The Organizational Implementation of Information Systems: Towards a New Theory

Prof.Aashish Hasmukhbhai Kacha
Computer Science &Engineering, Ph.D. Scholar in Rai University, Ahmedabad
Campus Director at Shree Swaminarayan Gurukul Gynbag International School, Junagadh

Abstract

We intuitively refer to the term planning as the deliberation process that chooses andorganizes actions by anticipating their expected effects. This deliberation aims at Satisfying some pre-defined requirements and achieving some prestated objectives. The intuition is that actions are executed in a given domain. They make the domainevolve and change its state. For instance, in a robot navigation domain, an actionmoving the robot changes its position; in the case of a microprocessor, an instruction
can be viewed as an action that changes the value of the registers; a web service forbooking flights can receive a message with a flight reservation confirmation, and thisis an action that changes its state.The deliberation process can organize actions in different ways. For instance, movinga robot to a given room and then to the corridor is an example of a sequentialorganization of actions; executing an instruction depending on the result of the executionof a previous one is an example of a conditional combination of actions;requesting for a flight reservation until a seat is available is an example of an iterativecombination.Actions are organized and combined with the aim to satisfy some requirements on the evolution of the domain. An example of a requirement for a mobile robot is that of "reaching a given room", while a requirement for a flight service can be that of "never exceeding a given number of overbooking". Automated Planning is the area of Artificial Intelligence that studies this deliberation process computationally. Its aim is to support the planning activity by reasoning on conceptual models, i.e., abstract and formal representations of the domain, of the effects and the combinations of actions, and of the requirements to be satisfied and the objectives to be achieved. The conceptual model of the domain in which actions are executed is called the planning domain, combinations of actions are called plans, and the requirements to be satisfied are called goals. Intuitively, given a planning domain and a goal, a planning problem consists in determining a plan that satisfies the goal ina given domain.The framework is defined along the three main components of the planning problem:domains, plans, and goals.

Domains

We allow for nondeterministic domains, i.e., domains in which actionsmay have different effects, and it is impossible to know at planning timewhich of the different possible outcomes will actually take place. We also allowfor partial observability. It models the fact that in some situations the stateof the domain cannot be completely observed, and thus cannot be uniquely determined.A model with partial observability includes the special cases of fullobservability, where the state can be completely observed and thus uniquely determined,and that of null

observability, where no observation is ever possible atrun time.

Plans

We define plans where the action to be executed in a given state can dependon available information about the history of previous execution steps. Thedefinition is general enough to include sequential plans, i.e., plans that are simplysequences of actions, conditional plans, i.e., plans that can choose a differentaction depending on the current situation at execution time, iterative plans thatcan execute actions until a situation occurs. We can have plans that depend ona finite number of execution steps (finite-memory plans), as well as plans thatdo not depend on the previous execution steps (memory-less plans). In general,plan executions result in trees (called execution trees) whose nodes correspondto states of the domain.

Our framework is general enough to represent a relevant and significant set ofplanning problems. Classical planning can be modeled with deterministicdomains, plans that are sequences of actions, and reachability goals. Inaddition, our framework is well suited for modeling certain forms of planning underuncertainty and incomplete information, which are being recently addressed in theresearch literature and are relevant to several real-world applications. Indeed, nondeterministicdomains model uncertainty in action effects, while partial observabilitymodels uncertainty in observations. For instance, the so-called conformant planning can be modeled with nondeterministic domains, null observability,sequential plans, and reachability goals. Contingent planning canbe modeled with nondeterministic domains, conditional plans, and reachability goals.Planning for temporally extended goals can be modeled withnondeterministic domains, history dependent plans, and goals that represent desiredevolutions of the domain.

Related Work

The use of interfaces is well known across severalprogramming languages and software programming concepts, downloaded components. For a mission critical system it isimportant to ensure the safety and reliability of the host systemover which these safety mechanisms have been implemented.There are several issues in the way components become faulty,one of the most common being cyber attacks. One possiblescenario is that the components may be altered or replaced withmalicious content while they are still in transit over a network.To ensure that the components are received in the form theyhave been sent is our goal while downloading (note that this isnot a way of ensuring the credibility of the source from whichthese components are downloaded). As noted earlier,traditional approaches on detecting vulnerability would simplyblock or prevent the faulty component from executing in thehost environment. This is not the solution to our problem, asmission-critical systems cannot afford complete blocking ofcomponents which were probably from a credible source butaffected by malicious content during their transit (over anetwork, etc). The whole concept of survivability hinges on thebasis of providing the system with capabilities to continueexecution of the component after faulty parts have beensuccessfully removed (or rendered harmless) by providinggood recovery

schemes. Thus the focus is on improvedaccuracy in detection and recovery of malicious components—a must for effective survivability. The dynamic testingstrategies described below are collectively termed as MultipleAspect Testing due to the nature of the testing itself—a multifunctionaland extendible approach designed to achieve moreof such accuracy in detection and recovery.

In large mission-critical systems where speed andperformance is a vital concern, it is best to filter potentiallymalicious content even before the system proceeds to test for

such content. As a first level of filtering in our system, we canhave the system download multiple copies of each componentrequired by the host machine (preferably onto multiple parallelmachines or servers) and check for coherence in their contentby comparing certain parameters. Thus for every component Nwe have several downloads, say N1, N2, N3 and N4. Each ofthese components are checked for parameters such ascomponent size, last modified time, number of modulescontained in each, source information, etc., and a comparison ismade between each of them to check for coherence or matchesin the values for each mentioned parameter. Now, if componentN4 does not match the properties of the other downloadedcomponent copies N1, N2 and N3, then since the latter form amajority—in this case, 75%, we can discard N4 by assuming

that it may be a component whose contents may have beenmodified over the network during its transit. The percentage ofacceptance here, which we call acceptance threshold, was 75%,as we saw already, but this could vary based on the criticalityof the system. For example, an extremely critical system wouldhave an acceptance threshold of 90%, requiring nine out ofevery ten component copies to hit concurrency in the parametervalues (thus consequently increasing the number of copiesdownloaded for each component download). It is extremelyimportant to note that although the valid components are

accepted in this case, they are far from being allowed toexecute in the host machine, because this technique doesnothing to establish the validity of the source site. Thus, one ormore of the accepted components can be marked in this casefor further testing as described below.

Establishing Source Credibility

The acceptance threshold can be used to obtain statistics onthe credibility of the sites from which the components are being downloaded. For example, if a particular site consistentlyproduces low acceptance rates, then the problem may not justbe with the network-based attacks. The components emanatingfrom the source site itself may be implemented withinconsistencies and malicious content for some copies of each

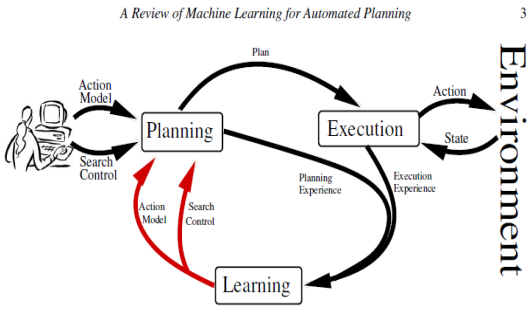
component copy sent. Thus, we can define a critical acceptancethreshold here that is a value below which a source site wouldbe marked not trustworthy, and all further downloads from thissite would be blocked. Thus the system would have toconsistently incorporate decision-making based on past experience as well.

Automated planning

Automated Planning (AP) is the branch of Artificial Intelligence that studies the computational synthesisof ordered sets of actions that perform a given task. AP appeared in the late ’50s as the result of studiesinto state-space search, theorem proving and control theory to solve the practical needs of roboticsand automatic deduction. The STanford Institute Problem Solver STRIPS (Fikes and Nilsson, 1971),developed to be the planning component for controlling the autonomous robot Shakey (Nilsson, 1984),perfectly illustrates the interaction of these influences. From the days of Shakey to now, AP has produced
accepted standards for representing planning tasks and efficient algorithms for solving them (Ghallabet al., 2004). In the last decade AP systems have been successfully applied to real world problems
such as planning space missions (Nayak et al., 1999), managing fire extinctions (Castillo et al., 2006)or controlling underwater vehicles (Bellingham and Rajan, 2007).Despite these successful examples, the application of AP systems to real world problems is stillcomplicated, mainly because of two knowledge definition problems:

Automated planners require an accurate description of the planning task. These descriptions includea model of the actions that can be carried out in the environment, a specification of the state of theenvironment and the goals to achieve. In the real world, the execution of an action may result innumerous outcomes, the knowledge of the state of the environment may be partial and the goals maynot be completely defined. Generating exact definitions of the planning tasks in advance is unfeasiblefor most real-world problems.

_ Off-the-shelf planners often fail to scale-up or yield good quality solutions. Usually, the search for asolution plan in AP is a PSpace-complete problem (Bylander, 1991, 1994). Current state-of-the-artplanners try to cope with this complexity through reachability analysis by (1) grounding actions and(2) performing a search process guided by domain-independent heuristics. Nevertheless, when thenumber of objects is large, the resulting ground search trees cannot be traversed in a reasonable time.Moreover, where heuristics are



poorly informed –like in the case of some domains with strong subgoals
interactions– this kind of analysis is misleading. Domain-specific search control knowledgehas been shown to improve the scalability of planners in these situations (Bacchus and Kabanza,2000; Nau et al., 2003). Defining search control knowledge is usually more difficult than definingthe planning task because it requires expertise, not only in the task to solve, but also in the planningalgorithm.

AP aims to produce solvers that are effective when faced with different classes of problems. AP producessolvers that exploit environment dynamics models to reason on different tasks in different environments.An AP task is defined by

two elements:The domain, that consists of the set of states of the environment S together with the set of actions Athat indicates the transitions between these states. The problem, that consists of the set of facts s0 2 S that indicates the initial state of the environmentand the set of facts G _ S that indicates the goals of the AP task.A solution to an AP task is a sequence of actions a1; : : : ; an. This sequence of actions corresponds to asequence of state transitions s0; : : : ; sn such that ai is applicable in state si□1 and produces state si, sn isa goal state; that is G _ sn. The cost of the solution is the sum of the action costs. A solution is consideredoptimal when it minimizes the expressionPni=1 cost(ai).

Specifying accurate action models to address AP tasks in the real world is a complex exercise. Even intraditionally easy-to-code planning domains– like Blocksworld– it is hard to specify the actions' potentialoutcomes when the environment is non-deterministic. In extreme cases, like planning for the autonomouscontrol of a Mars Rover (Bresina et al., 2005) or an underwater vehicle (McGann et al., 2008), thisknowledge cannot be specified because it is as of yet unknown. In these situations, the success of the APsystems fully depends on the skills of the experts who define the action model. With the aim of assistingexperts, the planning community is developing systems for the acquisition, validation and maintenance ofAP models. ML seems to be a useful tool for these systems to automate the extraction and organization ofknowledge from examples of plans, execution traces or user preferences.When the AP task is accurately defined, it looks like it should be easy to tackle by searching for a path ina state-transition graph, a well-studied problem. However, in AP this state-transition graph often becomesso large that search is intractable. The complexity of the algorithms to solve this type of problems growswith the number of states, which is exponential in the number of problem variables (number of objectsand predicates of the problem). Prior to the mid '90s, planners were unable to synthesize plans with morethan ten actions in an acceptable amount of time.In the late 90's, a significant scale-up in planning took place due to the appearance of the reachabilityplanning graph (Blum and Furst, 1995). This discovery allowed for the development of powerful domain independentheuristics (Hoffmann and Nebel, 2001a; Bonet and Geffner, 2001) that were computablein polynomial time. Recent discoveries– such as the automatic extraction of search landmarks (Porteousand Sebastia, 2004) and the automatic construction of symbolic Pattern Data Bases (Edelkamp, 2002) improve planners' speed and quality performance. Planners using these advances can often synthesize one hundredaction plans in seconds, yet scalability limitations are still present in AP and even well-studieddomains –like the Blocks world– become challenging for planners when there is a relatively large numberof objects. On the one hand, current domain-independent heuristics are expensive to compute. This effectis more evident in domains where heuristics are misleading. In these domains planners spend most oftheir planning time computing useless node evaluations. On the other hand, given that these domain independenttechniques are based on action grounding, the planners' search trees become intractable whenthe number of problem objects and/or

action parameters reaches a certain size. These problems makeDomain-independent planners' application to various real problems difficult. Logistics applications needto handle hundreds of objects as the same time as hundreds of vehicles and locations (Florez et al., 2010)which makes computing evaluation functions in every search node unfeasible. In this case, ML has a roleto play in capturing useful control knowledge skipped by the domain-independent techniques.

Learning Planning Action Models

AP algorithms reason about correct and complete action models that indicate the state-transitions ofthe world. Building planning action models from scratch is difficult and time-consuming, even for AP experts. An alternative approach is to use ML so people do not have to hand-code the action models.This section reviews ML techniques for the automatic definition of action models for AP. The reviewclassifies the techniques by the stochasticity of the actions effects and by the observability of the state ofthe environment.

1. Action effects. In many planning tasks deterministic world dynamics cannot be assumed. This is thecase with planning domains that include stochastic procedures such as the toggling of a coin or therolling of a dice, or non-deterministic outcomes, such as robot navigation in the real world.

2. State observability. In many planning tasks handling a complete and exact description of the stateof the environment is inconceivable. Parts of the current state may be confused or missing due tosensors' failures or to their inability to completely sense the world. For example, when controlling arobot in the real world.Accordingly, we defined four categories for AP modelling: deterministic actions in fully observableenvironments; deterministic actions in partially observable environments; stochastic actions in fullyobservable environments; and stochastic actions in partially observable environments. Despite the factthat other classifications are possible– like for instance by grouping according to the the learningtarget (preconditions, effects, conditions of effects, probabilities of outcomes,) we believe thisone is useful for planning purposes because each class corresponds to a different planning paradigm.the classification of the planning action modelling systems by giving some exampleimplementations. This table does not intend to be an exhaustive enumeration so the systems in the tableare just a sample. The rest of the section describes the systems belonging to each of the four categories indetail.

The Learning Task

1. Knowledge representation. In terms of AP, the suitable languages for representing deterministicactions are the languages defined for classical planning. The Planning Domain Definition Language(PDDL) (Fox and Long, 2003; Thi´ebaux et al., 2005; Gerevini et al., 2009b) is the standardrepresentation language for classical planning. It was created as the planning input language forthe International Planning Competition (IPC1) to standardize the planning representation languagesand facilitating comparative analysis of diverse planning systems. Although several of the systemsreviewed in this paper use other representation languages, we have used PDDL to illustrate ourexamples. A PDDL action a consists of a tuple

MODEL	FEATURES		IMPLEMENTATIONS
	Strengths	Weakness	
Deterministic effects **Full** state observability	• Learning complexity is theoretically bounded • Efficient planning algorithms • Complete coverage of the learning examples	• Poor expressiveness	LIVE (Shen and Simon, 1989), EXPO (Gil, 1992), OBSERVER (Wang, 1994)
Deterministic effects **Partial** state observability	• Complete coverage of the learning examples	• Poor expressiveness • Inefficient planning algorithms	ARMS (Yang et al., 2007), (Amir and Chang, 2008), (Mourão et al., 2008), LOCM (Cresswell et al., 2009)
Probabilistic effects **Full** state observability	• Rich expressiveness • Efficient planning algorithms	• Non-existent online learning	(Oates and Cohen, 1996), TRAIL (Benson, 1997), LOPE (García-Martínez and Borrajo, 2000), (Pasula et al., 2007), PELA (Jiménez et al., 2008)
Probabilistic effects **Partial** state observability	• Rich Expressiveness	• High planning and learning complexity	(Yoon and Kambhampati, 2007)

Implementations of systems for planning action modelling

<head(a),pre(a),eff(a)> where,_ head(a) is the head of action a containing the action name and the list of typed parameters,pre(a) is the set of action preconditions. This set represents the facts that needs to be true for theapplication of action a and,eff(a) is the set of action effects. This set includes the facts that are no longer true after theapplication of action a (known as del effects and represented by del(a)) and the facts made trueby the application of action a (known as add effects and represented by add(a)).

Implementations

The LIVE system (Shen and Simon, 1989) was an extension of the General Problem Solver (GPS)framework (Ernst and Newell, 1969) with a learning component. LIVE alternated problem solving withmodel learning to automatically define operators. The decision about when to alternate depended onsurprises, that is situations where an action effects violated its predicted model. EXPO (Gil, 1992)generated plans with the PRODIGY system (Minton, 1988), monitored the plans execution, detecteddifferences in the predicted and the observed states and constructed a set of specific hypotheses to fix thosedifferences. Then the EXPO filtered the hypotheses heuristically. OBSERVER (Wang, 1994) learnedoperators by monitoring expert agents and applying the version spaces algorithm (Mitchell, 1997) to theobservations. When the system already had an operator representation, the preconditions were updated byremoving facts that were not present in the new observation's pre-state; the effects were augmented byadding facts that were in the observation's delta-state.All of these early works were based on direct liftings of the observed states. They also benefit from

experience beyond simple interaction with the environment such as exploratory plans or external teachers,but none provided a theoretical justification for this second source of knowledge. The work recentlyreported in (Walsh and Littman, 2008) succeeds in bounding the number of interactions the learner mustcomplete to learn the preconditions and effects of a STRIPS action model. This work shows that learningSTRIPS operators from pure interaction with the environment, can require an exponential number ofsamples, but that limiting the size of the precondition lists enable sample-efficient learning (polynomial inthe number of

actions and predicates of the domain). The work also proves that efficient learning is alsopossible without this limit if an agent has access to an external teacher that can provide solution traces ondemand.In the review we focused on learning STRIPS-like action models but others systems have tried to learnmore expressive action models for deterministic planning in fully observable environments. Exampleswould include the learning of conditional costs for AP actions (Jess Lanchas and Borrajo, 2007) or thelearning of conditional effects with quantifiers (Zhuo et al., 2008).

Conclusions

Today encouraged by the applications of AP to real-world problems and the maturity of relationallearning, there is a renewed interest in learning for planning. In 2005 the International Competition onKnowledge Engineering for Planning Systems (ICKEPS) began and in 2008 a learning track was openedat IPC.Workshops on planning and learning have taken place periodically at the International Conference on Automated Planning and Scheduling. ML seems to be once again one of the solutions to the challengesof AP.In the paper we focused on two of AP's challenges: defining effective AP action models and definingsearch control knowledge to improve the planners performance. For the first challenge, we reviewedsystems that learned action models with diverse forms of preconditions and effects. These systemsgenerally present learning algorithms that are effective when the appropriate learning examples arecollected although it is still not clear how to automatically collect good sets of learning examples inAP. When learning results in imperfect action models, there are few mechanisms for diagnosing the flawsof the models or algorithms to robustly plan with them.For the second challenge, we reviewed different forms of search control knowledge to improve off-the shelfplanners. This kind of knowledge has been shown to boost the performance of planners in particulardomains. Learning effective search control knowledge over a collection of domains is still challengingsince different planning domains may present very different structures. We also reviewed learning searchcontrol for HTN planners which is a more expressive form of control knowledge that uses a different planning paradigm based on domain-specific problem decomposition. In this planning paradigm, searchcontrol knowledge and action models are not separated in the domain theory's definition.

The paper also reviewed techniques for RRL, a form of RL that, like AP, uses predicate logic torepresent states and actions. Though current RRL techniques can solve relational tasks, they are focused on achieving a particular set of goals and present generalization limitations in comparison with learningfor AP techniques. Furthermore, they have problems collecting significant experience for complex taskslike the ones traditionally addressed in AP, where achieving a particular goal may undo previously satisfiedgoals.

The application and combination of new ML algorithms for AP is still an openissue. ML is constantly producing new learning algorithms with potential applications for AP. In thecase of relational learning AP has only benefited from algorithms for classification and regressionbut existing

relational learning algorithms cover almost all machine learning tasks. Algorithms forrelational clustering or for inferring association rules (Raedt, 2008) have still not been explored inAP. ML algorithms for propositional data can be adapted to the relational setting used in AP (Mour˜aoet al., 2010). Another example of applying new ML algorithms to AP is the use of kernel functionsto effectively match planning instances (Serina, 2010). Despite all this new ML technology, learningalgorithms present biases that affect the learning process. Research in ML for AP can take up newresearch directions to study the combination of different learning algorithms to reduce the negativeeffect of a given algorithm bias.

This review focused on learning two inputs of the AP process, the action model and knowledge forsearch control. There is a full set of techniques that aim to capture information about the third input to theAP process, the problem. Plan recognition techniques (Charniak and Goldman, 1993; Bui et al., 2002) tryto infer an agent's goals and plans by observing the agent's actions. The plan recognition task traditionallyassumes there is a library with the space of possible plans to be recognized but recent works do not requirethe use of this library. These works (Ram'ırez and Geffner, 2009; Ram´ırez and Geffner, 2010) can alsocompute the goals that account for the observed examples using slightly modified planning algorithms.Further studies are needed to analyze the relationship of plan recognition techniques with the learningtechniques for AP reviewed in this paper.

References

I. Aler, R., Borrajo, D., and Isasi, P. (2002). Using genetic programming to learn and improve control knowledge. Artificial Intelligence, 141(1-2):29–56.

II. Amir, E. and Chang, A. (2008). Learning partially observable deterministic action models. Journal of Artificial Intelligence Research, 33:349–402.

III. Bacchus, F. and Kabanza, F. (2000). Using temporal logics to express search control knowledge for planning. Artificial Intelligence, 116(1-2):123–191.

IV. Barto, A. and Duff, M. (1994). Monte carlo matrix inversion and reinforcement learning. In Advances in Neural Information Processing Systems 6, pages 687–694.

V. Bellingham, J. and Rajan, K. (2007). Robotics in remote and hostile environments. Science, 318(5853):1098–1102.

VI. Bellman, R. and Kalaba, R. (1965). Dynamic Programming and Modern Control Theory. Academic Press.

VII. Benson, S. S. (1997). Learning Action Models for Reactive Autonomous Agents. PhD thesis, Stanford University.

VIII. Bergmann, R. and Wilke, W. (1996). PARIS: Flexible plan adaptation by abstraction and refinement.

IX. In Workshop on Adaptation in Case-Based Reasoning. ECAI-96.

X. Bertsekas, D. P. (1995). Dynamic Programming and Optimal Control. Athena Scientific.

XI. Bertsekas, D. P. and Tsitsiklis, J. N. (1996). Neuro-Dynamic

Programming (Optimization and Neural Computation Series, 3). Athena Scientific.

XII. Blockeel, H. and De Raedt, L. (1998). Top-down induction of first-order logical decision trees. Artificial Intelligence, 101:285–297.

XIII. D. A. Fisher and H.F. Lipson, "Emergent Algorithms-A New Method for Enhancing Survivability in Unbounded Systems," Proceedings of the 32nd Annual Hawaii International Conference on System Sciences, Maui, HI, Jan. 5-8, 1999 (HICSS-32), IEEE Computer Society, 1999, http://www.cert.org/research/

XIV. J. Knight, M. Elder, and X. Du, "Error Recovery in Critical Infrastructure Systems," Proceedings of the 1998 Computer Security, Dependability, and Assurance (CSDA'98) Workshop, Williamsburg, VA, November 1998.

XV. H. Lipson and D. Fisher, "Survivability -- A New Technical and Business Perspective on Security," Proceedings of the New Security Paradigms Workshop (NSPW'99), Caledon Hills, Ontario, Canada, September 21-24, 1999.

XVI. N. Mead, R. Ellison, R. Linger, et al., "Survivability Network Analysis Method," SEI Technical Report: CMU/SEI-00-TR-013, September 2000.

XVII. Joon S. Park and PratheepChandramohan, "Component Recovery Approaches for Survivable Distributed Systems," In Proceedings of the 37th Hawaii International Conference on Systems Sciences (HICSS-37), Big Island, Hawaii, January 5-8, 2004.

XVIII. Joon S. Park and Judith N. Froscher, "A Strategy for Information Survivability," In Proceedings of the 4th IEEE/CMU/SEI Information Survivability Workshop (ISW), Vancouver, Canada, March 18-20, 2002.

XIX. M. Chen, E. Kiciman, E. Brewer, and A. Fox, "Pinpoint: Problem Determination in Large, Dynamic Internet Services," In Proceedings of the IEEE International Conference on Dependable Systems and Networks, DSN, 2002.

XX. M. Milenkovi?, A. Milenkovi?, and E. Jovanov, E, "Using instruction block signatures to counter code injection attacks," ACM SIGARCH Comput. Archit. News, 33, 1, Mar. 2005.

XXI. D. L. Oppenheimer and M. R. Martonosi, "Performance Signatures: A Mechanism for Intrusion Detection," Proceedings of the 1997 IEEE Information Survivability Workshop, San Diego, California, 1997.

XXII. Joon S. Park, GautamJayaprakash, and Joseph Giordano, "Component Integrity Check and Recovery Against Malicious Codes," In Proceedings of the 20th IEEE International Conference on Advanced Information Networking and Applications (AINA), Workshop on Trusted and Automatic Computing Systems

(TACS), Vienna, Austria, April 18-20, 2006.

XXIII. A. Ghosh, J. Voas, "Inoculating Software for Survivability," Communications of the ACM, July 1999

XXIV. S. Jajodia, C. McCollum, and P. Ammann, "Trusted Recovery," Communications of the ACM, 42(7), pp. 71-75, July 1999.

XXV. P. Liu, P. Ammann, and S. Jajodia, "Rewring Histories: Recovering from Malicious Transactions," Distributed and Parallel Databases, 8(1), pp. 7-40, January 2000.