Abstraction

In recent years, the importance of software is increasing and the functions carried by software also become complicated. Therefore, at the time of software design, we divide system into modules by functions, and realize the request of the user as software through the messages between the modules. The designer of software assumes user's use cases and determines the patterns of the message transmission and reception between the modules. However, if the system has a module which cannot understand its internal structure and must grasp behavior only from interface specifications, software designer cannot finish grasping the behavior of the module and the unexpected patterns of the message transmission and reception between the modules cause problems. For examples, modules provided on business are open to external interfaces, but the internal structure is unknown. In such modules, natural language specifications are provided in the form of interface specifications, and the designer grasps the behavior of the module. However, since descriptions of natural languages often involve ambiguity, it is sometimes impossible for the designer to grasp the behavior of the module correctly. As a result, it leads to leakages of the patterns of the message transmission and reception between the modules and the unexpected problems.

In this research, we propose a method to detect unexpected problems. In the research, we focused on the fact that the unexpected problems was caused by the ambiguity of specifications, and analyzed existing specifications. In chapter 3, results of analysis, we arranged the information obtained at designing software from specification and defined the general form of the specifications. And, in chapter 4, we also defined a method for generating a state transition model from the general specifications. Next, in chapter 5, in order to clarify how ambiguity affects model generation, we arranged the part where ambiguity is included in specifications and the influence on model generation. When ambiguity is included in the description of the specifications, it is not possible to reflect collect behavior of the module in the state transition model. If there is an omission of the reflection which behavior of the module is reflected in the model, there is a possibility that it will lead to unexpected behaviors. Therefore, in the proposed method, in order to eliminate the omissions of the reflection of module behavior on the model, we reflect all possible behavior of the module at the time of model generation. In chapter 6, in order to reflect the behavior formally, at the method, we patterned the reflection on the modules based on the arranged result of the part where ambiguity is included in specifications and the influence on model generation. As a result, the model reflects all the possible behaviors that can be read from the specifications on the model, and it has become possible to prevent omissions from ambiguity. However, this causes another problem. Incorporating all behaviors may cause too many patterns of the message transmission and reception between the modules to consider. Then, we model modules which make up the system and the messages between the modules, and apply model checking. Therefore, we can exhaustively verify by model checking and detect the patterns of messages between the modules as counter examples.

In chapter 7, we did an experiment of the research method. In the experiment, we targeted the embedded software development of in-vehicle audio system that I actually experienced in the past. The audio system consists of three layers of modules, one of which is provided on business with specifications. There are

problems in the module to be designed latently. In the experiment, the three layers of modules ware represented by the state transition modules and applied model checking. In chapter 8, we considered the result of experiment. As a result, it was possible to detect the latent problems, and the effectiveness of the method was demonstrated.