Loose Programming with PROPHETS

Stefan Naujokat, Anna-Lena Lamprecht, and Bernhard Steffen

Dortmund University of Technology, Chair for Programming Systems, Dortmund, D-44227, Germany
{stefan.naujokat,anna-lena.lamprecht,bernhard.steffen}@cs.tu-dortmund.de

Abstract. Loose programming is an extension to graphical process modeling that is tailored to automatically complete underspecified (loose) models using a combination of data-flow analysis and LTL synthesis. In this tool demonstration we present PROPHETS[1], our current implementation of the loose programming concept. The first part of the demonstration focuses on the preparative domain modeling, where a domain expert annotates the available services with semantic (ontological) information. The second part is then concerned with the actual loose programming, where a process modeler orchestrates the services without having to care about technical details like correct typing, interface compatibility, or platform-specific details. The orchestrated process skeletons are treated as loose service orchestrations that are automatically completed to running applications.

1 Introduction

In service-oriented software development approaches, the specification of (business) processes usually requires detailed knowledge of the available services, their behavior and capabilities. Our concept of loose programming [1] aims at providing easy access to and experimentation with (often unmanageably large) libraries of services. With loose specification, process designers are given the opportunity to sketch their intents roughly, while the backing data-flow analysis and linear-time synthesis handle the concretization automatically.

PROPHETS[1] extends the graphical modeling framework jABC [2] by the loose programming concepts. To enable loose specification and synthesis on a given library of services, semantic information on the services, i.e., a *domain model* is needed. Therefore, there are two user roles defined to work with PROPHETS: While the *domain expert* provides information on services and data types, the *process developer* uses it to semi-automatically create workflows.

In the following, Section 2 explains the domain modeling concepts by means of a simple example domain. Then, Section 3 presents how the domain model is applied for the synthesis of loosely specified processes. Section 4 concludes with a short discussion of related and future work.

[1] PROPHETS is available for download at `http://prophets.cs.tu-dortmund.de`. This site also provides technical documentation and further information on loose programming.

J. de Lara and A. Zisman (Eds.): FASE 2012, LNCS 7212, pp. 94–98, 2012.

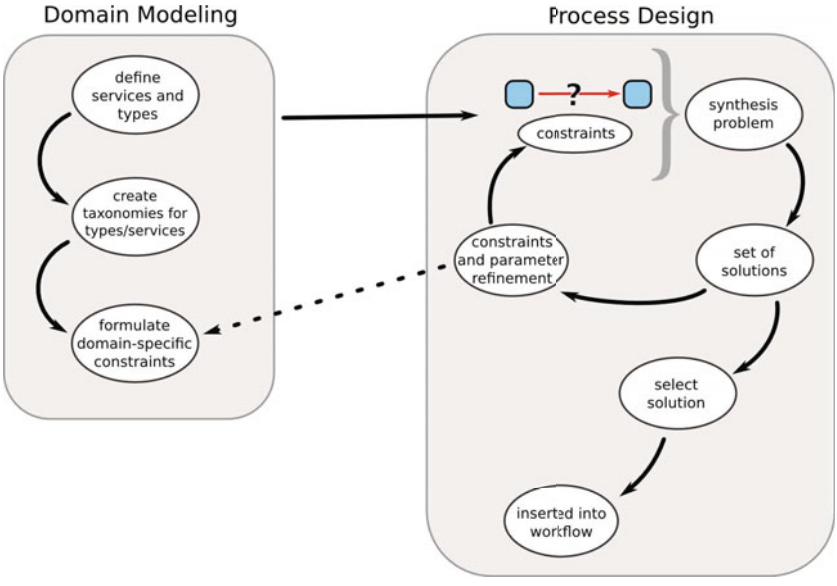

Fig. 1. The workflows of the two user roles involved in loose programming

2 Modeling the Domain

The here presented domain literally corresponds to the 'Hello World' example common to all programming languages. Our domain consists of three services: 'SayHello' sends a message, while 'Understand' receives one, with the language of both being configurable. Naturally, the latter service can only understand the message if it is in its known language. Therefore, the third service, 'Translate', can convert a message from one language into another. Unfortunately, not all language combinations are directly translatable. Only translations from one country's language to its direct neighbors' languages are valid (here limited to Western Europe). In our example scenario, the process developer wants to model a process that sends a message in one language and receives it in another, but he is not familiar with the geography of Europe. The domain expert has this information and provides the semantic annotations to the three services as well as (possibly) further constraints for the composition of services.

Setting up the domain model consists of three major steps (see Fig. 1):

1. At first, the domain expert has to create the service definition file. This mainly requires the identification of symbolic names for types and services, and the behavioral description of the services in terms of their input and output types. Multiple possible type combinations for one service (as it is the case here) simply lead to multiple entries in the service definition file.
2. Secondly, the domain expert may define taxonomies on the types and services. Although this is not strictly required, it may be useful for further structuring of the domain. Here, it might make sense to group all 'Translate'

service instances into one group, all 'SayHello' service instances into another etc.. The types (which are languages in this example) can, for example, be grouped according to language families.

3. Finally, the domain expert may define general domain-specific knowledge by global formulas that are used as constraints for every synthesis. In this example it makes sense to prevent the synthesis from utilizing any of the 'SayHello' services as part of the solution. This becomes necessary, because the synthesis algorithm tries to solve the loose specification by satisfying the input requirements of the target SIB. As the 'SayHello' services have no input requirements, they can be used anywhere to produce every language. The solution problem would be solved formally, but nothing has been actually translated. Such a solution would not be acceptable for the process developer.

3 Process Synthesis

Process design with the jABC (cf. Fig. 2) consists of taking SIBs[2] from the SIB library (A), placing them at the graph canvas (B) and connecting them with directed labeled edges (branches) according to the flow of control. Configuration (i.e. setting parameters) of SIB instances is done using the SIB inspector (C). A model that is defined in this way can then directly be executed and debugged with the integrated interpreter (D). In addition to this kind of complete specification, the PROPHETS plugin enables the process developer to mark one or

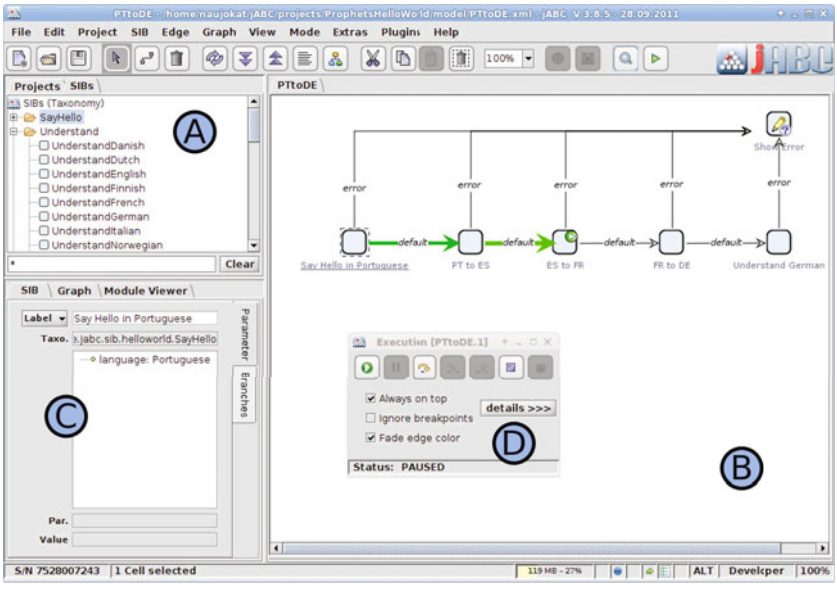

Fig. 2. Overview of jABC GUI elements

more branches as *loosely specified*. PROPHETS' synthesis is then applied to each of the loose branches to replace them by concrete realizations (see Fig. 3)

The plugin determines the start types for the synthesis automatically by performing a data-flow analysis on the process model. The types that are available at the source of the loosely specified branch are used as initial state for the synthesis. As goal types the synthesis uses the input types of the loosely specified branch's target SIB.

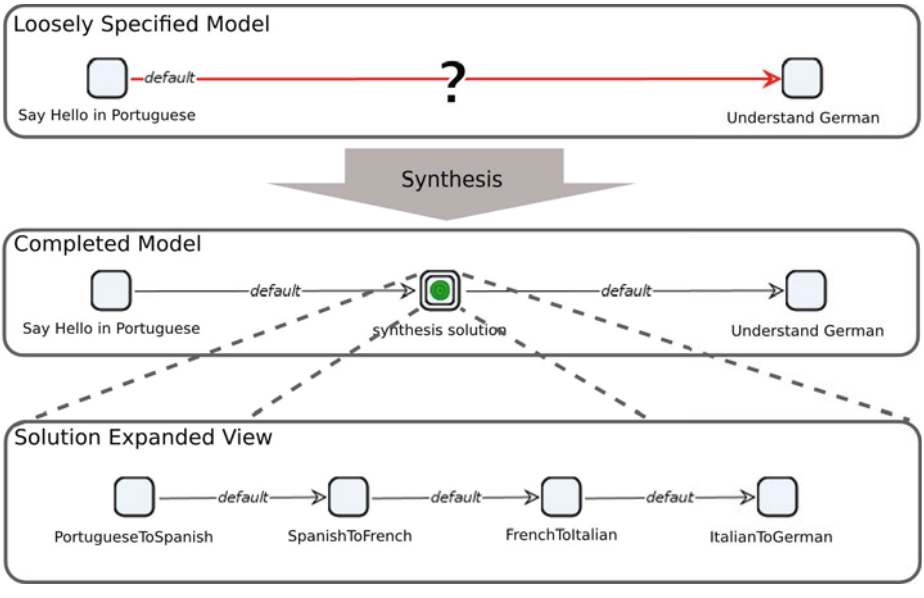

Fig. 3. Loosely specified process and completed model after synthesis

In addition to the inferred start and goal types, the synthesis can be guided by constraints in SLTL [3,4]. However, the expertise of a process designer in the jABC usually covers rather knowledge on business processes than software programming, and likewise we assume that specification of process requirements with formulas in temporal logic is beyond his interests. Therefore, PROPHETS incorporates a concept for template-based constraint specifications. The templates, which can easily be defined and extended by the domain expert, present a description of the constraint in plain natural language to the process designer. The description contains variable parts, which are translated into drop-down boxes for the process designer to assign values.

A process designer can also profit from a specified domain without using the synthesis feature. If a PROPHETS service definition exists, a jABC model can be automatically verified. The plugin then checks if all SIBs have their required

[2] A *SIB* (*Service Independent Building Block*) forms a wrapper for any kind of service that is used in the jABC.

types (input types) available on execution, whatever execution path might lead to this SIB. This is done by a combination of the previously mentioned data-flow analysis and GEAR [5], the model-checking-plugin for the jABC.

4 Conclusion

The here presented example is kept very simple on purpose, so that despite the limited space in this paper, we can elaborate on both of the two basic concepts when working with loose programming. More complex domains, especially in the context of bioinformatics analysis workflows, have shown the applicability of our approach [6,7]. Furthermore, the flexible architecture allows one to change (and even synthesize) the synthesis process itself in order to adapt to special needs of the domain in question [8]. In fact, PROPHETS supports self-application: loosely defined synthesis processes can be completed and executed. Subject of ongoing research are the improvement of the synthesis performance with domain-specific heuristics as well as further concepts for the automatic creation of the domain model, e.g. by learning from service logs or exploiting structural information about the service domain.

References

1. Lamprecht, A.L., Naujokat, S., Margaria, T., Steffen, B.: Synthesis-Based Loose Programming. In: Proceedings of the 7th International Conference on the Quality of Information and Communications Technology (QUATIC). (September 2010)
2. Steffen, B., Margaria, T., Nagel, R., Jörges, S., Kubczak, C.: Model-Driven Development with the jABC. In: Hardware and Software, Verification and Testing. Volume 4383 of LNCS. Springer Berlin / Heidelberg (2007) 92–108
3. Steffen, B., Margaria, T., Freitag, B.: Module Configuration by Minimal Model Construction. Technical report, Universität Passau (1993)
4. Steffen, B., Margaria, T., von der Beeck, M.: Automatic synthesis of linear process models from temporal constraints: An incremental approach. In ACM/SIGPLAN Int. Workshop on Automated Analysis of Software (AAS'97) (1997)
5. Bakera, M., Margaria, T., Renner, C., Steffen, B.: Tool-supported enhancement of diagnosis in model-driven verification. Innovations in Systems and Software Engineering **5** (2009) 211–228
6. Lamprecht, A.L., Naujokat, S., Steffen, B., Margaria, T.: Constraint-Guided Workflow Composition Based on the EDAM Ontology. In: Proc. of the Workshop on Semantic Web Applications and Tools for Life Sciences. Volume 698., Berlin, CEUR Workshop Proceedings (December 2010)
7. Lamprecht, A.L., Naujokat, S., Margaria, T., Steffen, B.: Semantics-based composition of EMBOSS services. Journal of Biomedical Semantics **2**(Suppl 1) (2011)
8. Naujokat, S., Lamprecht, A.L., Steffen, B.: Tailoring Process Synthesis to Domain Characteristics. In: Proceedings of the 16th IEEE International Conference on Engineering of Complex Computer Systems (ICECCS). (2011)