Scientific Research and Essays Vol. 6(16), pp. 3504-3513, 19 August, 2011
Available online at http://www.academicjournals.org/SRE
DOI: 10.5897/SRE11.813
ISSN 1992-2248 ©2011 Academic Journals

Full Length Research Paper

Modeling and non-functional analysis of service-oriented architectures using AADL

Raziyeh Aminpour, Vahid Rafe* and Mohsen Rahmanei

Department of Computer Engineering, Malayer Branch, Islamic Azad University, Malayer, Iran.

Accepted July 7, 2011

Nowadays, service-oriented architectures (SoA) are increasingly used to develop dynamic enterprise systems. Due to the increasing need for high quality services, it is desirable to consider different qualitative aspects in this architecture (for example security, availability, reliability, fault tolerance, etc.). As architecture analysis and design language (AADL) can be used in the analysis of partially defined architectural patterns with limited architectural details, it is suitable for specifying large-scale complex systems. In this paper, we present a formal approach to model functional and non-functional aspects in service oriented applications through AADL language. We will explain how different parts of SoA can be specified through AADL and how different non-functional aspects can be analyzed using the proper existing tools for AADL.

Key words: service-oriented architectures, architecture analysis and design language, analysis, non-functional aspects.

INTRODUCTION

Service-oriented architectures (SoA) provide a flexible platform to develop dynamic enterprise systems. SoA provide a standard in which automated service publication and discovery at runtime is possible. It means that whenever a service provider cannot provide a service with the required quality any longer, the service requester can search for and switch to a new service provider (Baresi et al., 2006). Designing such highly dynamic architectures is a complex task since designers have to consider both functional and non-functional requirements. The functional requirements deal with component structures, interaction and reconfiguration mechanisms among components while non-functional requirements are dealing with quality aspects of the systems such as security, availability, reliability, fault tolerance, etc. Even though there are different works to alleviate these complexities by proposing different modeling frameworks and architectural styles, there is still room for improvement. Most of the existing work is concerning the modeling functional aspects while a few of them considering non-functional requirements in their proposals. Modeling and analysis non-functional properties are more complex for developing distributed systems (Baier and Katoen, 2008). In this paper we present an approach to formally model service oriented applications using AADL. We describe how different parts involved in a service-oriented application can be specified through AADL. Similar to an architectural style, we propose a vocabulary of design elements in AADL (for example component and connector types, data elements, etc.), a set of configuration (reconfiguration) policies and finally the analysis approach to assess non-functional properties using the existing tools to analyze AADL descriptions [for example OSATE (Hansson et al., 2010)].

We can assess different non-functional aspects for the designed models like: security, safety, reliability. It is also possible to perform the following analysis: checking model sanity, checking connections binding consistency and checking for consistency of port properties on connections.

MATERIALS AND METHODS

The AADL language

AADL (Feiler et al., 2006; SAE International and SAE Standards,

*Corresponding author. E-mail: v-rafe@araku.ac.ir.

2009) is an internationally standardized architecture description language. It is configured by the set of non-functional properties applied to each model element. It is developed for modeling software system architecture and conducting analysis and verification of its non-functional behavior. The behavior of a system is defined by means of dispatching invariants and communication patterns. The language is useful across domains where real-time, embedded, fault tolerant, secure, safety critical, software-intensive systems are developed. System components are composites that can consist of other systems as well as of software or hardware components. Furthermore, AADL supports automated system integration via tools from fully specified AADL models when source code is provided for the software components. This standard provides formal modeling concepts for the description and analysis of application systems architecture in terms of distinct components and their interactions. The AADL is component-centric and allows to:

i) Specify and analyze real-time embedded systems, complex systems and specialized performance capability systems.
ii) Map software onto computational hardware elements (Feiler et al., 2006).

The AADL standard includes runtime semantics for mechanisms of exchange and control of data including:

i) Message passing.
ii) Event passing.
iii) Synchronized access to shared components.
iv) Thread scheduling protocols.
v) Timing requirements.
vi) Remote procedure calls.

AADL provides an extensible core language which is composed of well-defined semantics and both graphical and textual syntaxes representations (Thöne, 2005). In addition, dynamic reconfiguration of runtime architectures can be specified using operational modes and mode transitions. An AADL model such as thread dispatching condition, interface specifications and how components are interconnected can be used to describe non-functional aspects of components as performance, schedulability and reliability (Gilles and Hugues, 2009). Functional aspects (algorithmic/behavioral specifications) are attached separately as source code by means of AADL properties. For example, thread components for specifying and analyzing schedulability include the predeclared execution property of periodic, aperiodic (event-driven), background (dispatched once and executed to completion) and sporadic (paced by an upper rate bound) events (Feiler et al., 2006). AADL provides components to describe computer system architectures and these components have precise semantics. Components have a type and one or more implementations. Software components include data, subprogram, thread, thread group and process. The hardware components include processor, memory, bus and device.

The system abstraction represents a composite of software, execution platform or system components. System abstractions can be organized into a hierarchy that can represent complex systems of systems. AADL components interact exclusively through defined interfaces. A component interface consists of directional flow through:

i) Data ports for unqueued state data.
ii) Event data ports for queued message data.
iii) Event ports for asynchronous events.
iv) Synchronous subprogram calls.
v) Explicit access to data components.

Interactions among components are specified explicitly. For example, data communication among components is specified through connection declarations. Application components have properties that specify timing requirements such as period, worst-case execution time, compute deadlines, initialize deadlines, space requirements, arrival rates, and characteristics of data and event streams. The original language concepts and key specification elements of AADL are summarized in Figure 1. In AADL, components are defined through type and implementation declarations. A 'component type' declaration defines a component's interface elements and externally observable attributes (that is, features that are interaction points with other components, flow specifications and internal property values). A 'component implementation' declaration defines a component's internal structure in terms of subcomponents, connections, subprogram call sequences, modes, flow implementations and properties. Components are grouped into application software, execution platform and composite categories. Packages enable the organization of AADL elements into named groups.

The SoA style

Service-oriented architecture (SoA) is a method underlying systems development and integration where system functions are grouped around business processes and are packaged as interoperable services (Jonnaganti, 2009). In a service-oriented architecture, business components expose their functionality as services over a network to other components. A service is equipped with a description of the provided functionality including information about the interface and how to access it. Recently, the service-oriented paradigm has become very popular under the label of 'web services' (Champion et al., 2002; Chen, 1976). As shown in Figure 2, SoA involves three different roles: service providers, service requesters and discovery agencies. The service provider runs the service and publishes the service description to 'discover agency'. The 'discovery agency' enables dynamic service discovery and allows requesters to access the service (Thöne, 2005). Since service providers and requesters usually do not know each other in advance, the service descriptions are published via third-party 'discovery agencies'. They categorize the descriptions and deliver them in response to queries issued by service requesters. As soon as the 'service requester' retrieves a service description that meets its requirements, it can use it to interact with the service (Pilioura and Tsalgatidou, 2001). Service-oriented architectures are very dynamic and flexible: components and services are loosely coupled and use standardized communication protocols. As the service descriptions are exchanged at runtime, 'service requesters' can dynamically switch from unsatisfactory services to those providing better functionality or quality. To consider SoA-specific features like service discovery in our architecture models, in this paper we define an architectural framework which formally describes the concepts of service oriented computing.

Following the idea of a platform hierarchy, the SoA-specific style extends the more generic, component-based style. This means that SoA contain components and connectors, too, and they support the same message-based communication mechanisms. The framework can be continuously used along with the following platform mechanisms:

1) A 'service provider' can publish service descriptions to 'discover agency'.
2) A 'discovery agency' receives publication requests and stores the attached service descriptions.
3) The 'service requester' sends a service query to the 'discovery agency'.
4) The 'discovery agency' can receive such queries, search for an appropriate service description and send the query result back to the requester.
5) The service requester receives and safes the description and is directly connected to the relevant service.

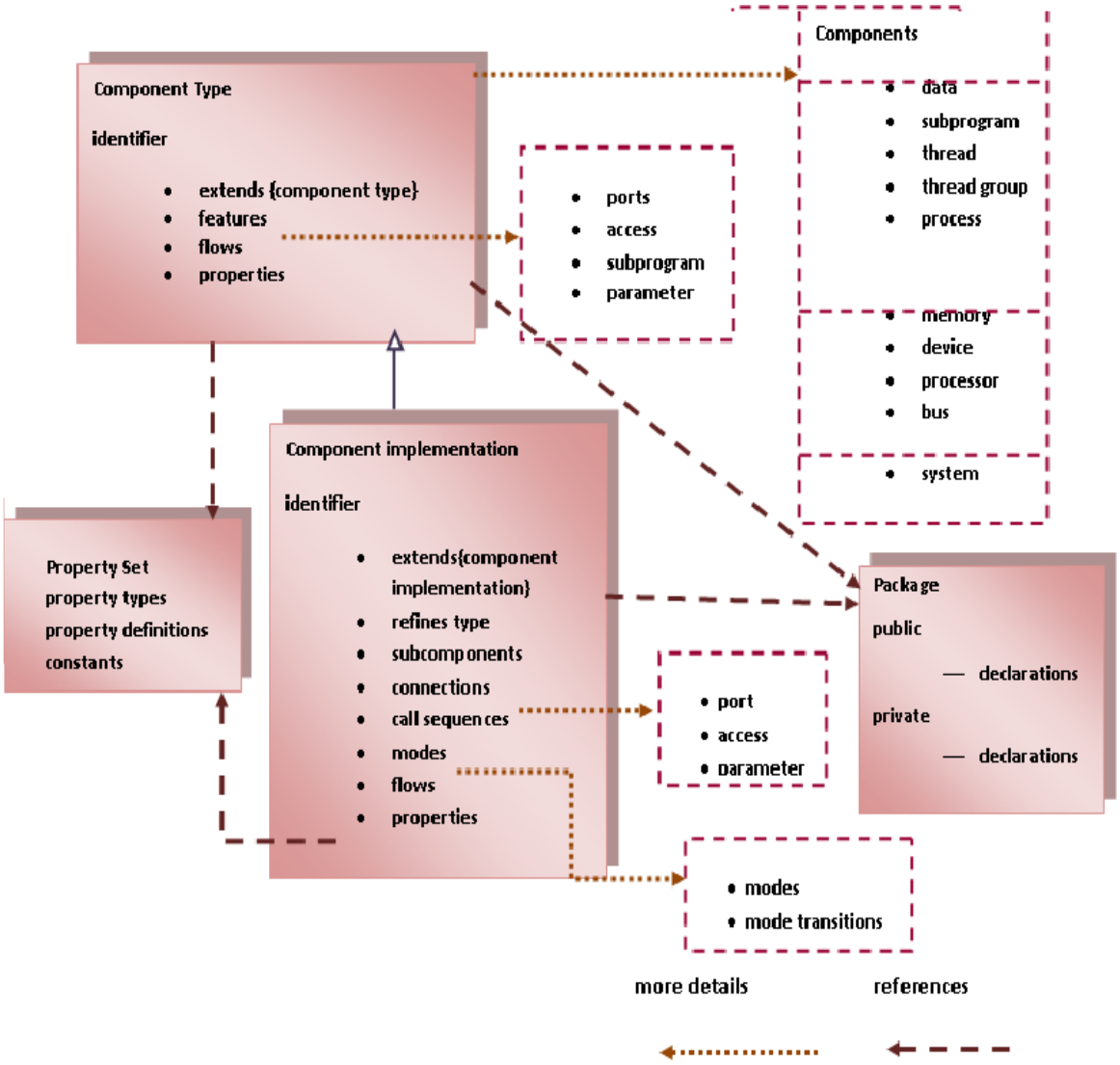

Figure 1. Summary of AADL elements: Feiler et al. (2006).

OUR PROPOSED APPROACH

Service provider, 'discovery agency' and 'service requester' can be modeled as special components using appropriate subtypes of component and component type. A 'service provider' publishes service descriptions to a 'discovery agency'. The service description describes a specific service because, in SoA, descriptions refer to deployed, addressable service rather than to service type. We can summarize our approach to model service oriented applications through AADL in the following steps:

Steps 1

Specify the SoA system through three processes that is 'service provider', 'discovery agency' and 'service requeste'r (Figure 3). As it is shown, the connection between these three processes along with the proper ports and interfaces is defined in section 'connection' based on the architecture presented in Figure 2. Communication between processes is defined in Part SoA system. Overall the system has three operational modes. Initialize mode is the initial mode which is used to activate the system. Discovery agency and 'service provider' are active in both modes.

Step 2

Specify service provider and 'discovery agency' as a periodic processes as shown in Figures 4 and 5 respectively. All services that can provide a service are declared in Part Service_Provider.imp. So to define a particular system, it needs the service providers to be declared again.

Step 3

Determine the two aperiodic threads for searching and adding services in 'discovery agency' as shown in Figure 5.

Step 4

Communication between the 'service provider' and the 'discovery

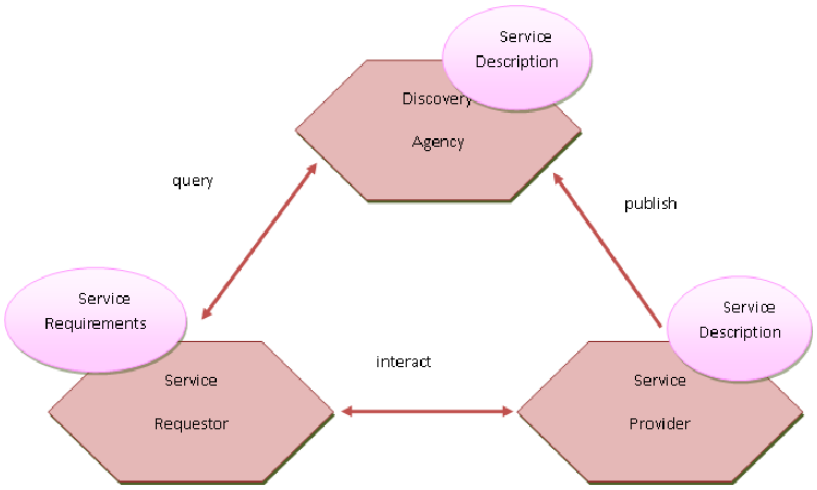

Figure 2. Roles in a service-oriented architecture, cf.: Champion et al. (2002).

```
system SOA
 features
 Init_Done: in event port;
end SOA;
system implementation SOA.imp
 subcomponents
 SR: process Service_Requester.imp ;
 DA: process Discovery_Agency.imp in modes (Query, Int);
 SP: process Service_Provider.imp in modes (Query, Int);
 connections
 event data port SP.new_Service -> DA.new_Service          in
modes (Query, Int);
 event data port DA.query_Result -> SR.query_Result        in
modes (Query);
 event data port SR.new_query -> DA.new_query              in
modes (Query);
 port group SR.interact_out -> SP.interact_in       in modes (Int);
 port group SP.interact_out -> SR.interact_in       in modes (Int);
 event port SP.Act_Service -> DA.Act_Service in modes (Query, Int);
 modes
 Initialize: initial mode ;
 Query: mode ;
 Int: mode ;
 Initialize -[ Init_Done ]-> Query;
end SOA.imp;
```

Figure 3. SoA system.

```
process Service_Provider
 features
 new_Service: in out event data port Service_record;
 Act_Service: out event port;
 interact_in: port group basic::input_PT;
 interact_out: port group basic::output_PT;
 --properties
 --Dispatch_Protocol => Aperiodic;
 end Service_Provider;
 process implementation Service_Provider.imp
            --here we declare services that are required
            --subcomponents
             --Connections
            --modes
 end Service_Provider.imp;
```

Figure 4. Service provider.

```
process Discovery_Agency
 features
 new_Service:        in out event data port Service_record;
 Act_Service: in out event port;
 new_query:          in out event data port ServiceQuery_record;
 query_Result: in out event data port Service_record;
end Discovery_Agency;
process implementation Discovery_Agency.imp
 subcomponents
 Service_addition: thread SEI_Service_addition;
 Service_search: thread SEI_Service_search;
 connections
 event data port new_Service -> Service_addition.new_Servise;
 event data port new_query -> Service_search.new_query;
 event data port Service_search.query_Result -> query_Result;
 event port Act_Service -> Service_addition.Act_Service;
 event port Act_Service -> Service_search.Act_Service;
 end Discovery_Agency.imp;
```

Figure 5. Discovery agency.

agency' is modeled via the event data port as shown in Figures 4 and 5.

Step 5

Specify service requester as periodic processes as shown in Figure 6.

Step 6

Determine an aperiodic thread in 'service requester' to receive services from 'service provider' as shown in Figure 6.

Step 7

Determine a periodic thread in 'service requester' to send out request to 'discovery agency' as shown in Figure 6.

S

The 'requester' thread and the 'interaction' thread in the process of 'service requester' are active in 'query' mode and 'int 'mode respectively.

```
process Service_Requester
 features
 new_query:        in out event data port ServiceQuery_record;
 query_Result:     in out event data port Service_record;
 interact_in:      port group basic::input_PT {
Required_Connection => false; };
 interact_out:     port group basic::output_PT {
Required_Connection => false; };
 properties
 Dispatch_Protocol => Periodic;
 Period => 30 Ms;
end Service_Requester;
process implementation Service_Requester.imp
 subcomponents
 Requester: thread Requester          in modes (Query);
 Interaction: thread Intraction       in modes (Int);
 connections
 event data port Requester.new_query -> new_query in modes
(Query);
 event data port query_Result -> Requester.query_Result in
modes (Query);
 event port Requester.Activation -> Interaction.Activation in
modes (Int);
 port group Interaction.interact_out -> interact_in {
Required_Connection => true in modes (Int); };
 port group interact_out -> Interaction.interact_in {
Required_Connection => true in modes (Int); };
 modes
 Query: initial mode ;
 Int:       mode ;
```

Figure 6. Service requester.

```
data ServiceDescription
end ServiceDescription;

data ServiceRequirements
end ServiceRequirements;
```

Figure 7. Data type.

Step 8

Communication between the 'service requester' and the 'discovery agency' is modeled via the event data port as shown in Figures 5 and 6.

Step 9

Communication between the 'service requester' and the 'service provider' is modeled via the port group as shown in Figures 4 and 6.

Step 10

Enable aperiodic threads through the event port as shown in Figures 3, 4 and 5.

Step 11

Specify service description and service requirements as data type as shown in Figure 7.

Service description is defined by the type of data which can be as a new service to be sent in the form of a service record to 'discovery agency'. Also, service requirement is defined by the type of data which can be sent in the form of a query to the 'discovery agency'.

Case study

To show how our modeling framework can be used to model different systems, we have considered an ATM machine. Figure 8 shows an ATM machine which has been described using our proposed modeling approach. It consists of three particular services to an ATM machine in the form of a thread. As previously mentioned, the Service_Provider.imp is the main part that should be changed. Services available include: 'balance', 'withdrawal' and if the user

```
process implementation Service_Provider.imp
        subcomponents
         Balance:  thread returnBalance in modes (Blnc);
         Withdrawal: thread group Withdrawal.imp in modes
(WithDrwl);
         ExitSP:    thread ChooseExit  in modes (Exit);
        Connections
         C1: event data port Balance.new_Service ->
new_Service in modes (Query,Int);
         C2: event data port Withdrawal.new_Service ->
new_Service in modes (Query,Int);
         C3: event data port ExitSP.new_Service ->
new_Service in modes (Query,Int);
         C4: event port interact_out.event_out ->
ActiveThread        in modes(Int);
         C5: event port ActiveThread -> Balance.BalanceSrv
        in modes (Int,Blnc);
         C6: event port ActiveThread ->
Withdrawal.WithDrwSrv      in modes (Int,WithDrwl);
         C7: event port ActiveThread -> ExitSP.Choose_Exit
        in modes (Int,Exit);
         C8: data port Balance.ShowBalance ->
interact_out.data_out in modes (Int);
         C9: data port Withdrawal.GrantMoney ->
interact_out.data_out in modes (Int);
         C10: event port ExitSP.InsertCard ->
interact_out.event_out in modes (Int);
        modes
         Query: initial mode;
         Blnc:      mode;
         WithDrwl: mode;
         Exit:      mode;
         Int:       mode;
         Int -[ interact_in.event_in ]-> Blnc;
         Int -[ interact_in.event_in ]-> WithDrwl;
         Int -[ interact_in.event_in ]-> Exit;
 end Service_Provider.imp;
```

Figure 8. ATM machine specified through AADL.

```
thread returnBalance
        features
         BalanceSrv: in event port;
         ShowBalance: out data port Basic::int;
         Exit:      out event port;
         new_Service: out event data port Service_record;
        properties
         SEI::SecurityLevel => 8;
         SEI::safetyCriticality => 8;
end returnBalance;
```

Figure 9. Return balance thread.

wants to cancel is 'exit'. As it is shown, Service_Provider.imp is composed of five operational modes in which the 'query' and 'int' modes are related to the entire system. Initially, the system is in the 'query' mode and in this mode information about the services is sent to 'discovery agency'. After that, 'service provider' enters to the 'int' mode. Then, the 'service requester' will request a special service and depending on the requested service and using the activated ports, the system enters to the desired mode. returnBalance thread defines the balance for the user. As it is shown in Figure 9, it has an input event port that is used to activate the returnBalance. It also has an output event data port that uses it to introduce new service to 'discovery agency'. Also, 'show_balance' data port is used to display the user account balance and 'exit', event port is used to prompt completion and enables ChooseExit thread. Withdrawal thread

```
thread group Withdrawal
        features
        WithDrwSrv:        in out event port;
        GrantMoney:        in out data port Basic::Money;
        new_Service: out event data port Service_record;
        properties
        SEI::SecurityLevel => 8;
        SEI::safetyCriticality => 8;
       end Withdrawal;
thread group implementation Withdrawal.imp
        subcomponents
        StnOrOth: thread ChooseStnOrOth in modes
(WithDrwl);
        Stn:      thread StandardP    in modes (Standard);
        Oth:      thread OthersP      in modes (Other);
        Money:  thread GrantMoney in modes
(Standard,Other);
        Exit:     thread ChooseExit   in modes
(Standard,Other);
        Connections
        C6: event port WithDrwSrv                    ->
StnOrOth.WithDrwSrv        in modes (WithDrwl);
        C7: event port StnOrOth.StandardDrw          ->
Stn.StandardDrw    in modes (Standard);
        C8: event port StnOrOth.OtherDrw
       -> Oth.OthersDrw            in modes (Other);
        C9: data port Stn.Select_StandardMoney       ->
Money.GrantMoneySrv in modes (Standard,Other);
        C10: data port Oth.Select_OthersMoney        ->
Money.GrantMoneySrv in modes (Standard,Other);
        C11: data port Money.GrantM
                -> GrantMoney      in modes
(Standard,Other);
        C12: event port Money.Exit                   ->
Exit.Choose_Exit in modes (Standard,Other);
        C13: event port Exit.InsertCard
       -> WithDrwSrv       in modes (Standard,Other);
        modes
        WithDrwl: initial mode;
        Standard: mode;
        Other:    mode;
        WithDrwl -[ StnOrOth.StandardDrw ]-> Standard;
        WithDrwl -[ StnOrOth.OtherDrw ]-> Other;
end Withdrawal.imp;
```

Figure 10. Withdrawal thread.

```
thread ChooseExit
        features
         new_Service: out event data port Service_record;
         Choose_Exit: in event port;
         InsertCard: out event port;
        flows
         ExitFlow: flow path Choose_Exit -> InsertCard;
         properties
         SEI::SecurityLevel => 8;
         SEI::safetyCriticality => 8;
end ChooseExit;
```

Figure 11. Choose thread.

group is composed of five threads which pay requested amount of money to the user as shown in Figure 10. According to the user request to use standard amounts provided by the ATM machine or to enter his/her desired amount, two threads are executed respectively, that is the StandardP and 'othersp' threads. GrantMoney thread pays money to the user and finally ChooseExit thread terminates operations. The ChooseExit thread is also used for the cases in which the user is not willing to continue operations or operations terminated (Figure 11).

Defining the 'security' and 'safety' levels for this system is very important. Using the features 'security level' and safety 'criticality' in this language, we can define the security level in components and subcomponents. As it is shown in this example, we have considered the level of safety and security equal to 8.

RESULTS AND DISCUSSION

These existing analyses are made available through a number of tools such as OSATE, SPICES, Stood, etc. Additional analysis can be implemented (for example, through OSATE since it supports plug-in development in an eclipse environment) (Hansson et al., 2010). As we mentioned earlier, these tools will automatically check the non-functional behavior and help us in creating a system with high security level. Architectural modeling can efficiently be used to verify security of system architectures and thus gain confidence in the system design. Using AADL and OSATE, the SEI has developed analytical techniques to represent standard security protocols for enforcing confidentiality and integrity and model and verify security using system architecture according to flow-based approaches early and often in the life cycle. Security as an architectural concern crosscuts all levels of the system (application, middleware, operating systems and hardware). Security requires intra- and inter-level verification and has immediate effects on the runtime behavior of the system, specifically on other dependability attributes. The designer seeks to ensure that the software applications do not compromise the confidentiality of the secure information they are exchanging. The following types of security verification and analysis are available as OSATE plug-ins:

i) Basic confidentiality principle-access should only be granted if given the appropriate security clearance.
Ii) Need-to-know principle access should be granted only to a resource if there is a need. Controlled sanitization lowering the security level of an object or subject should only be authorized and performed by a privileged subject.
iii) Non-alteration of object's security require a subject using an object as input should not alter the security level of the object, even if the object is updated as an output from the subject.
iv) Hierarchical condition: (1) a component has a security level that is the maximum of the security levels of its subcomponents, and (2) all connections are checked to determine whether the source component of a connection declaration has a security level that is the same or lower than that of the destination component (Hansson et al.,

2010). This tool from the Part in which system implementation provides an instance which different analysis can be done on it. In fact, the system will create instances based on spatial-defined architecture.

When we are defining the security and safety levels, the system checks if it can create spatial instances with this level of security or not. If the security is defined at a component level and have not defined for the associated subcomponents, the system will report an error and states the instance produced at 0 level, when we want to check security. This means that the security level must be considered in a component and all its associated subcomponents. Of course this feature is more visible when we want to bind software to hardware. By mapping the entities of a software architecture (that is, processes, threads and partitions) to a hardware architecture (consisting of, for example, CPUs, communication channels and memory), we can ensure that the hardware architecture supports required security levels. In addition security check and safety check, the analyst can perform the following analysis:

i) Checking model sanity.
ii) Checking connections binding consistency.
iii) Checking for consistency of port properties on connections.
iv) Checking property values for circular property references.
v) Checking ports for required connections.

In this paper, we tried to check all items stated on the code written for proposed approach and ATM machine and to specify an approach that will satisfy all the aforementioned cases.

Related work

Chaari et al. (2008) present a non-functional parameters-based framework for web service discovery and selection. The developed framework relies on an ontology-based categorization of service non-functional property. The selection process is achieved via peer-to-peer interactions among a NFP-based matching engine (NME) and community services. This framework can be used only for web services. Gönczy et al. (2007) present a metamodel to create a 'service-oriented architecture' with reliable message delivery. The formal techniques and graph transformation were used to achieve this goal. This approach only considers reliable messaging without considering the other kind of faults in an SoA -based application. The authors briefly look at this system and quality aspects in them. This approach also considers only one aspect of the non-functional properties and other aspects are ignored in this proposal. Rafe et al. (2009), Rafe and Mahdian (2011) and Mahdian et al. (2009) uses the style presented in (Gönczy et al., 2007) to consider different type of non-functional properties. They extend the formal style by adding new elements to the type graph and new graph transformation rules to consider new configurations. Even though using this approach, one can model different types of non-functional properties (for example fault tolerance and security); there is no proper approach to analyze them. Using this approach, one can analyze different functional properties through model checking on graph transformation system (Rafe et al., 2010; Baresi et al., 2008; Rafe and Rahmani, 2009).

Garcia and Toledo (2007) present architecture for fault tolerant and service-based business processes. The approach extends the basic web service architecture with the inclusion of broker and monitor components and a UDDI extension to create BPMS. Although the proposed extensions cause a significant additional cost, it is acceptable because of the offered benefits. To extend its business process architecture, they use two extra components. Also, they detect faults that occurred in the system.

Conclusion

In this paper, we proposed a formal framework to specify service-oriented applications. To do this, we have used AADL language to express SoA-based applications. We have proposed a framework which helps designers to model and express different parts of the system through AADL. Then, we explained our approach to analyze different non-functional properties on the models. Using the existing tools for AADL descriptions we can analyze different non-functional properties such as real-timeness, fault tolerance, security etc. As our future work, we have a plan to complete the work by presenting the approach through a style. To ease the using of the approach by different designers we have a plan to define the style by UML through proposing proper stereo types. Hence, it is possible to use the UML in the front while a formal and precise basis (that is ADDL) as the background.

REFERENCES

Baier C, Katoen JP (2008). Principles of Model Checking. Library of Congress Cataloging-in-Publication Data.

Baresi L, Heckel R, Thöne S, Varró D (2006). Style-Based Modeling and Refinement of Service-Oriented Architectures: *A Graph* Transformation-Based Approach. J. Software Syst. Modeling., 5: 187-207.

Baresi L, Rafe V, Rahmani AT, Spoletini P (2008). An Efficient Solution for Model Checking Graph Transformation Systems. Electronic Notes in Theoretical Computer Science (ENTCS). Elsevier Science B.V. 213: 3-21. ISSN: 1571-0661.

Chaari S, Badr Y, Biennier F, Favrel J, Ben AC (2008). Framework for Web Service Selection Based on Non-Functional Properties. In Proc. of Int. J. Web Serv. Pract., 3(1-2): 94-109.

Champion M, Ferris C, Newcomer E, Orchard D (2002). Web Service Architecture. W3C Working Draft, World Wide Web Consortium. Available: http://www.w3.org/TR/2002/WD-ws-arch-20021114/.

Chen V (1976). The entity-relationship model - toward a unified view of data. In Proc. of ACM Transactions on Database Systems, 1(9-36).
Feiler PH, Gluch DP, Hudak JJ (2006). The Architecture Analysis and Design Language (AADL): An Introduction. Technical Note. Carnegie Mellon University.
Garcia DZG, Toledo MBF (2007). An Architecture for Fault Tolerant and Service-based Business Processes. In Proc. of Brz. W. Business Process Management. Gramado. RSLiving experience through web. in conjunction with IEEE 11th Int. Conf. on Computational Science and Engineering.
Gilles O, Hugues J (2009). Towards Model-based optimizations of Real-Time systems, an application with the AADL. 15th IEEE Int. Conf. on Embedded and Real-Time Computing Systems and Applications.
Gönczy L, Kovács M, Varró D (2007). Modeling and Verification of Reliable Messaging by Graph Transformation Systems. Elect. Notes Theor. Comput. Sci., 175(4): 37-50.
Hansson J, Lewis B, Hugues J, Wrage L, Feiler P, Morley J (2010). Model-Based Verification of Security and Non-Functional Behavior using AADL. In Proc. of IEEE Secur. Priv., 8: 43-49.
Jonnaganti V (2009). An Integrated Security Model for the Management of SoA: *Improving the attractiveness of SoA Environments through a strong Architectural Integrity*. University of Gothenburg. Department of Applied Information Technology Gothenburg. Sweden. Report No. 2009-055. ISSN: 1651-4769.
Mahdian F, Rafe V, Rahmani AT, Rafeh R (2009). Modeling Fault Tolerant Services in Service-Oriented Architecture. in Proc. of Third IEEE International Symposium on Theoretical Aspects of Software Engineering (TASE09). China. pp. 319-320.
Pilioura T, Tsalgatidou A (2001). E-services: Current technology and open issues. In Proc. of the 1st W. Technologies for e-Services. TES 2001. 2193(LNCS): pp. 1-15.
Rafe V, Mahdian F (2011). Style-based modeling and verification of fault tolerance service oriented architectures. Procedia-Computer Science. Elsevier Science B.V, 3(1): 972-976.
Rafe V, Rahmani AT (2009). Towards Automated Software Model Checking Using Graph *Transformation Systems* and *Bogor.* J. Zhejiang University- Science A (JZUS)., (8): 1093-1105.
Rafe V, Rahmani AT, Baresi L, Spoletini P (2009). Towards Automated Verification of Layered Graph Transformation Specifications. J. IET Software., 3(4): 276-291.
Rafe V, Rahmani AT, Rafeh R (2010). Formal Analysis of UML 2.0 Activities Using Graph Transformation Systems. Int. J. Software Eng. Knowl. Eng., 20(5): 679-694.
SAE International, SAE Standards (2009). Architecture Analysis and Design Language (AADL). AS5506. November 2004. Available: http://www.sae.org/technical/standards/AS5506/1. May 25.
Thöne S (2005). Dynamic Software Architectures: A Style-Based Modeling and Refinement Technique with Graph Transformations. Ph.D. Thesis. University of Paderborn.