View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by UEL Research Repository at University of East London

University of East London Institutional Repository: http://roar.uel.ac.uk

This paper is made available online in accordance with publisher policies. Please scroll down to view the document itself. Please refer to the repository record for this item and our policy information available from the repository home page for further information.

To see the final version of this paper please visit the publisher's website. Access to the published version may require a subscription.

Author(s): Bresciani, Paolo; Giorgini, Paolo; Mouratidis, Haralambos; Manson, Gordon.
Article title: Multi-agent systems and security requirements analysis
Year of publication: 2004
Citation: Bresciani, P. et al. (2004) 'Multi-agent systems and security requirements analysis' In Lucena, C et al (eds) Software engineering for multi-agent systems II: research issues and practical applications 2004 vol. 2940 pp. 35-48
Link to published version: http://dx.doi.org/10.1007/b96018
DOI: 10.1007/b96018

Multi-Agent Systems and Security Requirements Analysis

Paolo Bresciani[1], Paolo Giorgini[2], Haralambos Mouratidis3, and Gordon Manson[3]

[1]ITC-irst, via sommarive 18, I-38050, Povo, Trento, Italy
bresciani@itc.it
[2]Department of Information and Communication Technology, University of Trento, Italy
paolo.Giorgini@dit.unitn.it
[3]Department of Computer Science, University of Sheffield, England
{haris, g.manson}@dcs.shef.ac.uk

Abstract. Agent Oriented Software Engineering (AOSE) is a software paradigm that has grasped the attention of researchers the last few years. As a result, many different methods have been introduced to enable developers develop multi-agent systems. However, so far, security requirements have been mainly neglected, and the common approach towards the inclusion of security within a system is to identify security requirements after the definition of the system. This approach has provoked the emergence of computer systems afflicted with security vulnerabilities. In this paper we propose an analysis, based on the measures of criticality (how critical an actor of the system is) and complexity (represents the effort required by the actors of the system to achieve the requirements that have been imposed to them), which aims to identify possible bottlenecks of a multi-agent system with respect to security. An integrated agent-based health and social care information system is used as a case study throughout this paper.

1 Introduction

In a world that becomes more and more reliant on software systems, security is an important concern. Private information is stored in computer systems and without security, organizations are not willing to share information or even use the technology. In addition, possible security breaches can cost huge amount of time and money.

Following the wide recognition of multi-agent systems, agent-oriented software engineering has been introduced as a major field of research. Many agent-oriented software engineering methodologies have been proposed [1,2] each one of those offering different approaches in modeling multi-agent systems. However, only few attempts [3] have been made to integrate security issues within the development stages of methodologies.

Security requirements are generally difficult to analyse and model. It is difficult to analyse because many times security requirements conflict with functional requirements and many trade offs are required. Performing such trade offs can be

painful and time-consuming and it requires software and security engineering expertise. In addition, there is lack of developers' acceptance and expertise for secure software development.

Usually the goal will be to provide as much security as possible trading sometimes security concerns with other functional and non-functional requirements. To better achieve this goal, agent-oriented software engineering methodologies must help developers, through a systematic approach, to determine how complex is for each part (actor) of the system to achieve the security requirements, and also identify the most critical actors of the system with respect to security. Such an approach will help developers to perform trade offs between security and other functional and non-functional requirements based on quantitative measurements and thus minimizing the risks of putting in danger the security of the system.

Within a multi-agent system, more likely, different agents will play different roles and, with respect to security, some will be more critical than others. In addition, some agents of the system might have been overloaded (assigned more security requirements than they can handle) and thus fail to satisfy some of the security requirements assigned to them.

Developers must be able to identify, through a systematic approach and without much security knowledge, such cases and redefine the design of the system in such a way that none of the agents of the system are overloaded and all the security requirements assigned to the agents of the system are satisfied.

In this paper we propose an approach based on the concepts of criticality and complexity, and we indicate how such a process can be integrated within the early requirements analysis stage of the Tropos methodology. This work is within the context of the Tropos project [2] and our aim is to provide a clear and well-guided process of integrating security and functional requirements throughout the whole range of the development process. Section 2 provides an overview of Tropos methodology, and also introduces the electronic Single Assessment Process (eSAP) system case study. In Section 3, we describe the process of analysing the complexity and criticality of a system with respect to security, and we present an algorithm to reduce the complexity and/or the criticality of the "overloaded" actors. Finally, Section 4 presents some concluding remarks and directions for future work.

2 Tropos Methodology

Before we can describe our approach, we think it is necessary to provide an overview of Tropos methodology and how security can be integrated to it. Tropos is an agent oriented software engineering methodology, tailored to describe both the organisational environment of a system and the system itself, employing the same concepts throughout the development stages. The Tropos methodology is intended to support all the analysis and design activities in the software development process, from the application domain analysis down to the system implementation [2]. Using Tropos, developers build a model of the system-to-be and its environment that is incrementally refined.

Tropos adopts Yu's i* model [4] which offers the concepts of actors, goals, tasks, resources and social dependencies for defining the obligations of actors (dependees) to other actors (dependers). Actors have strategic goals and intentions within the system or the organisation and represent (social) agents (organisational, human or software), roles or positions (represent a set of roles). A goal represents the strategic interests of an actor. In Tropos we differentiate between hard goals (only goals hereafter) and soft goals; the latter having no clear definition or criteria for deciding whether they are satisfied or not. A task represents a way of doing something. For example a task can be executed in order to satisfy a goal. A resource represents a physical or an informational entity while a dependency between two actors indicates that one actor depends on another to accomplish a goal, execute a task, or deliver a resource.

Tropos covers four stages of software development: **Early Requirements** analysis consists of identifying and analysing the stakeholders and their intentions. Stakeholders are modeled as social actors, while their intentions are modeled as goals that, through a goal-oriented analysis, are decomposed into finer goals, which eventually can support evaluation of alternatives. **Late Requirements** analysis consists of analysing the system-to-be within its operating environment, along with relevant functions and qualities. The system is introduced as an actor and the dependencies between the system and the other actors of the organization are explicitly modeled. These dependencies define the system's requirements. **Architectural Design** describes the system's global architecture in terms of subsystems (actors) interconnected through data and control flows (dependencies). During this stage, new actors are introduced in the system as a result of analysis performed at different levels of abstraction. In addition, capabilities needed by the actors to fulfill their goals and tasks are identified. **Detailed Design** deals with the specification of each architectural component in terms of inputs, outputs, control and other relevant information. Tropos faces the detailed design stage on the basis of the specifications resulting from the architectural design stage and the reasons for a given element can be traced back to the early requirements analysis.

The security process in Tropos consists of analyzing the security needs of the stakeholders and the system in terms of security constraints [5] imposed to the stakeholders (early requirements) and the system (late requirements), identifying secure entities [5] that guarantee the satisfaction of the security constraints, and assigning capabilities to the system (architectural design) to help towards the satisfaction of the secure entities.

In our work [3, 5] we define security constraints as constraints that are related to the security of the system whereas secure entities represent any secure goal/task/resource of the system [5]. Security constraints can be categorized into Positive –they influence the security of the system positively (e.g., *Allow Access only to Personal Information*) – or negative – they influence the security of the system negatively (e.g., *Send information plain text*).

To make the process easier to understand, we consider as an example the electronic Single Assessment Process (eSAP) case study first introduced by Mouratidis et. al [6]. The eSAP case study involves the development of an agent-based health and social care system for the effective care of older people. Security in such a system, as in any health and social care information system, is very important

since revealing a medical history could have serious consequences for particular individuals. Taking into account a substantial part of the eSAP, we have defined the following stakeholders for our case study: The Older Person (OP) actor is the older person (patient) that wishes to receive appropriate health and social care. The Professional actor represents health and/or social care professionals involved in the care of the Older Person. The DoH actor represents the English Department of Health, which is responsible for the effective care of the Older Person. The Benefits Agency actor is an agency that helps the Older Person financially, and the R&D Agency represents a research and development agency interested in obtaining medical information.

During the early requirements analysis stage, the dependencies, the goals and the security constraints between these actors can be modeled using Tropos actors' diagram as shown in Figure 1[1]. In such a diagram each node represents an actor, and the links between the different actors indicate that one depends on another to accomplish some goals.

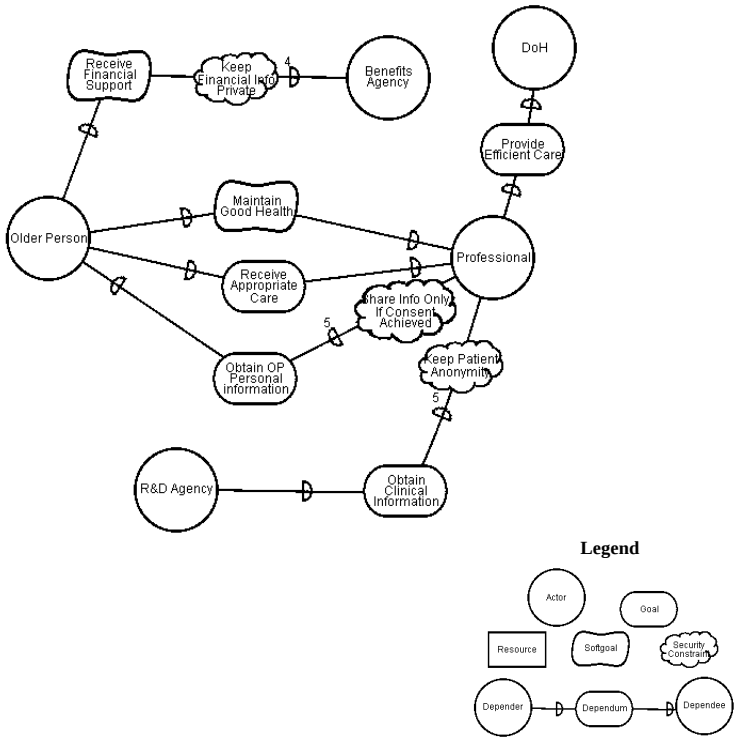

Fig. 1. The actor diagram of the eSAP system

In our example, the *Older Person* depends on the *Benefits Agency* to *Receive Financial Support*. However, the *Older Person* worries about the privacy of their

[1] The numbers next to the security constraints represent the criticality of the constraint (Section 3).

finances so they impose a constraint to the *Benefits Agency* actor, to keep their financial information private. The *Professional* depends on the *Older Person* to *Obtain Information*, however one of the most important and delicate matters for a patient (in our case the *Older Person*) is the privacy of their personal medical information, and the sharing of it. Thus most of the times the *Professional* is imposed a constraint to share this information if and only if consent is achieved. One of the main goals of the *R&D Agency* is to *Obtain Clinical Information* in order to perform tests and research. To get this information the *R&D Agency* depends on the *Professional*. However, the Professional actor is imposed a constraint (by the *Department of Health*) to *Keep Patient Anonymity*.

3 Criticality and Complexity

In the previous section we have briefly described a process of analysing the security of an organisational setting taking into consideration some *security constraints*, which are imposed by the different stakeholders. However, more likely different *security constraints* are having different impact on the security of the system. That is, one constraint might put in danger the security of the system in a level that must be satisfied even if it involves a trade off with some other functional or non-functional requirements, whereas other constraints might not be as important. As a result, different actors of the system impact the security of the system differently according to what security constraints have been imposed to. Thus, it is important to provide an analysis that identifies the impact each actor has on the security of the system. In doing so we need to define how critical each *security constraint* is for the overall security of the system. We call this measure, security criticality[2] and we define it as follows:

Security Criticality *is the measure of how the security of the system will be affected if the security constraint is not achieved*.

Security criticality allows us to evaluate how critical each actor of the system is with respect to security. This will help us to identify the security bottlenecks of the system, and refine it by taking into consideration the different impact that each actor has on the security of the system. We differentiate between ***ingoing*** and ***outgoing*** security criticality. Ingoing security criticality is the security criticality that actors assume when they are responsible for achieving a security constraint. On the other hand, the outgoing security criticality represents the security criticality of the achievement of a constraint for the imposer.

In order to calculate the criticality of the system, we consider the dependencies and we assign a value for each security constraint (see numbers next to security constraints in Figure 1). These values were assigned after closely studying the system's environment and after discussing them with the stakeholders. In the case of an open secure dependency (a dependency that has no security constraints attached to it), we assign a value of zero both for the ingoing and outgoing criticalities.

[2] Criticality has been introduced by E. Yu in [4]

In this example we have assumed that criticality obtains integer values within the range 1-5, where 1 = very low, 2 = low, 3 = medium, 4 = high, 5 = very high. However, the range of acceptable values can change and it depends on each developer. For example, developers might decide it is better for them to assign values within the range 1-20. This will provide them with more accurate ratings of the criticalities.

In addition, a maximum value of criticality is defined for each actor taking into account, the actor's abilities, their available time, and the responsibilities they have in the organization.

As mentioned above, security criticality allows us to evaluate how critical each actor of the system is with respect to security. Nevertheless, we need to be able to evaluate how much effort is required by each of the actors to achieve their security constraints. To perform such an evaluation, we introduce the concept of security complexity and we identify it as follows:

Security Complexity *is the measure of the effort required by the responsible actor for achieving a security constraint.*

Considering security complexity helps to design sub-systems to support actors that might be in danger not achieving some security constraints, and therefore put in danger the overall security of the system. This means, if an actor is overloaded with security responsibilities, some of the security constraints should be delegated to another existing actor of the system, or if this cannot happen, the developer should introduce another actor and delegate some of the security constraints of the "overloaded" actor.

In order to be realistic, we need to take into account both the system and security complexity, where ***System Complexity*** is defined as the measure of the effort required from the dependee for achieving the dependum [7]. This is necessary since it might be the case that an actor's security complexity is high, however since their system complexity is very low, they are capable of achieving all the security constraints. On the other hand, there might be cases where an actor's security complexity might be low but their system complexity is high and therefore they might not be able to achieve all the security constraints imposed to them. Thus, by taking into consideration both system and security complexity we can identify more precise the degree of achievement of the security complexity.

In addition, an important factor in (realistically) calculating the overall complexity is time. It might be the case that an actor can achieve different (secure) goals sequentially, so in this case it would not be realistic to sum up the individual values of complexity in order to evaluate the overall complexity of the actor. Sum up all the different complexity values would be realistic only if all the goals should achieved at the same time. However, in the real world this will be more likely the case of an organization (department) in which different agents work, than the case of a single agent.

Similar to criticality analysis, we have assumed that complexity (system and security) can obtain integer values within the range 1-5, where 1 = very low, 2= low, 3=medium, 4=high, 5=very high. Also similarly to criticality, a maximum value of (overall) complexity is defined for each actor.

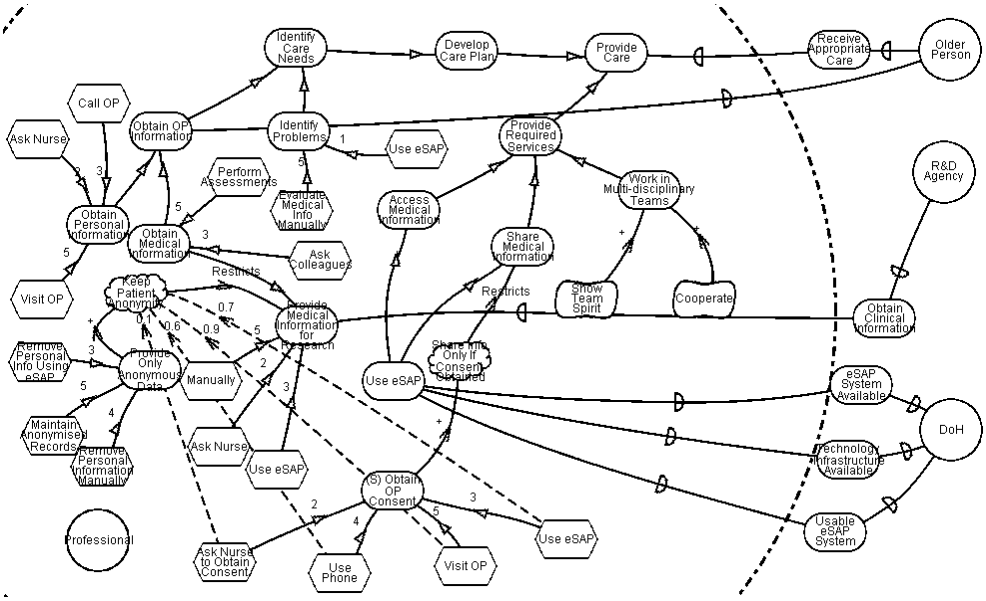

Fig. 2. Rationale diagram of the Professional actor

To be able to precisely assign values for security and system complexity, each actor of the system and their security constraints and goals respectively must be further analysed. This is necessary because the security constraints and the goals modeled in the actors' diagram (figure 1) are quite superficial and it is difficult to evaluate their complexity. Therefore, many different alternative tasks might be considered for their satisfaction, each with different complexity value. To cope with this, we are extending our analysis, by further analysing (for each actor involved in our system) the security constraints (for the security complexity) and the actor's goals (system complexity), together with the different alternatives that can satisfy them. This kind of analysis, apart from helping us to define more precisely the values for complexity, it provides a basis to choose between different alternatives that can be employed for the satisfaction of security constraints and the actor's goals, something very important in justifying the trade offs between security and the functional requirements of the system.

For this analysis, we are employing Tropos rationale diagrams [4]. Differently than actors' diagram, which focuses on the external relationships between the actors of the organization, each rationale diagram analyses the internal goals, security constraints and dependencies of each actor (figure 2). In order to calculate the values of security complexity for each actor, different weights have been assigned to the different relationships involved in the satisfaction of the security constraints (secure goals), that have been imposed to the actor, and the actor's strategic goals. For reasons of simplicity in this paper we have assumed weights can obtain integer numbers in the range of 1-5 (1 being the lowest value with respect to complexity and 5 the highest).

In addition, in the cases where the dependum is a soft goal, minimal system complexity values are assumed. This is the minimal effort requested from the depender to achieve the soft goal. This has been decided since the concept of a soft goal has no clear criteria for whether there are satisfied or not, and as such we cannot assign a precise value required for achieving the soft goal.

For our case study, the rationale diagram of the Professional actor is shown in figure 2. As it can be seen from the figure, different alternatives can be considered for the satisfaction of the security goals imposed to the actor as well as the actor's strategic goals. For example, to identify problems, the Professional can evaluate info manually or use eSAP. For each of those alternatives we have assigned a value as shown in figure 2. In addition, the contribution of each alternative to the other functional and security requirements is shown in figure 2 (as dashed line links). To denote the contributions of the different alternatives, we employ a quantitative approach presented by Giorgini et al [8]. Thus each contribution receives weights between 0 and 1, where 0 means the alternative puts in maximum danger the security or the functional requirement, while 1 means the alternative completely satisfies the security or the functional requirement. To keep the diagram simple and understandable we denote contributions to the Keep Patient Anonymity security constraint, only from the Obtain OP Consent secure goal alternatives (figure 2).

For example, the *Share Information Only if Consents Obtained* security constraint of the *Professional* actor is satisfied by the *Obtain OP Consent* secure goal. However, this goal can be achieved by considering different alternatives, each one of those alternatives having a different security complexity weight. Thus, the Professional can *Visit OP*, *Use Phone*, *Use eSAP*, or *Ask a Nurse* to obtain the consent of the Older Person. These tasks have been assigned with different weights of complexity according to how much effort is required from the Professional to achieve them. Thus, in the above-mentioned tasks we have assign weights of 5,4,3 and 2 respectively. However, in deciding which task is best suited, developers should also consider how this task affects (if it affects) other requirements of the system. For example, although the *Ask a Nurse* is the less complex task for the *Professional* and the obvious choice from the point of view of complexity, it is worth considering that the involvement of a nurse could contribute negatively to the *Keep Patient Anonymity* security constraint also imposed to the *Professional* actor. This could put in danger the privacy of the *Older Person*, an undesired effect for our system. Thus, we have decided in this case to choose the *Use eSAP* task, since it requires the less effort (apart from the *Ask a Nurse*) and also it helps towards the older person's privacy. When all the different options have been considered and a choice about which one is best suited have been made, the next step is to calculate the overall complexity for each actor. This process takes part alongside with the calculation of the criticality for each actor.

In order to analyze the complexity and criticality with respect to security, we firstly calculate, for each actor involved, the complexity and the criticality. Then, if some actor assumes a greater value of complexity and criticality than the maximum value they can assume, we want to reassign some security constraints to different actors of the system in order to reduce the complexity or the criticality of the "overloaded" actors. In other words, the problem we want to solve is: "how to reassign one (or more) goals of actors whose complexity/criticality is greater that their maximum complexity/criticality limit?", that is, how can we reconfigure the topology of the

actor diagram in order to end up with a "balanced" configuration? Of course we would like to solve the problem by means of minimal topology modifications. In fact, many solutions may be found by radically redesigning the diagram, but these shouldn't be considered as first choice solutions.

To take into account these needs, we propose in Figure 3 the `Rebalance` algorithm that, given a representation of an actor diagram and its constraints, is capable to produce a new configuration (if it exists), in which the constraints are satisfied. For the shake of simplicity, the presented algorithm considers only the complexity and not the criticality. However, it is relatively easy to extend the algorithm to consider both the complexity and the criticality.

Let us assume there are `m` dependums and `n` actors. Moreover, let us suppose that the fact that different actors may fulfill the different dependums, is coded by means of a cost matrix `CoM[1..n,1..m]` where, for each actor `i` and dependum `j`, the cost for `i` to fulfill `j` is `CoM[i,j]`. This cost may be different for different actors fulfilling a given dependum (not all the actors have the same level of skills) and, in particular, it may be infinite (`MAXINT`) for some actors (not all the actors can fulfill a given dependum). On the other hand, the vector `M_CoV[1..n]` provides the maximum complexity that each actor can hold. It is worth mentioning that the matrix `CoM` and the vector `M_CoV` are constant data provided with the analysis of the domain.

In addition, the actor diagram topology, is described by means of a variable `A[1..n,1..m]` of booleans where a "`1`" in position `(i,j)` means that the dependum `j` is assigned at the actor `i`. Of course, for each dependum `j` there is one and only one "`1`". The actor load defined by the current topology is computed by the Function:

$$\texttt{Compl(i,A)}=\sum_{j=1}^{m}\texttt{CoM[i,j]A[i,j])}$$

The core of the algorithm is given by the Function `Try_One_Actor` that tries to rearrange the matrix `A` in order to accommodate the load of actor `i` below its maximum complexity capacity, starting to analyze dependum `j` first. It iteratively considers possible reassignments for dependum `j` to other actors that can fulfill it without exceeding their maximum capacity. This possibility is tested by the Function:

$$\texttt{Fits(A,l,j)=(Compl(l,A)+CoM[l,j]} \leq \texttt{M\_CoV[l])}$$

The problem is recursively scaled down by considering also other dependums (`j+1`) if the reassignment of the current one (`j`) is not sufficient or not possible. Backtrack is required in case the current reassignment of `j` to `l` is useless.

The above presented core function, considers only one overloaded actor. However, it can be extended to consider more overloaded actors (see the Function `Reballance_Intransitive in Figure 5`). Such function is recursively called (with possible backtrack) only in the case at least one of the overloaded actors can be re-balanced. Backtrack allows us to iteratively consider all the overloaded actors in turn as the first to be processed. In fact, the solution may depend, in a

generic —even tough very idiosyncratic— case, by the processing order. The recursion takes care for considering the other overloaded actors.

Finally, if a solution involving the redistribution of dependums from actor to actor requiring that recipient actors have not to be re-balanced themselves cannot be found by means of the Function `Reballance_Intransitive` (Figure 5), the more generic and entry point Function `Rebalance` try to consider also the possibility of transitively affect the load of recipient actors even over their maximum capacity, by calling the Function `Try_Transitive` (Figure 4). In this case the adjustments can be spread all over the matrix, implying radical topology redesign. Minimizing modifications became now more difficult even to be defined, and, in the current version of the algorithm, no particular claim is done, except that termination and the production of one solution (if it exists) is guaranteed. Termination is guaranteed by the fact that each dependum is reassigned at most once (there is no need to reassign it more than once; again, of course, the use of backtracking allow us to test all the re-assignments).

```
CONST m:integer; {# of dependums}
      n:integer; {# of actors}
      M_CoV:array[1..n] of real;
      {max cost for each actor}
      CoM: array[1..n,1..m] od real;
      {the effort for actor i to provide goal j}

GLOBAL VAR VISITED_DEP: set of visited dependums;
           {initially empty}
           A: array[1..n,1..m] of boolean;
           {the assignament matrix properly
            initialized to reflect diagram
            topology}

LOCAL VAR SET_OF_UNBALLANCED: set of actors;

Function Rebalance(var A: ass_matrix): boolean;
begin
   result:=Reballance_Intransitive(A);
   if result=fail then
   begin
      SET_OF_UNBALLANCED:={i|Compl(i,A)>M_CoV[i]};
      copy_of_A:=A;
      while result=fail   and
            not empty(SET_OF_UNBALLANCED)  do
      begin
     i:=POP(SET_OF_UNBALLANCED);
     result:=Try_Transitive(i,A);
     if result=fail then A:=copy_of_A
      end
   end;
   RETURN result
end;
```

Fig. 3. The reassignment algorithm

```
Function Try_One_Actor(i,j: integer; var A: ass_matrix):boolean;
begin
```

```
   if j>m then RETURN fail;
   result:=fail;
   l:=0;
   if A[i,j]=1 then
      while Compl(A,i)>M_CoV[i] and l<n do
      begin
      l++;
             if l<>i and Fits(A,l,j) then
      begin
         copy_of_A:=A;
         A[l,j]:=1; A[i,j]:=0;
         if Compl(A,i)>M_CoV[i] then
         begin
            result:=Try_One_Actor(i,j+1,A);
            if result=fail then A:=copy_of_A
         end
         else result:=OK
      end
      end;
   if result=OK then RETURN result
     else RETURN Try_One_Actor(i,j+1,A)
end;

Function Try_Transitive(i: integer; var A:ass_matrix):boolean;
begin
   result:=fail;
   copy_of_A:=A;
   j:=0;
   if not empty(VISITED_DEP) then
      while j<m and result=fail do
      begin
      j++;
      if A[i,j]=1 and not j in VISITED_DEP then
      begin
         push(j,VISITED_DEP);
         l:=0;
         while l<n and
                  (Compl(i,A)>M_CoV[i] or
                   result=fail)             do
         begin
            l++;
            if l<>i and CoM[l,j]<MAXINT then
            begin
               A[l,j]:=1; A[i,j]:=0;
result:=Reballance(A);
               if result=fail then A:=copy_of_A
            end
         end;
         if result=fail then VISITED_DEP:=VISITED_DEP-{j}
      end
      end;
   RETURN result
End
```

Fig. 4. The functions `Try_One_Actor` and `Try_Transitive`

```
Function Reballance_Intransitive (var A:ass_matrix):boolean;
begin
```

```
result:=fail;
SET_OF_UNBALLANCED:={i|Compl(i,A)>M_CoV[i]};
if empty(SET_OF_UNBALLANCED) then result:=OK
else
begin
   copy_of_A:=A;
   while result=fail and
         not empty(SET_OF_UNBALLANCED) do
   begin
   i:=POP(SET_OF_UNBALLANCED);
   if Try_One_Actor(i,1,A)=OK then
   begin
      result:=Reballance_Intransitive(A);
      if result=fail then A:=copy_of_A
   end
    end
end;
RETURN result
end;
```

Fig. 5. The function `Reballance_Intransitive`

4 Conclusions

In this paper we have presented an analysis for evaluating the degree of complexity and criticality of the actors of the system, with respect to security. Such an analysis provides a valuable process for the developers of multi-agent systems in order to identify possible security bottlenecks. In addition, we have proposed an algorithm to reduce the complexity or the criticality of the "overloaded" actors.

Our analysis helps to justify possible trade offs between security and functional requirements. By knowing how critical an agent is with respect to security a decision can be made. Our aim is to provide a clear well guided process of integrating security and functional requirements throughout the whole range of the development stages. Such a process must use the same concepts and notations throughout the development phases. The ability to identify the bottlenecks of a multi-agent system with respect to security and justify the decisions behind possible trade offs between security and functional requirements can definitely help towards this aim.

It is worth mentioning that in this paper we only consider security requirements. Nevertheless, our approach can be easily adapted to deal with other non-functional requirements.

This work is an ongoing research. The presented analysis covers only the requirements stage of the Tropos methodology. We are working towards extending our analysis to the next stages of the methodology, since such an analysis can help in the later stages of the development. For example, criticality and complexity can help us to decide for different architectural choices during the architectural design stage of the methodology, such as the choice between mobile and static agents.

In addition, we are working towards the development of a process that will allow developers to assign weights to different alternatives in case the different stakeholders disagree on the assignment.

The present version of the algorithm guarantees to find a solution that requires the reassignments of dependums of overloaded actors only, if it exists. Otherwise, a solution with transitive reassignments is in any case provided (if it exists), although we cannot at present guarantee it is the best. We believe that, possibly after small improvements, the algorithm can provide the "best" solution. We foresee to work to prove this fact. Moreover, our future research plan includes also the study of the complexity of the algorithm, and its implementation and test.

Acknowledgements

The third Author is grateful to the RANK Foundation for the funding of his research project, in which this work was carried out.

References

[1] C. Iglesias, M. Garijo, J. Gonzales, "A survey of agent-oriented methodologies", *Intelligent Agents IV*, A. S. Rao, J. P. Muller, M. P. Singh (eds), Lecture Notes in Computer Science, Springer-Verlag, 1999

[2] J. Castro, M. Kolp and J. Mylopoulos. "A Requirements-Driven Development Methodology," In *Proc. of the 13th Int. Conf. On Advanced Information Systems Engineering (CAiSE'01)*, Interlaken, Switzerland, June 2001.

[3] H. Mouratidis, P. Giorgini, G. Manson, "Modelling Secure Multiagent Systems", (to appear) in the Proceedings of the 2nd International Joint Conference on Autonomous Agents and Multiagent Systems, Melbourne-Australia, July 2003

[4] E. Yu, "Modelling Strategic Relationships for Process Reengineering", PhD thesis, Department of Computer Science, University of Toronto, Canada, 1995

[5] H. Mouratidis, P. Giorgini, G. Manson, I. Philp, "A Natural Extension of Tropos Methodology for Modelling Security", Proceedings of the Agent Oriented Methodologies Workshop in OOPSLA 2002, Seattle-USA, November 2002

[6] H. Mouratidis, i. Philp, G. Manson, "Analysis and Design of eSAP: An Integrated Health and Social Care Information System", in the Proceedings of the 7th International Symposium on Health Information Managements Research (ISHIMR2002), Sheffield, June 2002

[7] M. Garzetti, P. Giorgini, J. Mylopoulos, F. Sannicolo, "Applying Tropos Methodology to a real case study: Complexity and Criticality Analysis", in the Proceedings of the Second Italian workshop on "WOA 2002 dagli oggetti agli agenti dall'informazione alla conoscenza", Milano, 18-19 November 2002

[8] P. Giorgini, J. Mylopoulos, E. Nicchiarelli, R. Sebastiani. "Reasoning with Goal Models", in the Proceedings of the 21st International Conference on Conceptual Modeling (ER2002), Tampere, Finland, October 2002.