Bulletin de la Société Royale des Sciences de Liège, Vol. 85, 2016, *p.* 348 - 357

Planning in Self-Adaptive Systems by using Constraint Satisfaction Problem

Mehdi ZIARI[1,*], Ladan SAEIDI[2], Eslam NAZEMI[3]
[1] Shahid Beheshti University
[2] Shahid Beheshti University
[2] Shahid Beheshti University

Abstract

Being self-adaptive is a subject which helps software in systems in order to respond to the surrounded changes that are not fully known while designing. Uncertainty over time is a challenge we face with when designing a dynamic adaptive system (DAS). Dynamic adaptive systems keep observing their environment and try to adapt their behavior to the changes made in the environment. Different techniques and strategies are used in requirements and decision-making models in Self-Adaptive Systems (SAS_s). Goal-achieving strategies have different effects on non-functional requirements (soft goals). The final decision on choosing the strategy is based on the realization time calculation, and on the total of more success possibilities in satisfying the non-functional requirements. In this paper, a new method is presented for decision making, using Constraint Satisfaction Problems (CSP). Evaluation results show the superior performance of the design phase, using the proposed method in performing time and the accuracy of decisions.

Keywords: Self-adaptive systems, Non-functional Requirements, Planning, Constraint Satisfaction Problem

1. INTRODUCTION

A self-adaptive system can revise its behavior according to changes occurred in the environment. In fact, a self-adaptive system must continually have its environment under control and adjust itself accordingly. The subject is that it is not necessary for a self-adaptive system to monitor all angles and aspects of its environment. The question arises here is that if the system finds out its environment is not optimal enough, what must be done? Perhaps in such a situation the environment should maintain a series of high level requirements whose satisfaction is not dependant on the environmental conditions. What and how limited a self-adaptive system should do, is on requirements engineering. In fact requirements engineering should identify the possible adjustments and limitations they are compatible with. In other words, requirements engineering should encounter the present lack of pragmatism because the information about the operating environment in future is incomplete so the requirements may need changes in response to the environmental changes at runtime [1]. Therefore it is clear that the requirements are in a dynamic adaptive system and are constantly subject to changes over time [2].

Planning phase in self-adaptive systems, which is in fact an equivalent to Analysis and Designing, is responsible for providing and implementing specific demands for the self-management system. Therefore, the decision-making phase is essential for preserving self-redressing, self-optimizing, self-configuring and self-protecting in automated systems [3].

A self-optimizing system should be able to identify and adapt itself to the functions analysis. Performance-based strategies play an important role in introducing autonomic computing systems and communicate directly with the decision-making process. An autonomous system must find some ways to improve its performance, identifying and finding opportunities in order to do better and more efficiently in the field of costs and performance [3].

One of the main challenges a self-adaptive system is faced with when making decisions, is uncertainty over the operating time. Target models are used at model requirements and decision making in self-adaptive systems [4, 5, 6, 7]. The target models argue the performance of all functional and non-functional requirements [8]. The Probability Theory can also be used to explain the lack of complexity in naturally satisfying non-functional requirements. The strategy chosen to achieve the objective of a non-functional requirement can be dependent on satisfying that non-functional requirement [9].

The final decision about which strategy to be used, is based on calculating time and total of more successful possibilities in satisfying a non-functional requirement. Claim is already used [7, 8] as a mark over the uncertainty time. In fact Claim uses a framework called Dynamic Decision-making Networks (DDNs) [9]. DDNs process the development of decision-making networks, which process the development of Bayesian networks [9]. Decision-making networks develop Bayesian networks in order to provide a mechanism for rational decision making by combining possibilities. In fact DDNs provide a guiding principle for logical and rational decision making in uncertain and unknown environments [13]. In general, research in this field takes advantage of possibility devotion to target models [14].

Claim is used to express more clearly the designing assumptions that a system has faced with at runtime, and its impact is on achieving the goals of the system. It can be proved at runtime that some designing-time assumptions are either incorrect or have no credits at all. If additional information is available, Claim can to be used as a mark of uncertainty that can be resolved at runtime [9].

It is shown in [7] how Claims are used at runtime to raise the level of system satisfaction in the fulfillment of the objectives by choosing dynamically among goal-achieving strategies. Claim credit verification is based on observations and investigations. At runtime, when the observations prove Claim is not used yet, the adaptive system sets up itself with another goal-achieving strategy [9]. Although Claim guides us somehow to select the appropriate achieving strategy, but it is very high in error rate and consumption of time.

Indeed, this strategy acts on the probability that experts have assigned, and prioritizes goal-achieving strategies this way, then begins the job with the strategy which has the highest priority. In fact Claim is used as a stationary target to make assumptions; however, it has dynamic benefits, too [7].

Now this is possible that Claim faces with a wrong conclusion on the way to satisfy non-functional requirements (i.e. a conclusion opposite what Claim has expected). In such a situation Claim has to abandon this strategy and choose the next one with the highest priority, and start the job again.

Some situations may happen over the course, e.g. Either the objective and non-functional requirements are satisfied using the first strategy, or the wrong result forces us to use other strategies. It may go ahead and all the possibilities offered by Claim lead to errors, in which case we fail; in other cases, the consumed time to achieve the goal may change from low to high.

This article is a comprehensive and complete review of using CSP in decision-making, and shows the advantages of using CSP and being more efficient than Claim.

In this article a vacuum cleaner simulator and its environment and performance conditions are used to determine how non-functional requirements are influenced when Claim is used in the implementation process of the vacuum cleaner, and to what extent the goals are expected to be achieved. Then Claim is replaced with the CSP method and the benefits of its use are described. At the end, all the processes are gathered together in a table.

The remainder of this paper is organized as follows: CSP in section 2, the proposed idea: the use of CSP in planning in section 3, the description of a problem (a vacuum cleaner) in section 4, simulation in section 5, conclusion in section 6 and finally aknowledgment in section 7.

2. CONSTRAINT SATISFACTION PROBLEM

Constraint Satisfaction Problems are defined as a series of variables and limitations on the amounts these variables can select. To solve these problems a set of unique amounts must be devoted to the variables, so that all constraints are satisfied. And selecting the amounts of the variables should be in a way that different variable values are compatible with each other according to the defined constraints.

Further analysis of these issues done when determining the value of each variable, prevents future rollbacks and finds the solutions to the problem in fewer steps. Even under certain conditions some constraint satisfaction problems can be solved without rollback [10, 11].

Some constraints are applied by experts for all of the situations that the vacuum cleaner simulator has indicated that may encounter. These constraints cause all conditions that is considered to be well satisfied, in fact these constraints is our range problem. The decision-making carefulness can be considered absolutely flawless and one can be sure that it is not possible for algorithm to fail, and the ultimate goal, which is to clean up all the environment, happens very well because by applying the algorithm used here, neither a box is hidden for the vacuum cleaner, nor a situation happens in which the vacuum cleaner goes into an extreme box and the job remains unfinished because we know experts have applied some functions for all situations may happen, and they take the vacuum cleaner out of those situations successfully, and all conditions will be satisfied with respect to the variables .

3. THE PROPOSED IDEA: THE USE OF CSP IN PLANNING

CSP creates some limitations for existing conditions in order to achieve the goal. These conditions bound the achieving-goal strategy to obey only the condition which is intended for and not to act out of the constraints. So when an expert considers them for goal-achieving strategies, they can be sure the purpose as well as non-functional requirements will be fulfilled and also all conditions will be satisfied.So considering the conditions, the failure probability is zero and we will not face the return process. Only in equal conditions, if Claim goal-achieving strategy with the highest priority fails, it is assured that using CSP saves the time. But if this strategy succeeds, must specify the functions in CSP, and the time for satisfying the goal must be calculated and then compared with the first method.

However, if the constraints are applied with high skills and the best constraint is considered for each situation, it is sure that using CSP acts more optimal than Claim in any situations.

In the vacuum cleaner case, the method of using Claim proposals has already been verified. But in the CSP method, the vacuum cleaner operates in a way that it may encounter obstacles, wasted materials, and clean boxes while moving to clean the designated path fully. Now CSP has to define

some constraints for these situations with regard to the issue of conditions and variables of the problem.

If we assume the environment is a 5 x 5 matrix, for example, and the vacuum cleaner gets to an obstacle at the end of the first row, which direction should it choose in order not to cause a problem in cleaning the other boxes and to be the most efficient way as much as possible? Therefore some constraints can be considered for other boxes.

Studies show that CSP increases speed and success possibilities in dynamic adaptive systems with constraints applying some functions for different conditions at runtime.

4. THE DESCRIPTION OF A PROBLEM (A VACUUM CLEANER)

A vacuum cleaner robot has been used for this part. It is assumed that this robot is in a 5 x 5 matrix and is aimed at cleaning the entire matrix. Two non-functional requirements can be considered for this robot (In this example non-functional requirements are the conditions of CSP problem):

1. To avoid hitting obstacles along the way (damaging
   People in the house)
2. To reduce the energy costs
   The goal (cleaning the house) can happen using two different implementation strategies:

1. To clean at night
2. To clean when the house is empty

At runtime, if the vacuum cleaner finds out someone is home or it may hit an obstacle, Claim avoids hitting obstacles, and the reasoning engine evaluates the results and selects the cleaning strategy of 'when someone is not at the house', which is of a lower priority. In this case, infrastructure monitoring realizes at runtime that not only the energy cost, which is a non-functional requirement, is not reduced, but also the vacuum cleaner may hit other obstacles at the house; so the second non-functional requirement may not be satisfied as well. In such a case, the goal (cleaning the house) fails because there are not any other options suggested by Claim that even have a lower priority. Even if the third proposal with a lower priority was available and it met the main goal and also the non-functional requirements, again it is clear how much time and effort would be spent on reviewing the proposals, and how much delay would occur in achieving the goals.

In general, in this method if the vacuum cleaner faces an obstacle, some options will be proposed by Claim. According to these proposals, some possibilities are given to different paths by the experts. In this case, the vacuum cleaner chooses the path that has the highest possibility at the first place and then continues its main job. If the proposed path does not satisfy the goal, which is passing the obstacle and verifying all the boxes in the matrix and cleaning them, the vacuum cleaner will have to come back all the way it has already taken and choose the next highest-priority option offered by Claim and do the same job again. Another situation that may arise for the vacuum cleaner is to fall in an infinite loop, in which case the goal has failed.

5. SIMULATION

Simulation for the vacuum cleaner is started using the language C#. A 5 x 5 matrix is created using DataGridView to represent the environment in which the vacuum cleaner is supposed to begin its job. This matrix receives some information randomly from a file to specify in which box the

wasted materials (No. 1) and in which boxes the obstacles are (No. 2), and which boxes are clean (No. 3) and do not need any specific activities by the vacuum cleaner.

The vacuum cleaner move is considered linearly, which means if it does not face with any obstacles in the whole way, it starts from the box (0,0) and moves towards right and when it gets to the box (0,4), it will move down and then to the left, and so on until it gets to the end of the matrix, and its only job in this case is to get the waste and clean the room (box) in which it is located.

Now if the vacuum cleaner faces obstacles along the way, the case is different; here CSP arises and the constraints for various conditions matter more. The general algorithm intended for this mode is that the vacuum cleaner constantly checks the box ahead and whenever notices there is an obstacle in the next box and it cannot move forward, then makes a semi-circle move around the box and gets to the next one. In fact, when the vacuum cleaner notices an obstacle in the next box, calculates the location of the one after the obstacle and tries to get there. This algorithm has been considered in order not to miss any boxes by the vacuum cleaner and to make sure that all boxes have been checked. Another advantage of this method is that when we get to the end of the path, there is no need to go back and check boxes to make sure they are all clean. In Fig. 1 you can see a state in which the vacuum cleaner is faced with an obstacle and the algorithm it implements.

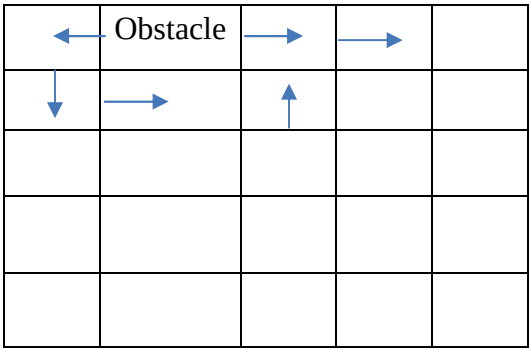


Fig. 1: An example of the vacuum cleaner dealing with obstacles

In this case, when the vacuum cleaner is at the box (0,0), it notices that there is an obstacle in the next box. So instead of moving forward, it goes down on the path and turns around the obstacle and gets to the box next to the obstacle and continues its main way. When obstacles are placed in the boxes at the sides, certain conditions occur for each one, that are in our control. For example, in Figure 2 you can see one of these states:

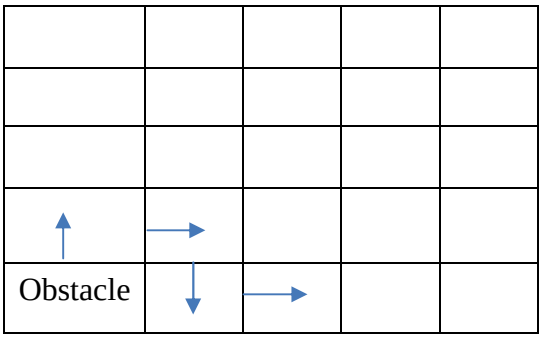


Fig. 2: An example of the vacuum cleaner dealing with obstacles in the next box

By applying these constraints, for every case which happens, there are some conditions already applied by experts, so we are sure that the vacuum cleaner does not fall in an infinite circle and can continue its job. We are also sure that the whole environment is checked by the vacuum cleaner and there is no need to a rollback in the middle or at the end of the path.

Now two graphs are drawn to compare CSP and Claim, and to show the more optimal performance of CSP. Fig. 5 shows the number of obstacles on the horizontal axis and time on the vertical one. The intended space is a 5 x 5 matrix. This graph is intended in identical conditions for both methods, i.e. the place of obstacles in calculating the runtime in CSP and Claim has been the same.

Runtime for facing one to five obstacles is calculated in CSP and shown in blue in Fig. 5. There are also two proposals by Claim in this evaluation and the result is shown in red :

- When the vacuum cleaner gets to an obstacle, it shifts moving towards down and then continues its path. If the obstacle is at the end of the matrix, it moves back for one box, and then goes down. When it gets to the last one, it goes up in the opposite direction to the box next to the obstacle and then returns to the last box. This is shown in Fig. 3.
- If the vacuum cleaner notices that the first proposal is violated, when it gets to the last box, it takes the whole path in the opposite direction and makes sure all the boxes are clean, and then returns. This is shown in Fig. 4.

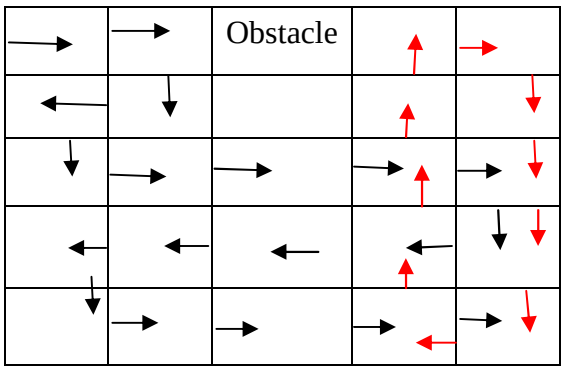
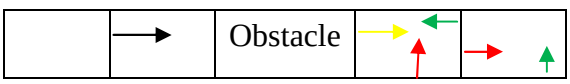


Fig. 3: The first proposal in Claim (black arrows first, then red)



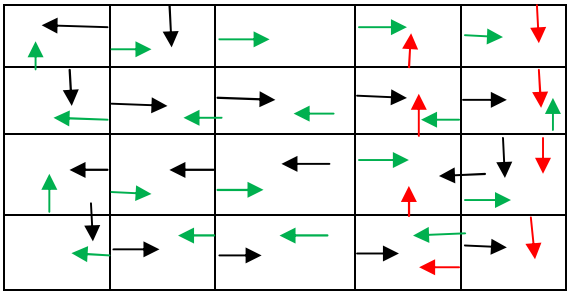


Fig. 4: The second proposal in Claim
(black arrows first, then red, then green, return to the yellow arrow)

Fig. 6 is also brought after Fig. 5 All the terms but one in this figure are considered exactly identical to the ones in Fig. 5. The only difference is the number of boxes in the matrix, which is considered a 6 x 6 one in this figure. But the result is again similar to that of Fig. 5. In fact it can be said that Claim is not dependent on the number of boxes. It does not mean that if there are more than two obstacles in a 6 x 6 matrix, Claim fails and our goals are not achieved. But in this case, CSP keeps going based on the constraints experts have intended for it with regard to the conditions of the problem and achieves our goals, and the number of obstacles cannot be a disruption in CSP.

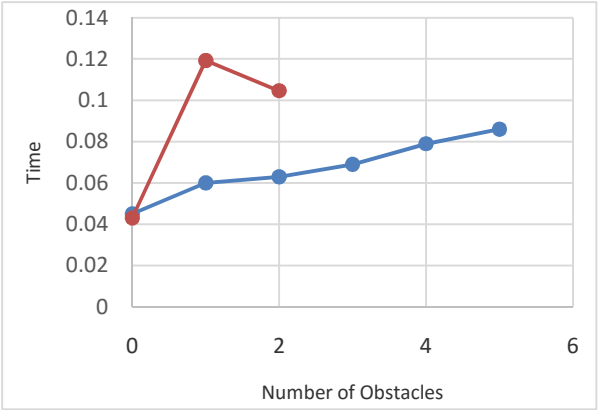


Fig. 5: Number of obstacles in time (blue: CSP - red: CLAIM, space is 5 x 5)

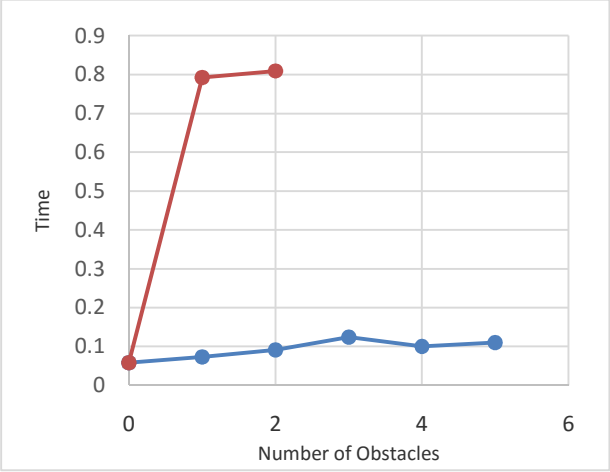


Fig. 6: Number of obstacles in time (blue: CSP - red: CLAIM, space is 6 x 6)

In Table I you can see a Comparison of Algorithms. If three or more obstacle exists in the path, Claim is violated after some moves and the cleaning up fails.

Table 1: Time span on cleaning up the whole environment (*space is 6 x 6*)

The Algorithm	*One obstacle*	*Two obstacles*	*Five obstacles in the path*
CSP	**0.05671 s**	**0.06082 s**	**0.08599 s**
Claim	**0.11093 s**	**0.10468 s**	**Fails after some moves**

Based on the obtained results, it is found out that in CSP:

- Decision-making accuracy based on the degree of accuracy in applying the constraints in order to satisfy the conditions, can be quite high.
- The final result is to achieve all the goals of the system.
- The imposed constraints in this article are in a way that there will not be any rollbacks.
- The consumed time and energy is much lower than that of Claim.

And it can be summarized for Claim that:

- If the first proposed path by Claim succeeds and achieves our goal, the runtime must be calculated in both methods to see which one has functioned more efficiently.

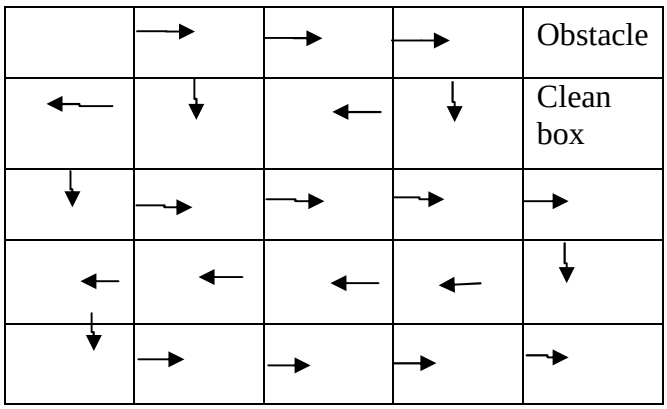


Fig. 7: An example of Claim's success in the first proposed path

In Fig. 5, the obstacle is at the last box. Assuming the box underlying the obstacle is clean, it can be said that Claim's first proposed path is a success. Now comparing the runtime in each method determines which one is more optimal.

- If Claim's first proposed path fails and the vacuum cleaner is forced to go back and check the next path, assuming that the second proposal will be a success, again we can be sure that CSP will certainly be more efficient than Claim, because the obstacle moves more in Claim than in CSP.
- Claim has usually limited proposals and there is possibility for failure in this way, but we are sure that goal is achieved in CSP.

It can be finally claimed that CSP functions more efficiently by increasing the number of either obstacles or boxes in a matrix. But in Claim, regardless of increasing the number of boxes in a matrix, by putting more than three obstacles in the path, and with respect to its proposed models described earlier in section 6, it fails and the goals are not achieved.

6. CONCLUSION

In this article we looked at self-adaptive systems and offered the two methods CSP and Claim in decision-making phase and checked them. Claim had already been used as a mark in self-adaptive systems. Showing the advantages of using CSP and checking them in details, we proved that, at the same conditions in a self-adaptive system, using CSP improves the consumed time and energy in decision-making. According to the assessments made, using this method assures us while making decisions that the job will certainly be done and will not fail along the way. However, using Claim may fail in the middle of the way and main goals and non-functional requirements will not be achieved and satisfied.

7. AKCNOWLEDGMENT

It is necessary to appreciate of Self-Star Research group of Shahid Beheshti University (Faculty of science and computer engineering) that works on Self-Adaptive software systems.

References

[1] B.H. Cheng, R. de Lemos, H. Giese, P. Inverardi, J. Magee, J. Andersson, B. Becker, N. Bencomo, Y. Brun, B. Cukic, G.D.M. Serugendo, S. Dustdar, A. Finkelstein, C. Gacek, K. Geihs, V. Grassi, G. Karsai, H.M. Kienle, J. Kramer, M. Litoiu, S. Malek, R.Mirandola, H.A. M¨uller, S. Park, M. Shaw, M. Tichy, M. Tivoli, D. Weyns and J. Whittle, "Software Engineering for Self-Adaptive Systems: A Research Roadmap," Software Engineering for Self-Adaptive Systems. Lecture Notes in Computer Science, vol. 5525, pp. 1–26. Springer, 2009.

[2] R. de Lemos, H. Giese, A. Hausi, H.A. M¨uller, M. Shaw,J. Andersson, M. Litoiu, B. Schmerl, G. Tamura, N. Villegas, TH. Vogel and D. Weyns, "Software Engineering for Self-Adaptive Systems: A Second Research Roadmap, " Software Engineering for Self-Adaptive Systems II, pp. 1-32. Springer Berlin Heidelberg, Springer, 2013.

[3] M. Maggio, H. Hoffmann, M. Santambrogio, A. Agarwal and A. Leva, A Comparison of Autonomic Decision Making Techniques, s. Technical Report MIT-CSAIL-TR-2011-019, Massachusetts Institute of Technology. USA, Massachusetts April, 2011.

[4] H.J. Goldsby, P. Sawyer, N. Bencomo, D. Hughes and B.H. Cheng, "Goal-based modeling of dynamically adaptive system requirements, " IEEE Int. Conference on the Engineering of Computer Based Systems, ECBS,2008.

[5] N.A. Qureshi, and A. Peini, "Engineering adaptive requirements," Workshop on Software Engineering for Adaptive and Self-Managing Systems, SEAMS 2009.

[6] A. Lapouchnian, "Exploiting Requirements Variability for Software Customization and Adaptation," Ph.D. thesis, University of Toronto 2011.

[7] K. Welsh, P. Sawyer and N.Bencomo, "Towards requirements aware systems: Runtime resolution of design-time assumptions," Automated Software Engineering (ASE), 2011 26th IEEE/ACM International Conference , pp. 560–563, 2011.

[8] L. Chung, B.A. Nixon, E. Yu, and J. Mylopoulos, Non-Functional Requirements in Software Engineering, vol. 5. Springer Science & Business Media, 2012.

[9] B. Nelly and B. Amel, Supporting Decision-Making for Self-Adaptive Systems: From Goal Models to Dynamic Decision Networks, Springer-Verlag Berlin, 2013.

[10] E. Freuder, "A Sufficient Condition for Backtrack-Free Search," Journal of the Association for Computing Machinery, Vol. 29,PP.24-32, 1982.

[11] A. Mackworth, "Consistency in Network of Relations," Artificial Intelligence Journal, Vol.8, PP.99-118, 1977.

[12] J. Pearl, Probabilistic reasoning in intelligent systems: networks of plausible inference. Morgan Kaufmann Publishers Inc., San Francisco, 1988.

[13] S.J. Russell and P. Norvig, Artificial intelligence: A modern approach, 2nd edn. Prentice Hall series in artificial intelligence. Prentice Hall, 2003.

[14] E. Letier, and A. van Lamsweerde, Reasoning about partial goal satisfaction for requirements and design engineering. SIGSOFT Softw. Eng. Notes 26, 2004.