designer* tool.

The prerequisites for creating a PIA Project using the PIA Designer tool are as follows:

- A compatible ASCE version needs to be installed. The oldest compatible ASCE version is v4.0.77.
- The PIA ASCE schemas are installed in the schemas' directory located in the user's *Application Data* directory (i.e., in the *schemas* sub-directory of the *ASCE User Settings* directory). An example of the absolute path of the schemas subdirectory is given below:  
C:\Documents and Settings\\Application Data\Adelard\ASCE\4.0\schemass
- The PIA ASCE plug-ins are installed in the plug-ins' directory located in the user's *Application Data* directory (i.e., in the *plugins* sub-directory of the *ASCE User Settings* directory). An example of the absolute path of the plug-ins subdirectory is given below:  
C:\Documents and Settings\\Application Data\Adelard\ASCE\4.0\plugins
- The PIA ASCE plug-ins are enabled. This is achieved using the *ASCE Plugin Manager*, which can be started from the *Tools* menu item in ASCE.

### C.1 Part 1 – Create PIA Project ASCE network

The first phase in the process of creating a PIA Project is to build a PIA Project ASCE network. The following steps describe this initial phase.

1. Start the ASCE tool
2. Using the option *New* from the *File* menu item, create a new ASCE network using the *PIA\_Project* ASCE schema.
3. Save the network using the ASCE tool *Save* option. The network is to be saved in the folder where all PIA projects are to be stored, under a name of choice. After saving the new project, a folder with the chosen project name will be created as a subfolder in the PIA projects folder. The saving of the PIA project ASCE network will set the project's attributes *PIA\_Project\_Name* and *PIA\_Projects\_Location*, which are modelled as ASCE network status-fields.

NB: The GUI of the ASCE tool does not allow the user to create a PIA project in the same absolute path that already exists.

4. Start the ASCE plug-in *PIA – Edit network status-fields* from the *File* menu option (Figure 12). Fill in the values for the PIA project attributes, which are modelled as network status-fields, to take on the values corresponding to the newly created project. The status-fields are as follows:
  - *PIA\_Project\_Name* – the value of this status-field has been already set when saving the ASCE network.
  - *PIA\_Projects\_Location* – the value of this status-field has been already set by saving the ASCE network.
  - *Mobius\_Projects\_Location* – this status-field specifies the absolute path of the folder where all Möbius projects are saved. This path is not necessarily the same as the path where the *PIA Designer* projects are saved. If the default value of this status-field exists (i.e., if it has been specified in the *PIA\_Project* schema), it needs to be overridden.
  - *Simulator\_Template\_Location* – this status-field specifies the absolute path of the Möbius template model. The simulation model of every new PIA project is created based on this template model. If the default value of this status-field exists (i.e., if it has been specified in the *PIA\_Project* schema), it needs to be overridden.
  - *Simulator\_Input\_Files\_Folder\_Name* – this status-field specifies the name of the folder where the inputs to the Möbius simulation model of the newly created PIA project are to be stored. The default value of this status-field is *D226inputs*.
  - *Simulation\_Plugins\_Conf\_Folder* – this status-field specifies the name of the folder where the configuration files of the *simulation plug-ins* are stored. Simulation plug-ins are self-contained pieces of code that enhance the functionality of the Möbius template model. Each simulation plug-in is a dynamically linked library which has an associated data file. The accepted file types of the simulation plug-ins' data files are *.txt* and *.xml*.

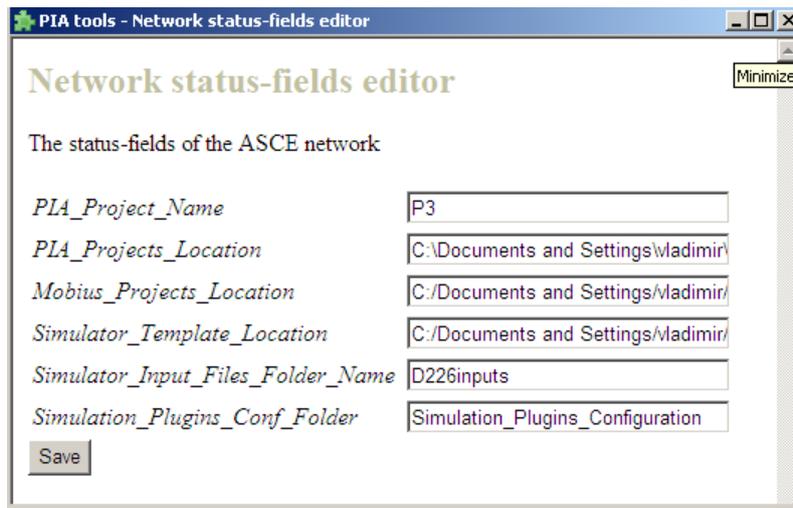


Figure 12: The Network status-field editor plug-in

5. Click anywhere on the ASCE modelling area for the helper HTML form, and the associated ASCE plug-in will be started (the title of the form is *PIA tools – PIA Project validation*, Figure 13).

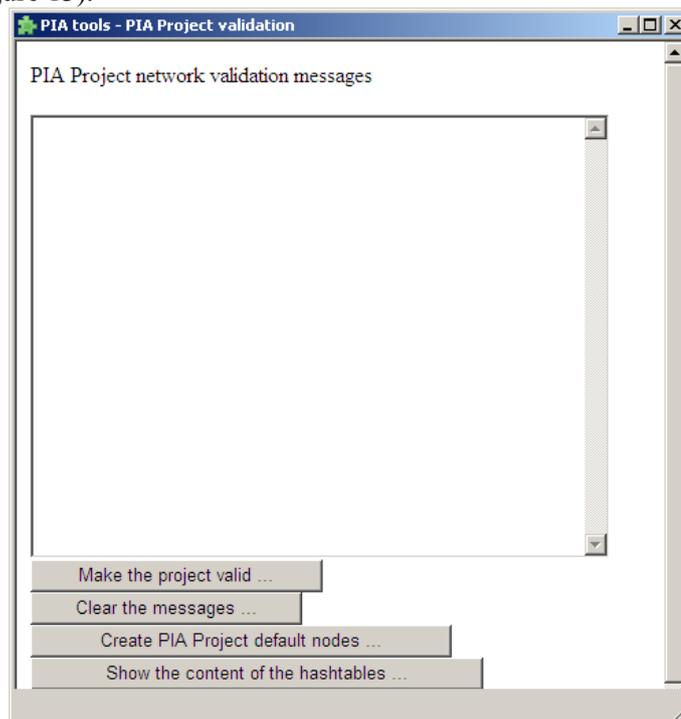


Figure 13: The HTML form used for the validation when creating a new PIA project

6. Create the default set of PIA ASCE nodes by pressing the *Create PIA Project default nodes* button. These nodes represent the ASCE networks that are necessary in any non-trivial PIA project. The default PIA Project nodes are as follows:
  - A node representing default *Intra\_PN* network (named initially *Default\_IntraPN*) is created. The *Intra\_PN* ASCE network will not be created in the project folder until the node in the project view is renamed (a meaningful name, such as *Power1* if a power critical infrastructure is being modelled, is supposed to be given).

Every time a node representing Intra\_PN network is created in the PIA Project ASCE network, the corresponding *Intra\_SA* node will be created programmatically (the convention is that the name of the Intra\_SA node will be composed of the name of the Intra\_PN network and the suffix “SA”). In this way, the user is not expected to create any of the Intra\_SA nodes manually.

- A node representing *StateTransitions* network. The StateTransitions.xml network will be created in the project’s folder. The name of this network is assumed to be the same for all PIA projects.
  - A node representing *SimulationPlugins* network. The SimulationPlugins.xml network will be created in the project’s folder. The name of this network is assumed to be the same for all PIA projects.
  - A node representing *InterCIPN* network. The InterCIPN.xml network will be created in the project’s folder. The name of this network is assumed to be the same for all PIA projects.
  - A node representing *InterCISA* network. The InterCISA.xml network will be created in the project’s folder. The name of this network is assumed to be the same for all PIA projects.
7. The user is expected to confirm the creation of each of the *default* ASCE networks before them being created in the project folder.
  8. The ASCE networks created in the project folder will have some of their respective network status-fields automatically set, e.g. network status-fields for the project path and the project name will be set programmatically.
  9. The user creates an Intra\_PN node for each service organisation modelled in the particular PIA project.
    - The user specifies the name of the network in the UserTitle property of the corresponding node.
    - For each Intra\_PN network the user specifies the corresponding ASCE schema, by choosing a value from the list given in the *Schema\_Name* status-field. By convention, the names of the Intra\_PN schemas end with the suffix “PN”.
    - Once the name of the node representing an Intra\_PN network is given, and the name of the corresponding schema specified, the empty Intra\_PN ASCE network will be created in the project’s subfolder.
    - The addition, as well as the update, of a node in the PIA project network is supported by the event-handling procedures of the following events thrown by the ASCE tool: *AfterNodeAdd* and *AfterNodeUpdate*. For further information see the ASCE tool documentation.
  10. Once a user deletes a node, say an Intra\_PN node, from the PIA project network, the corresponding Intra\_PN network from the project folder will be deleted.
    - The deletion of an Intra\_PN network node will be followed by the programmatic deletion of the corresponding Intra\_SA node – the user will be asked if the Intra\_SA node is to be deleted or not.
    - The deletion of the ASCE nodes is supported by the handling of the following event thrown by the ASCE engine: *AfterNodeDelete*. For further information see the ASCE tool documentation.

11. After the user has specified all the networks for the particular PIA project, he/she closes the PIA project ASCE network and commences the building of the individual ASCE networks saved in the project folder.

## C.2 Part 2 – Populate the State Transitions network view

The next phase in building a PIA Project is used to define the state transitions and associated parameterisations i.e. create *state machines*, for all entity kinds used in the study. This phase consists of the following steps:

1. Open the empty *StateTransitions* network, which was created as part of the phase Part 1 – Create PIA Project ASCE network.
  2. Create an *Entity\_Kind* node for each entity kind modelled in the PIA Project (for example, if we are modelling two service organisations, a power and a telecommunications service, a set of entity kinds could be: *HighVoltageCabin*, *MediumVoltageCabin*, *AddDropMultiplexer*, *HighVoltageTrunk* and *MVtoTelcoSite*)
  3. For each *Entity\_Kind* create the set of *State* nodes, each of which represents a state value associated with that entity kind. For example, the state space of an entity kind might consist of the following state values: *OK* and *Failed*.
  4. Create state transitions between state nodes – for each state transition an ASCE link connecting a pair of nodes should be drawn. This includes self-referential state transitions (characterising the cases when no state change occurs).
  5. Parameterise the state transition values, using the ASCE plug-in *PIA Tools – Link status fields editor* (Figure 14). The plug-in can be started from the list of options offered from a context of an ASCE link (this list of options is displayed once the user right-clicks the chosen ASCE link).
- These state transition parameterisations represent the default values for the corresponding entity kind. All instances of a particular kind take on these values, if not specified differently.

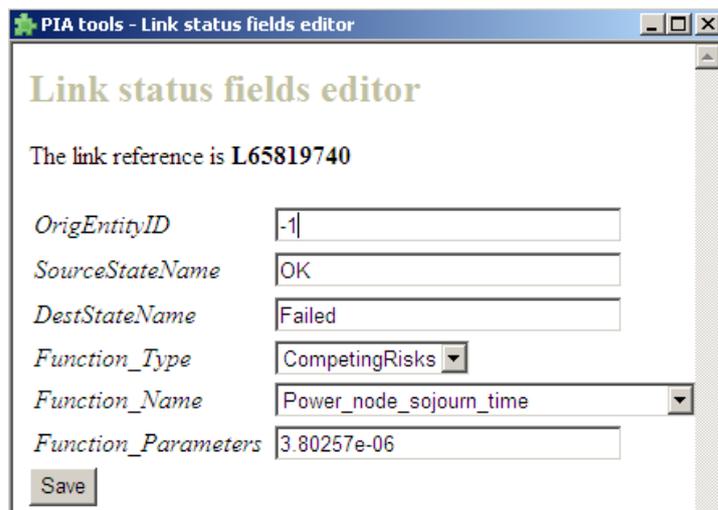
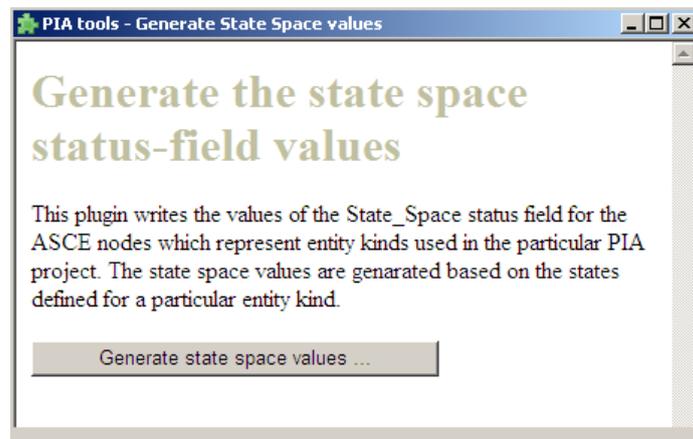


Figure 14: The Link status-field editor plug-in

6. Make sure that the value of the node status-field *State\_Space* for each entity kind node is populated with a comma-separated list of the state names associated to the entity

kind. For example, if there are two states (e.g. *Failed* and *OK*, as in a minimal case) in the state space of an entity kind, enumerate these two state values as *Failed*, *OK* in the status-field value.

- The values of the *State\_Space* status-field have to take exactly the same values as the names of the corresponding states, i.e. the capitalisation is significant. The ASCE plug-in named *PIA – Generate State Space values* (Figure 15) should be used to automate this task. This ASCE plug-in can be started from the Edit menu item of the ASCE tool.
- Please note that the *State\_Space* status-field is meaningful only for the nodes of type *Entity\_kind*.



**Figure 15: The Generate State Space values plug-in**

### C.3 Part 3 – Populate Physical Network (PN) views

#### C.3.1 Populate Intra PN views

For each *Intra\_PN* network defined in the PIA project perform the following:

1. Open up the *Intra\_PN* network. An *Intra\_PN* network displays a topology associated with a particular critical infrastructure service modelled in the PIA project.
  - Make sure that nodes which are *coupling points* have the value of *True* set for the status-field *IsInterCICouplingPoint* (Figure 16). This value must be set to *False* by default for any type of entity.
    - Similarly, of course, make sure that the nodes that are not coupling points have the value of the status-field *IsInterCICouplingPoint* set to *False*.
    - For the nodes belonging to another critical infrastructure service the node type of which is *foreign\_ci\_node*, the value of the status-field *IsInterCICouplingPoint* is always *False*.
  - Make sure that links which are *coupling points* have the value of *True* set for the status-field *IsInterCICouplingPoint* (this value must be set to *False* by default for any type of entity).
    - Similarly, make sure that the links that are not *coupling points* have the value of the status-field *IsInterCICouplingPoint* set to *False*.
    - No links belonging to different, i.e. *foreign*, services are shown in the *Intra\_PN* networks.

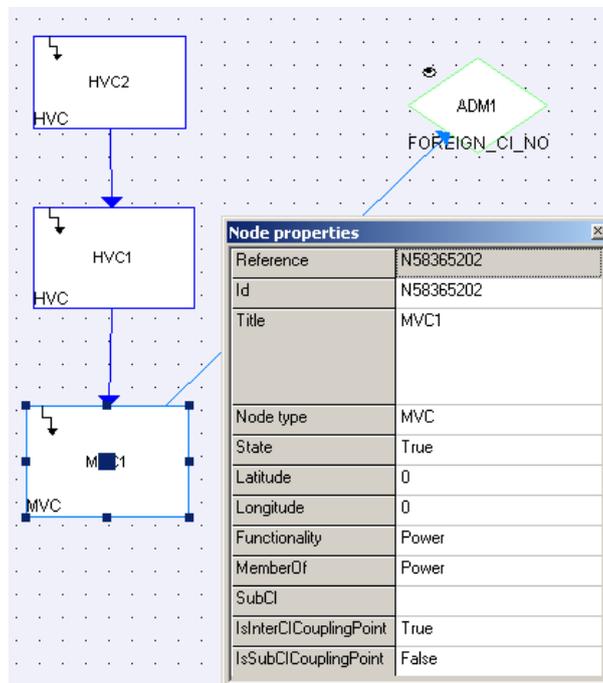


Figure 16: The status-fields for the node MVC1 of an *Intra\_PN* ASCE network

- Using the plug-in *PIA\_PropagateNetworkChange.xml*<sup>4</sup>, the addition of new nodes, as well as deletion of the obsolete ones, is automatically propagated to the project's state transition ASCE network.
  - For this plug-in to work as intended, the set of *Intra\_PN* network schemas defined in the project have to be specified in the plug-in's list of applicable ASCE schemas – the tag *<applicable-schemas>* in the plug-in's header needs to be updated.

### C.3.2 Populate Inter PN view

After all the *Intra\_PN* networks have been populated, open the empty *InterCIPN* network, which was created as one of the project's default nodes in the PIA project network (see Part 1 – Create PIA Project ASCE network). Populate the *InterCIPN* ASCE network using all services that are part of the study. In a minimalist example every service is represented by a single physical entity, which is defined as a coupling point. The following steps are to be performed when populating the *InterCIPN* network:

1. Create an ASCE node of type *Service* for each critical infrastructure service organisation modelled in the PIA project (e.g. if we are modelling a power service we will create a service node in the *InterCIPN* network and give it a name, for example, *Power1*).
2. Define the cross-references between the *InterCIPN* and the corresponding *Intra\_PN* views, i.e. define the value for the *IntraPNNetworkName* node status-field of each node of type *Service* in the *InterCIPN* view.
3. Execute the plug-in *PIA – Inter Service Links* (Figure 17) for each service node in order to display its corresponding *coupling points*<sup>5</sup>. The plug-in can be started from the Edit menu item of the ASCE tool.
4. Connect manually the coupling points between the services, using the link type labelled *Supplies*.

NB: Creation and population of the *InterCIPN* network, as well as the *InteCISA* network, is not essential for execution of the simulation model. This is because the data supplied as input to the simulation model can be extracted from the *Intra\_PN* and *Intra\_SA* networks belonging to the PIA project.

---

<sup>4</sup> The ASCE plug-in *PIA\_PropagateNetworkChange.xml* does not have an associated GUI. Instead, once the ASCE plug-in is enabled, it listens to the events thrown by the ASCE engine, such as *AfterNodeAdd* and *AfterNodeDelete*.

<sup>5</sup> Using the latest version of the PIA ASCE plug-in *PIA – Inter Service Links*, only the nodes from the *Intra\_PN* networks can be shown to be the *coupling points* belonging to a particular service. In a future version of the ASCE plug-in, the functionality will be enhanced so that physical links can be shown as coupling points of a service.

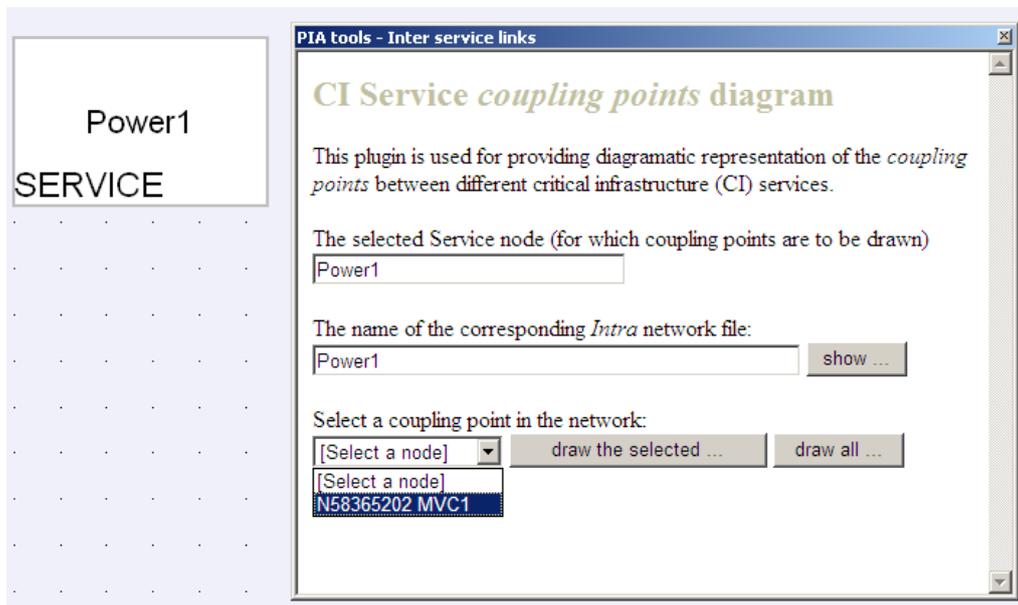


Figure 17: The Inter-service links (coupling points) plug-in

## C.4 Part 4 – Populate Stochastic Association (SA) network views

### C.4.1 Populate Intra SA views

For each *Intra\_SA* network defined in the PIA project perform the following:

1. Open up the *Intra\_SA* network. An *Intra\_SA* network displays a stochastic associations' view for a particular critical infrastructure service modelled in the PIA project.
  - As a starting point, use the topology as defined in the corresponding *Intra\_PN* view. The user can programmatically import the entities, both nodes and links, from the *PN* view when generating the *SA* view – this activity is supported by the PIA ASCE plug-in *PIA – Import PN view entities* (Figure 18). An ASCE node in the *Intra\_SA* view represents either a physical node or a physical link from the corresponding *Intra\_PN* view. For the intended functioning of this plug-in, it is necessary that the meaningful values for *UserTitles* of the *PN* nodes and *PN* links have been specified in the *Intra\_PN* network. If, for example, one of the links in the *Intra\_PN* view has an empty *UserTitle*, this entity will not be imported in the *Intra\_SA* view.
  - Create the *SA* links in the ASCE network. An ASCE link, in an *Intra\_SA* network, represents a *stochastic association* between two physical entities.
    - Parameterise scaling factors: a stochastic association can be unidirectional, represented by the *Intra\_SA* link type *affects*, or bidirectional (reciprocal), represented by the *Intra\_SA* link type *interdependent*. A link of type *affects* displays a single parent-child relationship, and thus only the status-field *Src\_ScalingFactor* is to be populated. A link of type *interdependent* displays a two-way parent-child relationship between a pair of physical entities, and thus both *Src\_ScalingFactor* and *Dst\_ScalingFactor* status-fields are to be populated. The values of the *Src\_ScalingFactor* and *Dst\_ScalingFactor* are set using the PIA ASCE plug-in *PIA tools – Link status fields editor*.
  - Make sure that nodes from the *Intra\_SA* network (please note these nodes represent either physical nodes or physical links defined in the corresponding *Intra\_PN* network) which are *coupling points* have the status-field *IsInterCICouplingPoint*<sup>6</sup> set to *True*. This value is set to *False* by default for any type of entity.
    - Similarly, of course, make sure that the nodes that are not coupling points have the value of the status-field *IsInterCICouplingPoint* set to *False*.
    - For the nodes belonging to another critical infrastructure service, the node type of which is *foreign\_ci\_node*, the value of the status-field *IsInterCICouplingPoint* is always *False*.

---

<sup>6</sup> The attribute, i.e. the *status-field* in ASCE tool terminology, named *IsInterCICouplingPoint* is defined for the ASCE nodes in an *Intra\_PN* networks too. There are, thus, two interpretations of the term *coupling point*: in *Intra\_PN* networks the attribute defines if an entity is physically connected to at least one entity from another service, while in an *Intra\_SA* network the attribute defines if an entity is stochastically associated with at least one entity from another service.

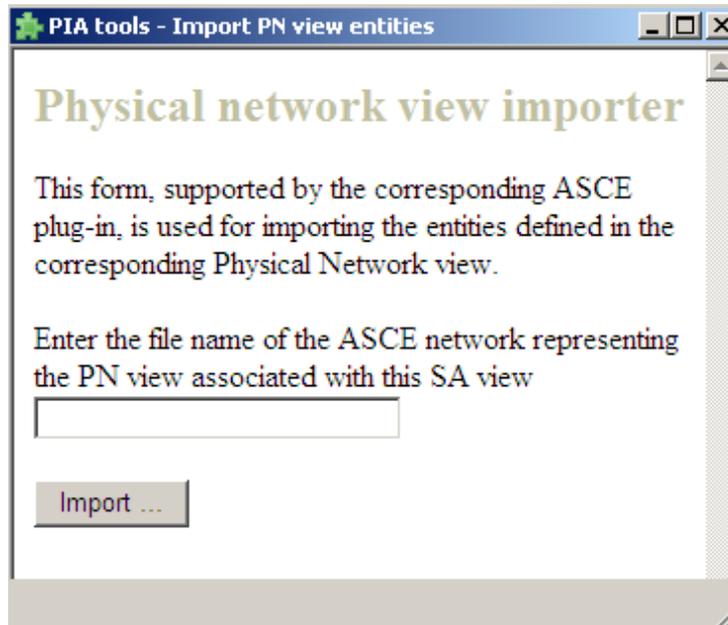


Figure 18: The Physical Network views import plug-in

#### C.4.2 Populate Inter SA view

After all *Intra\_SA* networks have been populated, open the empty *InterCISA* view, which was created as one of the project's default nodes in the PIA project network (see Part 1 – Create PIA Project ASCE network). Then populate the *InterCISA* view with the nodes representing all critical infrastructure services that are part of the PIA project. In a minimalist example, every service is represented by a single SA entity, which is defined as a coupling point. The following steps are to be followed when populating the *InterCISA* network:

1. Create an ASCE node of type *Service* for each critical infrastructure service organisation modelled in the PIA project (e.g. if we are modelling a power service we will create a service node in the *InterCISA* network and give it a name, for example, *PowerISA*).
2. Define cross-reference between the *InterCISA* view and the corresponding *Intra\_SA* network view, i.e. define the value of the *IntraSANetworkName* node status-field for each service in the *InterCISA* view.
3. Execute the plug-in *PIA – Inter Service Links* for each service node to create the corresponding SA *coupling points*. The plug-in can be started from the Edit menu item of the ASCE tool.
4. Connect manually the SA coupling points between the services. Use an appropriate link type for this activity: use the link type labelled *Affects* for the unidirectional relationship between the coupling points, or the link type labelled *Interdependent* for the reciprocal relationships between the coupling points.

NB: Creation and population of the *InterCIPN* network, as well as the *InterCISA* network, are not essential for execution of the simulation model. This is because the data supplied as input to the simulation model can be extracted from the *Intra\_PN* and *Intra\_SA* networks belonging to the PIA project.

### C.5 Part 5 – Configuration of the Simulation Plug-ins

Each PIA project uses a set of *simulation plug-ins*. A simulation plug-in<sup>7</sup> is a standalone piece of code, i.e. a dynamically linked library, which extends the functionality of the PIA simulation model template. The categories of simulation plug-ins are as follows: Initialisation, Deterministic, Trace, Rate and Reward. In each PIA project there must exist one, and only one, Initialisation plug-in, and there are zero to many simulation plug-ins belonging to the other categories.

The ASCE engine plug-ins *PIA – Configure Simulation Plug-ins* and *PIA – Simulation Plug-in Entities Mapping* are used for the configuration of the simulation plug-ins for a particular PIA project. Each PIA Project contains a subfolder named *Simulation\_Plugins\_Configuration* – it is the default location where the configuration files of simulation plug-ins are stored. The steps for populating the SimulationPlugins ASCE network are as follows:

1. Open the SimulationPlugins ASCE network, which was created as one of the project's default nodes in the PIA project ASCE network (see Part 1 – Create PIA Project ASCE network). Each PIA Project has one SimulationPlugins ASCE network located in the project's folder.
2. For each simulation plug-in repeat the following:
  - Create an ASCE node representing the simulation plug-in (e.g. of type Initialisation plug-in).
    - Specify the values for the status-fields *DLL\_Path* and *Data\_File\_Name*. The former determines the absolute path to the dynamically linked library of the simulation plug-in (the *dll* name is specified by the UserTitle of the ASCE node representing the simulation plug-in), while the latter specifies the name of the data file associated with the particular simulation plug-in – every simulation plug-in has a *data/initialisation* file associated with it. This data file is produced by the simulation plug-in developer.
  - Execute the ASCE plug-in *PIA – Configure Simulation Plugins* (Figure 19) by choosing the corresponding option from the ASCE Edit menu item. As a result, the file conventionally named *Plugin Paths and Names.txt* will be created in the project's subfolder *Simulation\_Plugins\_Configuration*. This file contains the list of absolute paths to all the simulation plug-ins used in the particular PIA project, and for each plug-in the absolute path to its data file is specified.
  - The execution order of the simulation plug-ins belonging to the Deterministic category is significant. PIA Designer user specifies the particular ordering of Deterministic plug-ins by using the GUI shown in Figure 20. The GUI is displayed once the PIA Designer user presses the button labelled *Deterministic plugins ordering* from the HTML form of the PIA ASCE plug-in *PIA – Configure Simulation Plug-ins*.

---

<sup>7</sup> The term *plug-in* is used for describing different parts of the PIA toolkit. Beside the simulation plug-ins there are ASCE tool plug-ins, which are standalone pieces of a scripting language code that enhance the functionality of the ASCE engine. ASCE plug-ins have been used extensively for building the functionality of the PIA Designer tool.

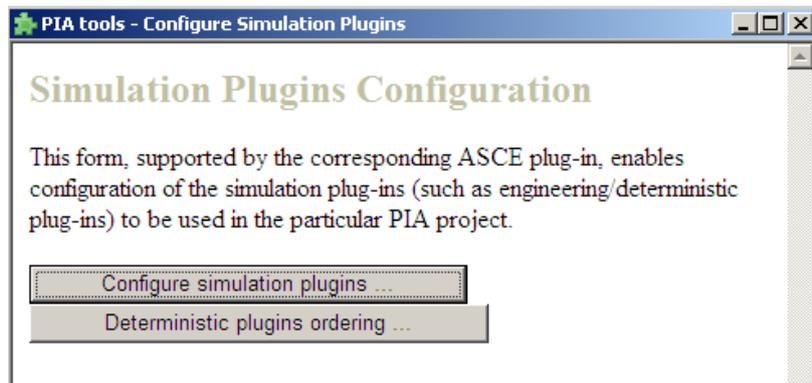


Figure 19: The plug-in used for configuration of the Möbius-based simulation study

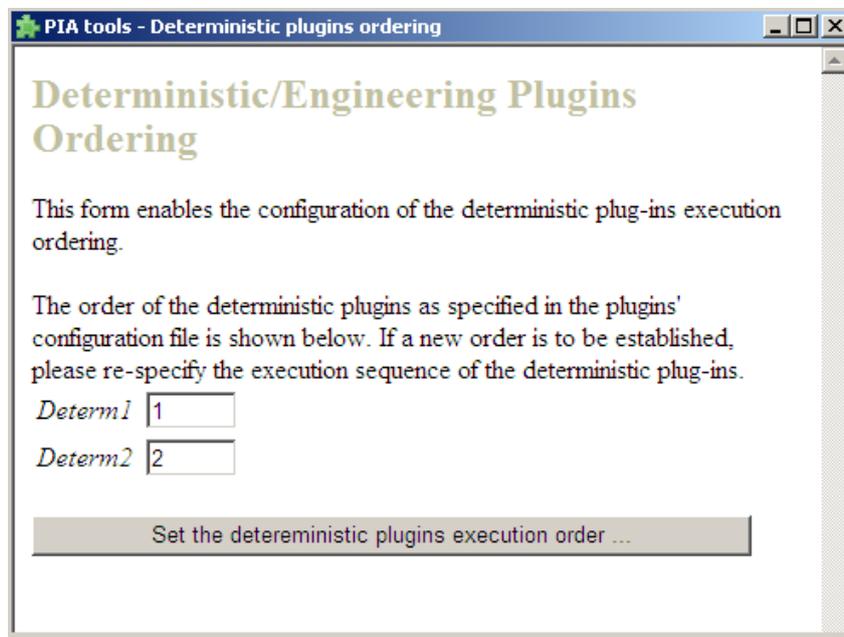


Figure 20: The plug-in used for specifying the particular execution order of Deterministic plug-ins in a PIA project

3. For each simulation plug-in belonging to the Deterministic category, a mapping between the entities used in the plug-in and the entities used in the PIA Project has to be established. The ASCE plug-in named *PIA – Simulation Plugin Entities Mapping* (Figure 21) is used for creation of such a mapping.
  - Specify the value for the status-field *IDs\_File\_Name*. This status-field, which is relevant for the deterministic simulation plug-ins only, indicates the name of the XML file that contains the IDs and names of the entities used by the deterministic plug-in. By convention, the file is stored in the Project's subfolder *Simulation\_Plugins\_Configuration*. Each such file is assumed to have been produced by the developer of the Deterministic plug-in and PIA Designer user places it in the dedicated project folder. This file is used to enumerate the entities of the particular Deterministic plug-in in the GUI of the ASCE plug-in *PIA – Simulation Plugin Entities Mapping*.

- Execute the *PIA – Simulation Plugin Entities Mapping* plug-in by choosing the corresponding option after right-clicking on the ASCE node representing a deterministic plug-in. An HTML form with the entities used in the deterministic plug-in will be loaded. Create a mapping between entities used in the plug-in and the entities belonging to the specific PIA project – this is achieved by the PIA Designer user who pairs a deterministic plug-in entity, displayed in the left column of the HTML form, with the corresponding PIA Project entity, displayed in the right column of the HTML form. In a general case, not all plug-in entities will necessarily be mapped to a PIA project entity.
  - The entity mappings for all deterministic simulation plug-ins are stored in a single file (conventionally named *Plugins ID Mappings.txt*) in the project's subfolder *Simulation\_Plugins\_Configuration*. The file contains the pairs of mapped ID values for each Deterministic plug-in: the first ID value represents the entity's ID as used in the deterministic plug-in, and the second value represents the ID value as used in the PIA project.

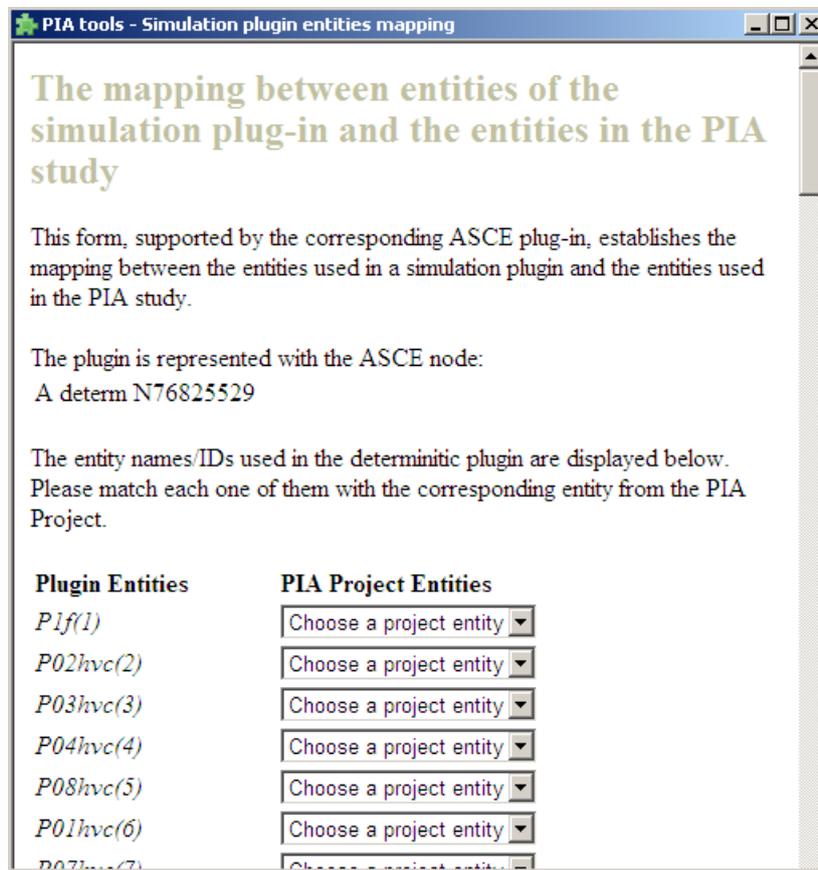


Figure 21: The plug-in for mapping entities defined in a Deterministic plug-in to entities from the PIA project

### C.6 Part 6 – Creation and configuration of the PIA simulation study

1. Open the PIA project ASCE network. Use the ASCE plug-in *PIA – Generate Simulation Study* (Figure 22) to create the Möbius-based simulation study for this PIA project. The simulation study is based on the ASCE networks defined in the PIA project. Before the simulation study is created, the plug-in makes the project's *StateTransitions* network consistent with all the *Intra\_PN* networks defined for the particular project.
  - When the user chooses the option *Generate simulation study* the following is performed:
    - The PIA ASCE plug-in assigns unique UserID values to all physical entities defined in the PIA project, stepping through each *Intra\_PN* network.
    - The consistency between the *StateTransitions* network and the underlying *Intra\_PN* networks is enforced automatically – the *StateTransitions* network is updated to contain the data consistent with the information modelled in the *Intra\_PN* networks.
    - The Möbius-based simulation project is created together with the necessary input files. The Möbius project is based on the PIA template model.
  - It should be noted that the PIA Designer user can perform a check to examine if inconsistencies between the *StateTransitions* and *Intra\_PN* networks exist without choosing the option *Generate simulation study*. This is performed using the *Check* button. Additionally, the user can explicitly force the consistency between the *StateTransitions* network and the project's *Intra\_PN* network by pressing the *Reconcile* button.
  - It is necessary that the *StateTransitions* network contains a set of default parameter values for state transitions of each entity kind.
    - The parameters are *F\_Type*, *F\_Name* and *F\_Params*

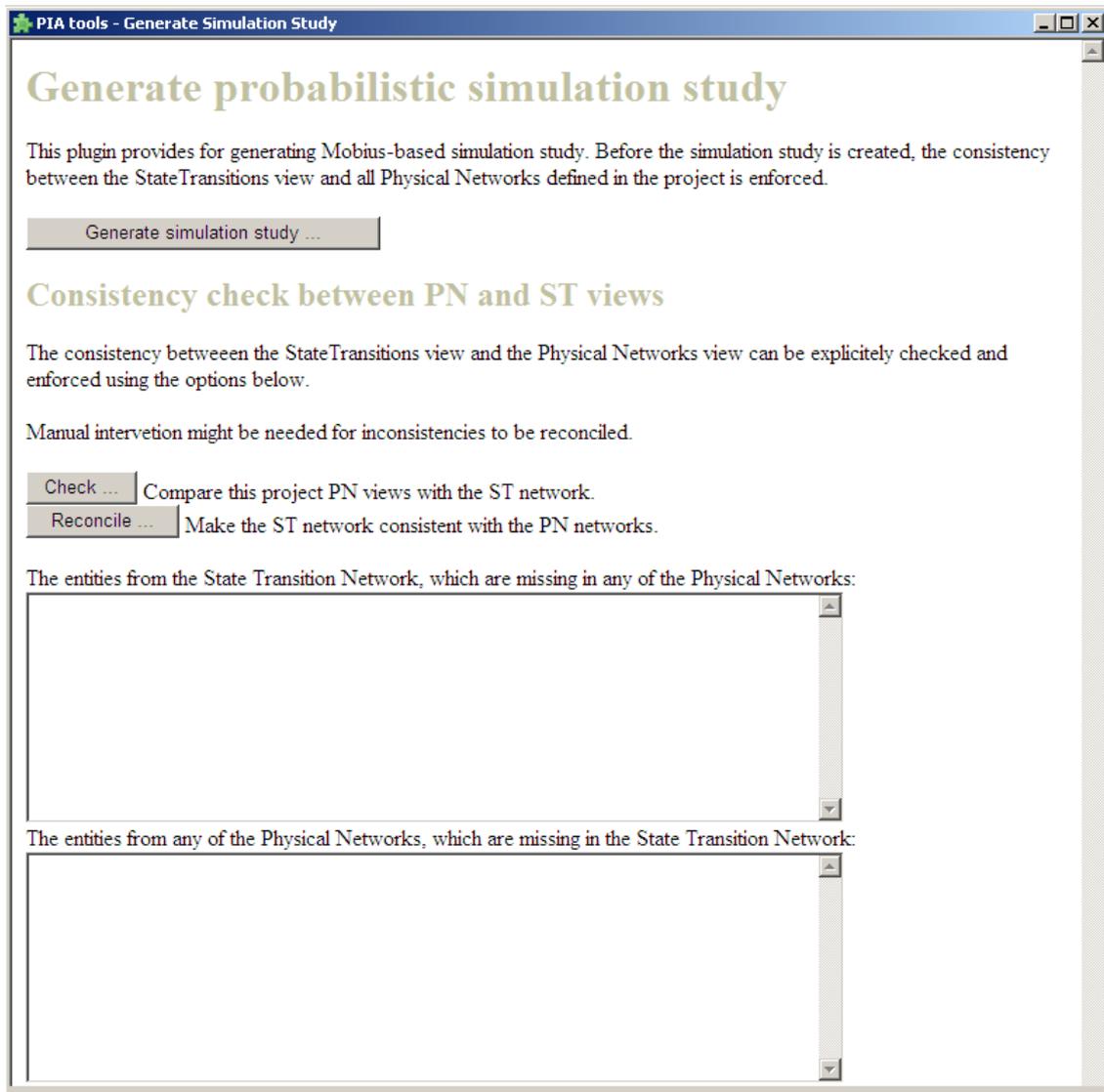


Figure 22: The plug-in for generation of the Möbius-based simulation project