DiálogoP - A Language and a Graphical Tool for Formally Defining GDPR Purposes

Evangelia Vanezi(✉), Georgia M. Kapitsaki, Dimitrios Kouzapas, Anna Philippou, and George A. Papadopoulos

Department of Computer Science, University of Cyprus, Nicosia, Cyprus
{evanez01,gkapi,dimitrios.kouzapas,annap,george}@cs.ucy.ac.cy

Abstract. The notion of *processing purpose*, as set out in the EU General Data Protection Regulation (GDPR), comprises a crucial part of a software system's privacy policy. Processing purposes are meant to characterize the usage of personal data within a system. In this work, we propose a formal type language for defining purposes as the communication exchanges between a system's entities, based on *session types* enhanced with privacy notions. In order to provide software engineers with the means to easily define processing purposes, we encode the formal language syntax to a UML-based domain model and we present DiálogoP, a tool that supports the graphical model definition and subsequently translates it into formal language definitions.

1 Introduction

The European Union applied the General Data Protection Regulation (GDPR) [2] in order to face the challenge of protecting personal data. The GDPR imposes the notion of *purpose* in privacy policies, defining that *"personal data shall be collected for specified, explicit and legitimate purposes and not further processed in a manner that is incompatible with those purposes"*.

In software engineering, validation of privacy policies against software systems is either done by testing techniques, or by human auditing. Alternatively, a formal verification method can guarantee the compliance of a system to its privacy policy, by relying, for instance, on type checking techniques. Proposals for validating privacy requirements in software systems include [9], where automata are used as a formalism for designing and enforcing privacy policies on social networks. Other works aim to extract and model requirements out of regulatory text [12], or to model laws by using primitives, such as roles and norms, and the relationships between them [5]. In [8], the authors present RSLingo4Privacy Studio, a tool for supporting the specification and analysis of privacy policies in software systems, using the RSL-IL4Privacy domain specific language proposed in [1], for the specification of privacy-aware requirements. In turn, [7] employs UML diagrams to define the design of a software's architectural structure, facilitating software engineers in implementing the desired functionality and achieving

F. Dalpiaz et al. (Eds.): RCIS 2020, LNBIP 385, pp. 569–575, 2020.
https://doi.org/10.1007/978-3-030-50316-1_40

privacy by design and by default. Finally, we mention [11], where a two-tiered UML representation of the GDPR is presented, aiming towards a cost-effective method that will help the business sector systems achieve GDPR compliance.

Our work shares similar aims to the above-mentioned works and complements them by adopting a formal approach towards rigorously verifying that software systems comply to their privacy requirements. We propose thus, a language for formal specification of processing purposes in software systems based on session types [3,4,10] and the Privacy Calculus [6]. Furthermore, we propose a graphical modelling language for privacy policies and a methodology for transforming graphically defined privacy policies into formal type language definitions. We present DiálogoP, a tool for defining purposes as diagrams through a domain model and a toolbox, without requiring knowledge of the formal language. We utilise Domain Specific Languages (DSL) covering privacy purpose for any software system. Each diagram is translated into a formal specification using the introduced transformation process.

Use Case: A Healthcare System. As a use case, we consider an automated medical system for hospitals, for which the responsible authority, i.e. the Ministry of Health, defines the functionality and the privacy policy, and each hospital is responsible to implement its own system. A part of the system handles appointments as follows: the system knows the full name, the date of birth and the hospital id number of each patient that has an appointment. At the entrance, a patient scans her hospital identity card, which stores the above data. The system compares the data from the card and the booked appointment and, if they match, it allows the patient to proceed with the visit. The data cannot be used in any other way. The system retrieves from the database the patient's record, i.e. all visits and diagnoses, and forwards that information to the doctor without reading the data. The doctor reads the data, examines the patient and adds information for symptoms and latest diagnosis to the record.

2 A Formal Language for Defining Purposes

We build on *multiparty session types* [4], to formally define processing purposes, enriching them with the notion of personal data stores inspired by the Privacy Calculus [6]. Session types are based on message exchanges between the entities of the system. In our language, in each purpose definition a number of distinct sessions can be set up, able to run in parallel and be interleaved, the entities participating in each session and their interactions. Entities belonging to the same session can exchange messages including personal data either by directly sending them to each other, or based on conditions' evaluation. Such interactions should occur in a specific defined sequence within each session. In order to integrate the notion of *personal data* we exploit the structure of *personal data stores*, first defined in the Privacy Calculus [6], including a unique id and a set of data. Each store has an owner entity. Other entities of a system may obtain references to stores, and use them to access the respective personal data.

Figure 1 presents the syntax. Metavariable, g, denotes ground types such as integer and boolean. Exchange types, U, include ground types, g, annotations, α, and private data types, $\mathsf{pd}[\mathsf{U}]^{\alpha}$. Annotations, α, are identifiers used to annotate private data. Private data type, $\mathsf{pd}[\mathsf{U}]^{\alpha}$, is used to type a reference on private data storage that contains private data of type U. We also assume labels for true, $\mathtt{tt}$, and false, $\mathtt{ff}$, and a set of participants $\mathsf{p}, \mathsf{q}, \ldots \in \mathcal{P}$. A global type, G, defines a multiparty session type able to describe privacy purpose. Global type end is the termination type. Type $\mathsf{p} \to \mathsf{q} : U.G$ describe that participant p sends to participant q an exchange value U and then proceeds with global type G. Global types include $\alpha \to \mathsf{p} : U.G$ and $\mathsf{p} \to \alpha : U.G$ describing the reading from or the writing to a store annotated by α personal data of type U, respectively. Finally, conditional branching type, $\mathsf{p} \to \mathsf{q} : (U \ op \ U')\{\mathtt{tt} : G_1, \mathtt{ff} : G_2\}$, describes the choice that takes place on participant p based on the type of the expression $U \ op \ U'$ and, moreover, participant q is informed and proceeds accordingly.

$$
\begin{array}{rrcll}
\text{Ground Types} & g & ::= & \mathsf{int} \mid \mathsf{bool} \mid \ldots & \\
\text{Exchange Types} & U & ::= & g \mid \mathsf{pd}[\mathsf{U}]^{\alpha} \mid \alpha & \\
\text{Global Types} & G & ::= & \mathsf{end} & \text{Termination} \\
 & & \mid & \mathsf{p} \to \mathsf{q} : U.G & \text{Value Exchange} \\
 & & \mid & \alpha \to \mathsf{p} : U.G & \text{Personal Data Read} \\
 & & \mid & \mathsf{p} \to \alpha : U.G & \text{Personal Data Storage} \\
 & & \mid & \mathsf{p} \to \mathsf{q} : (U \ op \ U')\{\mathtt{tt} : G_1, \mathtt{ff} : G_2\} & \text{Conditional Branching} \\
\text{Operators} & op & ::= & = \mid \geq \mid \leq &
\end{array}
$$

Fig. 1. Multiparty session types for privacy purpose

3 DiálogoP - A Graphical Tool for Defining Purposes

Aiming to simplify the procedure of specifying purposes we elevate the definition to diagrams that are subsequently automatically translated to session types through DiálogoP. The tool was developed using Eclipse Sirius, through Obeo Designer[1] and EMF (Eclipse Modeling Framework) modelling.

Meta-Model. The first step was to create an EMF meta-model, shown in Fig. 2[2], the source code of which is then generated and imported into new modelling projects, allowing to define purposes as diagrams and dictating the structure that such diagrams should have. *PDataStoreReference*, *PersonalData*, and *Operator*, were defined as types. *Purpose* is the main kind of entity, composed of a set of *Sessions*, composed in turn out of one or more *Ends* and sets of *Communicating Entities*, *Messages*, and *Conditions*. Communicating entities may be

[1] https://www.obeodesigner.com/.

[2] Full-size figures can be found at http://www.cs.ucy.ac.cy/seit/dialogop/.

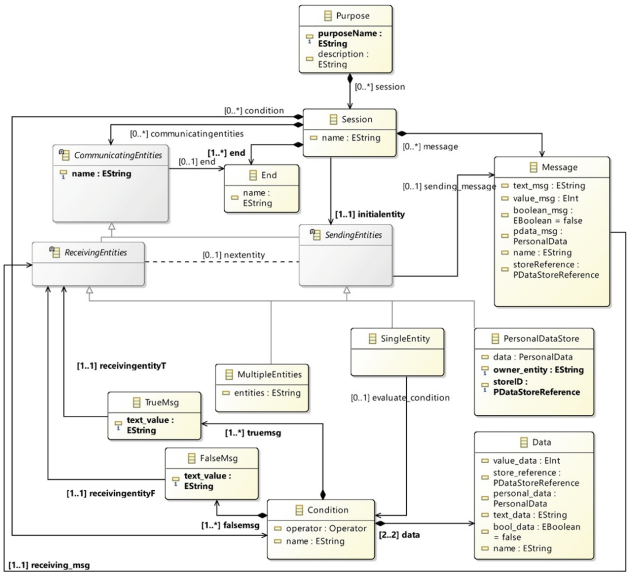

Fig. 2. DiálogoP metamodel

either *Receiving Entities* or *Sending Entities*. Each session needs to have precisely one initial entity, that will be a Sending Entity. Sending Entities may be *Single Entities* or *Personal Data Stores*. Receiving Entities may be *Single Entities*, *Personal Data Stores* or blocks of *Multiple Entities*. Conditions are composed of two sets of *Data* that will be compared based on an operator, as well as exactly one message for evaluating the condition as *True* and exactly one message for *False*. Sending Entities can be connected to at most one Condition or at most one Message, Messages can be connected to exactly one Receiving Entity, Receiving Entities can be connected directly to a Sending Entity, and all entities can be connected to an End.

Graphical Designer. New modelling projects can be created by the user in order to design models, either via the tree view or via the palette. The tree form allows the addition of children and siblings for each model entity. The tree structure root is a *Purpose* entity that can only have *Session* entities as children. In each *Session*, new children can be added including *Single Entities*, *Multiple Entities*, *Personal Data Stores*, *Messages*, *Conditions* and *Ends*, as shown in Fig. 3. All other entities cannot have any children added, except a *Condition* that can have *Data*, *True Message* and *False Message* entities as children. Each added entity has a set of properties that can be set up by the user, including its attributes values and relations with other entities. Different class instances are represented by different icons[3]. Figure 4 shows the palette that includes the

[3] Iconset source: https://www.iconfinder.com/iconsets/message-and-communication-sets, under Creative Commons License Attribution 3.0 Unported (CC BY 3.0).

Nodes section with all entities, and the *Edges* section, with all relations. The attributes are set. To add a node, one should first click on the respective icon from the palette, and then click on an already existing entity which is higher in the hierarchy and already placed on the canvas. Figure 5 displays the purpose diagram for the healthcare use case.

Translation into Formal Session Types. The purpose model is extracted in an XML (Extensible Markup Language) format and is fed into the custom made parser, implemented in Java, to be translated into session types definition. Two main classes, *PurposeEdge* and *PurposeEntity*, are used. The parser recognises all nodes and edges (with *xmi:type* equals to *"diagram:DEdge"*) by their *ownedDiagramElements* tag, and subsequently all inner tags *semanticElements*, from the XML document. All recognised objects with their data, are stored in lists. The lists are then parsed, recognising sessions, and the sequence of actions beginning with the initial entity and following all edges towards the end of the session by calling function *findPath*. Each situation is then handled in a different manner, e.g. in the case of conditional statements the function is called recursively for the two possible paths. Figure 6 shows the translation of the running use case to session types, as given automatically by the tool.

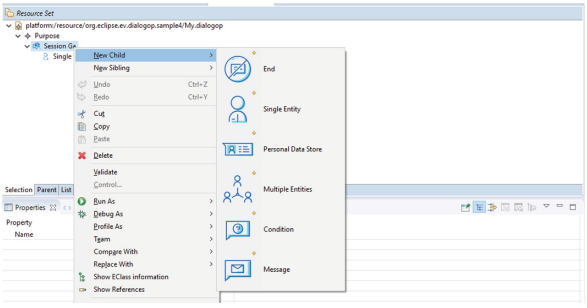

Fig. 3. New entities in a session

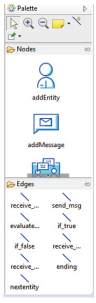

Fig. 4. Palette

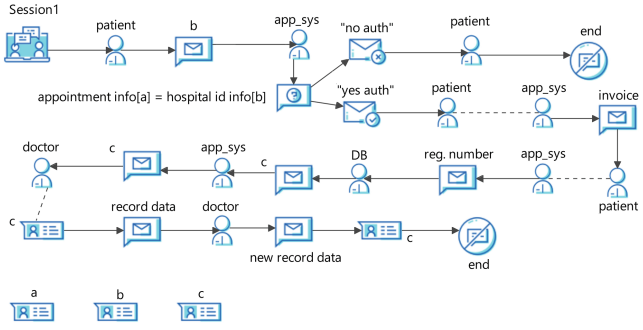

Fig. 5. The graphical purpose

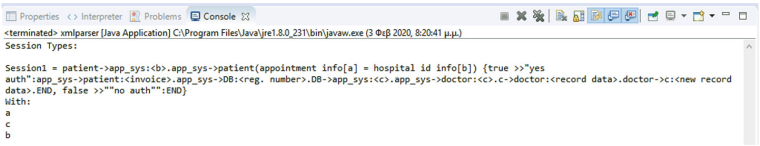

Fig. 6. The translation output

4 Conclusions

We have presented our work towards an automated process for compliance of software systems to privacy policies, focusing on the GDPR notion of purpose. We have defined a formal language for purpose using session types, and we have created a graphical modelling environment that allows the definition of purposes for systems using the introduced meta-model, and the automatic transformation to the formal language. As future work, we intend to use the output to perform compliance analysis, and to extend our approach to other areas of GDPR.

References

1. Caramujo, J., da Silva, A.R., Monfared, S., Ribeiro, A., Calado, P., Breaux, T.: RSL-IL4Privacy: a domain-specific language for the rigorous specification of privacy policies. Requir. Eng. **24**(1), 1–26 (2019). https://doi.org/10.1007/s00766-018-0305-2
2. European Parliament and Council of the European Union: General data protection regulation (2015). Official Journal of the European Union
3. Honda, K., Vasconcelos, V.T., Kubo, M.: Language primitives and type discipline for structured communication-based programming. In: Hankin, C. (ed.) ESOP 1998. LNCS, vol. 1381, pp. 122–138. Springer, Heidelberg (1998). https://doi.org/10.1007/BFb0053567
4. Honda, K., Yoshida, N., Carbone, M.: Multiparty asynchronous session types. In: ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, pp. 273–284 (2008)
5. Ingolfo, S., Siena, A., Mylopoulos, J.: Nómos 3: reasoning about regulatory compliance of requirements. In: IEEE Requirements Engineering Conference, pp. 313–314. IEEE (2014)
6. Kouzapas, D., Philippou, A.: Privacy by typing in the π-calculus. Logical Methods Comput. Sci. **13**(4), 1–42 (2017)
7. Mougiakou, E., Virvou, M.: Based on GDPR privacy in UML: case of e-learning program. In: International Conference on Information, Intelligence, Systems & Applications, pp. 1–8. IEEE (2017)
8. Ribeiro, A., da Silva, A.R.: RSLingo4Privacy studio-a tool to improve the specification and analysis of privacy policies. In: ICEIS, vol. 2, pp. 52–63 (2017)
9. Pardo, R., Colombo, C., Pace, G.J., Schneider, G.: An automata-based approach to evolving privacy policies for social networks. In: Falcone, Y., Sánchez, C. (eds.) RV 2016. LNCS, vol. 10012, pp. 285–301. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-46982-9_18

10. Takeuchi, K., Honda, K., Kubo, M.: An interaction-based language and its typing system. In: Halatsis, C., Maritsas, D., Philokyprou, G., Theodoridis, S. (eds.) PARLE 1994. LNCS, vol. 817, pp. 398–413. Springer, Heidelberg (1994). https://doi.org/10.1007/3-540-58184-7_118
11. Torre, D., Soltana, G., Sabetzadeh, M., Briand, L.C., Auffinger, Y., Goes, P.: Using models to enable compliance checking against the GDPR: an experience report. In: 2019 ACM/IEEE International Conference on Model Driven Engineering Languages and Systems, pp. 1–11. IEEE (2019)
12. Zeni, N., Kiyavitskaya, N., Mich, L., Cordy, J.R., Mylopoulos, J.: GaiusT: supporting the extraction of rights and obligations for regulatory compliance. Requir. Eng. **20**(1), 1–22 (2015). https://doi.org/10.1007/s00766-013-0181-8