Hierarchical, Model-Based Risk Management of Critical Infrastructures

F.Baiardi[+*], C.Telmon[*], D.Sgandurra[*]

(+) Polo G.Marconi La Spezia (*)Dipartimento di Informatica, Università di Pisa, Pisa, Italy

Corresponding Author

F.Baiardi, Dipartimento di Informatica, Università di Pisa, L.go B.Pontecorvo 3 56127, Pisa, Italy, f.baiardi@unipi.it

ABSTRACT: Risk management is a process that includes several steps, from vulnerability analysis to the formulation of a risk mitigation plan that selects countermeasures to be adopted. With reference to an information infrastructure, we present a risk management strategy that considers a sequence of hierarchical models, each describing dependencies among infrastructure components. A dependency exists any time a security related attribute of a component depends upon the attributes of other components. We discuss how this notion supports the formal definition of risk mitigation plan and the evaluation of the infrastructure robustness. A hierarchical relation exists among models that are analyzed because each model increases the level of details of some components in a previous one. Since components and dependencies are modeled through a hypergraph, to increase the model detail level, some hypergraph nodes are replaced by more and more detailed hypergraphs. We show how critical information for the assessment can be automatically deduced from the hypergraph and define conditions that determine cases where a hierarchical decomposition simplifies the assessment. In these cases, the assessment has to analyze the hypergraph that replaces the component rather than applying again all the analyses to a more detailed, and hence larger, hypergraph. We also show how the proposed framework supports the definition of a risk mitigation plan and discuss some indicators of the overall infrastructure robustness. Lastly, the development of tools to support the assessment is discussed.

Keywords: Critical infrastructure, risk assessment, dependency, countermeasure, cost effectiveness, automatic tool

1 INTRODUCTION

Most problems posed by risk management of critical infrastructures arise because of the very large number of infrastructure components and of the interdependencies among both these components and those of distinct critical infrastructures (Alberts & Dorofee 2002, Anderson 2001, Barber & Davey 1992, CORAS 2000). The analysis of interdependencies is simplified if a uniform approach can be applied to components of the same or of distinct infrastructures. While interdependencies are critical, the complexity of the overall analysis, i.e. the number of operations or the amount of work the analysis requires, depends upon the number of components. The very large number of components to model real world systems can be handled through a hierarchical modular strategy where the assessment considers more and more detailed models of the infrastructure components and of their relations. In this way, the assessment reaches the required level of detail through a sequence of models, each describing in more details a component of the previous one. This strategy can simplify and speed up the assessment provided that newly introduced model can be analyzed separately from the previous one and that the results of the corresponding assessment can be integrated into those of the previous ones. As an example, if distinct components of the same, more abstract, model are decomposed and the resulting models can be separately analyzed, then these analyses can be implemented in parallel and this reduces the overall time of the analysis. Hence, it is important to define conditions that determine when a hierarchical approach can be safely applied.

In the following we discuss the application of a hierarchical approach to models based upon the notion of security dependency among components in the case of attacks implemented by intelligent, i.e. goal oriented, threats to achieve one of a predefined set of goals. (Baiardi et al. 2006). A security dependency from a set of source components to a destination one arises any time a security attribute in the destination, such as availability, integrity or confidentiality, depends upon those of the source components. Security dependencies are fundamental to trace all the effects of a vulnerability, i.e. of a defect in a component (NAIC 2004, Amman et al 2002). To this purpose, we follow the cascades due to the dependencies starting from the components that are affected by the various vulnerabilities and analyze their influence on complex attacks. A complex attack is a goal oriented sequence of elementary attacks, where each elementary attack exploits a single vulnerability.

This generalizes cascade failure models (Anderson 2001), because it characterizes a dependency through several security attributes. We believe that the resulting approach is fully general even if only information and communication infrastructure are discussed in the following. In particular, we introduce sufficient conditions that define if the assessment has to be completely repeated after a hierarchical decomposition or only the newly introduced hypergraph is to be considered. These conditions are not related to information and communication infrastructures as they only depend upon the notions of security dependency and of complex attack. We discuss alternative sufficient conditions as well as problems posed by the definition of necessary and sufficient conditions.

This paper is organized as follows. Section 2 discusses the notion of dependency and how an assessment can deduce information of interest for any risk management strategy from a model that describes the infrastructure components and their security dependencies through a hypergraph. In particular, this section describes how risk mitigation plan can be defined and introduces some indicators to evaluate the robustness of an infrastructure and to rank alternative countermeasures. Section 3 describes a hierarchical approach to the assessment and it shows how the assessment can be simplifies by considering a sequence of hypergraphs, each increasing the detail level of the previous one, and by integrating the results of the analysis of the current hypergraph with previous results. This section introduces alternative sufficient *decomposability conditions* to determine those cases where a hierarchical approach can be effectively applied because only the newly introduced hypergraph has to be analyzed. We also discuss the problems posed by the definition of conditions that are both sufficient and necessary. Section 4 briefly discusses the design of tools to support the assessment and, in particular, of those that select the countermeasures to be implemented to reduce the risk at an acceptable level for the infrastructure owner (Baiardi et al 2005, CORAS 2000, Swiler et al. 2001). A simple case study of a hierarchical decomposition is briefly discussed in Section 5.

2 MODEL BASED RISK MANAGEMENT

After describing the abstract modeling of an information and communication infrastructure through labeled hypergraph, we show how this model may be exploited by the various analyses involved in a risk assessment. In particular, we focus on the discovery of complex attacks against the infrastructure as well as on the risk mitigation step that introduces the countermeasure to stop attacks. We show how the choice of countermeasures can be formalized in terms of partial order among sets of countermeasures. We recall that the whole framework described in the following is focused on intelligent attacks, i.e. attacks from intelligent threats trying to achieve some goals. We denote by user any source of attacks. In the most general case, any user has some legal rights on some infrastructure components and some illegal ones. Here and in the following, we assume that the complexity of the assessment evaluates the amount of work the assessment requires and that it is an increasing function of the number of nodes and arcs of the hypergraph that models the infrastructure.

2.1 *The Infrastructure Hypergraph*

The model we propose describes the infrastructure as a set of interdependent components, each consisting of some internal state and of operations on this state. (Dawkins et al 2002, IEC1025 1999, Moore et al 2001). For each component, three security attributes exist:

- confidentiality, *c*. Its control implies the ability of reading the component state;
- integrity, *i*. Its control implies the ability of updating the component state;
- availability, *a*. Its control implies the ability of managing the component, i.e. of determining who can invoke its operations.

Even if other attributes may be introduced, we only consider these three in the following. The model does not detail the inner state or the component operations and it represents user rights in terms of the attributes of the components the user control. As an example, a user U has a right <*Com, c* >, or control the confidentiality of Com, if U can read the inner state of Com. U owns the right <*Com, c* > if she can update the state of Com. Hence, the rights of a user is represented as a set of pairs <*component, attr* > where *attr* $\in \{c, i, a\}$.

A user controls an attribute because of either the operations she can invoke or dependencies from other components. Each dependency is characterized by the set of source components from where it originates, by a destination one and by an attribute for each component. Distinct dependencies correspond to alternative ways of controlling the same attribute or distinct ones of a component. A first example of dependency is the one from the component modeling an encryption key *K* to the component that models D_K, the data encrypted by *K*. The dependency shows that anyone controlling the confidentiality of *K* controls that of D_K as well. Also the availability of D_K may depend upon the integrity or the availability of K because D_K cannot be accessed if K cannot be recovered. If distinct data subsets are encrypted through distinct keys, then each subset is modeled as a component and each key controls one component. Furthermore, the control of a component depends upon that of the confidentiality of a distinct key. Instead, if just one key exists, we may introduce a single component and its confidentiality depends upon the key confidentiality. A second example of a dependency is the one of a web server connected to a corporate network through a network infrastructure including some routers and links. Anyone that controls the availability of the infrastructure components also controls the one of the server because it can deny the access to the server.

Components and their dependencies are described by the infrastructure hypergraph Ih(I), a labeled directed hypergraph that includes one node N(C) for each component C of I and one hyperarc from {n(C_1), ..., n(C_k)} to n(C) for each dependency from {C_1, ..., C_k} to C. As shown in Figure 1, for any hyperarc H of Ih(I), there is one label for each tail of H, to denote the attribute to be controlled, and one for the head of H, to denote the attribute of the component that is controlled.

Consider now S, the set of rights of a user U. Here, and in the following, we do not distinguish legal users from illegal ones and model any user as a potential, intelligent source of goal directed attacks. The rights in S do not fully describe the level of control of U on the infrastructure, i.e. the set of component attributes U controls, because this set also depends upon the dependencies modeled by Ih(I). The set of attributes U controls is equal to TC(S, Ih(I)), the transitive closure of rights in S according to Ih(I). The algorithm to compute TC(S, Ih(I)) selects an element <*C, w*> in S and it marks any tail of any hyperarc of Ih(I) leaving from N(C) and labeled by *w*. After examining all the elements in S, it selects any hyperarc H where any tail of H has already been marked. Assume that H exists, its head is labeled by $g \in \{c, i, a\}$ and the destination node is N(C_H). We mark N(C_H) and any tail leaving from it that is labeled by *g*. Furthermore, if *g=a*, i.e. the availability of C_H is controlled, we mark any tail leaving N(C_H) because U controls any attribute of C and can assign to herself any rights on C. The procedure is iterated till no further element of Ih(I) is marked. <*C, g*> belongs to TC(S, Ih(I)) if N(C) is marked and $g \in \{c, i, a\}$ is the label of the hyperarc used to reach N(C). As an example, in the hypergraph in Figure 1, the closure of the rights of a user that controls the confidentiality of C_7 and the integrity of C_8 results in the control of the availability of C_5, of the confidentiality of C_6 and of the integrity of C_4. The algorithm to compute the closure assumes that a user grants rights only to herself. The more general case is detailed in the following.

Ih(I) is the model of the infrastructure that the assessment uses. As detailed in the following, the model also include further information about the users and the components of Ih(I) as well as about the attacks of the users against the components

2.2 *The Evolution Graph*

A first analysis that is applied to Ih(I) is the one to discover how elementary attack can be composed into complex ones. An elementary attack is any action a threat, a human or natural source of attacks, may execute against a single infrastructure component. A complex attack is implemented by a sequence of elementary attacks to achieve a goal, i.e. to control a set of component attributes.

To show how complex attacks can be deduced from elementary ones, first of all, we assume that the vulnerabilities of each component C are known (Amman et al 2002) together with the elementary attacks each vulnerability enables against C (Swarup et al 2005, Philips & Painton Swiler 1998). Besides the enabling vulnerability, each elementary attack *A* is paired with at least three pieces of information: the target component, the pre and the post conditions of the attack. Pre(*A*), the precondition of *A*, is the set of rights required to execute *A*. Post(*A*), the post condition of *A* is the set of rights the user acquires if *A* is successful.

A complex attack is a sequence of elementary attacks that leads the infrastructure into a state where a user has achieved one of its goals and that results in an impact, i.e. it damages the infrastructure owner. To discover the sequences of elementary attacks of interest for the infrastructure users, we have to discover which sequences may occur and to evaluate the effect of each sequence. To this purpose, first of all we define *an infrastructure security state*, or state for simplicity, as a set of pair <user, user rights>. For each set of users, a state determines the component attributes each user controls. If St is an infrastructure state and U a user of the infrastructure, then R(U,St) denotes the set of rights of U in the state St.

To discover which sequences of elementary attacks the infrastructure users are going to implement and evaluate the global effect of each sequence, we consider that *each elementary attack of a sequence is executed* *in the infrastructure state resulting from the previous attacks of the same sequence*. Furthermore, if the execution of attack *A* by a user U in the state Is_1, results in a transition into the state Is_2, we have that:

1. R(U, Is_1) $\supseteq$ pre(*A*): a user can execute an elementary attack only if she owns any right in the attack precondition
2. R(U, Is_2)) $\supseteq$ post(*A*): after the successful execution of an attack, a user owns any right in the attack postcondition.

To fully define R(U, Is_2)), we take into account that the post condition of an elementary attack does not completely define the set of rights U acquires because this set also depends upon the dependencies described by Ih(I). In particular, the set of rights U owns after executing *A* in Is_1 is equal to the transitive closure of the union of the rights of U in Is_1 and the ones U has acquired because of the success of *A*. This is formally expressed by the relation

$$R(U, Is_2) = TC(R(U, Is_1) \cup post(A), Ih(I))$$

that shows that the transitive closure is a function of Ih(I). In other words, by executing *A,* U acquires not only the rights in post(*A*) but also those that depend upon post(*A*) in the infrastructure I. The latter set is a function of the dependencies among components as described by Ih(I). The set of rights U owns after an attack determines the further attacks U can implement.

In the following, we assume that the goals U wants to achieve through complex attacks are known for any user U. This implies that any user U is paired with:

- Init(U) the set of rights it initially owns, i.e. the rights it legally owns before executing any attack
- Goal(U), a set of sets of rights, where each set defines one goal U wants to achieve through complex attacks.

The evolution graph Eg(I) of the infrastructure I describes all the states produced by the sequences of elementary attacks that the users may implement to achieve their goals. Eg(I) is a direct acyclic graph where

1. each node *n* corresponds to a distinct state Is(*n*) of I,
2. there is an arc from the node n_1 to the node n_2 if a successful elementary attack executed in Is(n_1) fires a transition into the state Is(n_2),
3. a pair <*A, U*> labels any arc where *A* is the attack whose successful execution fires the corresponding state transition and *U* is the user executing *A*.

The *initial* node of Eg(I) represents the infrastructure state where no attack has been implemented and any user U only owns the rights in Init(U), i.e. its initial rights. Several initial states may exist if alternative infrastructure configurations assign distinct initial rights are assigned to each user. After introducing user goals, we denote a node n of Eg(I) and the corresponding infrastructure state Is(n) as *final* if there is at least one user U that has achieved at least one goal, i.e. there is an element of Goal(U) that is included in R(U, s). Both a final state and the corresponding graph node may be paired with *the impact for the owner of the infrastructure when the corresponding users achieve their goals*. In general, the impact depends upon the goals that have been achieved and on the users that have achieved them.

An *infrastructure evolution corresponds to a path in Eg(I) from an initial node to a final one* provided that the final node is the first one of the path where the users achieve their goals. The labels of the arcs in the path define a complex attack against the infrastructure.

An *autonomous evolution due to a user U* is an evolution where all the attacks are implemented by U and, hence, U belongs to any pair that labels an arc along the path. Instead, a *collusion* evolution corresponds to a path where distinct users cooperate by implementing the attacks of the path to enable one of them to reach a goal. The analysis of collusion evolutions is very important for critical infrastructures because they should be able to resist to joint attacks from several, cooperating, users. We say that an evolution interleaves sequences of elementary attacks if distinct users implement distinct subsets of elementary attacks in the evolutions.

Two evolutions are *equivalent* if the corresponding paths in Eg(I) lead to the same node, that is if the same users achieves the same goals, but their paths cross distinct nodes or distinct paths. Two equivalent evolutions are disjoint if each exploits at least one attack that the other does not exploit. Equivalent evolutions define alternative complex attacks, or attack strategies, of a user to achieve the same goal. The strategies may differ because either they implement distinct elementary attacks, if the equivalent evolutions are disjoint, or they execute elementary attacks in a distinct order. To prevent a user from achieving a goal, *all the corresponding equivalent evolutions should be stopped rather than a single one*.

As a first example of an evolution consider a user U that can achieve one of its goals by executing an elementary attack A. However, to execute A, U needs to control a privileged account on a computer node. Hence, at first U should control the confidentiality of a password of an account, then increase its level of privilege through a second elementary attack and then execute A. Besides the initial states and the final one, the corresponding evolution graph has, at least, two further states where U controls, respectively, the confidentiality of a password of a non privileged account and a privileged account. A further example of an evolution is the one that describes a complex attack against a Java Virtual Machine, JVM. In several cases, the first step of the corresponding evolution is an elementary attack to avoid, or confuse, the run time mechanisms for type checking of the JVM.

The notion of evolution is an important difference between our framework and those focused on reliability because it is strongly related to both and-or attack trees (Dawkins et al 2002) and goal oriented planning (Russel & Norving 2003) as it defines user attack plan against the infrastructure. In planning terminology, an infrastructure state corresponds to the current state of the world and the elementary attacks in a plan enable a user to achieve a goal by changing the infrastructure state. Any algorithm to compute the evolutions users may implement can exploit most planning algorithms and the corresponding heuristics to navigate in an intelligent way the search space that includes all the sequences of elementary attacks. As a counterpart, in general, planning algorithms are focused on the definition of the optimum plan or of optimal plans. Instead, a security assessment should discover all evolutions because the definition of a risk mitigation plan requires the knowledge of any evolution. An evolution graph conveys a more detailed information than attack graphs (Jha et al. 2002, Ritchey et al. 2002, Sheyner & Smith 2003, Sheyner 2004) because it distinguishes autonomous evolutions from collusion ones.

To build Eg(I), we can exploit a forward procedure that consider an initial state *is* and any elementary attacks *A* any user can implement in *is*. For each of such attacks, either both a new state of I and a new node of Eg(I) are generated or a previous state and the corresponding node are considered. In the latter case, *A* defines a further complex attack to reach the state. The procedure is repeated till no elementary attack can be applied in any state. After Eg(I) has been built, we can recognize which of its states are final because at least one user has achieved one of its goals. Notice that elementary attacks can be implemented in a final state as well by any user that tries to achieve other goals. An alternative, backward, procedure to build the graph is introduced in Section 4 that describes the tool we have developed to build Eg(I) and compute the evolutions of I. The inputs of this tool are the initial states, the hypergraph Ih(I), the vulnerabilities and the attacks they enable, a set of users, the initial rights and the goals of each user.

2.3 *Concurrency and Cooperation*

When considering several users that implement complex attacks against an infrastructure, two cases arise, concurrency and cooperation among attacks sequences.

Two attack sequences implemented by, respectively, U_1 and U_2 are concurrent if the users implement their sequences simultaneously and the overall sequence interleaves their elementary attacks, but there is no cooperation between the users. To simplify the overall analysis, we neglect these evolutions because they are stopped provided that we stop those they interleave and the latter evolutions have always to be stopped. Hence, evolutions of distinct users can be analyzed in isolation and only those that include labels with the same user can be considered. The cooperation between U_1 and U_2 that is described by a collusion evolution is more challenging because U_1 may grant to U_2 some rights that U_2 cannot achieve in isolation. By granting each other some rights, a set of users can implement complex attacks that no user can implement in isolation, i.e. through her own rights only. Hence, the complex attacks that may occur depend upon the set of the overall rights of the cooperating users rather than upon the rights of a single user.

As described in Sect. 4, the tool we have developed can handle cooperating users provided that the sets of cooperating users are known in advance.

2.4 *Risk Analysis*

An assessment should consider not only the complex attacks against the infrastructure and the corresponding evolutions, but also the overall risk for the owner. In turns, this risk depends upon the impact paired with the final state of an evolution and upon the probability that each complex attack occurs or, from another point, upon the probability of each evolution. In the following we describe how evolutions with a low probability or that result in a low impact are pruned from the evolution graph.

To evaluate the probability of an evolution, first of all we pair each elementary attack *A* with both the complexity of the actions to implement *A* and the resources it requires. This information can also be paired with any arc labeled by A. We use resources in a fairly broad sense that includes both computational resources, know how and abilities. Provided that we know the largest complexity a user U can afford, i.e. the most complex attack U can implement, we can filter out those elementary attacks that U cannot implement and use this information to prune Eg(I). At first, the pruning removes from Eg(I) an arc labeled by < *A*, U> for any attack A that U cannot implement. Then, all the nodes and arcs of Eg(I) that can no longer be reached from an initial state are removed as well.

A further pruning removes from Eg(I) all those nodes corresponding to states that can be reached by a number of steps, i.e. elementary attacks, larger that a predefined threshold. Lastly, we prune any final nodes describing any state S whose impact is too low with respect to either the resources the complex attacks to reach S require or the lowest impact the owner of the infrastructure is interested in avoiding. We suppose that for each user, a threshold is available that corresponds to the lowest impact the user is interested in achieving. After removing some final states, we also remove nodes and arcs that no longer belong to at least one path from an initial state to a final one.

The input of the last pruning strategy is historical information about the frequency of attacks to remove those with a very low frequency. We consider this pruning in a distinct step to take into account that sometimes it cannot be implemented because no historical information has been collected or is available. This case also includes the one where the assessment occurs before deploying the infrastructure.

2.5 *Risk Mitigation*

Risk mitigation is the step of the assessment that selects which elementary attacks should be stopped and, hence, which countermeasures should be implemented.

A countermeasure C(*A*) for an elementary attack *A* may exploit any combination of the followings:

- remove a vulnerability that enables *A*,
- update dependencies among components to prevent users from achieving all the rights in post(*A*),
- update the initial rights of some users,
- increase the resources *A* requires so that a smaller set of users can implement it.

The return of a countermeasure is defined as the damage it avoids. We are interested in static countermeasures that are applied before any attack may occur.

In terms of Eg(I), the adoption of a countermeasure for an elementary attack *A* implies the pruning of any arc labeled by *A*. In turns, this implies that any evolution that executes *A* is stopped because *A* can no longer be successful. Hence, to prevent any user from achieving its goals, the set of countermeasures to be adopted should stop any evolution. This implies that the countermeasures to be adopted define a cut set of Eg(I). Informally, let *Sc* be a set of countermeasures, A(*Sc*) the set of elementary attacks that are stopped, i.e. are no longer successful, because of the adoption of countermeasures in *Sc* and Arc(*Sc*) is the set including all the arcs labeled by the attacks in A(*Sc*). We are interested in those cases where Arc(*Sc*) is a cut set of Eg(I), i.e. no final node of Eg(I) can be reached from the initial ones after removing any arc in Arc(*Sc*). If Arc(*Sc*) is a cut set, then *Sc* is *a complete set of countermeasures*. A *complete set Sc* is *minimal* if none of its proper subsets is complete as well. An alternative definition of minimal introduces the cost of countermeasures so that a complete set of countermeasure *Sc* is minimal if the overall cost of any complete set is not lower than the one of *Sc* (Gordon & Loeb 2002, Schechter & Smith 2003). In the following, we show how a complete risk mitigation plan can be defined together with a planning of the adoption of the various countermeasures.

To define a risk mitigation plan, first of all we notice that cost effective risk mitigation should apply countermeasure in one minimal set only so that distinct minimal sets correspond to alternative risk mitigation plans. Furthermore, if a countermeasure C stops an evolution *e*, then it also stops any evolution *e'* equivalent to *e* provided that e and e' are not disjoint. Hence, to define a risk mitigation plan we can consider equivalent set of evolutions, where the equivalent set of *e* includes any evolution equivalent to and not disjoint from *e*. For the sake of simplicity, in the following, we denote as evolution any representative of its equivalent set.

We propose an approach that defines a risk mitigation plan by composing two steps:

1. define a partial order among subsets of each minimal set,
2. merge the resulting partial orders into a single partial order.

Risk mitigation plans are defined in terms of a partial order because of disjoint evolutions or, from another perspective, of distinct success paths that lead to the same final node of Eg(I). If several disjoint evolutions enable some users to achieve a goal G, stopping just a proper subset of these evolutions cannot prevent these users from achieving G. This implies that two countermeasures (vulnerabilities) are correlated if they stops attacks (are exploited by attacks) in equivalent but disjoint evolutions. The correlation occurs because the adoption of just one countermeasure is useless since *the user can achieve the same goal through an equivalent but disjoint evolution*. Hence, in general, the basic block to define a risk mitigation plan is not *a single countermeasure but a set of countermeasures* that stops not only an evolution *e* but also any evolution that is equivalent but disjoint with respect to *e*. Instead, all equivalent evolutions are stopped by any countermeasure for one evolution because they share any elementary attack. To take correlation into account, first of all we introduce the notion of *minimal subset*. Ms(M) is a minimal subset of a minimal set of countermeasures M if both the following conditions hold:

1. if the adoption of Ms(M) stops an evolution *e*, then it stops any evolution equivalent to *e* as well
2. no proper subset of Ms(M) stops all the evolutions that Ms(M) stops.

If Ms(M) is a minimal subset of M, then it is related to a set *sg* of goals of a user *U* because Ms(M) is the smallest subset of M that can stop all the evolutions enabling *U* to achieve the goals in *sg*. Minimal subsets play a critical role in risk mitigation because the adoption of a set of countermeasures *Sc* is cost effective only if *Sc* is a minimal subset. As a matter of fact, if *Sc* is not minimal then some of its countermeasures are useless because they do not prevent a user to achieve a goal and a smaller, and hence less expensive, set can avoid the same impact. *In other words, for any set of countermeasures Sc that is not a minimal subset, the application of the countermeasures in Sc has the same return of the largest minimal subset in Sc.*

For each minimal set of countermeasure M, we consider a partial order PO according to set inclusion among all the minimal subsets of M. The bottom of PO is the empty set Φ and the top is M. Any maximal chain of PO from Φ to M of length *n* corresponds to a *n-1* steps risk mitigation plan where the i-th step of the plan implements the countermeasures in the difference set between the i-th set and the (i+1)-th one.

Consider a minimal set of countermeasures $\{C_1, C_2, C_3, C_4\}$ and assume that each of C_1 and C_4 stops all the evolutions that result, respectively, in the goals g_1 and g_4. Instead, C_2 and C_3 stop two equivalent and disjoint evolutions resulting in g_{23}. Hence, a risk mitigation plan can prevent a user from achieving g_{23} only if both countermeasures are simultaneously applied. The corresponding minimal subsets are

$$\{C_1\},\ \{C_4\},\ \{C_2, C_3\},\ \{C_1, C_2, C_3\},\ \{C_2, C_3, C_4\},\ \{C_1, C_4\},\ \{C_1, C_2, C_3, C_4\}\ .$$

Figure 2 shows the resulting order. As an example, the maximal chain $\{C_4\}$, $\{C_2, C_3, C_4\}$, $\{C_1, C_2, C_3, C_4\}$ corresponds to the risk mitigation plan that implements at first C_4, then both C_2 and C_3 and, at last, C_1.

To take into account all the minimal sets of countermeasures when defining a risk mitigation plan, we consider the global partial order that merges all the orders resulting from each minimal set. The bottom of the global order is Φ, the empty set, while each minimal set is a maximum. Any maximal chain from Φ to one of the maximum of the order corresponds to an alternative risk mitigation plan that can stop all the evolutions and viceversa, i.e. for each plan there is a chain. Any minimal subset S belongs to a number of maximal chains that depends upon the number of minimal sets that include S.

The choice of the most appropriate plan depends upon not only the maximal chains but also financial parameters such as the amount of the resource to be invested in countermeasures, the distribution in time of these resources and the return of delaying an investment. We believe a framework should define the space of possible plans rather than focus on a unique strategy to choose one maximal chain and, consequently, one plan because these financial parameters fully determine both the optimal plan and the strategy to choose such a plan. Suppose, as an example, that the resources currently available do not support the implementation of all the countermeasures in a minimal set and little information is available on future investment. In this scenario we should privilege those chains resulting in some degrees of freedom in future choices. This corresponds to the adoption of a least commitment plan that is modeled by those chains that cross all the subsets from where any maximum, i.e. any minimal set, can be reached. In this scenario, the early choice of a chain that leads to one minimal set only may be inappropriate, because it freezes the set of countermeasures to be implemented even if little information is available on future investments. Any update to this plan corresponds to the choice of a disjoint chain but, in turns, this implies that some of the countermeasures previously applied are redundant, i.e. useless, in the new plan. Instead, if accurate information about future investments is available, we can choose the optimal chain for the time scheduling of the investment and neglect all the other ones.

Consider again the minimal set $\{C_1, C_2, C_3, C_4\}$ and assume that $\{C_1, C_2, C_5, C_6\}$ is another minimal set where C_5 and C_6 are countermeasures that stop the same evolutions stopped by C_3 and C_4. Here, $\{C_1, C_3, C_4\}$ and $\{C_5, C_6\}$ are some of the minimal subsets that appear in the global poset. Any strategy that applies at first either $\{C_5, C_6\}$ or $\{C_3, C_4\}$ commits itself to one minimal set, instead those that at first apply either $\{C_1\}$ or $\{C_2\}$ can freely choose any of the two minimal sets. The resulting partial order is shown in Figure 3.

The definition of risk mitigation in terms of cut sets is not original because it has been introduced in relation with fault tree (Nasser 1997, Papazoglou 1998, Rauzy 2001, Tang&Dugan 2004, Vesely 1981). Given a fault tree, a cut set includes the failures to be avoided to prevent the top event. A cut set is minimal if none of its proper subsets is a cut set as well. Obviously, when the set of events in a minimal cut set is known, the definition of a minimal set of countermeasures only requires the mapping of each event *e* into the countermeasure that prevent the occurrence of *e*. However, the definition of a cut set for an evolution graph generalizes that of a fault tree because it takes into account all the goals of a threat rather than a single top event. Similar considerations apply to approaches based upon event trees. Because of the simple mapping between fault and event trees from one side and evolution graphs on the other, we can exploit any result on fault and event trees as far as concerns the computation of the probability of a fault or of an event to determine the corresponding probability of an elementary attack or of a sequence of such attacks.

A further interesting application of minimal set of countermeasures introduces the notion of *fragility* of an infrastructure. Along the guidelines described in (Klamt & Gilles 2004, Klamt 2006), we can define Mr(I, C), the *mitigation ratio o*f an infrastructure I with respect to C as the ratio between Msc(I, C), the number of minimal sets of countermeasures of I that include C and Nsc the overall number of minimal sets, i.e.

$$Mr(I, C) = Msc(I, C)/ Nsc$$

As $Mr(I, C) \rightarrow 1$, it increases the probability that risk mitigation plans have to adopt C to stop all the complex attacks, and the corresponding evolutions, against the infrastructure. If Mr(C) is 1, then any risk mitigation plan that wants to stop all the evolutions is forced to adopt C. Under the assumption that each countermeasure prevents the attacks enabled by a distinct vulnerability, the average mitigation ratio for all the countermeasures defines an overall evaluation of the infrastructure mitigation ratio, i.e.

$$Mr(I) = \Sigma_C Mr(I, C)/|C|$$

where |C| is the number of countermeasures and of vulnerabilities. Mr(I) is equal to the average size of a minimal set of countermeasures divided by the number of countermeasures and it is a simple indicator of the percentage of countermeasures to be applied to make the infrastructure safe, i.e. of the vulnerabilities to be removed to prevent any user from reaching any goal. If $Mr(I) \rightarrow 1$, then most vulnerabilities have to be removed to stop all the evolutions. On the other hand, if $Mr(I) \rightarrow 0$ then most of the vulnerabilities do not belong to any minimal set so that all the evolution can be stopped by applying just a few of all available countermeasures. Notice that in the latter case, Mr(I) does not help in determining which are the countermeasures to be applied because it does not convey useful information about the number of countermeasures to be implemented to make the infrastructure safe. This information is conveyed by the sizes of the various minimal sets of countermeasures. However, it can be proved that Mr(I) decreases if we apply a countermeasure C such that $Mr(I, C) > Mr(I)$ and the other way around.

Mr(I, C) may be applied to rank the various countermeasures to be implemented when we are interested in delaying the choice of the risk mitigation plan because a larger value of Mr(I, C) implies that C belongs to a larger number of plans. In the limit case, the implementation of any countermeasure C where $Mr(I, C)=1$ does not constrain the choice of the risk mitigation plan because it belongs to any minimal set and, hence, to any plan. A further application of the ratio arises when a vulnerability V is discovered in between two successive risk assessments. If C(V) is the countermeasure to stop attacks enabled by V, Mr(I, C(V)) evaluates how critical V is and the urgency of applying C(V).

Lastly, it is worth noticing that both the previous mitigation ratios can be restricted to a subset of users. As an example, we can evaluate the mitigation ratio of a countermeasure with respect to attacks of users modeling insider threats or of users that have no legal rights on the infrastructure components and so on.

3 HIERARCHICAL DECOMPOSITION

The previous section has discussed how some steps of the assessment exploit the infrastructure hypergraph Ih(I). However, in a real infrastructure the amount of work, and hence of time, each analysis requires may be very large because of the number of nodes and of arcs of Ih(I). To face this problem, we consider a modular strategy based upon a hierarchical approach that initially models the infrastructure at a very high abstraction level through a simple hypergraph. As the assessment goes on, some or all the components are hierarchically decomposed into a larger number of components and further node and hypergraph are introduced to describe the dependencies among these components.

3.1 *Hierarchical Decomposition*

To describe a hierarchical decomposition suppose, as an example, that to achieve a more accurate representation of the infrastructure a new model is introduced that replaces a component C by a set of components Set(C). As a consequence, one node N(C) of the infrastructure hypergraph Ih(I) is replaced by a hypergraph Dec(C). This requires the definition of a new destination in Dec(C) for each hyperarc whose destination was N(C). Furthermore, any arc leaving from N(C) is replaced by a hyperarc leaving from nodes of Dec(C). A hyperarc may be required because a dependency may have several source components. Let us assume, as an example, that a hyperarc H1 has N(C) and other nodes as its sources. H1 is replaced by a hyperarc H2 such

1. that any source of H1 different from N(C) is also a source of H2
2. N(c) is replaced by some nodes in Dec(C).

This fully exploits the adoption of hyperarc because we can replace one source of any hyperarc by a set of sources without violating the definition of hyperarc. The same property does not hold for arcs of a graph. In other words, *a hierarchical approach is possible only because we have assumed that a dependency is modeled through a hyperarc rather than by an arc*. To evaluate how a hierarchical decomposition influences the assessment, we have to describe how it influences the evolutions of the infrastructure. To this purpose, we notice that a decomposition should assign any vulnerability of C to one in Set(C) and it may introduce further vulnerabilities in the components in Set(C). For each of this vulnerability, a set of elementary attack has to be computed.

The problem posed by a hierarchical decomposition is the repetition of previous analyses, in particular how to avoid the computation of all the evolutions and of the countermeasures, even those unrelated to the decomposition, anytime a node of Ih(I) is decomposed. This implies that, after a hierarchical decomposition, the set of evolutions has to be computed again and that we want to exploit any output of the computation of evolutions before the decomposition to simplify the new computation.

3.2 *Hierarchical Analysis*

To analyze how a hierarchical decomposition of C into Dec(C) influences the evolutions of the infrastructure, we consider the two ways a user may exploit to acquire rights and discuss each case in isolation. The two ways a user *U* has to control the attributes of a component C are

- a dependency having C as its destination,
- an attack against C.

In the first case, *U* acquires the rights because of the transitive closure of her current rights. In terms of Ih(I), there is at least one path in the transitive closure from an arc in In, the arcs having N(C) as their destination, to one in Out, the arcs leaving from N(C) and both arcs are labeled by the same attribute.

The decomposition of N(C) into Dec(C) does not influence the rights of *U*, and hence the evolutions, provided that there is a path across Dec(C) that connects In to Out. Notice that we can check the existence of the path since we know how the original hyperarcs have been affected by the decomposition. If the path does not exist, then the decomposition may prevent some evolutions, i.e. it is equivalent to the adoption of a countermeasure. However, any risk mitigation plan defined taking into account the original hypergraph is still valid because the decomposition does not introduce new evolutions. Instead, the cost effectiveness of the plan may have changed because some evolutions may no longer be possible. In other word, a minimal set of countermeasure may no longer be minimal.

Consider, as an example, the hypergraph in Figure 4a and assume that a transitive closure exploits a path that crosses N(C1). The decomposition of N(C1) shown in Figure 4b does not influence the transitive closure because there is a path from the arc labeled *i* that enters into Dec(N(C1)) to the one with the same label that leaves from Dec(N(C1)). Hence, a user that controls the integrity of C1 can exploit this right to control B integrity even after the decomposition.

Consider now an evolution Ev where a user U implements an elementary attack *A* against C. First of all, we consider how the decomposition influences *A* and the vulnerabilities that enable it. If we know the new pre and post condition of *A*, we can determine whether U rights in the state where U executes *A* suffice to implement the attack and whether the rights acquired through *A* enable U to implement the next elementary attacks in Ev.

Both the examples previously discussed are particular cases of the one where in an evolution a user exploits C to acquire some rights through either paths in Ih(I) that cross N(C) or attacks against C. Hence, starting from Init(C), a set of rights on C, a user may acquire a set Post(C). Hence, in the general case, a decomposition does not influence the overall evolutions if there is an evolution of Dec(C) starting from the state where U owns Init(C) and that ends in the one where it owns Post(C), i.e. Post(C) becomes the goal of U as far as concerns Dec(C). An evolution of Dec(C) is one that includes attacks to components in Dec(C) only and where the transitive closure is constrained to the same components. In the following, an evolution that involves only the components introduced by a decomposition will be denoted as a low level evolution while any evolution of the infrastructure before the decomposition will be denoted as a high level one. Since several decompositions may be applied during an assessment, the same evolution can be a low level one for some assessment and high level one for some other, distinct, assessments. Distinct sets Init(C) have to be used because the rights of some users on C depend upon the high level evolutions that are considered. However, since the evolution graph of the infrastructure has been computed before decomposing C, we know any right on C any user can achieve because of a high level evolution. This set is used as the initial one of a low level evolution. As an example, in the hypergraph in Figure 4, we are only interested in knowing whether a user that owns a right on C_1 can achieve also the right on C_5 required to continue the sequence of attacks of the corresponding evolution.

The previous condition can be rephrased by saying that any further right a user can acquire because of decomposition should not enable new high-level evolutions of the overall infrastructure. This implies that the decomposition should not introduce new complex attacks because it should not enable a user U to achieve a goal that U cannot achieve before the decomposition. In other words, the decomposition may transform a right on C into one on a component in Dec(C) but this right, together with current vulnerabilities, should not introduce further infrastructure evolutions. This constraint is fundamental to guarantee that only the low evolutions that involve the newly introduced components have to be computed and analyzed. In the following, this constraint will be denoted as the *decomposability* one. *If the decomposability constrain is violated, the new model has to be analyzed from scratch because the results of previous analyses do not longer hold.* Notice that if several hierarchical decompositions of distinct nodes of the same hypergraph satisfy the constraint, then they can be simultaneously applied to decompose the corresponding components and the new components can be analyzed in parallel to speed up the assessment. A simple, sufficient, condition for the decomposability constrain is the following one.

Decomposability Condition 1: No low level evolution results in new rights for any user.

This condition implies that the rights any user U acquires because any low level evolution are the same one U can achieve in some state of an high level one before the decomposition. This condition is very simple but rather severe because we are not interested in the further rights that U may acquire but in the further goals U can achieve because of these rights. The following is a more general condition:

Decomposability Condition 2: A low level evolution may result in new rights for a user only if these rights do not enable any elementary attack of a high level evolution.

This condition allows the set of rights of U resulting from a low level evolution to include even new rights provided that these rights do not belong to pre(A) for any elementary attack A against the infrastructure that could have been implemented before the decomposition. This is more general than the previous one but it is always a sufficient but not necessary one.

An even more general condition can be defined by taking into account that a user cannot implement some attacks because of the resources the attack requires. Also this condition is a sufficient one.

Decomposability Condition 3: A low level evolution may result in new rights for a user only if these rights do not increase the number of elementary attacks of high level evolutions the user can implement.

An important case is the one where any of the previous condition is satisfied because a countermeasure has been introduced. In this case, some low evolutions violate any conditions but a set of countermeasure is introduced to stop all these evolution. Hence, the results of the previous assessment are still valid provided that the countermeasures in the set are implemented. This is resumed in the following condition.

Decomposability Condition 4: Further countermeasures are introduced to stop any low level evolution that violates the previous conditions.

It is worth stressing that the resulting set of countermeasure may not be the minimal one because, as an example, the rights the user acquires do not result in new evolutions.

The definition of a sufficient and necessary condition is rather complex because it is related both to the low level evolutions that the decomposition introduces and to the user goal. In other words, the decomposability condition depends upon the relations between the original hypergraph and the transformed one as well as upon the property of the infrastructure users. Hence, a hierarchical decomposition enables some evolutions for some users that depend not only upon the goals of these users but also upon the vulnerabilities of, and the attacks against, components *not involved in the decomposition*.

3.2.1 *Example*

Consider the hypergraph shown in Figure 5.a, where x and y belong to *{i, c}* and assume that T is decomposed as shown in Figure 5.b. Furthermore, assume there is a vulnerability of C_2 that makes it possible to implement an attack to control the integrity of C_2 and that the precondition of this attack is the control of attribute x of C_2. Hence, both before and after the decomposition, the control of x of T results in the control of y of W. After the decomposition, this requires the implementation of an attack against C_2, but this attack does not increase the rights of the attacker. Consider now the decomposition in Figure 5.c and suppose that a vulnerability of C_3 enables an elementary attack A resulting in the control of the confidentiality of C_3 and that the precondition of A is the empty set. This implies that any user that can implement A can control the confidentiality of C_3 and this results in the control of x of W. As a consequence, the decomposition may have increased the rights of the attacker and this violates Decomposability Condition 1. Obviously, it may happen that the control of x of W does not enable further elementary attacks or, at least, that no user achieves one of its goals through these attacks, but this depends upon both the overall structure of Ih(I) and the user goals.

After defining the decomposability conditions, we notice that the notion of hierarchical decomposition of a component can be generalized to decompositions that replace a subset of components S by a larger subset S' provided that the decomposition does not introduce dependencies among nodes outside S and nodes in S'. In terms of Ih(I), a decomposition replaces a subset S of node of Ih(I) by a subset Dec(S) with a cardinality larger than the one of S provided that for each arc between a node in S' and a node n in Ih(I) outside S', there is one and just one arc between a node in S and n. Figure 6 shows a hierarchical decomposition of a subgraph that cannot be implemented by hierarchically decomposing single nodes of Ih(I).

3.3 *Checking the decomposability constraint*

No significant change has to be introduced into the assessment, or into the tools that support it, to check whether the decomposability constraint is satisfied. In fact, the set of rights of each user is computed independently of decomposition, because this set contributes to the definition of the infrastructure state that, in turns, underlies the one of infrastructure evolution. Hence, after decomposing a subgraph, we can check the constrain simply by analyzing infrastructure states to discover if in some state a user owns some rights on the components not involved in the decomposition it did not own before the decomposition. Since during an evolution, the size of a set of user rights increases in a monotonic way, it suffices to compare the final rights of the user after a low level evolution, i.e. one involving the components introduced by the decomposition, against those the user owns in a final state. A violation implies that some of the analyses of the assessment should be implemented again.

3.4 *Decomposition and Risk Mitigation*

The previous considerations about the evolutions that only involve the components introduced by the decomposition and about the adoption of countermeasures to stop low level evolutions suggest how risk mitigation can be handled.

Suppose, as an example, that a countermeasure has been adopted that stops an elementary attack against a component C. If C has been hierarchically decomposed, this implies that the low level evolutions involving components in Dec(C) should be stopped. This can be implemented either by a countermeasure that involves some of the components in Dec(C) or by preventing the user from achieving some of the rights required to implement the low level evolution of interest. In both cases, the strategies to select the countermeasure to be adopted may be the same adopted for high-level evolutions.

4 **PROGRAMMING TOOLS**

We briefly describe the tools we have developed to support the assessment. Among them, the most important ones are the evolution graph builder, the graph pruner and the countermeasure selector. Each of these tools will be briefly described in the following.

4.1 *Evolution Graph Builder*

Starting from some proper databases, this tool computes in several steps Eg(I) for the infrastructure I. Eg(I) is the input of the other tools.

The first database the tool exploits includes information on infrastructure components and on dependencies among these components. Starting from this information, at first the tool builds Ih(I) that is an input of the second step of the tool together with two further databases. The first database is the user, or threat database, that includes information about each user of interest. For each user, the database records the initial rights, the goals and the resources the user can exploit to implement her attack. The second database is the elementary attack one. For each elementary attack, this database records the enabling vulnerabilities, the resource it requires, the pre and the post conditions. By merging the information in these two databases, the tool can deduce the attacks each user can implement to reach its goals. Starting from this information and from Ih(I), the second step of the tool builds the evolution graph for each user. A first, forward, solution for building the graph has already been described. An alternative strategy is the backward one that analyses each user goal and finds the elementary attacks the user can implement to achieve this goal. Then, the preconditions of these attacks become the goals to be achieved. The overall procedure can be easily programmed in a logic programming framework. An advantage of a logic-programming environment is the primitive and efficient support of backtracking that is fundamental both to explore all the alternative way of composing elementary attacks to discover any complex attack that enables a user to reach its goal. The tool can handle cooperating attacks, i.e. collusion among users, in an approximated but conservative way. For each set CS of users that can collude, a further user U(CS) is introduced whose initial rights include any initial right of any user in CS and that can access any resources or information any user in CS can access. This is a worst-case assumption with respect to the case where a user grants one of its rights to another one, because now U(CS) may also acquire rights no user in CS owns.

After building the evolution graphs for the various users, the last step merges the graphs into Eg(I) that describes all the evolutions due by any user.

4.2 *Graph Pruner*

This tool removes nodes and arch from Eg(I) according to the strategies described in Section 2.4.

The pruning is implemented in two steps. The input of the first steps is a set of threshold that define, respectively, impacts too low to be of interest for a user, attack too complex to be executed and the largest number of elementary attacks in an evolution a user implements. After removing some arcs and nodes from the graph, the tool also removes nodes that cannot be reached from an initial state.

The second step removes all the elementary attacks that in the past have occurred with a frequency lower than a user specified threshold. The input of this step includes any available historical information about attacks against the infrastructure.

A two-step approach has been adopted to distinguish the attacks and evolutions that are removed because of available statistics on the user behavior from those that are removed because they have a very low probability of occurring due either to a very large complexity or a low impact. It also worth noticing that the overall complexity of the assessment is minimized provided that the tools exploit as soon as possible any available information about attacks complexity and impact. In our case, this implies that the information should be exploited by the graph builder. However, we believe that the decomposition of graph building and pruning into two tools mirrors in a more accurate way the two distinct kinds of information in the input of the tools. As a matter of fact, the builder mostly exploits long-term information about the infrastructure and the users, such as the dependencies and the user goals. Instead, the pruner mostly exploits short-term information, such as the impact of a complex attack or the complexity of the elementary attacks a user can implement. The set of attacks that are pruned may change very quickly, as an example because some tools that automatically implement an elementary attack become available. The adopted decomposition enables the analyst to evaluate alternative scenarios, each corresponding to distinct short-term values.

4.3 *Countermeasure Selector*

This tool selects, among available ones, the countermeasures to be applied to stop all the evolutions not pruned by the previous tool. It assumes there is at least one countermeasure for each elementary attack and that the cost of this countermeasure is known.

There are two algorithms the tool can apply, an approximated and an exact one. The approximated algorithm computes a cut set of Eg(I) by neglecting arc labels. As a consequence, the algorithm complexity is fairly low but it returns an upper bound on the investment to stop all evolutions because it neglects the fact that countermeasure for an elementary attack cuts all the arcs with the corresponding label. Hence, it does not take into account that stopping an attack shared among several evolutions stops all these evolutions. This implies that the algorithm returns a complete set of countermeasures but there is no guarantee that the set is a minimal one. However, if the overall cost of the set of countermeasures is acceptable for the owner, the exact algorithm need not be applied.

To take into account that the adoption of a countermeasure for an elementary attack removes all the arcs with the corresponding label, the exact algorithm computes a minimal complete set by solving an integer programming problem that has one constrain for each arc of the graph and a 0-1 variable for each attack and for each node of Eg(I). Currently, both the approximated version and the exact one exploit a standard solver for linear and integer programming problems.

5 AN EXAMPLE

As an example of a hierarchical decomposition consider an infrastructure where a set of user can connect to some server nodes through a personal device, i.e. an hand held device, that records information to authenticate the corresponding user and that runs some applications to connect with the server nodes. The server nodes receive the data from the user, implement a preliminary computation on this input and transmit the results to database nodes that store it together with further information received from other sources such as a cluster of workstations that applies some analysis to the data in the database. This fairly general and abstract model can describe a subset of SCADA infrastructure that control a water distribution or a pipeline system as well as an infrastructure that collect data from point of sale devices.

At this very abstract level, the assessment may describe each element of the infrastructure, i.e. a hand held device, a server node or a database one, through a few components that represent the main programs of the elements and data that the element manages. A further component may be introduced to model the interconnection structure. The dependencies among the components can be analyzed together with their vulnerability to discover the complex attacks against the infrastructure and the evolutions that enable the users to achieve some of their goals. Consider, as an example, an attack that results in the control of the confidentiality of components that describe the personal device. Because of dependencies, this attack may also result in the ability of authenticating to the server nodes and hence of accessing information in the database. A minimal set of countermeasure can be computed together with decomposition into non redundant subset for the abstract model. Given the very low number of components, the computation of a minimal set of countermeasures is rather efficient. Hence, an assessment can be applied in a cost effective way during the design of the overall infrastructure to determine a set of cost effective countermeasures. Obviously, the larger the number of vulnerabilities that the model describes, the larger the number of elementary attacks the user can implement. In turn, when and if a hierarchical decomposition is applied, the larger the probability that one sufficient decomposability constrain is satisfied because a good amount of information is considered even before decomposing the components. To exemplify this strategy, suppose that the assessment initially considers the model described by the hypergraph in Fig. 7.a that relates the information to authenticate a user to the infrastructure and to access database information. The model shows that to access the database requires some information to authenticate to the hand held device and some further information outside the device. Assume that a component vulnerability enables an attack *ais* against AuthInfS, the information to authenticate to the server. Hence, any user that controls the confidentiality of AuthInfH, the information to authenticate to the handheld device, can implement *ais* and then access and update information in the database. If this is the user goal, a one step evolution that implement *ais* suffices. An alternative evolution is to be exploited if the user does not control the confidentiality of AuthInfH. In this case, a further attack, *aih* against AuthInfH is required. Hence, now an evolution composes two elementary attacks, *ais* and *aih*, since they may be executed in any orders, we have two equivalent evolutions that differs only in the order they execute the attacks. Sine the evolutions that are not disjoint, one countermeasure can stop both of them.

Assume now that the component that models the HandHeld Device in Fig. 7.a is decomposed as shown in Fig. 7.b. The decomposition introduces two components that model, respectively, the data and the program of the device and a further one to model the physical memory of the device. The hypergraph resulting from the decomposition show that an attack *amem* that results in the control of integrity of the physical memory enable the user to control the confidentiality of data in the device. This implies that the execution of *amem* and the control of the confidentiality of AuthInfS, the information to authenticate to the server, results in the ability of reading and updating information in the database. Hence, a low evolution that implements *amem* enables a user that controls the confidentiality of AuthInfS but not that of AuthInfH to read and update database. This also holds for a user that can implement *amem* but not *ais*. Hence, the low level evolution increases the rights of a user that can implement *amem* but not *ais*. If such a user exists then the sufficient condition is violated. Obviously, this may be uninfluential if the access to the database is not a goal of the user or if does not enable the user to achieve its goals. If, instead, the ability of reading and updating the database is important, an encryption algorithm can be introduced to protect the data in the device memory so that an access to the physical memory does not suffice to implement the low level evolution that violates the sufficient condition.

Encryption is a way to satisfy Decomposability Condition 4 because we have adopted a countermeasure to stop some low level evolutions. In general, the choice of exploiting this condition or of computing the set of countermeasures by neglecting the previous results and by considering the more detailed model also depend upon the available budget since the most convenient but more complex approach may be adopted only when the current budget forbids the other one.

A further decomposition may be applied to the component that describes the database node. As an example this decomposition can model in more detail the interconnection structure by decomposing the node into several components that describe, respectively, the information system that stores and manages information and the network interconnection to access this system. The dependencies among components of the network interconnection are strongly related to the topology of the interconnection structure. As an example, by controlling a security attribute of a link L we may also controls some attributes of further links or routers connected to L. The decomposition of the database node can also be applied after the one previously described. To reach the abstraction level of interest, further decompositions that may be applied to the server nodes or to one of the components previously decomposed.

6 CONCLUSION

We have discussed a model based approach to the risk assessment of a critical infrastructure, where the model is focused on a hypergraph that describes the security dependency among components. Starting from this hypergaph, we can compute a graph whose paths describe the complex attack strategies that distinct, intelligent, threat can implement. Even if the case of information and communication infrastructure has been considered, the proposed approach is fully general.

To define in a formal way the various steps of the assessment and in particular the one that determines the optimal risk mitigation plan, we have introduced a mathematical framework based upon minimal sets of countermeasure and a partial order among subsets of countermeasures.

To simplify and speed up the assessment, we have proposed the adoption a hierarchical approach where each component may be decomposed into a number of simpler components. This is modeled by replacing a node of the hypergraph by a further hypergraph. We have introduced alternative conditions that define when the results of the assessment before the decomposition can be preserved and integrated with those of analysis focused on the newly introduced hypergraph and on the dependencies it describes. No necessary and sufficient condition has been defined because of the complexity of integrating the information about the new components and the new dependencies with those that describe the user goals.

Various programming tools have been introduced to support or implement the various analyses involved in the overall assessment.

Future developments of our work concern the evaluation of the effectiveness of the tools we have developed and further investigation of the conditions concerning sufficient conditions for the decomposability constraint.

7 REFERENCES

Alberts C., Dorofee A., *Managing Information Security Risks*. Addison-Wesley, 2002.

Anderson R.J., Security Engineering *A Guide to Building Dependable Distributed Systems*. John Wiley, 2001.

Ammann P., et al., Scalable, Graph-based Network Vulnerability Analysis, *9th ACM Conf. on Computer and Comm. Security*, Nov. 2002, Washington, DC, USA

Apostolakis, George E. and Lemon, Douglas M., A Screening Methodology for the Identification and Ranking of Infrastructure Vulnerabilities Due to Terrorism, *Risk Analysis*, Vol. 25, No. 2, pp. 361-376, April 2005

Baiardi F., et al. Constrained Automata: a Formal Tool for ICT Risk Assessment, *NATO Adv. Research Workshop on Information Security and Assurance*, Morocco, June 2005

Baiardi F., Pioli M., Suin S., Telmon C., Assessing the Risk of an Information Infrastructure through Security Dependencies, *First Workshop on Critical Information Infrastructure Security*, Samo, August 2006,

Barber B., Davey J., The use of the CCTA risk analysis and management methodology CRAMM. Proc. *MEDINFO92*, North Holland, pp.1589-1593, 1992.

Bertoli P., Cimatti A., Pistore M., Roveti M., and Traverso P.. MBP: a model based planner. *Joint Conf. on Artificial Intelligence, Workshop on Planning under Uncertainty,* 2001.

Boutilier C., Brafman R., Partial-order planning with concurrent interacting actions. *Journal of Artificial Intelligence Research*, 14:105-136, 2001.

Braynov S., Jadiwala M., Representation and analysis of coordinated attacks, *Proc. of the 2003 ACM Workshop on Formal Methods in Security Eng.*, Washington, Oct. 2003

Cuppens F., Mie'ge A., Alert Correlation in a Cooperative Intrusion Detection Framework, *IEEE Symp. on Security and Privacy*, p.202, May 12-15, 2002

CORAS: *A platform for risk analysis of security critical systems*. IST-2000-25031, 2000.

Dawkins J., Campbell C., Hale J., Modeling Network Attacks: Extending the Attack Tree Paradigm, *Proc. of Statistical and Machine Learning in Intrusion Detection*, June 2002.

Epstein, S. and Rauzy, A. Can we trust PRA? *Reliability Engineering and System Safety*, vol. 88, n.3, pp.195–205, 2005,.

Goldman R. P., Heimerdinger W., Harp S. A., Information Modeling for Intrusion Report Aggregation, *DARPA Information Survivability Conference*, June 2001.

Gordon L., Loeb M., The economics of information security investment. *ACM Trans. on Information and System Security* vol.5, n. 4, 2002. pp.438-457.

Grunske, L. and Kaiser, B. and Papadopoulos, Y., Model-driven safety evaluation with state-event-based component failure annotations , *Component-based software engineering : 8th international symposium*, St. Louis, MO, USA, May 14 - 15, 2005 Lecture Notes in Computer Science 3489, 2005 , pp. 33-48

Gulati, R.; Dugan, J.B., A modular approach for analyzing static and dynamic fault trees, *Reliability and Maintainability Symposium. 1997 Proceedings, Annual* , vol., no., pp.57-63, 13-16 Jan 1997

Haimes, Y. Y., 'Hierarchical holographic modeling, *IEEE Transactions on Systems, Man, and Cybernetics*, 11(9), 606–617, 1981.

Haimes, Y. Y.,, Kaplan S., Lambert J.H., Risk filtering, ranking, and management framework using hierarchical holographic modeling, *Risk Analysis*, vol.22, n. 2, pp. 383-97, Apr. 2002.

IEC 1025, Fault tree analysis (FTA) 1990.

Jajodia S., Noel S., O'Berry B., Topological Analysis of Network Attack Vulnerability, in *Managing Cyber Threats: Issues, Approaches and Challenges*, V. Kumar, J. Srivastava, A. Lazarevic (eds.), Kluwer Academic Publ., 2003.

Jha S., Sheyner O., Wing J., Two Formal Analysis of Attack Graphs, *15th IEEE Computer Security Foundations Workshop* , p.49, June 2002.

Klamt, S., Gilles, E.D., 2004. Minimal cut sets in biochemical reaction networks. *Bioinformatics,* 20, 226–234, Elsevier Press, 2004.

Klamt. S, Generalized concept of minimal cut sets in biochemical networks, *BioSystem*, 83 pp. 233–247, Elsevier Press, 2006

Manadhata P., Wing J. M., Measuring a System's Attack Surface, Technical Report CMU-CS-04-102, Jan. 2004

Manadhata P., Wing J. M., An Attack Surface Metric, *USENIX Security Workshop on Security Metrics* (MetriCon), Vancouver, BC, August 2006

Martin R. A., Managing vulnerabilities in networked system, *IEEE Computer*, Nov. 2001. p. 32 – 38

Moore P., Ellison R. J., Linger R. C., Attack modelling for information security and survivability, Carnegie Mellon University, Report CMU/SEI- 2001-TN001, 2001

National Infrastructure Advisory Council, *The Common Vulnerability Scoring System*, Final Report and Recommendations, Oct. 2004

Nasser S. F., Determination of minimal cut sets of a complex fault tree, *Computers & Industrial Engineering*, Volume 33, Issues 1-2,Oct. 1997, Pages 59-62.

Nicol D. M., Modeling and Simulation in Security Evaluation, *IEEE Security and Privacy*, v.3 n.5, p.71-74, Sept. 2005

Ning P., Cui Y., Reeves D. S., Constructing attack scenarios through correlation of intrusion alerts, *9th ACM Conf. on Computer and Communications* Security, Nov. 2002, Washington, DC, USA.

Ning P., Cui Y., Reeves D. S., Xu D., Techniques and Tools for Analyzing Intrusion Alerts, *ACM Trans. on Information and System Security (TISSEC)*, vol.7 n.2, p.274-318, May 2004

Papazoglou I.A, Mathematical foundations of event trees, *Reliability Engineering & System Safety*, Volume 61, Issue 3, , September 1998, Pages 169-183.

Phillips C., Painton Swiler L., A graph-based system for network-vulnerability analysis, *Workshop on New Security Paradigms*, p.71-79, Sept.1998.

Ritchey R., et al., Representing TCP/IP Connectivity For Topological Analysis of Network Security, *18th Annual Computer Security Applications Conf*, p.25, Dec. 2002.

Russell S., Norving P., Artificial Intelligence: a Modern Approach, Prentice Hall, 2003

Schechter S., Smith M., 2003. How much security is enough to stop a thief?. *Proc. of the Financial Cryptography Conference*, Guadeloupe, Jan. 2003.

Sheyner O.M., et al., Automated Generation and Analysis of Attack Graphs, *IEEE Symposium on Security and Privacy*, May 12-15, 2002.

Sheyner O. M., *Scenario Graphs and Attack Graphs*, Carnegie Mellon University, Report CMU-CS-04-122,2004.

Swiler L.P., Phillips C., Ellis D., Chakerian S., Computer-Attack Graph Generation Tool, *Proc. of the DARPA Information Survivability Conf*, June 2001.

Swarup V., Jajodia S., Pamula J., Rule-Based Topological Vulnerability Analysis, *3rd Int. Workshop on Mathematical Methods, Models and Architecture for Network Security*, S.Petersburg, Sept. 2005.

Tang Z., Dugan, J. B.: Minimal Cut Set/Sequence Generation for Dynamic Fault Trees. Annual Reliability and Maintainability Symposium, Los Angeles, 2004

Templeton S. J., Levitt K., A requires/provides model for computer attacks, *Proc. of the Workshop on New Security Paradigms*, Sept. 2000, Ballycotton, County Cork, Ireland

Rauzy A.: Mathematical foundations of minimal cut sets. IEEE Transactions on Reliability, vol.50, (2001) 389-396.

Vesely, W. E., Goldberg, F. F., Robert, N. H., and Haasl, D. F. Fault tree handbook. Technical Report NUREG 0492, US Nuclear Regulatory Commission, 1981.

Wing J.M., Scenario Graphs Applied to Network Security in *Information Assurance: Survivability and Security in Networked Systems* , D.Tipper, P. Krishnamurthy, Yi Quian, and J. Joshi, eds, Morgan Kaufmann Publishers.

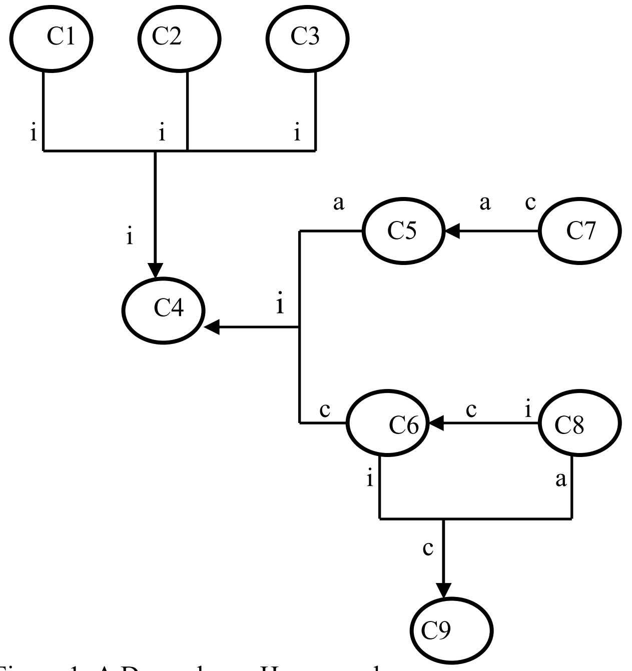


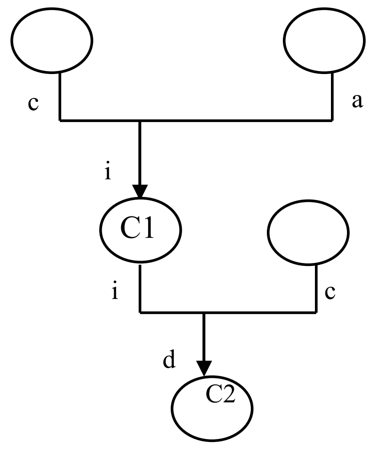
Figure 1. A Dependency Hypergraph

$$\begin{tikzcd}
 & \Phi \arrow[dl] \arrow[d] \arrow[dr] & \\
\{c1\} \arrow[d] \arrow[dr] & \{c4\} \arrow[dl] \arrow[dr] & \{c2, c3\} \arrow[dl] \arrow[d] \\
\{c1, c4\} \arrow[dr] & \{c1, c2, c3\} \arrow[d] & \{c2, c3,c4\} \arrow[dl] \\
 & \{c1, c2, c3,c4\} &
\end{tikzcd}$$

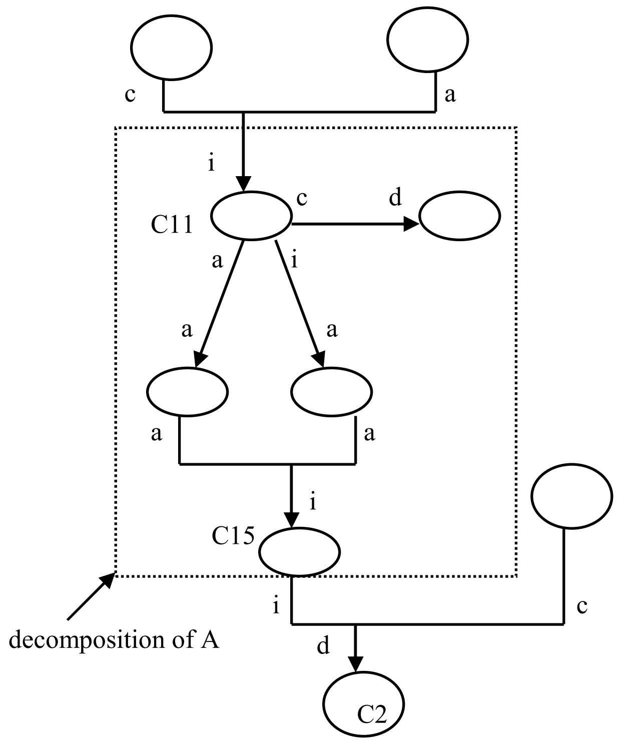
Figure 2. Lattice Generated by the the Nonredundat Subsets of a Minimal Set of Countermeasures

$$\begin{tikzcd}
 & & \Phi \arrow[dll] \arrow[dl] \arrow[d] \arrow[dr] & & \\
\{c5,c6\} \arrow[d] \arrow[dr] & \{c1\} \arrow[dl] \arrow[dr] \arrow[drr] & \{c4\} \arrow[dl] \arrow[d] \arrow[drr] & \{c2,c3\} \arrow[d] \arrow[dr] & \\
\{c1,c5,c6\} \arrow[dr] & \{c4, c5,c6\} \arrow[d] & \{c1,c4\} \arrow[dl] \arrow[dr] & \{c1,c2,c3\} \arrow[d] & \{c2,c3,c4\} \arrow[dl] \\
 & \{c1,c4,c5,c6\} & & \{c1,c2,c3,c4\} &
\end{tikzcd}$$

Figure 3. Global Lattice Generated by Taking any Minimal Set into Account

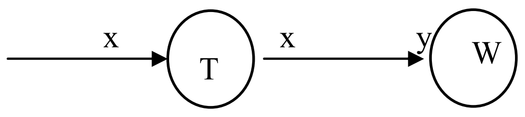


a) an infrastructure hypergraph

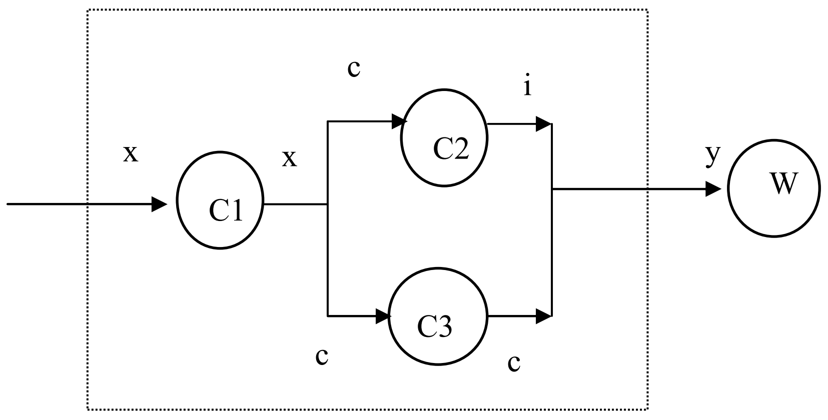


b) hypergraph after the decomposition of C1

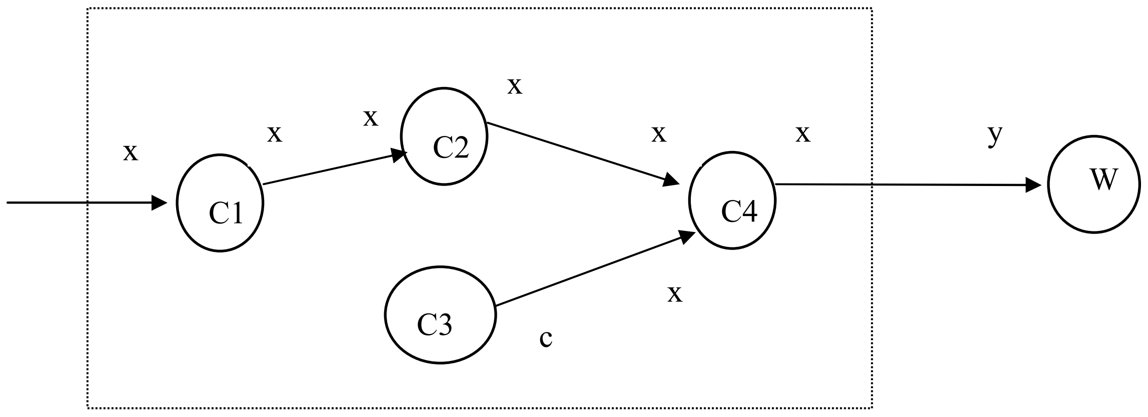
Figure 4 Hierarchical Decomposition of a Component



a) Original Hypergraph

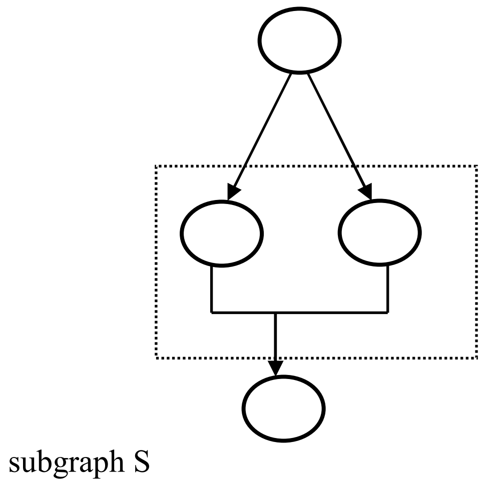


b) A First Hierarchical Decomposition

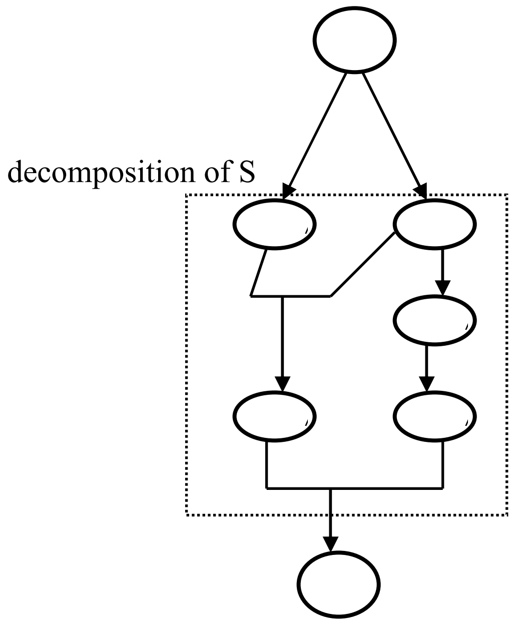


c) Alternative Hierarchical Decomposition

Figure 5. Hierarchical Decompositions of a Component

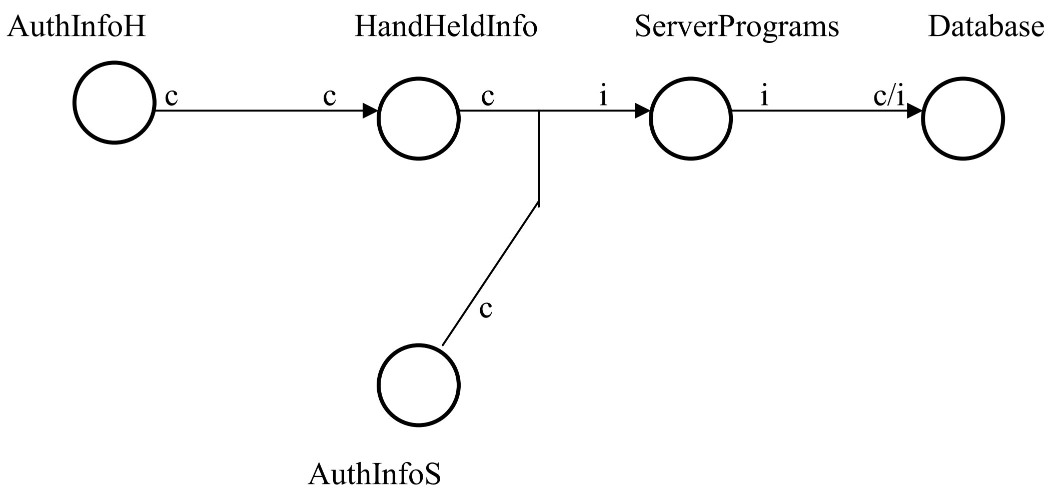


a) initial hypergraph, attributes are not shown

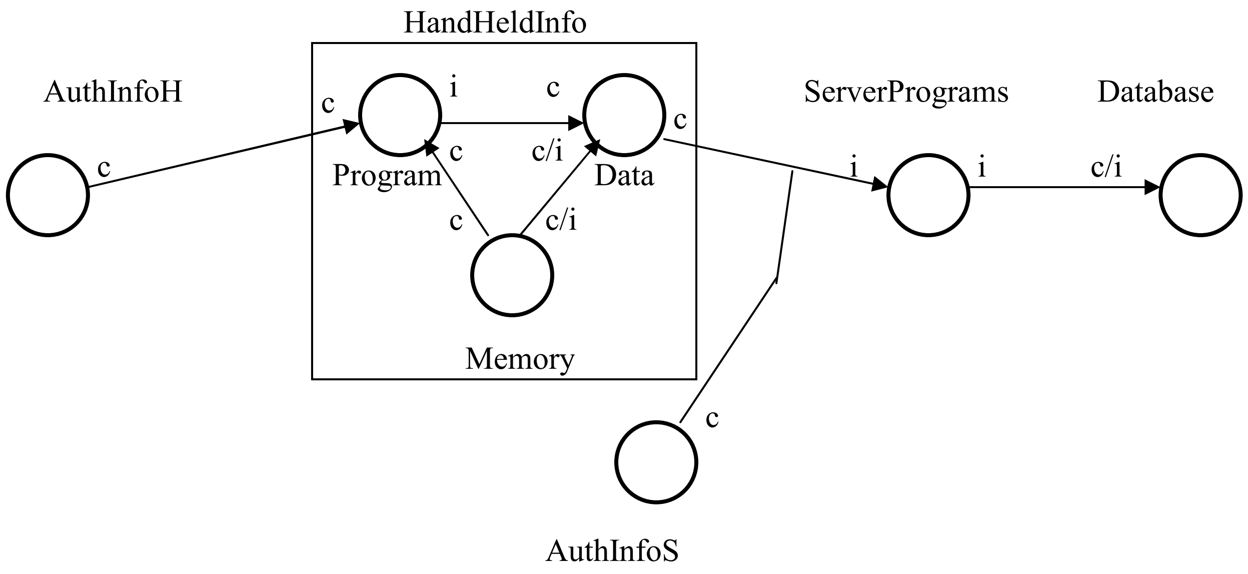


b) hypergraph after the decomposition

Figure 6. Hierarchical Decomposition of a Subgraph



a) A high level description of a (subset) of a system



b) A Hierarchical Decomposition of the HandHeldDevice

Figure 7. An Example