A Logical High-Level Framework for Critical Infrastructure Resilience and Risk Assessment

Sadie Creese	Michael H. Goldsmith	Adedayo O. Adetoye
e-Security Group,
International Digital Laboratory, University of Warwick,
Coventry, CV4 7AL, UK
E-mails: {s.creese, m.h.goldsmith, a.adetoye}@warwick.ac.uk

***Abstract*—Critical Infrastructures (CIs) play crucial roles in modern societies and our reliance on their proper functioning even in the face of accidental failures and deliberate targeted attacks makes their protection of paramount importance. A notable characteristic of CIs is their interdependencies that may exist at many levels and facets of the infrastructure, and which may sometimes lead to unforeseen and unexpected interactions. In particular, the underlying dependencies may induce domino effects in the propagation of failure impacts with devastating consequences and pose serious threats. Thus, technologies that provide new insights and visibility into the infrastructure dependencies, helping stakeholders to understand root causes and to predict the propagation of failure impacts are valuable for the assessment of risks and the engineering of resilience into the infrastructure. This paper presents a logical framework for the high-level modelling of CI dependencies along with analytical tools for automated reasoning about resilience properties of CIs.**

I. Introduction

Modern societies increasingly rely on the effective functioning of CIs to provide essential services such as telecommunications, transport, healthcare, energy, water and so on. It is thus crucial that CIs are resilient to natural and man-made assaults on its components. However, an important factor to note, from the perspective of resilience and risks, is the fact that the components within the CI are typically interconnected and dependent on each other in various ways that make it difficult to fully know the potential impact of failures. Furthermore, when we factor in the typical scale and complexity of modern CIs, determining the resilience of the infrastructure to attacks can be daunting. Even when we do know the dependencies that exist within the CI at a point in time, CIs are by nature emergent, which means that their organic growth and evolution may lead to unforeseen and subtle interdependencies, which can introduce new single-points-of-failure and pose other systemic risks.

In order to cope with some of these issues various simulation-based approaches (see [1]) have been proposed to help understand the dependency dynamics of CIs and help infrastructure owners to engineer resilience into their systems and to help governments to plan for local and national disasters. In most of the cases, simulation-based approaches are *bottom-up*: modelling the operational engineering details of infrastructure elements and connecting together the various CI elements in a simulation environment. The objective is that by perturbing the behaviour of the CI elements and their environments in the simulation, the resultant failure can give insight into the behaviour of the real network. However, a limitation of this approach is that the simulation needs to reflect fairly similar topologies to the real system and the models themselves need to be fairly detailed at the engineering level to yield results with reasonable fidelity. Because of the detailed engineering information required to build models, modellers are usually domain experts, who must have intimate knowledge of the operation of the infrastructure entities and their environments. This is one of the reasons why existing models are typically sector-specific. However, the nature of CIs is that they cross multiple sectors, which may have different models, cultures, and ways of reasoning about resilience issues. Merging results from multiple sectors into a coherent resilience plan is therefore a challenge.

Our analysis framework takes a top-down approach: driving the analysis from high-level abstractions of system interactions that can be easily mapped to business objectives to the necessary level of details to capture the system risk and resilience properties of interest. Because of modularity, models within our framework can be incrementally refined. We specifically note the distinction in this approach because managers do not typically conceptualise resilience and risk at the level of engineering details (which may not necessarily be available across sectoral and ownership boundaries) necessary to build high-fidelity simulation models. When the necessary details for a segment of the CI are not available, as is often the case, our top-down abstraction framework makes it relatively straightforward to substitute those unknown segments with abstract entities that can still be meaningfully reasoned about in terms of their failure impacts. This is enabled by a concept of *externalities*, which describes events or entities outside of an organisation's control, but which impact the organisation in some way and whose detailed internal structure or mode of operation may not be fully known but is only approximated.

Another important factor to consider about CIs is that ownership typically spans both the public and private sectors, and subsystems that rely on each other may be owned by competing organisations. However, certain systemic risks are better dealt with not only from within the individual ownership boundaries, but also at a system-of-systems level through joined-up risk mitigation – whereby organisations cooperate to make the overall system-of-systems more resilient to devastating attacks. Obtaining the necessary information for coordinating risk mitigation within the CI is a major challenge. We have begun to study how we may develop means within our framework whereby, when it is not desired or expedient to directly share information, infrastructure owners can meaningfully share artefacts of their own high-fidelity analysis with other CI owners or government to benefit from joined-up risk mitigation without compromising confidentiality.

It is however natural for CI subsystem stakeholders to ask various resilience and risk questions which pertain to the impact of failures on their own systems. We highlight some of such questions:

1) Where in my infrastructure and what assets are exposed under a given failure or attack scenario and how resilient is my system?
2) How resilient is my system to failures in other organisations that I rely on?
3) How sensitive are my assets and services to fluctuations in the capacities of suppliers?
4) How effective are my risk controls and what level of service continuity can they sustain in the face of attacks of various capabilities and capacities?

These questions address various aspects of a CI – such as vulnerability, resilience, robustness and effectiveness of risk controls. To answer questions such as these, we are developing a logical framework that is capable of high-level reasoning about dependencies within CIs, using this information in the evaluation of resilience properties of the system and the assessment of risks. The framework comprises a domain-specific modelling language for describing the CI and its dynamics, a formalism and calculus for automated reasoning about the interplay between dependencies, failures and controls within a model, and supporting tools which provide analytical *What If* capabilities for deriving infrastructure dependencies, root cause and effects, and impact propagation paths. The insight thus gained from the analytical tools are valuable for decision support for risk assessment, providing new visibility into the resilience properties of the infrastructure that otherwise would have remained difficult to determine due to unknown dependencies and the typical scale and complexity of the CI.

II. Conceptual Model

This paper presents the modelling and analysis framework that is being developed within the project *SATURN* (**S**elf-organising **A**daptive **T**echnology **U**nderlying **R**esilient **N**etworks)[1], a collaboration between UK industry and academia, which is sponsored by the UK Technology Strategy Board and the Engineering and Physical Sciences Research Council. The project *SATURN* seeks to develop methodologies for resilient, self-healing CIs through the following overarching objectives: to understand the nature and communicability of dependencies between organisations; to analyse consequent risks under various threat and vulnerability models, and in particular to estimate their impact across a system of systems; and to enable collaborative risk-mitigation strategies across a CI of heterogeneous organisations, and potentially to support agile and smart self-healing middleware.

To achieve these objectives, we have developed a top-down *SATURN conceptual model* through which we structure the CI ecosystem and its environment. The conceptual model is designed around organisations, which are structured into four strategic layers[2]: the *Enterprise, Information, Technology,* and *Physical* layers [2]. This top-down, high-level layering allows us to, for example, map business objectives at the enterprise layer of an organisation to entities in other layers such as the information and technology layers, and to study how failures originating from within the organisation or externally to it at any of those layers might impact the business objectives. The entities of interest in our model may be tangible or conceptual, and they include *Assets, Controls, Vulnerabilities, Risks*, and *Externalities*, which are mapped to the different layers of the organisation as Fig. 1 illustrates. We have developed a formal ontology which maps common CI assets, controls, vulnerabilities and risks to the *SATURN* layers. The objective is that the formal ontology will serve as a vehicle for information and knowledge sharing about vulnerabilities, dependencies, criticality and risks between CI organisations during collaborative risk mitigation.

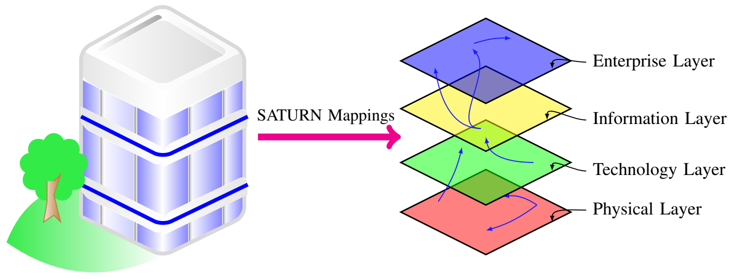

Fig. 1. Conceptualisation of a CI Organisation, mapping its assets, vulnerabilities, controls, and risks to *SATURN* Layers, and showing dependencies between entities.

While the concepts of organisational assets, risk controls, vulnerabilities, and risks are clear; we highlight the notion of *externalities*, which describe events and things, outside the organisation and which typically cannot be directly controlled by the organisation, that impact upon the organisation. We are primarily concerned about externalities that increase the risks to organisations. Externalities may thus be viewed as events or things originating from the organisation's environment which impact upon the organisation. Examples of externalities include natural and man-made disasters, physical and electronic cyber attacks, economic and regulatory environment, and so

[1]www.saturn-project.org.uk

[2]The *SATURN* layers presented in this paper have been chosen because of the types of assets, vulnerabilities and controls that may be found in those layers. We are currently consulting with our community of interest on how the layers map to their current organisational practices and whether refinements and additional layers may be necessary within various sectors of the CI.

on. Fig. 2 demonstrates how an externality at the technology layer of an organisation, in the form of an attack on a technology layer asset, may propagate through dependencies to other assets at the information layer, and ultimately to the enterprise layer. More concretely, this could be an attack on, say, a database system (technology layer asset), which corrupts say employee records or trade secrets (information layer assets), leading to wrong management decisions or wrong application of enterprise policies (enterprise layer assets) at the enterprise layer. The study of risks associated with the propagation of the effect of attacks and other incidences that are facilitated by dependencies within an organisation is one of our key concerns.

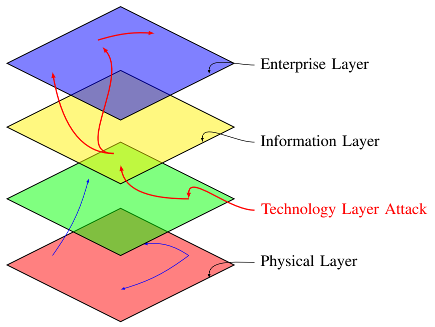

Fig. 2. The propagation of the effect of a Technology Layer attack across and within the layers of an organisation.

The layers in our conceptual model are described as follows. The *enterprise layer* circumscribes those aspects of an organisation that deal with the strategic, operational, legal, regulatory and human facets of all the activities carried out within the organisation. The *information layer* contains all the intangible information assets relevant for the proper functioning of the organisation, such as electronic data, intellectual property rights, digital content generated by the organisation and so on, which the organisation will want to protect from unauthorised assess and modification. The *technology layer* encompasses all the technological assets and mechanisms used in the fulfilment of the strategic business objectives of the organisation. These may include specific pieces of hardware assets such as computer servers, networks, and routers, as well as control mechanisms such as access control systems and electronic data protection mechanisms like cryptographic protection schemes. The *physical layer* circumscribes all the physical infrastructures and services necessary for the proper functioning of the organisation, such as water, power, buildings, land, physical barriers, road access networks and so on, which may be owned, leased, or shared with other organisations and third parties within or external to the CNI ecosystem.

Since each organisation is viewed as possessing an instantiation of the *SATURN* layers in a specific configuration, it is natural to ask how events happening at layers of one organisation within the CI ecosystem might affect those of another. In other words, we wish to know what sort of dependencies may exist between CI organisations, and whether we can systematically identify these, based on sectors of the CI ecosystem, and adaptively activate controls to more effectively mitigate risks. So, conceptually, we view the CI ecosystem as being made up of several "stacks" of *SATURN* layers interconnected by various dependencies that may cross layer boundaries within an organisation onto other stacks and layers in other organisations (see Fig. 3). The partitioning of the organisation into the layers allows us to study, for example, how the exploitation of vulnerabilities at the technology and information layer of an organisation might impact, say, the enterprise layer of another organisation; or how the mutual destruction of the physical layer infrastructures of two geographically co-located organisations might impact a third organisation. The reader should note that our analyses are not limited only to organisation-centric views and that we can reason about impacts, cause and effects, as well as risks, independently of their layer or organisational mappings, even though we have structured the CI ecosystem by organisations and layers. Indeed, in practice the view of infrastructure protection is often from the perspective of owning organisations (a single stack with associated externalities), but it may also be carried out as a partnership between CI organisations and the government (the multi-stack view).

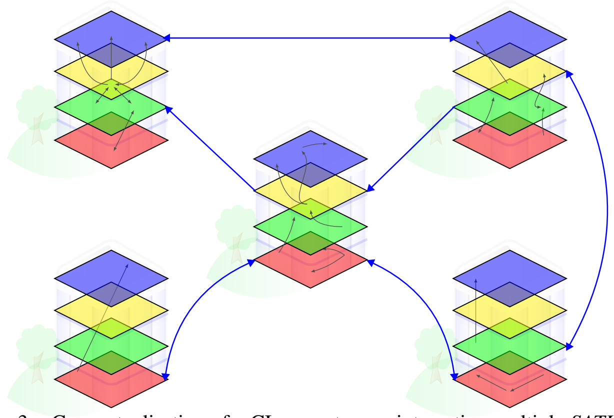

Fig. 3. Conceptualisation of a CI ecosystem as interacting multiple *SATURN* Stacks, showing intra- and inter-organisational dependencies at various layers.

III. Analytical Framework

Any analysis of the resilience properties of CIs must take into account the various dependencies that may exist within the infrastructure. However, the dependencies may be subtle, and the scale, complexity and emergent nature of the infrastructure system will make the manual maintenance of the dependency relations tedious. Since humans are not particularly best suited for this task, our analytical framework does not require the dependency relationships to be built directly, as is typically done in simulation-based approaches [1], rather, our modelling approach derives dependency relationships by logical deductions backed up by formal axioms about dependency relationships [3]. The axioms and rules for deriving dependencies are based on the architectural properties of the dependency relationships, for example, indirect dependencies are derived by applying transitivity rules to direct dependencies, and co-dependencies are derived when the failure of an entity leads to the simultaneous failure of multiple dependent entities [3]. The built-in dependency analysis can be augmented with model-specific rules for reasoning about cause and effects and impact propagation within the model.

We argue that extracting and reasoning about dependencies based on logical axioms and rules is conceptually simpler, less tedious and less error prone because it is automated. It also means that the effect of local changes to the model can be propagated to the whole model in a principled manner, which does not require the modeller to reason about the global impact of local changes, facilitating modularisation. This is very important, especially for large and rapidly evolving CIs where subtle dependencies may exist that are unknown to the managers, and also because the natural evolution of the infrastructure may introduce new, and break existing, dependency relationships so that the maintenance of the dependency model becomes tedious. Furthermore, since the modeller is only required to specify direct service relationships, and direct impacts of failures, the information needed to build the model are easier to obtain from system and risk managers. For example, information about which entities provide what services to others could be readily obtained from system managers, and impacts of externalities on entities within the organisation may be obtained from technical and other risk registers where appropriate.

In order to describe the CI environment within our analytical framework, we have introduced the domain-specific modelling language ***SDML*** (which stands for *SATURN D*ependency *M*odelling *L*anguage) as a front-end to the analytical framework. Currently, ***SDML*** serves as the primary means of interacting with our analytical framework, whereby the user can describe the architecture, properties, dynamics, and configuration of components within the infrastructure network and carry out automated reasoning about the impacts of failure on the network. To ease modelling, the ***SDML*** syntax is deliberately designed to have a natural language feel, but it is backed up by a formal semantics. A preliminary description of ***SDML*** is presented in [3]. We have developed an integrated development environment, based on the popular Eclipse platform, to provide advanced model editing and basic dependency graph visualisation. To make model development even more simple and natural, we are now working on a new 3D visualisation environment to serve as the primary basis for building the CI model through an interactive 3D graphical drag-and-drop environment to set up the CI model and visually carry out *What If* analysis within the environment. Internally, the visual model is translated to ***SDML*** and fed to the analytical engine.

*A. Some **SDML** Constructs*

To get a feel of the use of ***SDML*** to model a CI for the discovery of dependencies, and to reason about the propagation of failures in a *What If* scenario, we shall describe some key ***SDML*** constructs in this section. Because ***SDML*** is very close to natural language, the meaning of each of its constructs is obvious and intuitive from the syntax. However, a formal model pins down the precise semantics of the language.

The language uses the ***isA*** and ***hasProperty*** constructs to build type and object hierarchies, which is familiar from object-oriented systems. ***SDML*** also provides built-in types, mainly to simplify and support common problems and patterns in CI modelling and analysis. For example, the built-in data type *GPS* represents a global positioning system location, which is often used in location-based reasoning such as in studying the coverage and impact of floods and earthquakes. The *GPS* type allows the specification of the *longitude*, *latitude*, and *altitude* properties of a physical entity, and based on this information can calculate distances (through built-in operators) between entities based on their GPS locations. There is also a *GPSList* data structure that stores a list of GPS locations, useful for assets that span large GPS locations, such as a segment of road or railway. The ***hasProperty*** construct is used to specify that an entity has a property in ***SDML***. For example, the ***SDML*** declaration *Physical_Entity* ***hasProperty*** *location : GPSList* specifies that the entity *Physical_Entity* has a property *location*, which has a data type *GPSList*. This means that the a *Physical_Entity* may have a location property described as a list of GPS coordinates. Now suppose that we further declare that *Road* **isA** *Physical_Entity*, then the entity *Road* inherits the location property from the base type *Physical_Entity*. This hierarchical structuring is familiar from object-oriented design paradigm.

To facilitate high-level abstraction of the status of CI components, we have a distinguished built-in property *state*, which describes current state of the component. ***SDML*** has two built-in values that may be assigned to the *state* property of any entity: *Normal* or *Failed*, which can be used to reason abstractly about the operational status of the entity. When an entity, say A, is in the *Failed* state, then the predicate ***fails****(A)* holds. The user may also specify custom states through the ***hasState*** and ***hasFailState*** constructs. This may be useful for domain specific reasoning about CI entities, for example, in road network management, we may want to distinguish situations when the road is lightly congested but can still be used from a severely congested state, when it may be practically unusable. So, we may declare that *Road* ***hasState*** *LightlyCongested; Road* **hasFailState** *HeavilyCongested* to signal that, in addition to the states *Normal* and *Failed*, the entity *Road* may also have states *LightlyCongested* and *HeavilyCongested* and that *HeavilyCongested* is considered a failure state for roads. Hence, the predicate ***fails****(Road)* holds whenever *Road* is in the state *HeavilyCongested* or *Failed*. Note that the notion of *failure* generally represents when an entity is in an undesirable state. However, for events that are by nature undesirable such as flooding and earthquake, failure simply refers to their occurrence, so that **fails**(*Flood*) represents a flooding incident.

To reason about direct service provision between CI entities and direct impact of failures, we have respectively introduced the ***providesService*** and ***impacts*** constructs. So the declaration *Generator* ***providesService*** *Electricity* ***to*** *Machine* specifies the dependency of *Machine* on *Generator* for the supply of *Electricity* as a service. Similarly, the declaration *Flood*

impacts *Road* specifies that the physical externality *Flood* has an impact on *Road*. Without further refinement of the ***impacts*** statement to specify more precisely how flood impacts road, the default interpretation in the analytical tool will be that flooding makes the road fail: **fails**(*Flood*) $\Rightarrow$ **fails**(*Road*). We shall demonstrate shortly how declarations can be used to describe the dynamics of an infrastructure system, for example, we can refine the model to state precisely how *Flood* impacts *Road*, e.g. by causing heavy congestion, in which case the state of the road segment impacted by flood might be changed to *HeavilyCongested* during flooding in a nearby area. The reader should note that we can specify lists of entities in ***SDML*** constructs. For example, the configuration *Generator, National_Grid* ***providesService*** *Electricity* ***to*** *MachineA, MachineB* suggests that *MachineA* and *MachineB* both have *Generator* and *National_Grid* as redundant *Electricity* sources.

To capture the detailed behavioural model of entities and the dynamics of infrastructure systems, we have introduced the notion of *constraints*, which are constructs that can be used to describe the dynamics of the system under state changes. So, for example, to specify that the electricity sources to *MachineA* and *MachineB* are configured redundantly, we might specify a constraint which states that these machines fail only when all the electricity sources have failed:

```
Generator, National_Grid providesService Electricity to
MachineA, MachineB;
BeginConstraint MachineA, MachineB
   (forall s : Electricity  in  ServiceIn. fails(s)) => state = Failed ;
EndConstraint
```

This constraint applies to both *MachineA* and *MachineB*, as declared in the list of entities after the ***BeginConstraint*** header. The body of a constraint is usually a sequence of statements that describe how the states of an entity changes based on its interaction with its environment. Each declaration is of the form *Condition* => *Action*; or, for multiple actions, *Condition* => $\{Action_1; \ldots; Action_n;\}$. The actions typically specify state changes in entities when the condition on the left of => is satisfied. Within the body of the constraint, various sets of entities that interact with the constrained entity are made available. In this example, we used the set **ServiceIn**, which represents the set of services provided *to* the constrained entity. In this case, the set contains the *Electricity* services provided by the *National_Grid* and the *Generator*. Similarly, **ServiceOut** is the set of services provided *by* the constrained entity to others. Whereas, when reasoning about impacts, **ImpactIn** and **ImpactOut** are respectively the sets of entities that directly impact or are impacted by the constrained entity. Elements of these sets can be selected via *filters* and property constraints. In this example, the set **ServiceIn** is filtered by the *Electricity* service type. Hence, ***forall*** *s : Electricity* ***in ServiceIn*** selects only the services that are of type *Electricity*, which happens to be the entire set in this case. So the condition *(**forall** s : Electricity **in** **ServiceIn**. **fails**(s))* holds only when *all* incoming services of type *Electricity* fail - a redundant configuration. Existential quantifiers may also be used. For example, a condition (**exists** s : Electricity **in ServiceIn**. **fails**(s)) holds whenever any incoming *Electricity* service fails. The right hand side of => specifies the state change that happens in *MachineA* or *MachineB*: it assigns the built-in *Failed* value to the built-in *state* property of the entities whenever *all* their incoming electricity supplies fail. We could more succinctly achieve the same behaviour that the example constraint on *MachineA* and *MachineB* specifies by using the ***SDML requires*** construct instead. The statement *MachineA, MachineB* ***requires*** *Electricity* means that the machines require electricity to operate, and will fail if their electricity supply fails. Note that whenever an entity fails in the built-in *Failed* state, the default semantics is that all the services that it provides also fail (transitioned to the *Failed* state).

B. What If *Analyses*

An important capability for dependency discovery, and resilience and risk assessment in CI management is the ability to play *What If* games, to know how the infrastructure behaves under various event combinations. ***SDML*** provides constructs for carrying out automated *What If* analyses against an infrastructure model. In particular, we may query for the impact of failures of any set of entities on a given set of entities. Suppose that an infrastructure owner wishes to verify that a simultaneous failure of assets A and B cannot be induced by a combination of failures of some other entities, say, C, D, E, and F. Then, the *What If* specification to verify this requirement is:

```
BeginWhatIf
   over C, D, E, F assert{  not fails(A)  &&  not fails(B); }
EndWhatIf
```

In *What If* blocks the analytics engine searches the state space for conditions under which the statement(s) in the assert block holds. More precisely, all entities are initially configured to be in non-failed states, and subsets of the entities listed in the ***over*** clause[3] are made to fail to verify the effects of the failures on the validity of the assert block statements. In this example, starting from when all entities are in a non-failed state, non-empty subsets $X \subseteq \{C, D, E, F\}$ are then constructed such that for all $G \in X$, **fails**(G) holds, and the impact of that failure is propagated to the dependent entities within the model to discover combinations that lead to the simultaneous failure of A and B. If such combinations exist, the analysis tool reports back the failure propagation paths that led to the simultaneous failure. Various logical combination of model entity failures can be verified against. For example ***not fails****(A)* || ***not fails****(B)* finds the combination of states that lead the failure of *at least one* of A or B. Recall that **fails**(A) holds whenever A is in the (built-in) *Failed* state or any of the domain-specific fail states of A introduced through the ***hasFailState*** construct.

We may carry out type-based or property-based *What If* analyses to filter out interesting sets of assets for which we

[3]If the ***over*** clause is omitted, the set of all model entities is implied.

want to understand events and configurations under which they are potentially vulnerable. In the following *What If* queries:

```
locs isA GPSList; loc=gps1, gps2, ...;
BeginWhatIf
   over J, K, L, M assert{ exists A : Physical_Entity . not fails(A);}
   assert{(exists A : Physical_Entity .
         GPS.distance(locs, A.location)) < 10. not fails(A);}
EndWhatIf
```

the first assertion is a type-based analysis which looks for the combination of failures of entities J, K, L and M that can lead to the failure of any entity of type *Physical_Entity* within the model. The second query is property-based, and it uses the built-in *GPS* function that calculates shortest distance between two sets of GPS locations. Since the **over** clause is not used in the second ***assert*** statement, the assertion finds the combination of any set of entity failures that can lead to the failure of any *Physical_Entity* within the model that is situated, based on its *location* property, within a 10 *km* radius of the specified GPS locations *locs*. As the example suggests, advanced type and property-based criteria can be used to filter out the assets of interest for the *What If* analysis.

IV. Sample Model and Analysis

To demonstrate the modelling of the impact of CI dependencies and how it might pose risks, we consider an example that is based on the Buncefield oil storage and transfer depot explosion in the United Kingdom on 11 December 2005 [4]. Here we demonstrate in particular how a high-level abstraction of the system can help to quickly identify potential risks posed to target entities by events associated with other infrastructure entities. Because the Buncefield depot stores flammable assets, the potential risk of explosions is quite obvious, but the combination of events leading to the explosion and the resulting impacts on the local environment and economy may not be immediately obvious. Since the analysis tool can perform exhaustive search on the state space of the system, all possible combination of initial events that can lead to the target end states can be accounted for. The identification of such initial events is a key stage during risk analysis, where the risk analyst can then prioritise the events according to their likelihood of occurring.

In this example we have focussed on the impact of fire explosion on the depot and the adjoining areas. To illustrate the impact of co-dependencies due to geographical co-location, we also show, at a very high level, how an explosion at the depot might impact a local organisation. If such an organisation also provides services to other organisation, which may not be necessarily geographically located nearby, then we can study how the impact of the explosion may be propagated downstream by the geographical co-dependencies and the service dependencies. In this example we chose Northgate Information Solutions (*NIS*), looking at the associated problems from the perspective of its customers. *NIS* is a major provider of IT services to the human resources, local government, education and public safety markets, which had its headquarters located within the vicinity of the depot and was impacted at its physical, technology, information and enterprise layers by the explosion. Also, because *NIS* serves many other organisation, it is easy to see how the impact might have been far-reaching. For example, given that the event occurred in the run-up to the Christmas celebrations there was a real economic implications for families of staff at the organisations that *NIS* provided *Payroll* management services to.

According to the detailed review carried out into the causes of the explosion [4], late on Saturday 10 December 2005 a delivery of unleaded petrol from the Thames/Kingsbury pipeline (*TKP1*) started to arrive at Tank 912 (*T912*) within the Buncefield depot. At about 05:30 on 11 December, the safety system (*S912*, which we refer to as a type of *ShutoffSensor* in the model) in place to shut off the supply of petrol to the tank to prevent overfilling failed to operate. Petrol cascaded down the side of the tank and as overfilling continued, the vapour cloud formed by the mixture of petrol and air flowed over the protective wall around the tanks, dispersed and flowed west off site towards the Maylands Industrial Estate. At 06:01 on Sunday 11 December 2005, the first of a series of explosions took place. The main explosion, which appears to have been centred on the Maylands Estate car parks, was massive. The explosions caused a huge fire which engulfed more than 20 large storage tanks over a large part of the Buncefield depot. The fire burned for five days, destroying most of the depot [4].

The high-level model of the infrastructure entities of interest is shown in Fig. 4 and the associated dependency graph is shown in Fig. 5. The model specifies the behaviour of the *ShutoffSensors* as controls to prevent the fuel *Tanks* from overflowing, and how fire, depending on the proximity to tanks during overfilling might impact the tanks. Furthermore we show in the model, how fire depending on its magnitude and proximity might impact the *NIS* infrastructure. In practice this might be two models, one belonging to the risk managers at the Buncefield depot, and the other belonging to the risk managers of *NIS*. These two have to communicate risks in order for *NIS* in particular to benefit from a realistic account of the risk due to fire explosion posed to its organisation and services due to its location. However, we simplify this in the model by assuming perfect knowledge.

As can be seen from Fig. 4 the model specifies that a *ShutoffSensor* attached to a tank ensures that all the pipelines servicing that tank will be switched off as long as the *ShutoffSensor* itself has not failed. The behavioural constraint on *Tank* specify that if a pipeline servicing the tank is in the *On* state, and the tank itself is not full, then the state of the tank changes to *Full*. Clearly, this is a very coarse approximation of the dynamics of filling a tank, but since this behaviour is not relevant for the current analysis this approximation is sufficient. The point is that the modeller can later refine this behaviour modularly if so desired, for example, to model the time it takes for a *Tank* to overflow from empty. The second *Tank* constraint is similar to the first: if the tank is *Full* and one of its supplying pipeline is still on, then the tank transitions to the *Overflowing* state. The third and final constraint for the

```
Physical_Entity hasFailState Destroyed, Exploded;
Physical_Entity hasProperty location : GPSList;
Fire hasFailState Small, Explosion; Fire impacts Physical_Entity;
Tank,Pipeline, ... isA Physical_Entity; Tank  hasState NotFull, Full;
Tank hasFailState Overflowing; T912, T910, T915, ... isA Tank;
TKP1, TKP2, ...  isA Pipeline; S912, S910, S915, ... isA ShutoffSensor
S912 providesService OverflowShutoff to T912;
TKP1 providesService Petrol to T912; TKP1.state = On;
S910 providesService OverflowShutoff to T910;
TKP2 providesService Petrol to T910; TKP2.state = On;...
BeginConstraint ShutoffSensor
   (exists t : Tank in ServiceConsumers. forall p : Pipeline in
    t.ServiceProviders. not fails(this) && t.state==Full)
     => p.state = Off;
EndConstraint
BeginConstraint Tank
   (exists p : Pipeline in ServiceProviders. p.state==On
    && state==NotFull) => state = Full;
   (exists p : Pipeline in ServiceProviders. p.state==On
    && state==Full) => state = Overflowing;
   (exists f : Fire in ImpactIn. state==Overflowing &&
    GPS.distance(location, f.location) < 0.01)
     => { f.state = Explosion; state=Exploded; }
EndConstraint
NIS isA Physical_Entity; NIS.location=...; Fire.location = ...;
NIS providesService Payroll, ... to A, B, C, ...;
BeginConstraint NIS
   (exists f : Fire in ImpactIn. f.state==Explosion &&
    GPS.distance(location, f.location) < 1) => state = Destroyed;
   (forall s in ServiceOut. state==Destroyed => s.state = Failed;
EndConstraint
BeginWhatIf
   assert{ not fails(NIS);}
EndWhatIf
```

Fig. 4. ***SDML*** high-level model of the impact propagation of the Buncefield depot explosion to an adjoining organisation.

Tank type specifies that if there is any fire within 10 *meter* (0.01 *km*) radius of an *Overflowing* tank, then the fire turns into an *Explosion*, and the state of the tank itself is transitioned to *Exploded*.

In order to study the potential impact of failures in the Buncefield depot on an organisation such as the *NIS* that is located within a relevant radius of the depot, we "set fire" to the depot in the model and the *What If* analysis checks whether there can be any combination of events that can lead to the failure of *NIS*. The *NIS* organisation's infrastructure is modelled as providing *Payroll* services to other organisations $A, B, C, \ldots$. To model the impact of fire on *NIS*, the behavioural constraint specified for *NIS* in the model specifies that if there is a fire explosion within $1km$ radius of the *NIS* facility, then the impact is that facility will be destroyed: hence the state is set to *Destroyed*. Furthermore, as the second constraint on *NIS* specifies, the destruction of the infrastructure means that all the services provided from it will fail as a result.

The *What If* block specifies the analysis of interest, which is to find the condition under which *NIS* fails.That is, the events that can transition *NIS* into any of its specified *fail* states. But we could also check the impact of a tank explosion on other infrastructure component within Buncefield and the services that they provide, such as the supply of aviation fuel to Gatwick and Heathrow Airports. The result of the *What If* analysis will return obvious and less-obvious sequence of events that may lead to the desired end-states. In particular, in this example, one of such scenarios is when the overflow sensor *S912* fails and there is a fire within the vicinity of the tank *T912*. Such information can be useful to the managers of the Buncefield depot, although it may not come as a surprise to them. However, one of our objectives is to develop safe ways to share such information between organisations, so that organisations such as *NIS* in this case can adjust its risk exposure, contingency plans and disaster recovery plans accordingly. More generally, the sharing of such information can help CI organisations engage in meaningful joined-up risk mitigation activities.

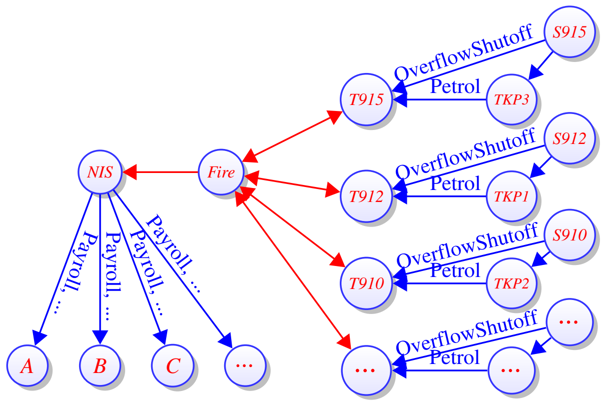

Fig. 5. Dependency graph of the Buncefield depot model.

V. Related Work

We briefly highlight here a few related frameworks for the modelling and analysis of CI systems. The CIMS© framework is an agent based CI modelling and simulation platform. CIMS© was developed to provide a portable and highly visual tools to identify and graphically display interdependency weaknesses and vulnerabilities to the critical portions of the infrastructure or operations [5]. Similarly to our approach, it allows high-level reasoning about CI states. CIMS© is fully-fledged simulation environment which allows network nodes to be connected external simulations or even physical sensor. A related tool from the developers of CIMS©, called CIPRSim, is a modelling and simulation framework that visually portrays the interactions between infrastructure components and allow users to link multiple hazard and specific critical infrastructure sector analysis modules, including physical components, through a distributed environment [6]. The DIESIS Project has developed ontology-driven techniques and tools for the characterisation of CI interdependencies. It uses a knowledge-based system to create abstractions of CI domains and it is designed to support a federated environment for simulating CI models [7]. In our framework, we use a formal ontology to map assets and other entities to the *SATURN* layers, and the ultimate aim is that our ontologies will also serve as a platform for information and knowledge sharing about vulnerabilities and risks between CI managers during joined-up risk mitigation.

VI. Conclusion

We have designed a framework for the high-level modelling of CIs underpinned by a *SATURN* conceptual model. The high-level framework allows us to carry out automated *What If* analyses for the discovery of dependencies and the assessment of resilience and risks. The analytical framework will give new insight and visibility into the dynamics and potential vulnerabilities of the CIs as a useful additional decision support platform for CI managers. The supporting analytical tools are being developed. The current implementation of the analytical engine uses Drools Expert [8] rule-based system[4] for reasoning about cause and effects and states, but we suspect that the rule engine might not scale to system models with extremely large state space and we are therefore planning to use the FDR CSP [9] model checker[5]. A key advantage of our framework is that it allows modular building of the infrastructure, allowing users to model the aspect of the system that they wish to study and to progressively refine the abstractions by introducing more states, and adding new constraints to the infrastructure model while the tool mechanises the discovery of dependencies and vulnerabilities through automated *What If* analyses. Moreover, the ability to reason about infrastructure systems in ***SDML***, a simple natural language-like language makes the system more accessible to a wider audience. We are also in the process of developing an interactive 3D interface for the visual modelling of the CI. A key area of future work is to support confidentiality-preserving sharing of information and knowledge about high-fidelity models developed by different CI organisation to facilitate meaningful joined-up risk mitigation, while minimising confidential information leakage. We argue that this will be valuable from a system-of-systems resilience engineering and risk management perspective, but not at the cost of too much sensitive information leakage, which currently constitutes a major barrier to the sharing of information between CI organisations and government.

[4]www.jboss.org/drools/

[5]http://www.fsel.com/

References

[1] P. Pederson, D. Dudenhoeffer, S. Hartley, and M. Permann, "Critical infrastructure interdependency modeling: A survey of U.S. and international research," Idaho National Laboratory, Idaho Falls, Idaho 83415, Tech. Rep. INL/EXT-06-11464, August 2006.

[2] A. Adetoye, S. Creese, M. Goldsmith, and P. Hopkins, "A modelling approach for interdependency in digital systems-of-systems security - extended abstract," in *5th International Conference On Critical Information Infrastructures Security*. Athens, Greece, LNCS 6712, Springer Verlag, 2010.

[3] A. O. Adetoye, S. Creese, and M. H. Goldsmith, "Analysis of dependencies in critical infrastructures," in *6th International Conference On Critical Information Infrastructures Security*. To appear, 2011.

[4] Lord Newton of Braintree, "The Buncefield Incident 11 December 2005: The final report of the Major Incident Investigation Board. Volume 1," Report of The Buncefield Major Incident Investigation Board, 2008.

[5] D. D. Dudenhoeffer, M. R. Permann, S. Woolsey, R. Timpany, C. Miller, A. McDermott, and M. Manic, "Interdependency modeling and emergency response," in *Proceedings of the 2007 Summer Computer Simulation Conference, SCSC 2007, San Diego, California, USA, July 16-19, 2007*, G. A. Wainer, Ed. Simulation Councils, Inc, 2007, pp. 1230–1237.

[6] S. Walsh, S. Cherry, and L. Roybal, "Critical infrastructure modeling: An approach to characterizing interdependencies of complex networks & control systems," in *Human System Interactions, 2009. HSI '09. 2nd Conference on*, may 2009, pp. 637 –641.

[7] V. Masucci, F. Adinolfi, P. Servillo, G. Dipoppa, and A. Tofani, "Ontology-Based Critical Infrastructure Modeling and Simulation," in *Critical Infrastructure Protection III*, C. Palmer & S. Shenoi, Ed., 2009, pp. 229–+.

[8] M. Proctor, "Relational declarative programming with JBoss drools," in *SYNASC*, V. Negru, T. Jebelean, D. Petcu, and D. Zaharie, Eds. IEEE Computer Society, 2007, p. 5. [Online]. Available: http://dx.doi.org/10.1109/SYNASC.2007.87

[9] M. Goldsmith, "CSP: The Best Concurrent-System Description Language in the World - Probably!" in *Communicating Process Architectures 2004*, I. R. East, D. Duce, M. Green, J. M. R. Martin, and P. H. Welch, Eds., Sep. 2004, pp. 227–232.