Towards the Automated Synthesis of Data Dependent Service Controllers

Franziska Bathelt-Tok and Sabine Glesner

Technische Universität Berlin, Germany
Software Engineering for Embedded Systems
bathelt-tok@soamed.de

Abstract. The treatment of data is a crucial step in service composition but it is currently done manually and informally. This makes the development process time-consuming, expensive, and error-prone, which is serious in safety-critical domains like the medical area. To overcome this problem, we present a novel approach for synthesizing data-dependent service controllers automatically based on composition and analysis methods for algebraic Petri nets. Consequently, our approach allows the automated, fast, and cost-efficient synthesis of correct controllers regarding data-dependent functional and safety-critical properties, which enables a reliable interoperability of medical devices.

1 Introduction

The idea of service-oriented architectures (SOAs) is to develop services with sophisticated functionality by composing simpler services appropriately. Services that are independently developed by different providers often cannot communicate with each other directly and cannot be adapted internally. Thus, it is necessary to develop another component, a controller, that routes and modifies messages depending on the exchanged data. It can be developed by a synthesis process that must ensure given data-dependent properties and, moreover, be correct with respect to safety-critical and functional requirements. Current approaches abstract from data and, consequently, model data-dependent choices as nondeterministic choices, which leads to invalid controllers. Thus, it is necessary to manually refine these controllers afterwards. This makes the controller development process expensive, time-consuming, and most of all error-prone.

To overcome this problem, we propose an approach to synthesize data-dependent controllers automatically based on algebraic Petri nets and a specification language that enables the presentation of causal dependencies of properties, which have to be fulfilled. It is ongoing work to show that the resulting controller is correct-by-construction and considers given data-dependent requirements. For our approach, we assume that, firstly, a formal representation as algebraic Petri nets of each service behavior is given. Secondly, we assume that the interface matching, which describes the mapping between the interfaces, is given. Thirdly, the set of data-dependent requirements to be fulfilled must be listed. Challenges of our work are the formal definition of the symbolic data treatment and the

A.R. Lomuscio et al. (Eds.): ICSOC 2013 Workshops, LNCS 8377, pp. 528–534, 2014.

generic data-dependency detection. To realize this and automatize the controller synthesis, we firstly translate the inputs into algebraic Petri nets as a uniform representation. This enables an easy understanding of the single component's behavior and of the causal dependencies between the single properties. Based on this, we compose the individual Petri nets to an over-all Petri net using the analysis methods Petri nets offer. As result, we achieve a single Petri net that fulfills all requested requirements, if it exists. Finally, we extract the required controller from this net. We have already implemented the translation step, and we are currently working on the implementation of the composition and the extraction step. The resulting controller

1. is synthesized automatically using the inputs,
2. handles different data types and considers data-dependent behavior,
3. ensures that all data-dependent properties are fulfilled,
4. is able to deal with synchronous and asynchronous communication, and
5. should be correct by construction w.r.t. safety-critical and functional requirements.

If we are able to verify the correctness, then we can apply our approach to safety-critical domains like the medical area, where different devices have to communicate with each other. As the single devices offer different functionalities that must be composed, we can shift the problem of enabling the interoperability of the medical devices to the more general problem of service composition.

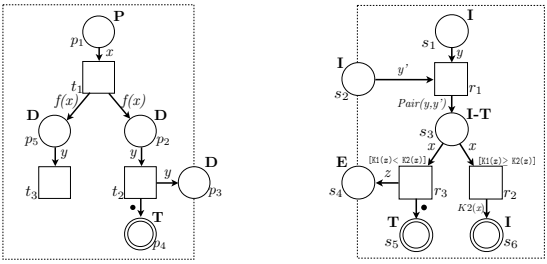
The rest of this paper is organized as follows. Based on a running example we sketch the main formalisms on which our approach is based in Section 2. In Section 3 we present our approach. We then discuss related work in Section 4. Finally, in Section 5, we conclude and give an outline of future work.

2 Background

In this section, based on a running example we give an intuition of algebraic Petri nets [Rei91] and a specification language, which we use in our approach.

2.1 Running Example

As running example, we focus on a medical scenario known as artificial pancreas. The artificial pancreas comprises two components to be composed reliably, a glucose sensor and an insulin pump that should react on the sensor but isn't able to do so at the moment. Figure 1 shows an excerpt of the behavior of the sensor (Fig. 1a) and the pump (Fig. 1b). The chemical component of the sensor reacts with the blood glucose and offers a current flow that is measured (x), transformed in a digital signal ($f(x)$), and then stored internally (at p_5) and sent to the environment (y). The insulin pump compares the required amount of insulin recieved by the environment (y') with the amount of insulin that is still available in the pump (y). If not enough insulin is available a warn signal is sent to the environment, otherwise the requested amount is injected via a catheter.



(a) glucose sensor (b) insulin pump

Fig. 1. Running example

2.2 Algebraic Petri Nets

The behavior of the components in Figure 1 is visualized by algebraic Petri nets. In this section, we give an intuition of their main elements using the representation of the insulin pump's behavior (see Figure 1b). Generally, *Petri nets* consist of places ($P = \{s_1, \dots, s_6\}$), transitions ($T = \{r_1, r_2, r_3\}$) and a flow relation ($F = \{(s_1, r_1), (s_2, r_1), \dots\}$). Transitions can be restricted by guards ($G = \{[K1(x) < K2(x)], \dots\}$) that define in which case a transition is active and can fire. If the nets are inscribed by algebraic specifications, then they are called algebraic Petri nets. An *algebraic specification* is a 4-tuple $SP = (S, OP, X, EQ)$ comprising a set of: sorts S, operations OP, variables X and equations EQ. In our case, sorts ($S = \{I, I-T, T, E\}$) assigned to places represent the data types of the signals abstractly. The used variables ($X = \{y, y', x, z\}$) symbolically represent data values and operations ($OP = \{Pair : I \times I \to I - T, \cdot :\to T \dots\}$) represent the modification of data. These elements are assigned to arcs. Apart from this, the set of equations ($EQ = \{Pair(y, y') = x, \dots\}$) defines the relationship between the arc inscriptions and, thus, restricts the possible domains of the data types. Finally, places are marked with structured tokens, which are tokens with properties defined by the structure of the sorts. Thus, we can consider primitives as well as complex data types.

2.3 Specification Language

The interface matching and the requirements describe different states of markings and causalities. Thus, we need a specification language that is able to cope with state and run properties. For our approach, we use so called run properties [Rei13]. We just give an intuition based on the property '$(p_1.X_P) \mapsto (p_4.X_T \wedge p_3.X_D)$' of our example (Figure 1a). This property states that the interface and the final state is reached from the initial marking, i.e. the glucose sensor is deadlock-free. Here, three elements are represented. Firstly, the *state property* ($p_1.X_P$) says that the place p_1 is marked with the data X_P. State properties can be combined using *propositional operators* ($\neg, \wedge, \vee, \to$). This is the case in the second part ($p_4.X_T \wedge p_3.X_D$). This states that there is a marking where X_T lies on p_4 and, at the same time X_D lies on p_3. Finally, we have the

leads-to operator $\mapsto$ as temporal component. Thus, the whole expression states that *whenever* p_4 *is marked by* X_P *there exists a marking at the same time or in the future where* p_4 *is marked with* X_T *and* p_3 *with* X_D.

3 Synthesis of Data Dependent Service Controllers

Since the sensor must be replaced every 4-7 days, an automatic controller synthesis is necessary to enable a reliable and cost-efficient interoperability of these devices and, thus, enhance the quality of life of people with type-1-diabetes.

As shown in Figure 2, our approach starts from the formal representation of the involved services given as algebraic Petri nets. We begin by extracting the relevant service behavior using the initial representation. This *behavioral abstraction* (Sec. 3.1) leads to a set of properties, expressed by the specification language, which have to be fulfilled at least. The main challenge is to ensure that all important properties are extracted and all information and dependencies are retained. Thus, a formal model for storing data and data-dependencies is needed. Our aim is to combine the resulting behavioral abstraction, the interface matching and the requirements the controller must fulfill. For this reason, we transform the properties into algebraic Petri nets (Sec. 3.2) using our generic and data-related transformation algorithm. Finally, we apply the well-defined composition and analysis methods of Petri nets for our composition algorithm (Sec. 3.3). As a result, we obtain the requested controller if one exists.

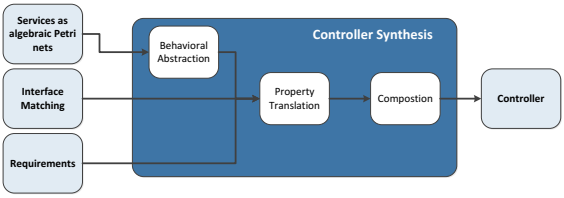


Fig. 2. Basic Idea of the Controller Synthesis Approach

3.1 Behavioral Abstraction

The formal representations of the services include their entire behavior and, thus, unused behavior as well, from which we abstract in this part. For our running example (Figure 1a) the internal storage of the measured glucose concentration, i.e., marking of p_5, is not important for the communication with the pump. To minimize the effort for the composition (Section 3.3), we just extract the properties that have to be fulfilled at least, especially the deadlock-freedom of the services, and define the set of equations w.r.t. the modifications of data. For this we use analysis methods for Petri nets, based e.g. on reachability graphs. This leads to a set of properties given in the specification language.

3.2 Property Translation

To get a uniform representation, we transform the properties into algebraic Petri nets. These properties are given by the result of the behavioral abstraction (e.g. deadlock-freedom), the interface matching (e.g. port p_3 is bound to s_2 (Fig. 1)) and the data dependent requirements (e.g. pump must inject if glucose value exceeds the limit G_{max}), and expressed in the specification language. For the translation, we already implemented a generic, data-related algorithm. Some of our defined rules describing the structural translation can be seen in Figure 3. These rules can be combined to form more complex rules. We have omitted guards restricting data domains and the inscription of arcs for reasons of clarity.

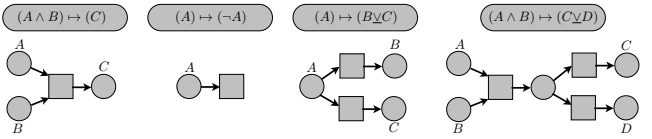


Fig. 3. Transformation rules

3.3 Composition

The basic idea of the composition algorithm (see Figure 4) is to compose the properties that are given as different algebraic Petri nets to an overall Petri net. Then, the controller can easily be extracted from this.

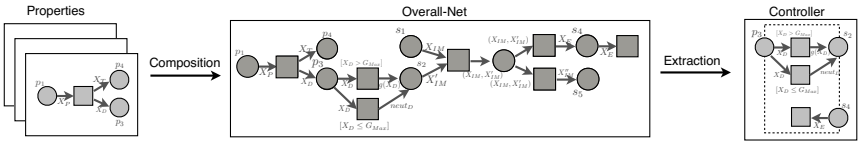


Fig. 4. Basic Idea of the Composition

In a first step, the algorithm chooses the smallest property (with the fewest places) out of the set of properties Pr and composes it with the empty set N. In each iteration, the smallest property $pr_{min} \in Pr$ that has at least one label (identifier of the place) that is present in N is chosen and composed to the net N. If there is exactly one label, the composition can be sequential or branching. Otherwise, if there is more than one match, we have to use the reachability analysis of Petri nets to decide if a composition is possible or not. Concerning this, we have to check if the reachability graph of the property pr_{min} is included in the reachability graph of N. The algorithm terminates if all properties are composed or if a property is found which cannot be composed. In the latter case, there exists no controller fulfilling all properties. In the first case, we can extract the controller from N by combining all elements between the interface places.

It is ongoing work to show that we get a reliable controller that fulfills all given properties, if it exists. The synthesis is done in three steps. Firstly, we derive necessary service properties using behavioral abstraction. The translation to Petri nets, which is already implemented, provides the basis for the composition. Within the composition, we are able to use the analyzing and composition methods Petri nets offer to decide if a property can be composed or not.

4 Related Work

There are several approaches dealing with service or component adaptation.

A controller synthesis process based on finite-state machines focusing on the protocol level is given in [YS97]. They do not consider data-dependent behavior and omit asynchronous communication. In [Bra05] a controller synthesis including the behavior-level that is based on the π calculus is presented, while the proposed approaches in [Aal09] and [Gie12] are based on Petri nets. However, the authors abstract from data and data-dependent behavior. Another approach [SG13] is based on model-checking and planning. It enables the time-related composition of services. They are able to represent clock values from type real, but this is not the kind of data we focus on. To the best of our knowledge, there is no approach that fulfills our requirements. In particular, the formal definition of data and data-dependencies are missing, so that an automated *data dependent* controller synthesis process is not available by now.

5 Conclusion and Future Work

In this work, we have addressed the problem of automatized controller synthesis based on data-dependent requirements. Our approach takes the formal representation as algebraic Petri nets of the involved services, the interface matching, and data-dependent requirements as inputs. The behavior of the services is reduced to the observable behavior, which is transformed to the specification language that we have introduced. Moreover, we have defined a translation from the specification language into Petri nets and have presented an algorithm for composing the abstracted behavior, the interface matching, and the requirements. We have already implemented the translation step and we are confident that with our approach, reliable service controllers can be extracted fully automatically, if they exist. Correctness-by-construction, which we still have to prove, makes our approach applicable in the medical domain.

In future work, we will formalize and implement our composition algorithm, the behavioral abstraction, and the controller extraction. Additionally, we will perform a case study, which is an extension of our running example.

References

[Aal09] van der Aalst, W.M.P., Mooij, A.J., Stahl, C., Wolf, K.: Service Interaction: Patterns, Formalization, and Analysis. In: Bernardo, M., Padovani, L., Zavattaro, G. (eds.) SFM 2009. LNCS, vol. 5569, pp. 42–88. Springer, Heidelberg (2009)

[Bra05] Bracciali, A., et al.: A formal approach to component adaptation. Journal of Systems and Software 74, 45–54 (2005) ISSN 0164–1212

[Gie12] Gierds, C., et al.: Reducing Adapter Synthesis to Controller Synthesis. IEEE T. Services Computing 5(1), 72–85 (2012)

[Rei91] Reisig, W.: Petri nets and algebraic specifications. Theoretical Computer Science 80, 1–34 (1991)

[Rei13] Reisig, W.: Understanding Petri Nets: Modeling Techniques, Analysis Methods, Case Studies, 230 p. Springer (2013) ISBN 978-3-642-33277-7

[SG13] Stöhr, D., Glesner, S.: Planning in Real-Time Domains with Timed CTL Goals via Symbolic Model Checking. In: IEEE TASE 2013 (2013)

[YS97] Yellin, D., Strom, R.: Protocol specifications and component adaptors. In: ACM Trans. Program. Lang. Syst. 19(2), 292–333 (1997)