LTSA-PCA: Tool Support for Compositional Reliability Analysis

Pedro Rodrigues, Emil Lupu, Jeff Kramer
{pedro.rodrigues, e.c.lupu, j.kramer}@imperial.ac.uk
Imperial College London, UK

ABSTRACT

Software systems are often constructed by combining new and existing services and components. Models of such systems should therefore be compositional in order to reflect the architectural structure. We present herein an extension of the LTSA model checker [1]. It supports the specification, visualisation and failure analysis of composable, probabilistic behaviour of component-based systems, modelled as Probabilistic Component Automata (PCA) [2]. To evaluate aspects such as the probability of system failure, a DTMC model can be automatically constructed from the composition of the PCA representations of each component and analysed in tools such as PRISM [3]. Before composition, we reduce each PCA to its interface behaviour in order to mitigate state explosion associated with composite representations. Moreover, existing behavioural analysis techniques in LTSA can be applied to PCA representations to verify the compatibility of interface behaviour between components with matching provided-required interfaces.

Categories and Subject Descriptors

D.2.2 [**Design Tools and Techniques**]: State diagrams; D.2.4 [**Software/Program Verification**]: Model checking, Reliability

General Terms

Design, Reliability, Verification

Keywords

Compositional Reliability Analysis

1. INTRODUCTION

Architectural Description Languages such as Darwin [4] have been used for specifying composite component structures and their bindings, complemented by Labelled Transition Systems to model and compose the non-probabilistic behaviour of components. Other tools, such as Möbius [5], can be used for performance and reliability analysis but are not able to represent and reason about probabilities of failures independently from the duration of actions. Current support for reliability analysis, *e.g.* PRISM [3], is based upon Discrete Time Markov Chain (DTMC) models of the entire system. The DTMC model of a composite component cannot be derived automatically from DTMC representations of its subcomponents. Furthermore, as probabilities of state transitions are obtained through system profiling, non-composable representations require profiling afresh after each change in the architectural configuration. In contrast, composable models allow independent profiling of each component.

We have previously defined Probabilistic Component Automata (PCA) [2] to address the aforementioned problems associated with non-composable models for probabilistic behaviour, while catering for the specification of failure scenarios, failure propagation and failure handling. To illustrate our approach and implementation we apply it to an e-commerce system adapted from [6]. The models used in this paper and the LTSA-PCA tool are available at `http://wp.doc.ic.ac.uk/dse/software/`.

2. PROBABILISTIC FSP

In order to extend the LTSA tool to support PCA models we need to define an extension to Finite State Processes (FSP). We denote this extension Probabilistic Finite State Processes (P-FSP), whose operator semantics are presented after introducing PCA.

2.1 Probabilistic Component Automata

PCA models distinguish between internal actions and the input/output actions that correspond to the provided/required interfaces of component-based models [4]. Input actions model the receiving end of a communication channel or methods that can be called (*e.g.* interface methods); whereas output actions model method invocations or the transmission of messages.

A Probabilistic Component Automaton is defined as $A = (S, q, A, \Delta, \mu)$, where:

- S is a set of states; q is the initial state;
- $A = A^{in} \cup A^{loc}$: A^{in} are input actions that follow reactive semantics; $A^{loc} = A^{int}$ (internal non-observable actions) $\cup A^{out}$ (output actions) are locally controlled generative actions;
- $\Delta \subseteq (S - \{\pi\} \times A \times S)$ is the set of transitions, π represents the error state;
- $\mu : (S - \{\pi\} \times A \times S) \in \Delta \rightarrow [0, 1])$: denotes the probability of reaching state s' from state s through the execution of action a.

ICSE '14, May 31 - June 7, 2014, Hyderabad, India

2.2 Prefix ' -> ' and Choice ' | '

The basic operators for specifying PCA models are prefix and choice. The prefix operator defines a transition between states. The transition consists of *a)* an action type: ? for input, ! for output, no-symbol for internal, $\sim$ for internal failures, $\sim$? for input failures and $\sim$! for output failures; *b)* the execution probability $< p >$, and *c)* the action label $a \in A$. While internal (output) failure actions are used to model the failure of the corresponding internal (output) action, input failure actions are used to model failure handling behaviour (see sub-section 2.3 for more details).

If a_i are locally controlled actions (internal or output), then $(\rho_i\ a_i\ \text{->}\ \ E_i \,|\ldots|\, \rho_n\ a_n\ \text{->}\ \ E_n)$ describes a PCA that initially engages in action a_i with probability ρ_i. In this case, the choice operator can be used to model `if` or `switch` statements. On the other hand, if a_i are input actions the control over which action the PCA engages in is dictated by the environment, *i.e.* other processes that output a_i. This construction can be used to model how the provided interfaces of a component are used by other components, which is only known for a specific architectural configuration.

The following P-FSP expression represents an example of a communication that may fail with a 3% probability.

```
!<0.97> logoutUserConfirmation -> <1.0> registerLogout
-> AUTH_SERVICE
| ~!<0.03> logoutUserConfirmation -> ERROR
```

2.3 Parallel Composition ' ∥ '

If P and Q are PCAs then $P \,\|\, Q$ represents the concurrent execution of P and Q. Synchronisation occurs between matching input and output actions, whereas internal actions are interleaved. Moreover, two PCAs are composed in parallel iff the following conditions hold: *a)* $E_P^{int} \cap A_Q = \emptyset$; *b)* $E_Q^{int} \cap A_P = \emptyset$; *c)* $E_P^{in} \cap E_Q^{in} = \emptyset$; *d)* $E_P^{out} \cap E_Q^{out} = \emptyset$. These conditions ensure that synchronisation occurs solely between a pair of input and output actions, thus representing the interaction between only two components. The composite PCA $P \,\|\, Q$ is defined as:

- If $P = \Pi$ or $Q = \Pi$, then $P\|Q = \Pi$;
- for $P = \langle S_1, q_1, A_1, \Delta_1, \mu_1\rangle$ and $Q = \langle S_2, q2, A_2, \Delta_2, \mu_2\rangle$ such that $P \neq \Pi$ and $Q \neq \Pi$;
- $P\|Q = \langle S_1 \times S_2, (q_1, q_2), A_1 \cup A_2, \Delta, \mu\rangle$, where Δ and μ are the smallest relation satisfying the following rules.

Rule 1 specifies communication between components is represented by the synchronisation between input and output actions, which then become an internal action in the composite model.

$$\frac{P \xrightarrow{(?a, p_a)} P' \,, Q \xrightarrow{(!a, p_{a'})} Q'}{P\|Q \xrightarrow{(a,\ \frac{p_a \times p_{a'}}{\eta})} P'\|Q'} \tag{1}$$

Rule 2 determines that when two components are concurrently executing internal actions, these actions are interleaved in the composite model. However, to preserve the generative semantics associated with locally controlled actions (*i.e.* the sum of the probabilities of all outgoing transitions labelled with locally controlled actions needs to be equal to 1), a normalisation factor η is applied to the probabilities of locally controlled actions actions [2].

$$\frac{P \xrightarrow{(a, p_a)} P'}{P\|Q \xrightarrow{(a,\ \frac{p_a}{\eta})} P\|Q'} \ ,\ \frac{Q \xrightarrow{(b, p_b)} Q'}{P\|Q \xrightarrow{(b,\ \frac{p_b}{\eta})} P\|Q'} \tag{2}$$

Rule 3 describes how input failure actions are used to handle failure scenarios.

$$\frac{\begin{array}{c} P \xrightarrow{(!a,\, p_a)} P' \,,\, P \xrightarrow{(\sim !a,\, p_f)} ERROR \\ Q \xrightarrow{(?a,\, p'_a)} Q' \,,\, Q \xrightarrow{(\sim ?a,\, p'_f)} Q'' \end{array}}{P\|Q \xrightarrow{(a,\ \frac{p_f \times p'_a}{\eta})} P\|Q''} \tag{3}$$

Rule 4 determines that when failure input actions are not specified, failure output actions are propagated as internal failure actions of the composition on the composite PCA, consequently leading to a global `ERROR` state.

$$\frac{P \xrightarrow{(!a,\, p_a)} P' \,,\, P \xrightarrow{(\sim !a,\, p_f)} \texttt{ERROR} \,,\, Q \xrightarrow{(?a,\, p'_a)} Q'}{P\|Q \xrightarrow{(\sim a,\ \frac{p_f \times p'_a}{\eta})} \texttt{ERROR}} \tag{4}$$

2.4 Re-Labelling ' / '

The previous operators support the specification of basic and composite components with single bindings to each provided interface. To cater for modelling of shared resources, the re-labelling operator / is used to replace the interface actions of a component with multiple transitions in order to support multiple bindings. Accordingly, the components that share the common resource need to replace their interface actions so that individual requests from each component can be distinguished.

Formally, the re-labelling operator applies a relation over action labels. When the relation is $\{a\} \times L$, $a \in A$, each transition $(s, a, s') \in \Delta$ is replaced by $\#(L)$ transitions labelled with the action labels in L. In addition to the re-labelling as used in LTS, the probability associated with those transitions is defined by $\frac{\mu(s,a,s')}{\#(L)}$, where $\mu(s, a, s')$ denotes the probability of reaching state s' from state s through the execution of action a.

2.5 Hiding ' \ '

When applied to a PCA P, the hiding operator $\backslash\{a_1, \ldots, a_x\}$ collapses, when possible, the transitions in P labelled with the action names $\{a_1, \ldots, a_x\}$, while maintaining the probabilistic reachability properties of the original process. This operator is used to reduce a component's PCA to its interface behaviour representation based upon the architectural configuration of the system. For instance, the reduced PCA does not contain behaviour associated with unbound provided interfaces as it corresponds to unused behaviour. Further details can be found in [2].

3. EXAMPLE

The e-commerce system used by Filieri *et. al* [6] consists of a web-service that sells merchandise and integrates three external web-services: authentication, shipping and payment. The original DTMC model [6] is a closed representation of the entire system and has not been constructed from the models of its components. Therefore, if the system configuration changes, a new DTMC representation needs to be defined which is both laborious and error prone.

To illustrate our approach, we assume that the system is constructed from the components shown in Figure 1. Based on the original DTMC model we have specified the behaviour of each component in P-FSP. For instance, the P-FSP for the Authentication Service is shown below and its graphical representation is depicted in Figure 2.

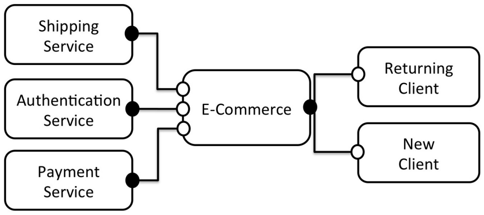

Figure 1: Architecture of e-commerce system [6]

```
AUTH_SERVICE =
  (?<1.0> loginUser ->  <1.0> checkCredentials ->
      (  !<0.97> authenticatedUser -> AUTH_SERVICE
      | ~!<0.03> authenticatedUser -> ERROR)
  | ?<1.0> signUpUser -> <1.0> createUser ->
      (  !<0.97> createdUser -> AUTH_SERVICE
      | ~!<0.03> createdUser -> ERROR)
  | ?<1.0> logoutUser -> <1.0> finishSession ->
      (  !<0.97> logoutUserConfirmation -> AUTH_SERVICE
      | ~!<0.03> logoutUserConfirmation -> ERROR) ).
```

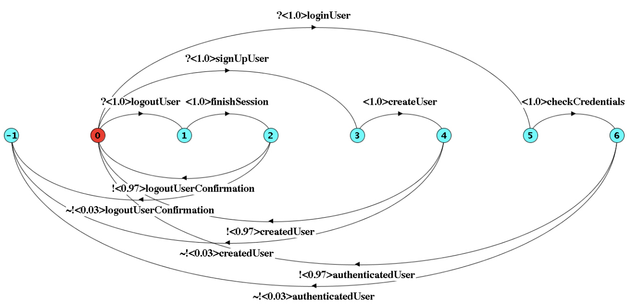

Figure 2: PCA of the Authentication Service

The architectural configuration of the e-commerce system determines the components and their bindings, which in turn determines how component instances are composed to construct the system (shown in Figure 3). For instance, as `E_Commerce` is a single-threaded component shared by two clients, its interface actions that represent the interactions with the two clients need to be re-labelled (section 2.4) using the following P-FSP expression.

```
||E_COMMERCE_TWO_CLIENTS = (E_COMMERCE / {
    {new.signUp}/signUp,
    {ret.login}/login,
    {new.confirmation, ret.confirmation}/confirmation,
    {new.searchProduct, ret.searchProduct}/searchProduct,
... } ).
```

Accordingly, the corresponding actions in the `NEW_CLIENT` and `RETURNING_CLIENT` PCAs also need to be re-labelled before composing them with the `E-Commerce`'s PCA.

In contrast with FSP, for which the hiding operator makes unobservable (internal) actions silent, we extend this operator for P-FSP (sub-section 2.5) to also delete non-observable transitions and propagate their probabilities to remaining transitions. Therefore, it can be used to reduce (*cf.* minimisation) the representation of each component to its interface behaviour, *i.e.* limited to input/output actions. The overall system representation can then be computed using the interface representation of each component, resulting in a smaller model. As the reduction algorithm preserves the probabilistic reachability properties of the system, the reduced representation can still be used for reliability analysis, while producing significant improvements in scaleability [2].

```
||E_COMMERCE_SYS = ( (E_COMMERCE_TWO_CLIENTS ||
    ret:RETURNING_CLIENT || new:NEW_CLIENT)
        || AUTH_SERVICE || SHIPPING_SERVICE
        || PAYMENT_SERVICE)
    @ {ret.login, new.signUp, ret.searchProduct,
        new.searchProduct, new.normalShipping,
        ret.normalShipping, new.expressShipping,
        ret.expressShipping, new.checkout,
        ret.checkout, new.logout, ret.logout}.
```

As PCA models cater for both failure scenarios and failure handling, we have further adapted the behaviour of the `E-Commerce` component to include handling behaviour for the failures of its required services. As a result, the composite PCA for the e-commerce system constructed using the above P-FSP expression and shown in Figure 3 does not contain failure transitions as they have all been handled, *i.e.* there are matching failure handlers in connected components.

4. ANALYSIS

The composite PCA representing the e-commerce system has full probabilistic information of the system and is automatically translated to a DTMC model for reliability analysis in the PRISM Model Checker [3]. The DTMC model is annotated with state variables to represent successful executions of the system (`finish`) and the occurrence of failures (`fail`). Therewith, the probability of the system successfully executing without failures for a single request is given by the following generic formula: `1 - (P = ?  [F fail & !finish])`. This analysis can be generalised to verify the likelihood of a system failure after N requests. In contrast to using a single failure probability for each component [7], PCA models result in a more accurate reliability analysis as probabilities of failure are associated with actions of a component representation. Moreover, PCA models are also more responsive to changes in the components execution profile.

The existing behavioural analysis tools in LTSA have been extended to be applied to PCA. For instance, safety analysis can be used to verify the existence of failure scenarios that have not been handled. The following is the shortest error trace for the Authentication Service produced by LTSA-PCA.

```
Trace to property violation in AUTH_SERVICE:
    ?<1.0>loginUser -> 5
    <1.0>checkCredentials -> 6
    ~!<0.03>authenticatedUser -> -1
```

Safety analysis is also used to verify the compatibility of two components with respect to their interface behaviour. For instance, wwo PCAs have incompatible interface behaviour when the components they model use different versions of a communication protocol. In this case, the components may block due to the following contexts: *a)* the two components are both waiting for a message to be sent by the other component, *b)* an unexpected message, with respect to the communication protocol, has been sent by one of the components and therefore cannot be synchronised. These scenarios can be analogously applied for methods calls. In both cases, these situations are represented as deadlock states in the composite behaviour model of the system, denoting that the system cannot progress further. Safety analysis produces

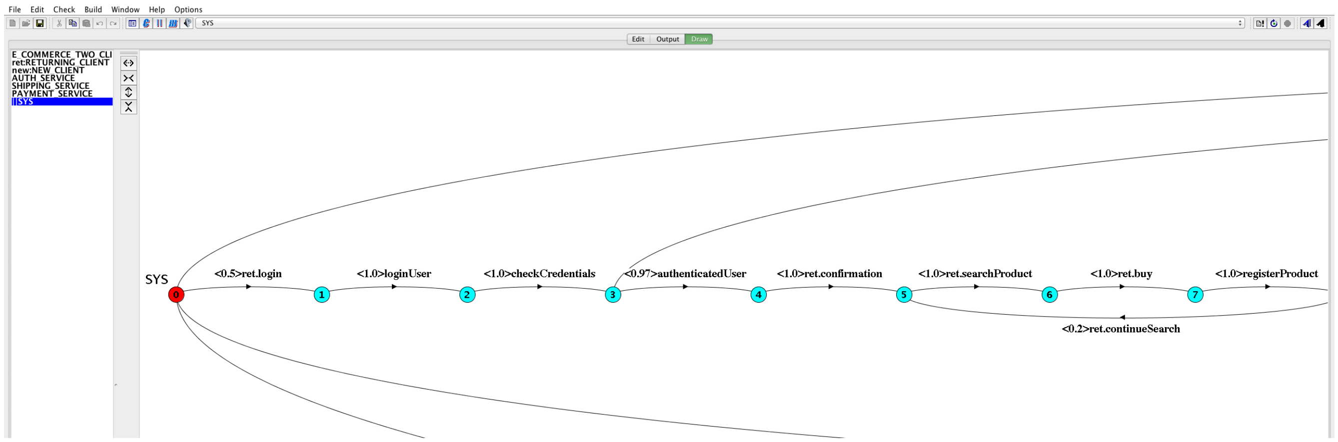

Figure 3: Composite PCA model of e-commerce system [6]

the shortest trace to a deadlock state, equivalent to the one presented for error traces.

5. PERFORMANCE EVALUATION

The performance gains for reliability analysis when using the reduction algorithm have been reported in [2]. Herein we compare the time to compile and compose probabilistic and non-probabilistic Finite State Processes. We report the results in table 1 for the e-commerce system and for some of the systems analysed in [2]. The overhead introduced by PCA approximately doubles the time to construct the composite PCA of each system as the parallel composition operator has to additionally consider action types and probabilities. Nonetheless, the efficiency gains produced by reducing each component representation before composition overcome the additional complexity of constructing PCA models.

Name	FSP	P-FSP	P-FSP & Red.
e-Commerce [6]	44ms	47ms	48ms
Tele Health [8]	42ms	46ms	47ms
Web-Server (5 Clients)	43ms	72ms	52ms
Web-Server (6 Clients)	75ms	125ms	54ms
Web-Server2 (3 Clients)	69ms	125ms	55ms
Web-Server2 (4 Clients)	780ms	1.9s	110ms

Table 1: Evaluation Results

6. CONCLUSION

Building upon the existing features of the LTSA tool, we have included tool support for PCA models by defining a probabilistic extension to FSP and extending the original operators to cater for the semantics of PCA models. All the existing tools for constructing, visualising and analysing composable models have also been extended to support PCA representations. Using PCA to model the probabilistic behaviour of a component results in a representation that is independent from the context in which it is deployed. Consequently, each component can be independently profiled and its representation can be reused for different system assemblies. When considering alternative architectural configurations, the behaviour model of each system configuration can be automatically constructed from the independent PCA representations of its components. As the reduction algorithm preserves the reliability properties of a PCA, the LTSA-PCA tool can be used for automatic construction of smaller composite models for different architectural configurations, thus significantly reducing the complexity of analysing the reliability of each one.

Finally, the tool currently requires the user to copy the translated model to PRISM when performing reliability analysis. We plan to fully integrate reliability analysis into LTSA-PCA by verifying pre-defined generic properties.

Acknowledgements

This work was supported by Fundação para a Ciência e Tecnologia under the grant SFRH/BD/73967/2010.

7. REFERENCES

[1] J. Magee and J. Kramer, *Concurrency - state models and Java programs (2. ed.)*. Wiley, 2006.

[2] P. Rodrigues, E. Lupu, and J. Kramer, "Compositional probabilistic reachability analysis for probabilistic component automata," *Technical Report 2014/2 - DoC, Imperial College London*, 2014.

[3] M. Kwiatkowska, G. Norman, and D. Parker, "PRISM 4.0: Verification of probabilistic real-time systems," in *CAV'11*, ser. LNCS, G. Gopalakrishnan and S. Qadeer, Eds., vol. 6806. Springer, pp. 585–591.

[4] J. t. Magee, "Specifying distributed software architectures," in *ESEC*. London, UK: Springer-Verlag, 1995.

[5] G. t. Clark, "The mobius modeling tool," in *Inter. Workshop on Petri Nets and Perf. Models*, 2001.

[6] A. Filieri, C. Ghezzi, and G. Tamburrelli, "A formal approach to adaptive software: continuous assurance of non-functional requirements," *Form. Asp. Comput.*, vol. 24, no. 2, pp. 163–186, Mar. 2012.

[7] R. Cheung, "A user-oriented software reliability model," *Software Engineering, IEEE Transactions on*, vol. SE-6, no. 2, pp. 118 – 125, march 1980.

[8] I. t. Epifani, "Model evolution by run-time parameter adaptation," in *ICSE*. Washington, DC, USA: IEEE Computer Society, 2009, pp. 111–121.