Views on UML Interactions as Spreadsheet Queries

Martin Gogolla[1] and Antonio Vallecillo[2(✉)]

[1] University of Bremen, Bremen, Germany
gogolla@informatik.uni-bremen.de
[2] University of Malaga, Malaga, Spain
av@lcc.uma.es

Abstract. This paper explores the use of table-based representation for artifacts occurring in model-driven development as opposed to graph-based representation. As an example for table-based representation of models, we explain how views on object interaction that are traditionally represented as UML sequence or communication diagrams can be realized by spreadsheet queries.

1 Introduction

Models in Model-Based Engineering (MBE) are graph structures and as such they are typically expressed using graph-based representations; e.g., class and object diagrams are represented as nodes and edges in a graph. This sort of representation permits a natural, faithful and comprehensive description of the models and of their views, and its operation by theories and tools. However, graph-based descriptions of large systems can become cumbersome and difficult to understand, query and analyze by human users due to their size and complexity [4,8].

The modelling world has few connections to the spreadsheet world, despite the fact that spreadsheets provide a widely used description technique. Spreadsheets are able to represent complex information in a clearly structured way in tabular form. Together with relational databases, now they probably constitute the most used way of presenting and manipulating information.

This paper explores the use of table-based representation for artifacts occurring in MBE, as opposed to their traditional graph-based representation. As an example, we show how object interaction diagrams that are traditionally represented as UML sequence or communication diagrams [9,12], can be naturally expressed in tabular form; and how views on such models can be easily realized by spreadsheet queries. In principle, the table-based representation we show here can be used for all other UML-like models as well, hence facilitating the creation of views and their representation in a more appropriate manner for many purposes.

2 Preliminaries

The context of our work is the UML tool USE (Uml-based Specification Environment) [5] that allows the developer to describe (a) system structure with a

P. Milazzo et al. (Eds.): STAF 2016, LNCS 9946, pp. 394–400, 2016.
DOI: 10.1007/978-3-319-50230-4_30

class diagram incorporating OCL (Object Constraint Language) invariants and (b) system behavior with OCL contracts, UML state machines and operation implementations in the language SOIL (Simple Ocl-like Imperative Language)

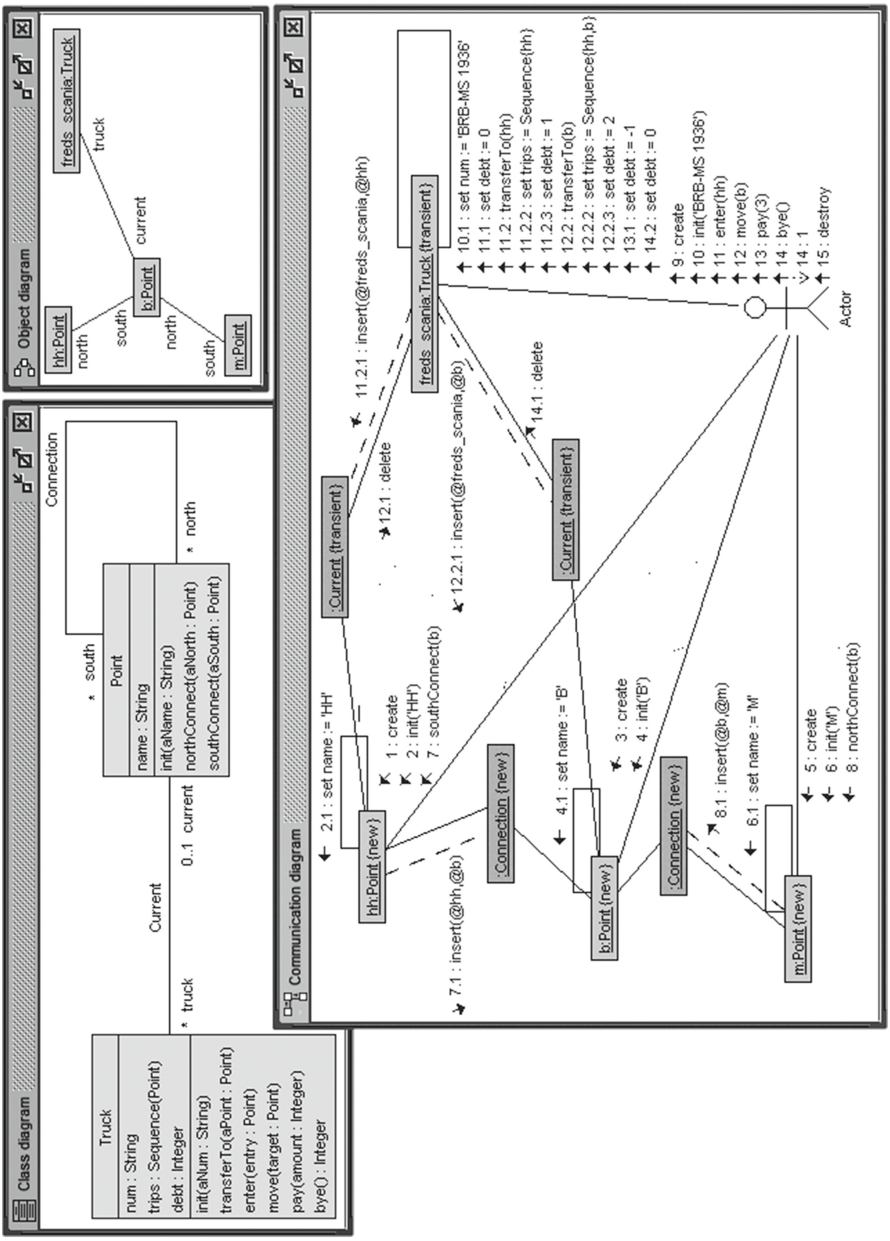

Fig. 1. Example interaction as UML communication diagram.

StrucNum	SpecificInteractionNotation	InstanciatedGenericInteractionNotation	Kind	FlatNum	Depth
...	...	...	...	...	...
[8]	m.northConnect(b)	[self=m].northConnect([aNorth=b])	call	15	1
[8.1]	insert(b,m) into Connection	insert(aNorth,self) into Connection	insert	16	2
[9]	create freds_scania:Truck	create [self=freds_scania]:Truck	create	17	1
[9.1]	freds_scania.num:=''	self.num:=''	assign	18	2
[9.2]	freds_scania.trips:=Sequence{}	self.trips:=Sequence{}	assign	19	2
[9.3]	freds_scania.debt:=0	self.debt:=0	assign	20	2
[10]	freds_scania.init('BRB-MS 1936')	[self=freds_scania].init([aNum='BRB-MS 1936'])	call	21	1
[10.1]	freds_scania.num:='BRB-MS 1936'	self.num:=aNum	assign	22	2
[11]	freds_scania.enter(hh)	[self=freds_scania].enter([entry=hh])	call	23	1
[11.1]	freds_scania.debt:=0	self.debt:=0	assign	24	2
[11.2]	freds_scania.transferTo(hh)	[self=freds_scania].transferTo([aPoint=hh])	call	25	2
[11.2.1]	insert (freds_scania,hh) into Current	insert (self,aPoint) into Current	insert	26	3
[11.2.2]	freds_scania.trips:=Sequence{hh}	self.trips:=self.trips->including(aPoint)	assign	27	3
[11.2.3]	freds_scania.debt:=1	self.debt:=self.debt+1	assign	28	3
[12]	freds_scania.move(b)	[self=freds_scania].move([target=b])	call	29	1
[12.1]	delete (freds_scania,hh) from Current	delete (self,self.current) from Current	delete	30	2
[12.2]	freds_scania.transferTo(b)	[self=freds_scania].transferTo([aPoint=b])	call	31	2
[12.2.1]	insert (freds_scania,b) into Current	insert (self,aPoint) into Current	insert	32	3
[12.2.2]	freds_scania.trips:=Sequence{hh,b}	self.trips:=self.trips->including(aPoint)	assign	33	3
[12.2.3]	freds_scania.debt:=2	self.debt:=self.debt+1	assign	34	3
[13]	freds_scania.pay(3)	[self=freds_scania].pay([amount=3])	call	35	1
[13.1]	freds_scania.debt:=-1	self.debt:=self.debt-amount	assign	36	2
[14]	freds_scania.bye()	[self=freds_scania].bye()	call	37	1
[14.1]	delete (freds_scania,b) from Current	delete (self,self.current) from Current	delete	38	2
[14.2]	self.debt:=0	self.debt:=0	assign	39	2
[14.3]	result:=1	result:=self.debt.abs()	assign	40	2
[15]	destroy freds_scania	destroy [self=freds_scania]	destroy	41	1

Fig. 2. Example interactions as spreadsheet.

that combines OCL expressions with control flow. Models can be executed by means of scenarios that can be documented with UML sequence and communication diagrams. These two diagrams are the UML form of showing object interactions.

Our running example is a simple variant of a toll collecting system for trucks moving on a road layout described in more detail in [5]. Figure 1 shows (a) the class diagram, (b) a communication diagram for a scenario with 15 major messages involving three towns (Hamburg `hh`, Berlin `b`, Munich `m`) connected by roads and one travelling truck as well as (c) an object diagram reflecting the system state after one particular message (number 12). Messages possess submessages indicated by structured numbers, e.g., 12.1, 12.2, 12.2.1, 12.2.2, 12.2.3.

The challenge for us was: how can we achieve a comfortable view mechanism on complex interactions consisting of message exchanges among objects? Different views may want to filter the interactions along different aspects like message kind (e.g., creating or destroying objects or inserting or deleting links), message order (through the numbering system) or involved object types.

3 Representing Interactions in Spreadsheets

We now want to explain how the graphically displayed message exchanges from the communication diagram in Fig. 1 can be given in table-based form. Figure 2 shows (part of) the object interactions in a spreadsheet. The columns describe

StrucNum	SpecificInteractionNotation	InstanciatedGenericInteractionNotation	Kind	FlatNum	Depth
			=create	<=16	
			=call	>=17	=1

StrucNum	SpecificInteractionNotation	InstanciatedGenericInteractionNotation	Kind	FlatNum	Depth
[1]	create hh:Point	create [self=hh]:Point	create	1	1
[3]	create b:Point	create [self=b]:Point	create	5	1
[5]	create m:Point	create [self=m]:Point	create	9	1
[10]	freds_scania.init('BRB-MS 1936')	[self=freds_scania].init([aNum='BRB-MS 1936'])	call	21	1
[11]	freds_scania.enter(hh)	[self=freds_scania].enter([entry=hh])	call	23	1
[12]	freds_scania.move(b)	[self=freds_scania].move([target=b])	call	29	1
[13]	freds_scania.pay(3)	[self=freds_scania].pay([amount=3])	call	35	1
[14]	freds_scania.bye()	[self=freds_scania].bye()	call	37	1

Fig. 3. Example query on interaction as spreadsheet query.

(1) the structured message number, (2) the message in instantiated form, (3) the message in generic form, (4) the message kind, (5) a flat message number, and (6) the message call depth. The distinction between columns (2) and (3) can be explained best by message 12.2.3 having flat number 34: the generic form shows the SOIL statement from the "move" implementation "self.debt := self.debt+1" whereas the instantiated form shows concrete, substituted values "self.debt := 2". The table-based representation contains even more details than Fig. 1, as attribute initializations and operation result values are shown. Every present information on messages is stated in the table. Apart from the depicted attributes, the table could contain further information, such as the sender and receiver of each message.

4 Views on Interactions as Spreadsheet Queries

Let us now turn to the question how complex interactions as present in Fig. 2 can be viewed and filtered for particular purposes. The developer will not be interested in all messages with all details, but will like to see only messages relevant in a particular context. For example, the developer might want to achieve a rough overview on the creation of the road layout and the operation calls in the later part of the scenario.

The spreadsheet query in the upper, grey-shaded part of Fig. 3 serves this purpose. The result of the query is stated in the lower part.

Thus, by expressing spreadsheet queries that can be formulated in the spreadsheet itself and that can use present constants and simple, but effective operators, the developer is able to interactively formulate requirements on the desired view and to satisfy the current information needs.

In order to compare this approach with similar queries on the UML metamodel, Fig. 4 shows the fragment on the UML metamodel that deals with Interactions. As we can see, every message requires the specification of several instances of class MessageEnd and of separate objects in case of arguments. For example, Fig. 5 shows how only one of the 41 messages in the Interaction is represented as an instance of the UML metamodel, namely the "enter(hh)" message. This illustrates the complexity required for navigating through these kinds of instances for expressing queries that can be easily specified in a spreadsheet.

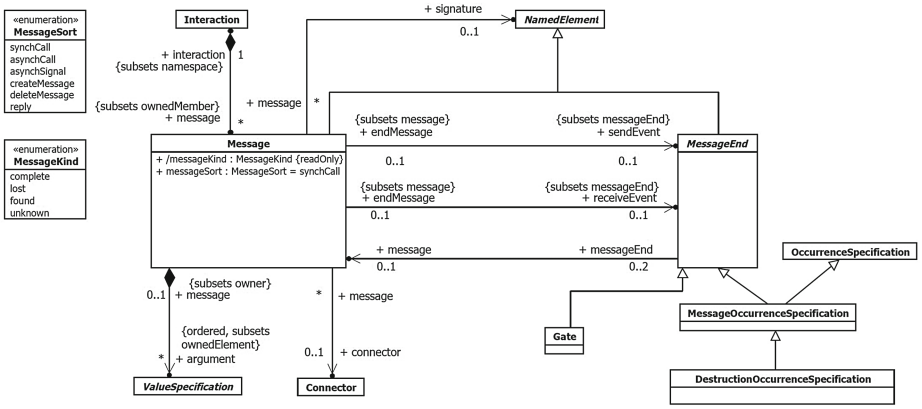

Fig. 4. UML interaction metamodel (from [9]).

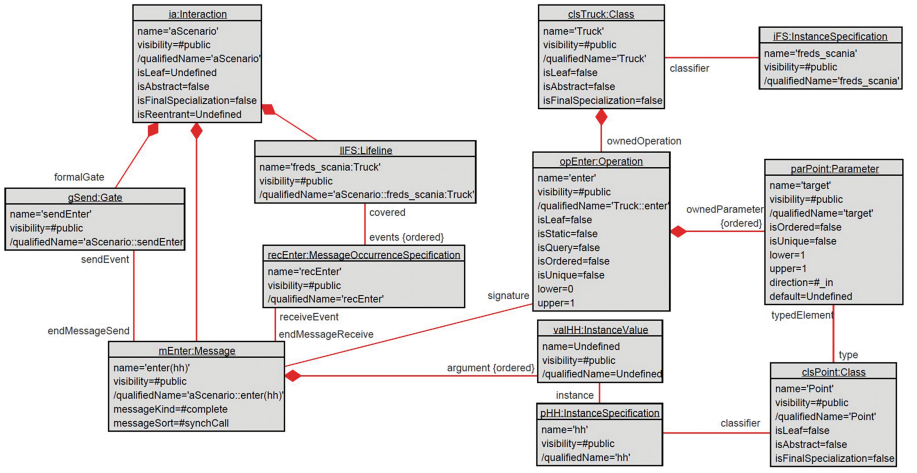

Fig. 5. Send message event as an instance of the UML metamodel (from [5]).

5 Conclusion

So far, most efforts for connecting the spreadsheet and Model-Based Engineering worlds have focused on representing spreadsheets as models [10] in order to make use of MBE concepts, mechanisms and tools to improve the specification, development, debugging, maintenance, and evolution of spreadsheets—see, e.g., [1,2] and many of the SEMS workshop series papers [6,7].

In this paper we have discussed an example that aims at showing the benefits of using a tabular representation of a model as opposed to its graph-based representation. In fact, models of non-trivial systems can become very complicated and their representation as spreadsheets can be quite simple and straightforward.

In this manner queries can be formulated in a natural and user-oriented way, too. Furthermore, we can use all the powerful operations provided by spreadsheets to implement some operations on the views, or even make use of advanced features for querying spreadsheets [3]. We could also keep them in sync, so that certain changes in the views (the spreadsheets) are propagated to the models, updating them accordingly.

Of course, each notation is more apt for particular goals, and probably the best approach consists of combining table- and graph-based representations, using one or the other depending on the kind of user querying the model [11], and on the operation we want to perform on the model. In the USE context we already count on graph- and table-based representations for object diagrams and interactions. Counting on spreadsheet-based representation of other aspects, such as complete object diagram evolution, would be desirable too.

Acknowledgements. This work is funded by Spanish Research Project TIN2014-52034-R and by Universidad de Málaga (Campus de Excelencia Internacional Andalucía Tech).

References

1. Bals, J., Christ, F., Engels, G., Erwig, M.: ClassSheets - model-based, object-oriented design of spreadsheet applications. J. Object Technol. **6**(9), 383–398 (2007)
2. Cunha, J., Fernandes, J., Mendes, J., Pereira, R., Saraiva, J.: MDSheet: model-driven spreadsheets. In: Proceedings of SEMS 2014, vol. 1209, pp. 31–33. CEUR (2014)
3. Cunha, J., Fernandes, J.P., Mendes, J., Pereira, R., Saraiva, J.: Embedding model-driven spreadsheet queries in spreadsheet systems. In: Proceedings of VL/HCC 2014, pp. 151–154. IEEE Computer Society (2014)
4. Gelman, A.: Why tables are really much better than graphs. J. Comput. Graph. Stat. **20**(1), 3–7 (2011)
5. Gogolla, M., Hamann, L., Hilken, F., Sedlmeier, M.: Modeling Behavior with interaction diagrams in a UML and OCL tool. In: Roubtsova, E., McNeile, A., Kindler, E., Gerth, C. (eds.) Behavior Modeling – Foundations and Applications. LNCS, vol. 6368, pp. 31–58. Springer, Heidelberg (2015). doi:10.1007/978-3-319-21912-7_2
6. Hermans, F., Paige, R.F., Sestof, P. (eds.): Proceedings of 1st International Workshop Software Engineering Methods in Spreadsheets (SEMS 2014). CEUR Proceedings, vol. 1209 (2014). http://ceur-ws.org/Vol-1209/
7. Hermans, F., Paige, R.F., Sestof, P. (eds.): Proceedings of 2nd International Workshop Software Engineering Methods in Spreadsheets (SEMS 2015). CEUR Proceedings, vol. 1355 (2015). http://ceur-ws.org/Vol-1355/
8. Kosslyn, S.M.: Understanding charts and graphs. Appl. Cogn. Psychol. **3**(3), 185–225 (2006)
9. Object Management Group: Unified Modeling Language (UML) Specification, version 2.5. OMG Document formal, 01 March 2015
10. Paige, R.F., Kolovos, D., Matragkas, N.: Spreadsheets are models too. In: Proceedings of SEMS 2014. CEUR, vol. 1209, pp. 9–10 (2014)

11. Sobreira, P., Tchounikine, P.: CSCL scripts: interoperating table and graph representations. In: Proceedings of CSCL 2013, pp. 165–168 (2013)
12. Wendland, M.-F., Schneider, M., Haugen, Ø.: Evolution of the UML interactions metamodel. In: Moreira, A., Schätz, B., Gray, J., Vallecillo, A., Clarke, P. (eds.) MODELS 2013. LNCS, vol. 8107, pp. 405–421. Springer, Heidelberg (2013). doi:10.1007/978-3-642-41533-3_25