論文 / 著書情報
Article / Book Information

題目(和文)	
Title(English)	Recovering a Summarized Sequence Diagram through Program Analysis
著者(和文)	野田訓広
Author(English)	Kunihiro Noda
出典(和文)	学位:博士(工学), 学位授与機関:東京工業大学, 報告番号:甲第11161号, 授与年月日:2019年3月26日, 学位の種別:課程博士, 審査員:小林 隆志,佐伯 元司,権藤 克彦,渡部 卓雄,林 晋平
Citation(English)	Degree:Doctor (Engineering), Conferring organization: Tokyo Institute of Technology, Report number:甲第11161号, Conferred date:2019/3/26, Degree Type:Course doctor, Examiner:,,,,
学位種別(和文)	博士論文
Category(English)	Doctoral Thesis
種別(和文)	論文要旨
Type(English)	Summary

（博士課程）
Doctoral Program

論 文 要 旨

THESIS SUMMARY

系・コース： Department of, Graduate major in	情報工学 系 情報工学 コース	申請学位（専攻分野）： Academic Degree Requested	博士 （ 工学 ） Doctor of
学生氏名： Student's Name	野田 訓広	指導教員（主）： Academic Supervisor(main)	小林 隆志
		指導教員（副）： Academic Supervisor(sub)	

要旨（英文 800 語程度）
Thesis Summary（approx.800 English Words ）

Program comprehension is a quite important activity in software maintenance. Software documentation is a vital source of information for program comprehension; however, it tends to be outdated or non-existent in many cases. Moreover, comprehending the behavior of an object-oriented system solely from its source code is cumbersome and error-prone due to its dynamism (e.g., dynamic bindings and reflections).

For aiding behavioral comprehension, recovering a sequence diagram from an execution trace, which reflects the actual state of a system, is a promising approach. However, owing to the massiveness of execution traces, reverse-engineered sequence diagrams are often afflicted by scalability issues.

There are many existing techniques for coping with the massiveness of reverse-engineered sequence diagrams. Most existing work focuses on compacting the vertical size of recovered diagrams or effectively exploring massive-scale diagrams (e.g., compaction of repetitive behavior in execution traces and interactive exploration). To address the scalability issues, decreasing the horizontal size of recovered diagrams is likewise essential. Nonetheless, few studies have addressed this point.
Thus, further development and improvement of reduction techniques that focus on the horizontal direction are needed.

In this dissertation, we propose a fully automated technique for recovering a summarized sequence diagram of a reasonable size, with a central focus on reducing the horizontal size of recovered diagrams. Our technique consists of three parts: (1) modeling an execution trace with our behavior model; (2) identifying core objects that play important roles in an execution scenario; (3) constructing object groups corresponding to concepts for recovering a summarized sequence diagram.

(1) Modeling an execution trace with our behavior model

Sequence diagram recovery techniques take execution traces as their input. Most existing studies on sequence diagram recovery develop their own tracers, and the formats of traces vary across those tracers.

We present a behavior model (B-model) that models the behavior of an object-oriented system for behavioral design recovery. With the B-model, we can formalize an execution trace in a tracer-independent format; this enables us to develop several trace analysis techniques independent of tracers. Besides, we define mappings from B-model elements to sequence diagram elements; a B-model event sequence can be easily converted into a sequence diagram, which fills the gap between the representation of an execution trace and that of a sequence diagram.

To demonstrate the feasibility and sufficiency of the B-model, we show an application, which calculates a slice of a recovered sequence diagram, built on the B-model. Through examples of slice calculation, we show that our slicing application successfully recovers sliced sequence diagrams of a reasonable size; it confirms that the B-model holds finer-grained information required for traditional program analyses, and can be a useful tool for behavioral design recovery.

(2) Identifying core objects

The most important concepts of a system are implemented by very few key classes. Comprehending the behavior of objects of such key classes is highly important in an early stage of program comprehension.

We present a technique for identifying core objects of such key classes, which plays important roles in an execution scenario, from a behavioral point of view. Our core identification technique consists of two steps. First, we identify and eliminate temporary objects, which are trivial to comprehend a behavioral overview of a system, by analyzing dynamic scopes of objects based on reference relations and lifetimes thereof. Then, we estimate the importance of non-temporary objects based on access frequencies, and thereby identify core objects.

The results of our experiment using traces generated from various types of open source software (OSS) show that our technique outperforms the state-of-the-art technique. In the experiment, our technique identified a total of 5--11 core objects within top 7--222 (108 on average) important objects, whereas that of the state-of-the-art technique ranged 148--4,378 (998 on average).

(3) Constructing object groups for recovering a summarized sequence diagram

To improve maintainability, in object-oriented programming, a concept is often divided into several classes by using design patterns. From the perspective of program comprehension, it increases the number of design elements to comprehend, and enlarges the amount of effort required for program comprehension.

We present techniques for constructing object groups and recovering a summarized sequence diagram. Our technique constructs object groups at a concept-level based on Pree's meta patterns that are the most primitive design patterns. We detect all the occurrences of Pree's meta patterns in program code, and then groups template and hook objects involved in the same meta pattern; this reunifies divided concepts as object groups.

Given the results of core object identification, we identify important object groups. By visualizing intergroup interactions only among important groups, we obtain a summarized sequence diagram of a reasonable size that depicts object interactions at a concept-level.

In an early stage of program comprehension where developers do not attain a detailed knowledge of a subject system, they prefer highly abstracted views to comprehend behavioral overviews rather than finer-grained views depicting detailed interactions. Because our summarized sequence diagram provides a behavioral overview (an abstracted view) of important object groups (concepts), our solution can be a valuable tool in an early stage of program comprehension.

The results of our experiments using various types of OSS show that our technique outperforms the state-of-the-art trace summarization technique in terms of reducing the horizontal size of a reverse-engineered sequence diagram. Regarding the quality of object grouping, under the condition of #lifelines < 30 (which is acceptable for manual investigation), our technique achieved an F-score (resp. a Recall) of 0.670 (resp. 0.793); this is 0.249 (resp. 0.123) higher than that of the state-of-the-art technique.

The runtime overhead incurred by our technique was 129.2% on average. This overhead is relatively small compared with recent scalable dynamic analysis techniques; thus, it is acceptable in a development phase.

Overall, our technique can recover a summarized sequence diagram with a small runtime overhead. The recovered diagram depicts a behavioral overview of important concepts in a subject system, which is expected to be a valuable tool in an early stage of program comprehension.