Scenario-based Specification of Car-to-X systems[1]

Joel Greenyer[2], Daniel Gritzner[2], Nils Glade[2], Timo Gutjahr[2], Florian König[2]

Abstract: Cyber-physical systems are found in many areas, such as manufacturing, transportation, or smart cities. They consist of many components which cooperate to provide the desired functionality. This need for cooperation causes complex interactions between components, which makes developing a cyber-physical system difficult. To support engineers developing such systems we have created a design- and specification method which makes it easier to develop reactive systems, especially cyber-physical systems. Object-oriented modeling combined with a scenario-based behavior specification provide an intuitive, yet precise way to formally specify reactive systems. We implemented a tool which uses this kind of formal specification to allow engineers to simulate, visualize, check, and execute their specifications. This helps with reducing development costs as inconsistencies, and thus defects in the system, are spotted early, when it is still relatively cheap to make changes to the system. We evaluated this technique by building a Car-to-X system, in which Raspberry Pi-based robots emulated autonomous cars.

Keywords: scenario-based specification, cyber-physical system, Car-to-X, simulation, realizability

1 Introduction

In many areas, for example, transportation, industry, and healthcare, we find cyber-physical systems (CPSs). CPSs consist of many cooperating components and are used to control complex processes in order to provide rich functionality. Each such process is typically controlled by multiple components and each component is involved in controlling multiple processes at the same time. This causes complex interactions between the components in a CPS. As users demand increasingly rich functionality of a CPS, these inter-component interactions become increasingly difficult to cope with when developing a cyber-physical system.

CPSs can also be found in smart cities, e.g., in the form of autonomous cars or cars with advanced driver assistance systems. These cars, which can be regarded as components of a CPS, need to cooperate to bring their passengers safely to their destinations. Specifying the correct behavior of each car in all situations is a difficult problem, especially when considering the dynamically changing relationships between cars at runtime.

In order to help engineers cope with the complexity of developing a cyber-physical system, we developed a formal method for specifying the behavior of a system. In our method we

[1] This work is funded by grant no. 1258 of the German-Israeli Foundation for Scientific Research and Development (GIF)

[2] Leibniz Universität Hannover, Fachgebiet Software Engineering, Welfengarten 1, D-30167 Hannover, Germany, {greenyer | daniel.gritzner}@inf.uni-hannover.de

define the system's behavior through a scenario-based description of the interactions of its components. Scenarios specify how a set of system components may, must, or must not react to external events.

In this paper we illustrate how to use ScenarioTools, an implementation of our method, to build a system based on Car-to-X communication. In this system autonomous cars, emulated by Raspberry Pi-based robots, needed to cooperate to safely pass an obstacle, i.e., to avoid a collision with each other. Our setup for testing is shown in Fig. 1 and was explained before [Gr15]. In this paper we focus on the specification methodology.

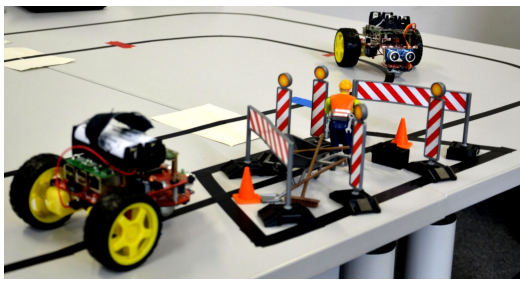

Figure 1: Raspberry Pi-based robots emulating autonomous cars

The paper is structured as follows. Section 2 explains scenario-based modeling with the help of an example. Our modeling process itself is discussed in section 3. Sections 4 and 5 are dedicated to related work and what how we plan to extend our design method and tools in the future. Finally, we provide a conclusion in section 6.

2 Scenario-based Modeling

Engineers typically think of scenarios, that include certain requirements, when trying to formalize requirements. This makes the behavior modeling with scenarios, which we describe in this section, an intuitive and natural process [Al14, GMMS12].

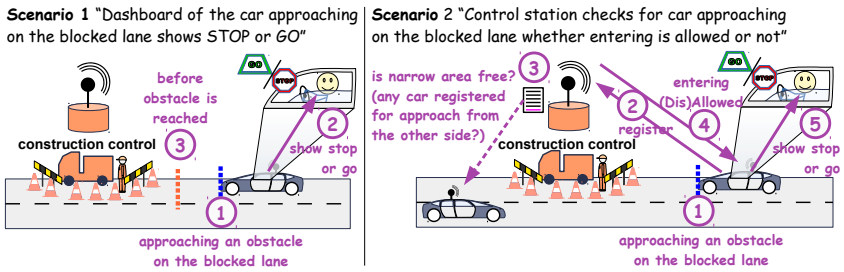

Figure 2: Sketches of scenarios

A scenario describes a short interaction between objects. The interplay of multiple scenarios, which may be active concurrently, then defines the system's behavior. This means a formal specification is the same as a set of formal scenarios. An example is shown in Fig. 2. The example describes an advanced driver assistance system. The first scenario describes what must happen when a car approaches an obstacle (1). The car's dashboard

must notify the driver whether he is allowed to pass the obstacle or whether he must wait (2). This notification must arrive before the driver actually reaches the obstacle (3). This is a scenario which may be discussed with stakeholders, who do not have a technical background, but are still interested in formulating requirements on the final system. The second scenario then refines the same situation from the point of view of an engineer. When a car approaches an obstacle (1), it registers itself at the obstacle's controller (2), which in turn decides (3) whether to allow the car to pass or not (4). The car can use the obstacle controller's reply to update the dashboard (5).

```
1  specification scenario DashboardOfCarApproachingShowingStopOrGo
2    with dynamic bindings [ bind dashboard to car.dashboard ] {
3        message env -> car.approachingObstacle()
4        alternative { message strict car -> dashboard.showGo() }
5          or        { message strict car -> dashboard.showStop() }
6        message env -> car.obstacleReached()
7    }
```

Listing 1: Example of a scenario written in the Scenario Design Language

Listing 1 shows a formal version of the first scenario in Fig. 2. The code example is written in our Scenario Design Language, which is based on the concept of objects exchanging messages. Keywords are used to define the desired behavior, e.g., strict in lines 4 and 5 means that either of those messages must occur before the message in line 6. Reaching the obstacle before updating the dashboard would create a safety violation of the system.

3 Specification and Analysis Methodology

This section discusses how to use ScenarioTools[3] to build a cyber-physical system. An overview of the process is shown in Fig. 3. ScenarioTools is a collection of Eclipse plug-ins based on the Eclipse Modeling Framework (EMF) and Xtext.

Building a system with ScenarioTools consists of three steps which are applied repeatedly in iterations until the desired state of the specification is achieved. The steps are specification, analysis and execution.

In the **specification step** an engineer adds or changes scenarios to specify the desired system behavior. EMF is used to model the objects whose behavior is specified and which represent the components of the system. We also integrated Henshin into ScenarioTools to allow engineers to define side-effects of messages to deal with structural dynamism of the object model at runtime.

Next follows the **analysis step** in which the engineer can either proceed with a manual simulation or automatic realizability checking of the specification. During a simulation the messages to be exchanged are manually chosen from a list as shown in the bottom-left of Fig. 4. This list is generated based on the current state of all active scenarios. It only includes messages which the specification currently allows. Simulations are based

[3] http://scenariotools.org/

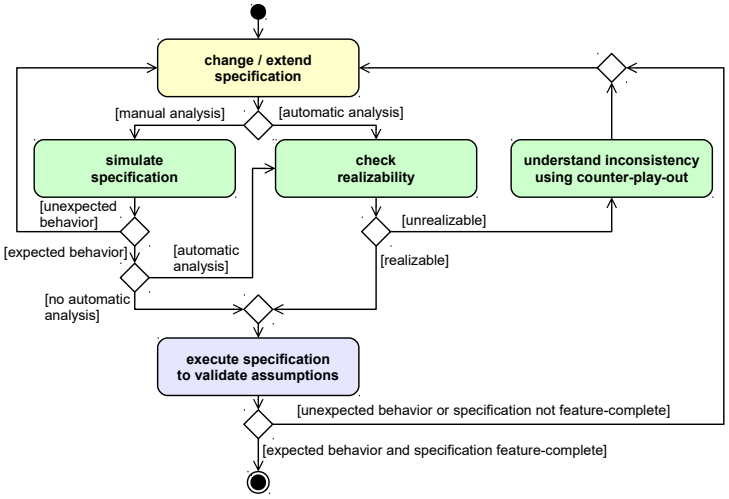

Figure 3: Using our methodology to build a cyber-physical system

on the play-out algorithm [BGP13]. This algorithm allows the system to perform any finite number of steps, represented by requested messages in the specification, between any two environment events, also represented by messages in the specification. Fig. 5 shows a visualization of the object model's state at runtime. This visualization highlights the last message and its effects. It also reflects structural changes such as cars changing lanes. This helps engineers understand structural dynamism of the object configuration. The visualization is based on SIRIUS and thus customizable for each system. If a simulation reveals that the system does not yet behave as expected the engineer can go back to refining the specification. If it behaves correctly instead, the engineer can still decide to do automatic realizability checking. Going directly to the execution step is possible as well.

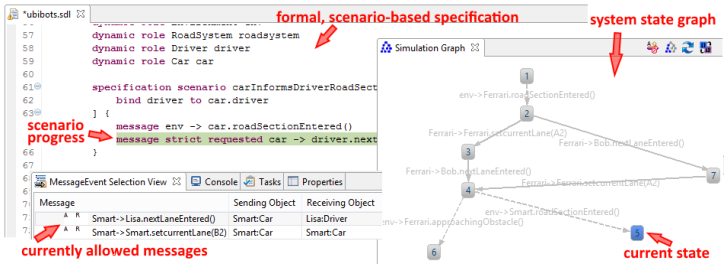

Figure 4: The simulation environment

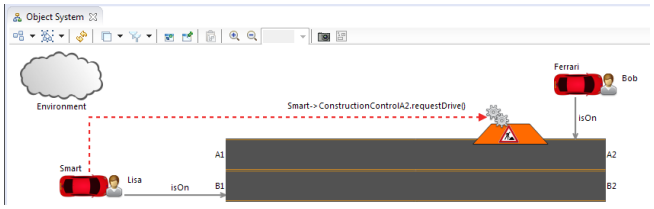

Figure 5: Visualization of the object model's state at runtime

For realizability checking [Gr13] SCENARIOTOOLS still generates the list of enabled messages but automatically chooses what message to pick next. It explores all possibilities in order to find a strategy, i.e., instructions defining when which object must send what message, in which all objects react correctly and as desired without causing any violations of the specification. If such a strategy exists, the specification is said to be realizable. If it is unrealizable, a counter-strategy, which describes how a system can be forced into a violation, can be created. A counter-play-out, i.e., a simulation with a limited message list restricted by a counter-strategy, then helps engineers to identify inconsistencies.

Finally, in the **execution step** the specification is executed on actual hardware in a sufficiently realistic environment. In our example in Fig. 1 Raspberry Pi-based robots played the roles of autonomous cars to test our Car-to-X specification. This enables testing whether assumptions made about the environment, such as when which message from the environment actually occurs, are actually valid. In SCENARIOTOOLS it is even possible to explicitly model environment assumptions as scenarios. This helps with understanding how the system must behave, because the environment it is in can be understood better.

4 Related Work

Similar scenario-based specifications techniques exist in the form of graphical Live Sequence Charts [DH01, HM03], and have been adopted in the form of *behavioral programming* in various programming languages [HMW12]. In joint research, we are extending these techniques to support structurally dynamic systems.

There has been extensive previous work on model-based and formal specification of CPSs, for example in the SPES 2020 project [Po12] and AutoFOCUS 3 [Ar15]. Also, RATSY [Bl10] uses formal specifications to synthesize implementations of reactive systems. To the best of our knowledge, however, there is no other previous work on directly executing scenario specifications at runtime.

5 Outlook and Research Roadmap

Realizing a cyber-physical system is a difficult task which becomes even more difficult if its components are physically distributed systems that need to cooperate yet execute their specific part of the system implementation independently. We already have a naive implementation of a distributed execution of a specification and we plan to improve it in order to make more economic use of available resources, e.g., network bandwidth.

While we already support counter-play-out to help engineers find and understand inconsistencies, we are working on offering additional tools to support engineers even more to reduce the time required to fix problems in the specification.

Cyber-physical systems are often found in safety critical areas in which human lives depend on the CPS being available and working correctly. However, little work has been done on dependability in the area of scenario-based specifications. We plan on expanding our method to include features to detect faulty behavior and how to recover from it.

6 Conclusion

In this paper we presented SCENARIOTOOLS and how to use scenario-based specifications to design cyber-physical systems using a Car-to-X example. Scenarios are an intuitive method for creating formal behavior specifications. They enable the use of powerful tools which assist engineers with designing consistent systems. This in turn reduces development costs as sources for defects can be removed early in the development process.

References

[Al14] Alexandron, G.; Armoni, M.; Gordon, M.; Harel, D.: Scenario-Based Programming: Reducing the Cognitive Load, Fostering Abstract Thinking. In: Proc. 36th Int. Conf. on Software Engineering (ICSE). pp. 311–320, 2014.

[Ar15] Aravantinos, V.; Voss, S.; Teufl, S.; Hölzl, F.; Schätz, B.: AutoFOCUS 3: Tooling Concepts for Seamless, Model-based Development of Embedded Systems. Joint proceedings of ACES-MB 2015–Model-based Architecting of Cyber-physical and Embedded Systems, p. 19, 2015.

[BGP13] Brenner, C.; Greenyer, J.; Panzica La Manna, V.: The ScenarioTools Play-Out of Modal Sequence Diagram Specifications with Environment Assumptions. In: Proc.12th Int. Workshop on Graph Transformation and Visual Modeling Techniques (GT-VMT 2013). volume 58. EASST, 2013.

[Bl10] Bloem, R.; Cimatti, A.; Greimel, K.; Hofferek, G.; Könighofer, R.; Roveri, M.; Schuppan, V.; Seeber, R.: RATSY–a new requirements analysis tool with synthesis. In: Computer Aided Verification. Springer, pp. 425–429, 2010.

[DH01] Damm, W.; Harel, D.: LSCs: Breathing Life into Message Sequence Charts. In: Formal Methods in System Design. volume 19. Kluwer Academic Publishers, pp. 45–80, 2001.

[GMMS12] Gordon, M.; Marron, A.; Meerbaum-Salant, O.: Spaghetti for the Main Course? Observations on the Naturalness of Scenario-Based Programming. In: Proc. 17th Conf. on Innovation and Technology in Computer Science Education. pp. 198–203, 2012.

[Gr13] Greenyer, J.; Brenner, C.; Cordy, M.; Heymans, P.; Gressi, E.: Incrementally Synthesizing Controllers from Scenario-Based Product Line Specifications. In: Proc. 9th joint meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering 2013. 2013.

[Gr15] Greenyer, J.; Gritzner, D.; Gutjahr, T.; Duente, T.; Dulle, S.; Deppe, F.; Glade, N.; Hilbich, M.; Koenig, F.; et al.: Scenarios@run.time – Distributed Execution of Specifications on IoT-Connected Robots. In: Proceedings of the 10th International Workshop on Models@Run.Time (MRT 2015), co-located with MODELS 2015. 2015.

[HM03] Harel, D.; Marelly, R.: Come, Let's Play: Scenario-Based Programming Using LSCs and the Play-Engine. Springer, 2003.

[HMW12] Harel, D.; Marron, A.; Weiss, G.: Behavioral programming. Commun. ACM, 55(7):90–100, July 2012.

[Po12] Pohl, K.; Hönniger, H.; Achatz, R.; Broy, M.: Model-Based Engineering of Embedded Systems - The SPES 2020 Methodology. Springer-Verlag Berlin Heidelberg, 2012.