International Journal of Computer & Electronics Research

A Combined Model Checking Approach for Services Safety

José M. Reyes Álamo[1], Aparicio Carranza[2], Benito Mendoza[3]
[1,2,3]Department of Computer Engineering Technology
New York City College of Technology
186 Jay Street
Brooklyn, NY 11201
[1]jreyesalamo@citytech.cuny.edu, [2]acarranza@citytech.cuny.edu, [3]bmendoza@citytech.cuny.edu

***Abstract*— Pervasive computing environments such as the smart home often rely on composite services to provide different functionalities. These services are often complex, handle sensitive data, and perform critical operations. This raises several concerns especially those related to the safety of interactions among different services. In this paper, we differentiate services based on their characteristics and categorize them as baseline or extended. We propose a model checking mechanism to ensure that services in both categories meet the safety criteria.**

***Keywords:* Model checking; service-oriented architecture; open workflow nets; safety.**

I. INTRODUCTION

A smart home is a house that integrates different technologies to assist the elderly and persons with special needs to stay home longer and live more independently [1]. As services in a smart home might perform critical operations or manage sensitive data, it becomes critical to provide a mechanism to verify that services and their interactions are safe.

In this paper, we propose such mechanism by relying on formal software analysis [2]. Our work integrates different model checking techniques to verify whether composite services and their interactions satisfy a safety criteria. In the literature we found different composition frameworks that use model checking techniques targeted especially toward web services (WS)[3].

Our solution follows a similar strategy however in our work the interactions checked are among services of heterogeneous service-oriented architectures (SOAs) instead of only checking WS. Other works on automatic composition frameworks for services of heterogeneous SOAs either formally check the individual services or the resulting composite services. Our work combines both approaches and integrates them into a service composition framework[4]. The safety criteria checked consist of the standard or those properties that all services must comply with, and custom criteria that are properties that apply only to composite services.

The rest of the paper is organized as follows. Section 2 presents background information. Section 3 shows the strategy and architecture to model check composite services. Section 4 presents conclusions and future work.

II. BACKGROUND

Several model checking approaches are used in this work for modeling and checking composite services and their interactions. One of the model checkers used is SPIN which is a popular model checker that uses the Process Meta Language (PROMELA) to specify the model of a system. PROMELA models are then checked for satisfiability with a set of properties that are specified using Linear Temporal Logic (LTL). For more details about the SPIN mode checker the reader can refer to [5, 6].

This work also relies on the use of another model checker Fiona, which is especially designed to analyze and interpret open Workflow Nets (oWFN) [7]. The use of oWFN is relevant as they are specially designed to model service-oriented architectures. By using oWFN we obtain an efficient modeling structure that reduces the number of states during analysis when compared to other structures like canonical Petri nets. Fiona takes as input an oWFN and checks a standard set of properties that includes controllability, absence of cycles, and absence of false nodes. There exist several tools that generate oWFN from workflow specification files [8, 9] that can be used with Fiona to check these properties. The following paragraphsin this section present background information on the theory of oWFN. As oWFN are based on Petri nets, we start defining what is a Petri net followed by the properties that make a Petri net and oWFN. We then follow with the definition of a controller, an automaton used to determine controllability.

Definition 1: A *Petri net N* consists of:

- A set *P* of places, represented as a circle
- A set *T* of transitions, represented as a rectangle
- A flow relation *F*, where $F \subseteq (T \times P) \cup (P \times T)$, represented by the edges
- A marking *m* that is a multiset $m: P \rightarrow N$ (where $m[p]$, represents the number of tokens in place *p*). A token is represented by a dot.
- A marking *m* enables a transition *t* if for each place *p* with $(p, t) \in F$, $m[p] \geq 1$. If enabled at *m*, firing *t* yields the marking *m'* with $m'[p] = m[p] - 1$ if $(p, t) \in F$ and $(t, p) \notin F$, $m'[p] = m[p] + 1$ if $(t, p) \in F$ and $(p, t) \notin F$, and $m'[p] = m[p]$ otherwise.

Fig. 1 shows a graphical representation of a Petri net, with its places, transitions, and markings that may result from the firing process.

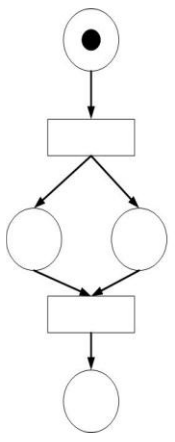
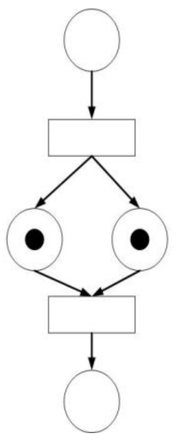
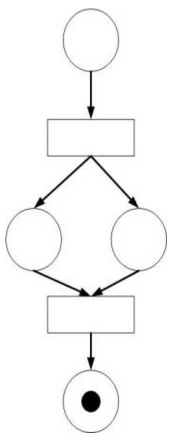

Figure 1: Graphical representation of a Petri net.

Definition 2: A Petri net N is an *open Workflow Net (oWFN)M* if:

- P is the disjoint union of the sets P_m, P_i and P_o, where:
    - P_m is the set of internal places
    - P_i is the set of input places with no incoming edges
    - P_o is the set the output places with no outgoing edges.
- $F \cap (P_o \times T) = \emptyset$
- $F \cap (T \times P_i) = \emptyset$
- F does not contain cycles (the transitive closure of F is irreflexive)
- M has a distinguished initial marking m_0
- M has a set Ω of distinguished final markings

Fig. 2 shows a graphical representation of an oWFN with its initial marking, input places and input transitions colored in orange, output places and output transitions colored in yellow, and final markings with only incoming edges colored in gray.

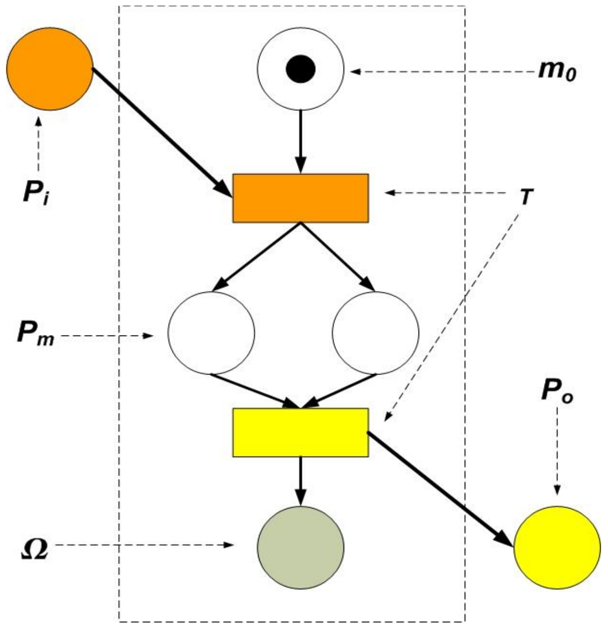

Figure 2: Graphical representation of an oWFN

Definition 3. Let M be an oWFN and $I \subseteq (P_I \cup P_O)$. A *controller* is an automaton connected to I containing the following elements:

- Alphabet *bags(I)*
- A set of states Q
- A move relation $\delta : Q \times bags(I) \rightarrow \rho(Q)$
- And an initial state q_0

Some of the safety properties that can be modeled and verified with oWFN are *weak soundness, soundness, usability, controllability, absence of cycles* and *absence of false nodes*. *Weak soundness* determines the possibility to reach an end state from each state reachable from the initial state. A *dead transition* is a transition that cannot fire. *Soundness* means that *weak soundness* holds and in addition there are no *dead transitions* on the net. *Usability* indicates that there exists an environment such that the oWFN of a service composed with the oWFN of the environment yields a weakly sound net. When a *controller* for the composition can be constructed from a given partner service for an oWFN and *soundness* holds this is known as *controllability*. The *absence of cycles* means that given a partner for an oWFN the composition does not create any cycle. A*bsence of false nodes* checks that given a partner for and oWFN, the input places matches the annotations of the corresponding output places. For our work, our safety criteria consist of checking an oWFN for *controllability, absence of cycles*, and *absence of false nodes*.

Several techniques can be used to check for *controllability* including computing a public view (*PV*), computing the interaction graph (*IG*) or computing the operating guideline (*OG*) [10]. A *PV* is an abstract representation of the operations and communication behavior of a service describing the public service interface. An *IG* is a structure that represents the controller point of view of a node where each node contains all states reachable at certain point. An *IG* can be seen as a hypothesis for the controller, representing feasible runs of a partner service. An *OG* is a structure that represents the behaviors of all possible strategies for sound executions. It describes how a partner service should behave instead of the actual service behavior. Computationally speaking of these three methods, computing the *PV* is the least costly but checking controllability is more expensive as *PV* suffers from the state explosion problem. The *PV* also reveals too much information about the service operations which may become an issue for applications with strong privacy requirements. The *IG* is relatively easy to compute and to verify, however it also reveals too much information about the possible states of the service. The *OG* is the hardest to compute but it is easy to verify. *OG* does not reveal information about the service itself as it describes how a partner should behave instead. Researchers in oWFN generally prefer*OG* because even though its computation is more expensive, this is a onetime operation. The extra computational cost is justifiable as the verification is efficient and also because the *OG* does not reveal information about the service itself, instead it describes the partner's expected behavior [11].

III. MODEL CHECKING COMPOSITE SERVICES

Our goal is to check the safety of composite services and their interactions in pervasive computing environments, using as an example the smart home. To achieve this, we classify services in the following categories:

- *Baseline services*: Services that are assumed to be part of the initial configuration whose safety criteria needs to be checked (e.g. notifications service).

- *Extended services*: Services that may be added, started, stopped, or removed at any time (e.g. monitor sugar levels every hour for two weeks).

For checking the safety of baseline services we use a semi-automatic approach that models services using PROMELA, and specifies its safety criteria using linear temporal logic LTL. We then check for compliance with the safety criteria using the SPIN model checker [5]. Fig. 3 presents the architecture where services are modeled and checked. The left-hand side shows baseline services and the right hand side the extended services. The Baseline Services Safety Requirements box at the top-left indicates all the requirements for a particular baseline composite service. Based on those requirements, the composite service is designed. Using the design, a model of the composite service is created using PROMELA. With these safety requirements, the safety criteria are modeled using LTL. By providing the PROMELA model and the LTL model, the SPIN model checker is used to verify whether the model complies with the specified safety criteria. If the model do not satisfies the safety criteria, it goes back to the design for model revision. If the model does satisfy the safety criteria, it passes to the implementation stage and finally the service is installed into the system.

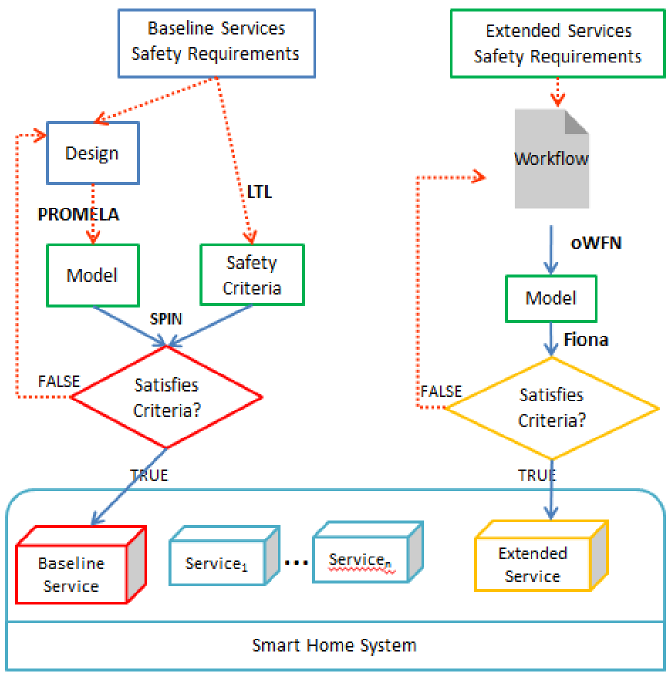

Figure 3: Model Checking Architecture for Baseline and Extended Services.

To model baseline services we used as an example the Medicine Information Support System (MISS), a system that integrates the doctor, the pharmacy, and the smart home to help the patient manage medications [12]. MISS ensures safety by checking drug interactions among medications, foods, and health conditions. The doctor, pharmacy, and smart home entities of MISS are modeled using PROMELA. We also modeled the medicine conflict database (MCD) and a client record request. The custom safety criteria to check are based on a set of statements from privacy laws modeled using LTL. For this work we modeled several statements from the Health Insurance Portability and Accountability Act (HIPAA) in the United States [13]. The law is modeled using a mapping of the legal language into formal language so that a computer system can understand it [14]. These LTL formulas are used as our custom safety criteria. In the models, a covered entity is defined as a patient, the government, or another entity that handles health records. The actual checking of whether MISS complies with the HIPAA law is performed using the SPIN model checker. The following is a list of some of the statements taken from the privacy law and modeled in LTL:

Statement 1: If a covered entity requests a patient's record, it will never receive an empty record.

LTL Formula:

#define p_1 (requestor == 2)
#define q_1 (patientData == true || restrictedData == true)
[] ((p_1) -> (<> (q_1)))

Explanation: The *requestor* variable indicates the requestor of the data and the value 2 represents a covered entity. The *patientData* and *restrictedData* variables represent the two components of a patient's record.

Statement 2: When patients request their medical record, only the patient's data is disclosed but never the restricted data.

LTL Formula:

#define p_5 (requestor == 1)
#define q_5 (patientData == true && restrictedData == false)
[] ((p_5) -> (<> (q_5)))

Explanation: The *requestor* variable indicates the requestor of the data with the value 1 representing the patient. The *patientData* and *restrictedData* variables represent the two components of a patient's record.

Statement 3: The doctor has to satisfy all the law requirements for covered entities over time.

LTL Formula:

#define p6 (doctorCE.patientReqs == true && doctorCE.adminReqs == true)
<>[] (p6)

Explanation: The *doctorCE. patientReqs* indicates whether the doctor is fulfilling the requirements for patients. The *doctorCE. adminReqs* indicates whether the doctor is fulfilling the administrative requirements.

For extended services, automatically checking the safety criteria is challenging. Therefore we modelled these services differently by relying on oWFN and checked the standard safety criteria using the Fiona model checker. Fiona [9] is a model checking tool that analyzes the safety of interactions among services modeled as oWFN [15]. Fiona can determine matching services, verify partner synthesis, and perform a check for controllability which is a minimal correctness criterion stating the existence of a behavioral compatible partner for the service. Fiona also provides the tools to create an adapter rule that serves as a mediator between different oWFN, mapping items from one oWFN into items in another. It can also check for absence of cycles and absence of false nodes. Fiona can produce the public view (*PV*), interaction graph (*IG*), and operating guideline (*OG*) of composite services automatically.

At the top-right of Fig. 3, the Extended Services Safety Requirements box indicates the requirements for a particular extended composite service. Based on these requirements the composite service and their interactions are specified as a workflow which is then modeled as an oWFN automatically. The oWFN model is then passed to the Fiona model checker and checked for *controllability, absence of cycles,* and absence of *false nodes*. Checking for *absence of cycles* avoids infinite calls to services. Checking *false nodes* helps to identify those nodes that violate their own annotation. For computing *controllability*, we primarily rely on the use of *OG*. In our work we also provide support for using *IG* but we generally do not use *PV*.

If the safety criteria are not satisfied, the service is not accepted. If the safety criteria are satisfied and no errors are found, the composition framework proceeds to automatically generate the implementation of the service. Finally, the composite service implementation is deployed into the system such as the Smart Home in our example.

IV. Conclusions and future work

We present a model checking approach to ensure the safety of composite services. Services are categorized as baseline or extended. Baseline services are assumed to be always up and running, while extended services may come and go. We present an architecture where safety criteria for both types of services are checked with the appropriate tool for each case: SPIN for baseline services and Fiona for extended services.

The model checking approach for extended services supports heterogeneous SOAs and it is integrated with the system allowing only implementation of services proven to be safe. Having a safety criteria and a checking mechanism is important as composite services in environments like a Smart Home might perform critical operation or handle sensitive data. Our approach and its integration into an automatic composition framework ensures that services are continuously fulfilling the safety criteria and that only new composite services that meet the safety criteria are accepted. We have developed a prototype of the system using a Smart Home as an example, and are currently expanding it to include more modeling scenarios and integrate other safety policies.

References

[1] N. Noury, G. Virone, P. Barralon, J. Ye, V. Rialle, and J. Demongeot, “New trends in health smart homes,” in *Enterprise Networking and Computing in Healthcare Industry, 2003. Healthcom 2003. Proceedings. 5th International Workshop on*, 2003, pp. 118–127.

[2] M. B. Dwyer, J. Hatcliff, R. Robby, C. S. Pasareanu, and W. Visser, “Formal Software Analysis Emerging Trends in Software Model Checking,” in *Future of Software Engineering, 2007. FOSE ’07*, 2007, pp. 120–136.

[3] H. Schlingloff, A. Martens, and K. Schmidt, “Modeling and Model Checking Web Services,” *Electron. Notes Theor. Comput. Sci.*, vol. 126, pp. 3–26, Mar. 2005.

[4] J. M. Reyes Álamo, H.-I. Yang, J. Wong, and C. K. Chang, “Automatic Service Composition with Heterogeneous Service-Oriented Architectures,” in *Aging Friendly Technology for Health and Independence*, Y. Lee, Z. Z. Bien, M. Mokhtari, J. T. Kim, M. Park, J. Kim, H. Lee, and I. Khalil, Eds. Springer Berlin Heidelberg, 2010, pp. 9–16.

[5] G. J. Holzmann, *The SPIN Model Checker: Primer and Reference Manual*. Addison-Wesley Professional, 2003.

[6] G. J. Holzmann, “The Model Checker SPIN,” *Softw. Eng.*, vol. 23, no. 5, pp. 279–295, 1997.

[7] T. Murata, “Petri nets: Properties, analysis and applications,” *Proc. IEEE*, vol. 77, no. 4, pp. 580, 541, 2002.

[8] N. Lohmann, “A feature-complete Petri net semantics for WS-BPEL 2.0 and its compiler BPEL2oWFN,” *WS-FM 2007 LNCS*, 2007.

[9] P. Massuthe and D. Weinberg, “Fiona: A Tool to Analyze Interacting Open Nets,” in *Proceedings of the 15th German Workshop on Algorithms and Tools for Petri Nets, AWPN 2008, Rostock, Germany, September 26–27, 2008*, 2008, vol. 380, pp. 99–104.

[10] K. Schmidt, “Controllability of open workflow nets,” *EMISA LNI Bonn. KÖLLEN Verl.*, vol. 75, pp. 236–249, 2005.

[11] P. Massuthe, W. Reisig, and K. Schmidt, “An Operating Guideline Approach to the SOA,” *Ann. Math. Comput. TELEINFORMATICS*, vol. 1, pp. 35–43, 2005.

[12] J. M. Reyes Álamo, H.-I. Yang, J. Wong, R. Babbitt, and C. Chang, “Support for Medication Safety and Compliance in Smart Home Environments,” *Int. J. Adv. Pervasive Ubiquitous Comput.*, vol. 1, no. 3, pp. 42–60, 2009.

[13] OCR, “Summary of the HIPAA Privacy Rule.” [Online]. Available: http://www.hhs.gov/ocr/privacy/hipaa/understanding/summary/. [Accessed: 07-Aug-2014].

[14] M. J. May, C. A. Gunter, and I. Lee, “Privacy APIs: Access Control Techniques to Analyze and Verify Legal Privacy Policies,” in *Proceedings of the 19th IEEE workshop on Computer Security Foundations*, 2006, pp. 85–97.

[15] N. Lohmann, “A Feature-Complete Petri Net Semantics for WS-BPEL 2.0,” in *Web Services and Formal Methods*, 2008, pp. 77–91.