Towards Trust-Aware and Self-Adaptive Systems*

Francisco Moyano, Javier Lopez
Network, Information and Computer Security Lab
University of Malaga, 29071 Malaga, Spain
{moyano,jlm}@lcc.uma.es

Benoit Baudry
INRIA Rennes Bretagne-Atlantique
Campus de Beaulieu, 35042 Rennes, France
Certus Software V&V Center
SIMULA RESEARCH LAB., Lysaker, Norway
benoit.baudry@irisa.fr

June 24, 2013

Abstract

The Future Internet (FI) comprises scenarios where many heterogeneous and dynamic entities must interact to provide services (e.g., sensors, mobile devices and information systems in smart city scenarios). The dynamic conditions under which FI applications must execute call for self-adaptive software to cope with unforeseeable changes in the application environment. Software engineering currently provides frameworks to develop reasoning engines that automatically take reconfiguration decisions and that support the runtime adaptation of distributed, heterogeneous applications. However, these frameworks have very limited support to address security concerns of these application, hindering their usage for FI scenarios. We address this challenge by enhancing self-adaptive systems with the concepts of trust and reputation. Trust will improve decision-making processes under risk and uncertainty, in turn improving security of self-adaptive FI applications. This paper presents an approach that includes a trust and reputation framework into a platform for adaptive, distributed component-based systems, thus providing software components with new abilities to include trust in their reasoning process.

*This work has been partially funded by the European Commission through the FP7/2007-2013 project NESSoS (www.nessos-project.eu) under grant agreement number 256980. The first author is funded by the Spanish Ministry of Education through the National F.P.U. Program. The authors want to thank Jean-Emile Dartois for his invaluable feedback.

1 Introduction

The Future Internet (FI) scenarios are bringing two important changes in the ICT world. On the one hand, the uprising of the service-oriented vision enables the on-the-fly improvement of the features offered to users. Applications become more dynamic and require rapid adaptations to meet new requirements or respond to environmental changes. On the other hand, the emergence of the Internet of Things (IoT) is bringing the seamless integration between the physical and the virtual worlds. As a consequence, both services and systems as a whole must adapt to dynamic changes in hardware, firmware and software, including the unpredictable arrival or disappearance of devices and software components.

The aforementioned reasons prevent system architects and designers to envision all possible situations an application will have to cope with, and the boundaries between design and runtime are blurring [6]. This calls for new software engineering approaches that allow keeping an abstract representation of a running system in order to reason about changes and drive dynamic reconfiguration, leading to the so-called 'models@runtime' paradigm [1].

Security is a crucial issue that must be addressed in order to guarantee the successful deployments of FI scenarios [12]. Increasing security in FI applications entails that trust relationships between components, applications and system environments cannot be taken for granted any more, and must be explicitly declared, monitored and changed accordingly. In fact, we argue that the management of these trust relationships, together with the notion of reputation, can drive the reasoning process required in self-adaptive systems.

This paper proposes incorporating the notions of trust and reputation into a platform for self-adaptive, distributed component-based systems. This provides developers with a development framework to implement trust-aware and self-adaptive applications, where software entities can reason about reconfiguration decisions in terms of their trust relationships and their reputation, enhancing the security of these applications.

The paper is structured as follows. Section 2 presents some works that can be related to ours. An introduction to models@runtime paradigm and the Kevoree platform is given in Section 3. A brief discussion on trust and reputation is presented in Section 4, whereas Section 5 discusses the approach for a trust-aware and self-adaptive framework. Finally, Section 6 concludes the paper.

2 Related Work

To the best of our knowledge, the idea of using trust in order to enhance reconfiguration decisions and improve security in self-adaptive systems is novel. However, we may find some works that aim to leverage traditional component- and service-based software systems by using some trust or reputation notions.

In Service-Oriented Architecture (SOA) environments, we find that trust is used for either protecting providers from potentially malicious clients or for shielding clients against potentially malicious providers (e.g. providers that

publish a higher Quality of Service (QoS) than offered). As an example of the first situation, Conner et al.[2] present a feedback-based reputation framework to help service providers to determine trust in incoming requests from clients. As an example of the second approach, Crapanzano et al. [3] propose a hierarchical architecture for SOA where there is a so-called super node overlay that acts as a trusting authority when a service consumer looks for a service provider.

Haouas and Bourcier [7] present a runtime architecture that allows a service-oriented system to meet a dependability objective set up by an administrator. System dependability is computed by aggregating ratings provided by service consumers regarding QoS attributes. Then, a reconfiguration manager may look up other available services to meet the dependability objective.

Yan and Prehofer [13] discuss a procedure to conduct autonomic trust management in Component-Based Architectures (CBA). Several quality attributes can be used to rate the trustee's trustworthiness, such as availability, reliability, integrity or confidentiality. Assessing these attributes requires defining metrics and placing monitors to measure their parameters. Finally, trust is assessed at runtime based on the trustor's criteria and is automatically maintained.

Herrmann and Krumm [8] propose using security wrappers to monitor components. The intensity of the monitoring activity by these wrappers is ruled by the component's reputation. This scheme was enhanced by Herrmann [9] in order to take the reputation of components' users into account so as to prevent deliberate false feedbacks.

3 Kevoree: A Models@Runtime Development Platform

Models@runtime [1] referes to model-driven approaches that aim to tame the complexity of software and system dynamic adaptation, pushing the idea of reflection one step further. Kevoree [4] is an open-source dynamic component model that relies on models at runtime to properly support the design and dynamic adaptation of distributed systems. Seven concepts constitute the basis of the Kevoree component metamodel, as shown in Figure 1. A node models a device on which software components can be deployed, whereas a group defines a set of nodes that share the same representation of the reflecting architectural model. A port represents an operation that a component provides or requires. A binding represents the communication between a port and a channel, which in turn models the semantics of communication. The core library of Kevoree implements these concepts for several platforms such as Java, Android or Arduino.

Let us suppose that a new mobile phone joins a smart grid system developed with Kevoree. It is then required to carry out an adaptation process to keep synchronized the abstract model of the running system with the actual system. The problem boils down to move from a current model to a new model (i.e. the target model). First, the target model is checked and validated to ensure a well-

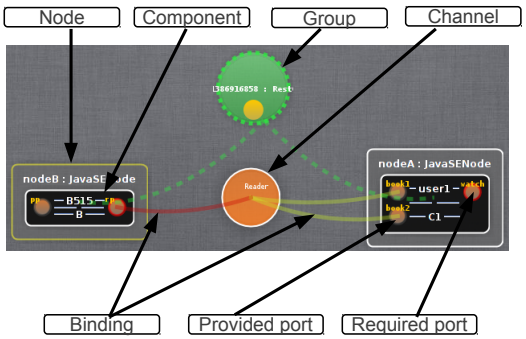

Figure 1: Kevoree Architectural Elements

formed system configuration. Then it will be compared with the current model that represents the running system. This comparison generates an adaptation model that contains the set of abstract primitives to go from the current model to the target one. In our example, these primitives will launch the mobile phone running platform (e.g. Android) and the group will inform the rest of nodes about the new joint node through gossip or paxos algorithms. Finally, the adaptation engine executes applies these abstract primitives. If an action fails, the adaptation engine rollbacks the configuration to ensure system consistency.

Up to now, Kevoree platform does not support reasoning over qualitative or security related concerns, and therefore any architectural element such as a node or a software component can join the system without further checks. Also, there is no criteria to guide the runtime changes. Our goal, as further explained in Section 5, is to provide these architectural elements with trust and reputation capabilities in order to allow better decision-making on reconfiguration.

4 Trust and Reputation Framework

Even though an agreed definition of trust is not proposed yet, an often cited definition is the one by Gambetta [5]: *trust is a particular level of the subjective probability with which an agent assesses that another agent or group of agents will perform a particular action, both before he can monitor such action (or independently of his capacity ever to be able to monitor it) and in a context in which it affects his own action.*

Whereas trust is subjective, reputation is often considered a more objective concept. The link between trust and reputation is sometimes fuzzy and has not been deeply studied yet. Jøsang [10] related both notions with the following statements: 'I trust you because of your good reputation' and 'I trust you despite your bad reputation'. Reputation can be considered then as a building block to set trust but, as stated by the second statement, it has not the final say.

The concept and implications of trust are embodied in the so-called trust models. These models define the rules that are to be used to process trust in

an automatic or semi-automatic way in a computational setting. We find two main groups of trust models, namely decision models and evaluation models[1].

We are particularly interested in evaluation models, which rely on the notion of trust or reputation metric to yield a trust or reputation value. Evaluation models include reputation models, propagation modes and behaviour models. Reputation models are those in which a score is derived from the aggregation of opinions of other entities in the system. This score is made public and can be looked up by any entity before making a trust decision. Regarding propagation models, also known as flow models, they assume that some trust relationships already exist and aim to use them in order to create new relationships, usually by exploiting trust transitivity. Finally, behaviour models are those where the trustor evaluates its trust in the trustee by personally observing its behaviour and interacting with it, and sometimes also building a beliefs-based mental state.

We build upon these concepts in order to incorporate trust and reputation into the self-adapting platform, as explained in the next section.

5 Towards a Trust-Aware Self-Adaptive Development Framework

The goal that we pursue is to provide developers with a development framework to build trust-aware and self-adaptive applications. This way, self-adaptability can also be guided by the trust relationships and entities reputation, increasing the security of the developed applications.

The approach proposed to achieve these goals is depicted in Figure 2. At design-time, the developer uses the trust API that encapsulates some of the trust-related concepts discussed in the previous section. This API provides ways to easily indicate that certain Kevoree type (e.g. node, component, channel, etc) is trust-aware by means of annotations, in the same way as Kevoree API does. For example, the following code snippet declares a class that is both a Kevoree software component and a trust-aware entity with a default reputation value.

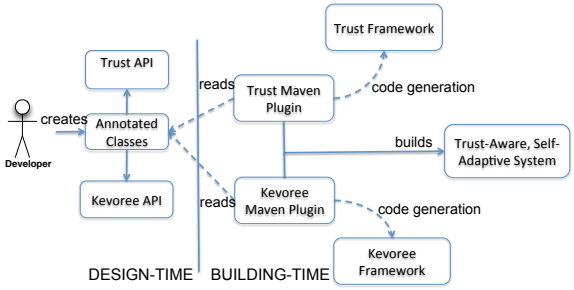

Figure 2: Approach for a Trust-Aware and Self-Adapting System Development

[1] A comprehensive domain analysis in trust, as well as a trust models classification, are presented in [11].

```
@ComponentType //annotation from Kevoree API
@TrustEntity[DefaultReputation = 0]// annotation from Trust API
public class HelloWorldComponent {
...
}
```

There would be other annotations to cover other aspects, such as the reputation and trust engines, which the developer should implement for each application. For example, the following code snippet declares that a Kevoree component is a trust engine that is to be applied to all the trust relationships with the trustees of the entity 'HelloWorldComponent'. Retriever and Pusher are interfaces provided by the Trust API, and their code is generated by the trust framework at building time.

```
@ComponentType
@TrustEngine[Entity=HelloWorldComponent]
public class EngineHelloWorld {
        Retriever ret;
        Pusher push;
        public void compute() {
                //retrieve variables from trust context model
                Variable var = ret.getVariable(name);

                // Specific computation of trust value
                ...

                //push trust value in trust context model
                push.addTrustValue(trustValue);
        }
}
```

Once the developer has annotated the classes with both Kevoree and trust information, maven[2] is used to call both the Kevoree and the trust frameworks. These frameworks generate code according to the annotation information passed to them by the Maven Kevoree and Maven Trust plugins[3].

Some of the duties of the trust framework consist of setting appropriate listeners to the Kevoree runtime model, in such a way that when two trust-aware nodes are linked through a channel, a trust relationship is initialized between them. The generated code by the trust framework is also in charge of knowing how to push and retrieve variables information to and from a trust context model that extends the Kevoree model with trust-related information.

The developer can then implement reconfiguration components, also by means of annotations, which take trust and reputation issues into account in order to make decisions. These reconfiguration components use Kevscript[4] to

[2]http://maven.apache.org

[3]The Maven plugin for Kevoree already exists and does not need to be modified. We just introduce the plugin for the trust framework.

[4]Kevscript is a script language to modify runtime models implemented with Kevoree.

change the current configuration in response to some trust events. The developer could specify that if a trust relationship falls below a certain threshold, the channel between the trustor and the trustee should be removed at runtime. Another example is to remove a node in case that its reputation falls drastically. Also, when a new node tries to join the system, it could be demanded to deploy certain trust-aware components to allow the rest of nodes to monitor its reputation and trust relationships.

As an example, Figure 3 shows a distributed system with three nodes. Each node is hosting four components: the first component represents the business logic of the node. The other components are in charge of reputation and reconfiguration issues. The variable producer component makes a ping regularly to the rest of nodes in order to measure their time to response and the packet loss rate, and it pushes this information into the trust context model. The reputation engine component retrieves these measures from the trust context model, and computes a final reputation score for a target entity by a weighted aggregation: $reputation = \alpha * myMeasure + \beta * othersMeasure$. Finally, the developer specifies in the reconfiguration component that when the reputation of one node, as measured by any node in the system, falls below a threshold, the node should be removed from the system.

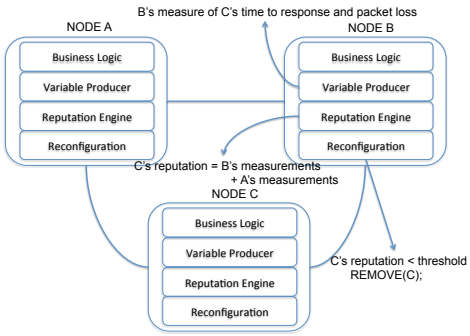

Figure 3: System Example

In this example, the reputation is computed with a metric that does not depend on the business logic, but in general, application-specific information can be used for trust and reputation assessment.

6 Conclusion

The complex scenarios that the FI is bringing raise new challenges for the software engineering community. Concretely, self-adaptability and security become first-citizen requirements for FI applications. In order to address these requirements, we propose to enhance a self-adaptive development framework with trust and reputation capabilities. This leads to two main contributions: first, developers are provided with the necessary tools to create trust-aware and self-adaptive

distributed applications. Second, the trust and reputation enrichment allows exploiting this data in order to make runtime reconfiguration decisions, increasing the security awareness of self-adaptive systems.

Future work entails finishing the implementation as well as extending the trust annotation API and the framework in order to support the definition of propagation models. Also, further exploration is required on how to automatically synthesize reconfiguration components that act on trust information.

References

[1] Gordon Blair, Nelly Bencomo, and Robert B. France. Models@ run.time. *Computer*, 42(10):22–27, 2009.

[2] William Conner, Arun Iyengar, Thomas Mikalsen, Isabelle Rouvellou, and Klara Nahrstedt. A trust management framework for service-oriented environments. In *Proceedings of the 18th international conference on World wide web*, WWW '09, pages 891–900, New York, NY, USA, 2009. ACM.

[3] C Crapanzano, F Milazzo, A De Paola, and G.L Re. Reputation Management for Distributed Service-Oriented Architectures. In *Self-Adaptive and Self-Organizing Systems Workshop (SASOW), 2010 Fourth IEEE International Conference on*, pages 160–165, 2010.

[4] François Fouquet, Olivier Barais, Noël Plouzeau, Jean-Marc Jézéquel, Brice Morin, and Franck Fleurey. A Dynamic Component Model for Cyber Physical Systems. In *15th International ACM SIGSOFT Symposium on Component Based Software Engineering*, Bertinoro, Italie, July 2012.

[5] Diego Gambetta. Can we trust trust? In *Trust: Making and Breaking Cooperative Relations*, pages 213–237. Basil Blackwell, 1988.

[6] Carlo Ghezzi. The fading boundary between development time and run time. In Gianluigi Zavattaro, Ulf Schreier, and Cesare Pautasso, editors, *ECOWS*, page 11. IEEE, 2011.

[7] Haouas Hanen and Johann Bourcier. Dependability-Driven Runtime Management of Service Oriented Architectures. In *PESOS - 4th International Workshop on Principles of Engineering Service-Oriented Systems - 2012*, Zurich, Suisse, June 2012.

[8] P. Herrmann and H. Krumm. Trust-adapted enforcement of security policies in distributed component-structured applications. In *Computers and Communications, 2001. Proceedings. Sixth IEEE Symposium on*, pages 2–8, 2001.

[9] Peter Herrmann. *Trust-Based Protection of Software Component Users and Designers*, volume 2692 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, Berlin, Heidelberg, May 2003.

[10] Audun Jøsang, Roslan Ismail, and Colin Boyd. A survey of trust and reputation systems for online service provision. *Decision Support Systems*, 43(2):618–644, March 2007.

[11] Francisco Moyano, Carmen Fernandez-Gago, and Javier Lopez. A conceptual framework for trust models. In Simone Fischer-Hübner, Sokratis Katsikas, and Gerald Quirchmayr, editors, *9th International Conference on Trust, Privacy & Security in Digital Business (TrustBus 2012)*, volume 7449 of *Lectures Notes in Computer Science*, pages 93–104, Vienna, Sep 2012 2012. Springer Verlag, Springer Verlag.

[12] Dirk van Rooy and Jacques Bus. Trust and privacy in the future internet - a research perspective. *Identity in the Information Society*, 3(2):397–404, August 2010.

[13] Zheng Yan and C Prehofer. Autonomic Trust Management for a Component-Based Software System. *Dependable and Secure Computing, IEEE Transactions on*, 8(6):810–823, 2011.