Increasing Software Infrastructure Dependability through a Law Enforcement Approach

Gustavo Carvalho¹, Rodrigo Paes¹, Ricardo Choren², Paulo Alencar³ and Carlos Lucena¹

¹Departamento de Informática – Laboratório de Engenharia de Software – PUC-Rio
Rua Marquês de São Vicente, 225 Rio de Janeiro, Brasil, 22453-900
{guga, rbp, lucena}@inf.puc-rio.br

²Departamento de Engenharia de Sistemas - Instituto Militar de Engenharia - IME
Praça General Tibúrcio 80, Rio de Janeiro, Brasil, 22290-270
choren@de9.ime.eb.br

³University of Waterloo, Computer Science Department, Computer Systems Group, Waterloo, Ontario, N2L 3G1 Canada
palencar@csg.uwaterloo.ca

Abstract*

Software systems are increasingly becoming distributed, open and ubiquitous assets. While open system components are often autonomous, they behave unpredictably when unforeseen situations arise. Taming this uncertainty is a key issue for dependable open software development. This work proposes a law enforcement approach that uses risk analysis to develop dependable open systems. We present law enforcement as a suitable technique to deal with dependability requirements in open systems. Laws impose execution rules and limits, creating a boundary of tolerated component autonomy and fostering the development of trusted systems. We also show that risk analysis methods can help the assessment of dependability alternatives. The approach models dependability requirements as risks that guide the specification of laws. Laws play an essential role in the system architecture and implementation developed to enforce the laws and support the approach.

Introduction

Recently, there is widespread interest in software technology and in its associated advances as they affect and stimulate the global economy. Software permeates every aspect of our lives, and is increasingly becoming distributed, open and ubiquitous assets.

Openness has led to software systems that have no centralized control and that are composed of autonomous entities (Agha, 1997). These entities may enter and leave the environment at their will, and they may even have conflicting interests (Fredriksson et al., 2003). Multi-agent auction systems are examples of such open and distributed applications (TAC, 2004; Zambonelli et al., 2003).

Further, open systems need to rely on critical infrastructures that constitute the backbone for the delivery of their essential services. This is not only because open systems are more prone for overloads, attacks or failures, but also because they need to deal with uncertainty (Neumann, 1995). While open system components are often autonomous, they behave unpredictably when unforeseen situations arise. Taming this uncertainty is a key issue for dependable open software development.

Law enforcement can be used as a suitable technique to deal with dependability requirements in open systems. Laws define interfaces for the components that can be present and interact in an open system. This interface imposes execution rules and limits, creating a boundary of tolerated autonomous behavior and fostering the development of trusted systems.

However, a high level of dependability usually has a side effect: low performance. This problem happens because to build dependable software developers generally include extra, often redundant, code to perform the necessary checking for exceptional states, and recovery from faults (Sommerville, 2004). Moreover, these additional design, implementation and validation efforts increase significantly the development costs.

On the other hand, risk analysis methods can help the assessment of dependability alternatives. They can be used as a criterion to establish an order of relevance or importance in dependable software development. We believe that risks can offer a structured method to specify, develop, monitor and maintain system requirements and to foster dependability.

This work proposes a law enforcement approach that uses risk analysis to develop dependable open systems. This approach models dependability requirements as risks. These risks guide the specification of laws that play an essential role in the system architecture and implementation developed to enforce the laws and support the approach. The Figure 1 shows an overview of our approach.

* We gratefully acknowledge the financial support provided by the CNPq as part of individual grants and of the ESSMA project (552068/2002-0).
The contribution of this paper is threefold. First, we propose using a risk based approach to deal with dependability. Second, we propose a conceptual framework to represent laws. At last, we propose mixing risk-based and law enforcement approaches to meet dependability attribute requirements.

The organization of this paper is as follows. In section 2, we discuss how risks can be used to structure dependability attributes. In section 3, we describe the law enforcement approach and the proposed architecture. Section 4 shows how risks are related to laws. In section 5, we show a case study that illustrates the approach. Related work is described in Section 6. Finally, our conclusions are in Section 7.

Structuring Dependability Attributes as Risks

Systematic exposition of the concepts of dependability consists of three parts: the threats to, the attributes of, and the means by which the dependability is attained (Avizienis et al., 2001). Besides threats, we aim to include beneficial consequences or opportunities that could also arise from system execution and are described as emergent behavior.

A dependability attribute can be modeled as risks, represented as a sequence, or a chain, of cause and consequence states (Figure 2). The chain of states helps to understand the sequence, tracing back to their origin. By this arrangement, it is possible to infer and analyze how we can gauge and alter the consequences, understanding their causes and the links between intermediary states.

We propose that this exposition should include risks to provide a structured method to specify, develop, monitor and maintain systems requirements and the existent challenges, opportunities, threats and limitations identified through the development of the solution. Briefly, this analysis is based on the identification, control and assessment of relationships between cause and consequence states, events and their characteristics.

For example, a cause of a bad consequence in a system is a point at which a failure or an attack can render the system incapable of continuing to satisfy its requirements for dependability attributes. As in a chain, in which a loss of a single link can destroy it entirely, a specific cause of failure can represent the potential for a disaster triggered at a single point (Neumann, 1995).

Non-functional dependability requirements derive new functional ones that specify how results may be avoided, given incentives or tolerated. In the proposed approach, functional and non-functional requirements could be expressed as laws and consequences that help the achievement of the objectives.

The dependability specification process consists of some activities, including the identification of cause types; their classification; the mapping of classes to metrics; and the identification of functional dependability requirements that reduce the probability of bad consequences or even give incentives for the occurrence of good results.

Nevertheless, it may become impossible to gather all the causes and consequences a priori and, therefore the risk assessment process begins after a relevant group of items has been identified. The assessment is performed considering the severity of each consequence, the probability that it will arise, and the probability that a result will be originated from it. For each item, the outcome of the risk assessment process is a statement of acceptability.

Methods, used to understand and analyze the chain of causes and consequences consider both directions of a chain (Figure 2). Analyzing the chain of consequences by looking in the direction of the past, the method tries to find out the origins of the problems or opportunities. This approach is concerned about answering why the consequence can happen. On the other hand, another approach is concerned with answering how some causes would influence the system behavior and the dependable attributes. This analysis begins by identifying the root causes and tries to derive the consequences through the evaluations of future states in the chain.

Some of the most important dependability attributes are reliability, safety, security and survivability. Next, we present a way to model these attributes as risks and the consequences of this approach.
Reliability
The chain of causes and consequences of reliability can be described as follows (Figure 3). Human errors and mistakes are the primary cause of faults. Due to a lack of understanding of what exactly is happening, sometimes the same fault can result in a normal behavior or in a system error, depending on the event that generates the transition. Likewise, errors can lead to failures only when the fault code is executed with inputs which expose the software failure. Therefore, the reliability can be measured as the probability that the input will cause erroneous outputs.

![Figure 3 – Chain of causes and consequences of reliability issues](image)

Table 1 shows an example of an availability risk, its consequences, the decisions made to deal with these consequences, and the goals that need to be achieved to implement the decisions.

<table>
<thead>
<tr>
<th>Risk Identification</th>
<th>Actor unavailability at the beginning of the process.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Risk Description</td>
<td>There may not be available actors in the system for the initial interaction between the participants. There are no available sellers at the moment which a buyer decides to start a negotiation.</td>
</tr>
<tr>
<td>Consequences</td>
<td>The system loses its purpose</td>
</tr>
<tr>
<td>Decision</td>
<td>For buyer agents: advertisement and promotions. For seller agents: good client base, minimize costs. Monitor and enhance system capability</td>
</tr>
<tr>
<td>Decision Goal</td>
<td>Maintain the system working.</td>
</tr>
</tbody>
</table>

Table 1 - Availability specification example scenario

Safety
To assure safety, you must ensure that accidents do not occur or that the consequence of an accident is minimal (Figure 4). Accidents are caused by hazards and have damages as consequences.

![Figure 4 – Chain of causes and consequences of safety issues](image)

Table 2 shows an example of a safety risk.

<table>
<thead>
<tr>
<th>Risk Identification</th>
<th>Risk of lack payment by buyer agents</th>
</tr>
</thead>
<tbody>
<tr>
<td>Risk Description</td>
<td>Buyer agent must pay for the acquired ticket. Payment may not happen.</td>
</tr>
<tr>
<td>Consequences</td>
<td>Non-payment implies in financial losses for sellers</td>
</tr>
<tr>
<td>Decision</td>
<td>If an agent does not pay, the agent will have the ticket cancelled and it will not be allowed to participate in future negotiations, unless... Each buyer agent has a negotiation insurance. If an agent does not pay, besides having the ticket cancelled, the company may be refunded. A mediator (human/software) may be used to solve conflicts.</td>
</tr>
<tr>
<td>Decision Goal</td>
<td>Provide a way to incentive and guarantee the air companies. Avoid bad payer agents to keep closing deals which hampers the system</td>
</tr>
</tbody>
</table>

Table 2 - Safety specification scenario

Security
Errors can lead to security loopholes, and damages can be caused exploiting this weakness (Figure 5). Some examples of different types of damages include denial of services (unavailable); corruption of programs or data (unreliable, unsafe and unavailable); and disclosure of confidential information (unreliable, unsafe and unavailable).

The chain of causes and consequences of this dependability attribute can be described as follows (Figure 5). Having control of a situation reduces vulnerability. Threats (circumstances) contribute to vulnerability. Vulnerabilities may lead to exposure (possibility) and attacks (exploitation).
Table 3 shows an example of a security risk.

<table>
<thead>
<tr>
<th>Risk Identification</th>
<th>Disguise – Fake identity – or Agent simulation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Risk Description</td>
<td>The interaction between agents requires the participants' identification. An agent may pose as another, taking advantage of this situation.</td>
</tr>
<tr>
<td>Consequences</td>
<td>Less system reliability</td>
</tr>
<tr>
<td></td>
<td>Losses for the participants</td>
</tr>
<tr>
<td></td>
<td>Financial losses and lack of trust in the application</td>
</tr>
<tr>
<td></td>
<td>User rejection as a consequence</td>
</tr>
<tr>
<td>Decision</td>
<td>Apply user identification and credentials verification policies on every interaction.</td>
</tr>
<tr>
<td></td>
<td>To interact, software agents will need an identification in the system and they will also have to guarantee the ID's authenticity.</td>
</tr>
<tr>
<td>Decision Goal</td>
<td>Provide a reliable user identification mechanism.</td>
</tr>
<tr>
<td></td>
<td>Increase the confidence on the systems interactions</td>
</tr>
</tbody>
</table>

Table 3 - Security specification scenario example

Sustainability & Survivability

A typical software system characteristic is that the properties and the behavior of its components are inextricably intermingled. The successful execution of each system component often depends on the execution of other related components. So, open system approaches must consider this characteristic and its influence to preserve the sustainability of other entities (Fredriksson et al., 2003).

Sustainability is the dependability property concerned with causing or allowing the open system to continue executing for a determined period (Fredriksson & Gustavsson, 2002). Its objective is to help the preservation or to guarantee a harmonious execution environment condition, that is, the behavior of one participant or a group of participants might not prejudice the possibility of other distributed software components are executing in a specific environment.

Sustainability is achieved by considering many other dependability attributes. For instance, to maintain a system sustainable it is desirable to keep it safe, reliable and secure or to provide means to avoid the occurrence or repetition of bad behaviors.

The scenario used to exemplify this attribute specification can be seen as the composition of Table 1, Table 2, and Table 3.

Survivability is concerned with continuing to deliver the service while the system is under attack or even while part of the system is disabled for security and availability issues (Sommerville, 2004). There are many ways to improve the survivability of systems including resistance to attack, attack recognition and recovery from damage techniques. Survivability is directly related with sustainability in open system context.

Law Enforcement Approach

In this work, we aim to translate the qualitative or even quantitative criteria presented by dependability attribute to a mechanism that will enforce behavioral rules. Rules are specified as laws and norms. Laws and norms are associated with well-established consequences that are subjected to any participant of an open system. These specifications aim to preserve the dependability attributes.

Laws are identified through a risk-driven analysis of the open system environment. The law enforcement mechanism allows control of the failures and benefits, and it also contributes to tame the uncertainty presented by open systems. This mechanism intercepts some interactions among distributed software components to control and audit the execution flow of conversations.

The enforcement approach aims to contribute to transition from unstable and very unpredictable open systems to the development of dependable, more stable and less unpredictable systems. A risk-based approach can facilitate the understanding of the complexity and the multiple variables involved in open systems development.

Laws Conceptual Model

In order to obtain a common understanding about laws, we propose a conceptual model. This model is intended for describing the elements that compose a law specification.
A law specification is composed of a description of interaction protocols, norms and time restrictions. These three elements are interrelated in a way that it is possible to specify interaction protocols using time restrictions, norms to control interaction protocols, or even create time sensitive norms.

Interaction protocols define the valid interactions that distributed software components can have and the context where the information exchanged must be interpreted. The specification of interaction protocols using laws allows the protocol enforcement and helps to acquire a better understanding about the problem.

Norms capture the behaviors that are allowable, forbidden, or obligatory. As we mentioned before, norms and protocols work together, complementing each other.

Digital clocks represent the time restrictions and they can be used with protocols or norms as well. Clocks could indicate that a certain period has elapsed producing clock-ticks events.

We use a state machine based approach to specify protocols. A protocol is composed of transitions. Transitions structure the change from previous state to next state, which define what actors can join in a specific part of the conversation, and which are the valid actions. Protocol transitions are activated by sending and receiving messages to software components. However, many other kinds of events can also activate or deactivate transitions. Examples of events are clock-ticks, arrival and sending of messages.

 Norm
 Transition
 Obligation
 Permission
 Prohibition

**Figure 6 - Transition and norms**

Norms prescribe how the distributed software components ought to behave, and specify how they are permitted to behave and what their rights are (Jones & Sergot, 1993). A norm is composed of obligations, permissions and prohibitions. An obligation defines the consequences that the distributed software component (DSC) actions within protocols will have in the future. For instance, the winner of an auction is obligated to pay the committed value. Permissions define the rights of a DSC in a given moment, e.g. the winner of an auction has permission to interact with a bank provider through a payment protocol. Finally, prohibitions define forbidden actions of a DSC in a given moment.

Each norm element (obligations, permissions and prohibitions) has an activation or deactivation condition and consequence. The conditions of activation and deactivation are logical expressions that are evaluated as true or false. The consequence is an instance of an obligation, permission or prohibition. The instance can carry information about the context where the instance was generated. A context is a set of values representing the past DSC interactions, the set of obligations, permissions and prohibitions, and any other value regarding the system execution.

**Law Enforcement Architecture**

We propose a law enforcement architecture to guarantee that the specifications will be obeyed (Figure 7) and we developed an infrastructure which includes some communication components that will be provided to DSC developers. This architecture is based on a mediator that intercepts every message and interprets the laws previously described. The main goal of this phase is to provide the infrastructure for the mediation of conversations between components. This phase is also responsible for developing the basic communication components and the interoperability concerns.

**Figure 7 - Law Enforcement Architecture**

Depending on the solution domain, it could be necessary to extend this basic infrastructure to attend system requirements. To develop dependable software, this infrastructure must implement some dependability techniques like fault tolerance, error handling and redundancy.

Distributed software components are independently implemented, i.e., this development can be done without a centralized control. The developers may have an a priori access to every specification, protocol descriptions or laws generated during the specification of the open system.

**Using Laws to Mitigate Risks**

In this section, we propose that the risks previously identified must be mitigated using the law enforcement approach and its related concepts. A software engineer has some alternatives or strategies to deal with risks. These strategies deal with the events that could cause the risk and the moment when actions are taken. For example, we can let the event that could cause the risk happen, ignoring its consequences; we can
alternatively avoid its occurrence by controlling the process before the execution of determined action, and finally, we can take an action after the occurrence of risks.

Some techniques could be used in conjunction with law enforcement approach to deal with risks (Laprie et al., 1995; Sommerville, 2004). Risk avoidance and risk tolerance are techniques used to minimize risk occurrence, to trap events before they result on bad consequences and to provide means to repair possible damages. Associated with these techniques, we can have a forward strategy to recovery damages or a backward strategy to recover from errors.

To illustrate how the law enforcement approach could be used to improve the dependability attributes, let us consider some examples on how a law enforcement controller can be used. Such a controller can be used, for example, as a checking facility that acts like a fault tolerance mechanism to improve reliability. Besides, the law enforcement approach can be seen a mechanism for assertion execution. In this case, it acts as a protection mechanism to ensure that some erroneous behavior are discovered and corrected before system services are affected. For instance, the interception should not allow a distributed software component that has broken a rule to continue interacting with others or it should even avoid the consequences of this action.

Besides, a law enforcement controller could provide sophisticated interlocks to improve safety. It supports control strategies that reduce the amount of time people have to spend in a hazardous environment.

Furthermore, a law enforcement approach should be used to validate and impose security policies established by the open system. It would impose, for example, restrictions on how and which components could interact with each other.

Finally, a law enforcement controller could be used as a mechanism for assuring that the sustainability laws, derived by the analysis of sustainability attributes and other functional requirements, are fulfilled by the open system participants.

**Case Study**

In this section, we present a specific example to illustrate the application of our approach and highlight its main features.

Suppose an airport where flight companies and passengers have an immersive environment for negotiating flight tickets. This environment is immersive in the sense that the goal of this environment is to enhance computer use by making many computers available throughout a physical environment, and also by making them effectively visible to as many users as possible. Users can have access to the airport services using systems like PDAs and mobile phones. Flight companies and passengers are represented by software agents. Software agents are DSC and they can enter or leave the environment at their own will (Zambonelli et al., 2003).

Flight companies offer tickets for commercial flights. The goal of a flight company is to sell the maximum number of tickets, to increase the user satisfaction and to charge them as much as possible. Passengers use palmtops when they arrive at the airport to buy flight tickets. Each passenger has a specific profile that defines his/her preferences concerning the destination, flight types, maximum acceptable ticket cost, and any other characteristics.

Passenger groups can bargain and get discounts when buying more tickets in a same negotiation process. A group should be formed considering common preference attributes of the participants specified in their personal interests, e.g., the same destination, price, comfort or time of departure.

In the whole negotiation process, a specific step exists for creating groups of interests, where the participant personal profiles are combined aiming to inform other participants that they have close interests. Using this information, it is possible to form a group with close preferences and this group can bargain discounts with the sellers.

By understanding the inherent risks in this case study, it is possible to specify interaction protocols and laws that will regulate the multi-agent system. Figure 8 show the steps related to the protocol specification. The user arrives physically at the airport; it tries to form groups to bargain discounts; it sends messages to the flight companies’ agents (FCA) to negotiate their values and attributes; it must pay the ticket to the air flight company which causes the FCA to emit the electronic ticket; it can check in using the electronic ticket, and it leaves the environment.

**Figure 8 - Agents Execution Scenario**

In this paper, we focus on the negotiation phase. The flow of conversation in this phase consists of the following steps:
1- The user agent sends a call-for-proposal message asking for flight tickets. That message contains information about attributes of the flight ticket. These attributes represent the passenger preferences and they are related to, for instance, the preferred departure time or degree of priority on low prices.

2- The flight company offers one ticket flight to the passenger who can, in turn, accept or reject the offer.

3- The user agent accepts the offer and sends a confirmation message.

4- The flight company produces an electronic flight ticket and sends it to the user agent.

5- The user agent receives the ticket and concludes the negotiation.

These steps describe the process, and we have used this description to execute the risk identification and law specification stages.

<table>
<thead>
<tr>
<th>Risk Identification</th>
<th>R1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Risk Description</td>
<td>No answers for call-for-proposals messages.</td>
</tr>
</tbody>
</table>
| Causes              | • C1. Overwork on flight companies’ system.  
                     | • C2. Crashing on flight companies’ system. |
| Consequences        | • Co1. The customer agent will be waiting indefinitely.  
                     | • Co2. The customer could be upset and not use the system anymore.  
                     | • Co3. If a crash is the cause of the problem, all negotiations between customers and flight companies are unable to continue. |
| Severity            | Medium |
| Priority            | Medium |
| Dependability attribute | Reliability and sustainability |

**Design Decision or Rule**

- We should specify a clock, which will start when the call-for-proposal message is send. After X seconds, the clock activates a norm allowing the customer agent to cancel the negotiation process without penalties.

**Rule Goal**

- Avoid annoying customers with long waiting times.
- Give a specified time for flight companies to execute its market strategies.

<table>
<thead>
<tr>
<th>Risk Identification</th>
<th>R2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Risk Description</td>
<td>Client takes too long to decide if he accepts or rejects the ticket offer. When flight companies offer flight tickets, they reserve a sit for the customer in a way that no other customer can buy that specific sit. However if the client takes too long to answer the request, the flight company could loose the opportunity for sell the ticket to another customer. On the other hand, the client also needs some time to decide whether to accept or reject the offer</td>
</tr>
</tbody>
</table>
| Causes              | • C3. Undecided customer.  
                     | • C4. Bad network function.  
                     | • C5. Bad customer’s software function. |
| Consequences        | • Co4. Flight companies could sell the sit to other customers.  
                     | • Co5. The flight company could be penalized with money losses caused by undecided passengers. |
| Severity            | High |
| Priority            | High |
| Dependability attribute | Reliability and sustainability |

**Design Decision or Rule**

- Specify a clock that after X seconds activates a permission to cancel the negotiation and gives the permission to the flight company. Create a norm forbidding future negotiation participations for the companies that do not follow the time restriction.

**Rule Goal**

- Give time for customer to decide about acceptance or rejection of offers.
- Protect flight companies against undecided customers.

Table 4 - Risk specification example

We have developed a scenario form that allows us to specify risks, to keep the tracing of causes and consequences, to describe which dependability attributes the consequences affect, and to specify the solution decision. A solution decision could be either a design solution or a law specification.

We use identifiers to represent risks, causes and consequences. In this way, it is possible to construct and identify the chain of causes and consequences related to the inherent risks. Below we show how we use the forms.

The scenario description and the initial risk analysis provide the information needed to specify the protocol. It is important to highlight that, although we are showing the risk identification and protocol specification as sequential steps, they are not. During the protocol specification, it is possible that we discover risks, and during the risk identification, we can discover some protocol elements as well.

Table 5 - Risk specification example
Figure 7 shows the protocol specification. The transition from state 0 to 1 indicates the call-for-proposal message, from 1 to 3 means the flight company sends an offer of flight ticket, from 3 to 4 indicates the acceptance by the customer, and finally, from 4 to 5 the client receives the electronic flight ticket. The other transitions represent alternative flows of conversation. The risk identification phase provides support to identify some of these flows.

![Figure 9 – Protocol specification](image)

Due to space limitations, we do not intend to present a comprehensive case study, but our goal is to highlight the main features of our software dependability approach. Our approach begins with the scenario description and continues through risks and protocol identification. The risk analysis generates new norms, associates clocks, and proposes some protocol modifications. All those concepts compose the laws of the system. Furthermore, we can specify many other laws to deal with the unpredictable behavior of the participant agents. This informal specification should be certainly specified, as one of our next steps, in a formal manner. However, the definition of formal law representations is out of the scope of this paper.

We have implemented the enforcement mechanism following the architecture of the Section 3 and developing a component that extends the Jade (Bellifemine et al., 2001) communication API to provide the redirection of messages. We have also developed an application to monitor the process execution. For example, this application shows the norm activation and deactivation and it collects some metrics that were previously specified. Figure 10 shows a screenshot of this monitoring application.

![Figure 10 – Monitoring tool screenshot](image)

**Related Work**

Law Governed Interaction (LGI) (Minsky & Ungureanu, 2000) proposes a mechanism to coordinate and control heterogeneous distributed systems. It is based on four basic principles: (1) coordination policies need to be enforced; (2) the enforcement needs to be decentralized; (3) policies need to be formulated explicitly rather than being implicit in the code of the agents involved and they should be enforced by means of a generic, broad spectrum mechanism; (4) and it should be possible to deploy and enforce a policy incrementally. However, this approach does not provide an explicit method to develop and evolve its law enforcement infrastructure.

One of the most interesting related works is the electronic institutions approach (Esteva, 2003; Rodriguez-Aguilar, 2001). In this approach, some concepts related to law enforcement were formalized, software tools were developed to facilitate the institution's design, a textual specification language called ISLANDER was defined, and an infrastructure that mediates agent interactions and enforces the institutional rules on participating agents was developed. Another contribution of this group is the method for rapidly building prototypes of large multi-agent systems using logic programming (Vasconcelos et al., 2004). This method advocates the use of all permitted interactions among the components of the systems. In contrast, the focus of our work is on structuring the development process of open middleware software, thus providing, during this process, development guidelines, structured as risks and dependability attributes.

Cole et al. (2001) proposes a way to identify laws in real world problems. However, his results do not deal with issues related to law enforcement and
specification. In addition, Mineau (2003) proposed that laws should be specified using a conceptual graphs approach. This approach supports the validation of rules and uses a very rich and expressive language but it does not propose any enforcement mechanism.

Conclusions

Tomorrow's infrastructures will have to face the challenge of survivability: delivering critical services in a timely manner in presence of overloads, attacks and failures (Ellison et al., 1999). This challenge has pushed to the point where proactive risk management is essential. It has become important that software engineers determine whether unwanted events may occur during the development and maintenance of a software system, and make appropriate plans to avoid or minimize the impacts of these events (Neumann, 1995).

In this work, we provide a means for translating the qualitative or even quantitative criteria presented by dependability attributes to a mechanism that will enforce behavioral rules. Rules are specified as laws. Laws are associated with well-established consequences that are subjected to any participant of an open system, and they are identified through a risk driven analysis of the open system environment. This mechanism permits to control the failures and benefits and it contributes to tame the uncertainty presented by open systems.

We believe that this paper represents an advance in the way that dependability attributes requirements can be met and the uncertainty of open environments can be tamed. However, many interesting extensions can be foreseen on the research side, including the formalization of risk specifications with their graphical counterparts, and the development of automatic tracking tools that can support the work of developers and help to improve the productivity of the whole process. Although it is not the focus of this paper, we have considered adding more formalism and methods for describing the laws and automatically generating solutions for verifying it.

Metrics (Fairley, 2002) could be specified in the risk identification phase and the automatic mechanisms provided by the law enforcement infrastructure are able to gather metrics and provide feedback on how efficient these metrics are. In this way, the approach could also help developers to specify system laws and to quantitatively measure their effectiveness.

As a future work, we intend to extend Anote notation (Choren & Lucena, 2004) to represent the design decisions, including the specification of system requirements considering information about risks, interaction protocols, norms activation and deactivation, and any other adaptation that could provide a better understanding of the solution. Furthermore, we also intend to provide a conceptual framework to aid the assessment of existing alternatives of law specification and enforcement mechanisms considering functionality, performance, cost and dependability system properties using for this purpose a risk driven approach.

References


