CEUR-WS.org/Vol-2293/jist2018pd_paper12.pdf

# Extraction of Functional Structure Graph from System Design Documents

Eiichi SUNAGAWA and Shinichi NAGANO

Toshiba Corp., Komukai-toshiba-cho 1, Kawasaki, Kanagawa 212-8582, Japan
{eiichi1.sunagawa, shinichi3.nagano}@toshiba.co.jp

**Abstract.** In system development, reuse of modules from existing systems is an important issue. However, such reuse becomes difficult in similar but customized applications because the relations among module functions are complicated. To address this problem, we aim to establish a technology that can extract, from design documents, a knowledge graph that represents the functional structure of a system and use it in development of other systems. In this paper, we introduce our knowledge-graph extraction framework and describe an experiment show a partial extraction.

**Keywords:** function graph, knowledge graph, ontology, machine learning

## 1 Introduction

Nowadays, software systems are rarely developed from scratch. Instead, most reuse existing modules. Rational Derivational Development and Software Product Line (SPL) development are typical styles of system development using extant modules. In doing such development, it is necessary to consider the functions and relations among available modules (e.g. their behaviors and functionality) along with their purposes, conditions, and requirements. However, migrating more complicated modules increases the difficulty for developers to acquire knowledge about the system functions because they have to read many documents, find the descriptions of functions of interest, and summarize them.

This research aims to support developers seeking to understand the functional configuration of systems, particularly those developed with reuse of several existing modules, by using a knowledge graph to represent how the system and its functions work. The goal is to simplify development of a common platform for a series of system products. We establish technology that can extract the knowledge graphs of system functions (function graphs) from design documents and integrate the graphs into a comprehensive model as a conceptual platform (Fig. 1) [1]. A graph of this type helps developers to share understanding of the system, to determine a common module that will be needed for various services, to grasp the status of ongoing development projects, to check the range of impact when adding a new module, to access necessary development assets, and to improve the efficiency of the development process.

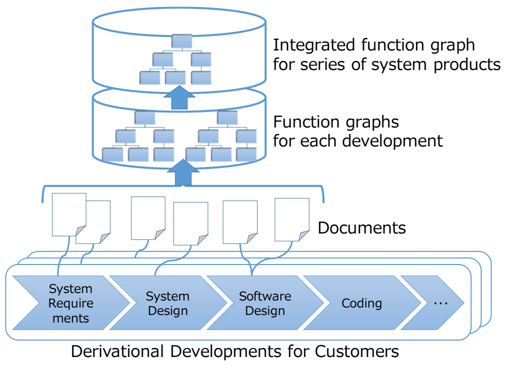

**Fig. 1.** Overview of Function Graph and Development Process

This paper introduces a part of our trial approach for extracting a knowledge graph of system functions from design documents. In Section 2, we explain the extraction method. In Section 3, we describe an experiment in which expressions are detected in functions and, in Section 4, we discuss the result. In Section 5, we summarize this paper.

## 2 Extraction of Function Graph from Documents

In this research, a function graph is a directed graph network in which nodes and links (edges) correspond functions and their relations, respectively. An example of a function graph is shown in Fig.2.

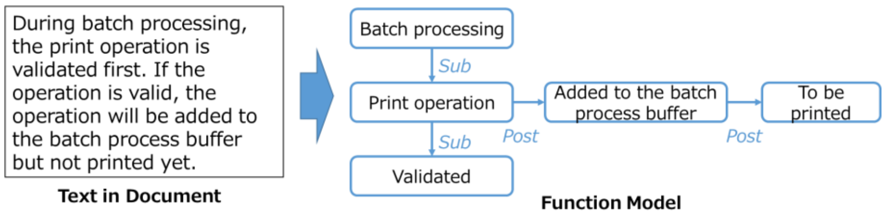
During batch processing, the print operation is validated first. If the operation is valid, the operation will be added to the batch process buffer but not printed yet.
Text in Document
Batch processing
Sub
Print operation
Added to the batch process buffer
To be printed
Sub
Post
Post
Validated
Function Model

**Fig. 2.** Example of Text and Function Graph

The nodes contain partial text from documents describing functions, such as the names of the functions or the process in which the functions are used. Three types of relations between functions are defined.

- *Same-as*: texts indicate the same function
- *(Super-)Sub*: one function includes the other as a sub-function
- *(Pre-)Post*: one function is executed after the other begins

A function graph is extracted from design documents semi-automatically by supervised learning model. An overview of the extraction is shown in Fig.3. First, texts in the document are analyzed and divided into phrases, which typically consist of several words. The phrases are converted into numerical vectors according to their words, meanings as defined in a domain ontology, and modifying relations. After that, each vector is classified into a class of expression for some function, or not. Finally, pairs of function phrases are classified into a class in which some type of relation (including a null relation) exists between the pair. To prepare the learning data, the system developers checked and corrected the results of classification and updated the learning data in an online fashion before each classification. Moreover, the developers updated the ontology for feature extension.

The phrases and their pairs are converted into feature vectors based on documents. However, developers determine which phrases are treated as functions and what relations are indicated by the phrases. Therefore, a function graph represents the developers' interpretation and understanding of the design documents.

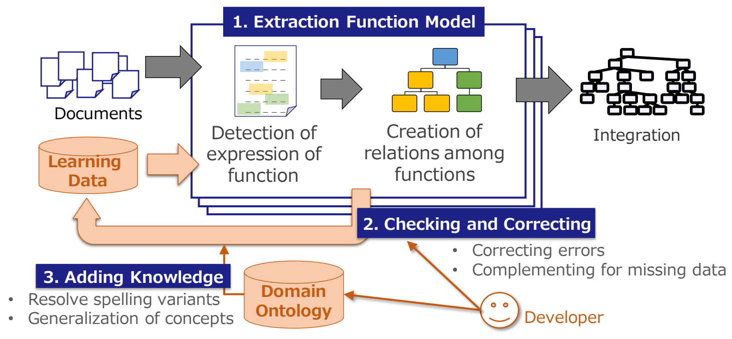
1. Extraction Function Model
Documents
Detection of expression of function
Creation of relations among functions
Integration
Learning Data
2. Checking and Correcting
Correcting errors
Complementing for missing data
3. Adding Knowledge
Resolve spelling variants
Generalization of concepts
Domain Ontology
Developer

**Fig. 3.** Process of Function Graph Extraction

Feature Models [2] and functional decomposition trees [3] have already been proposed also as a model for capturing the system hierarchically. These models are constructed basically by hand.

## 3 Experiment

A point-of-sales (POS) system in the retail domain is a representative product, composed of several modules and often customized for each retailer. Design documents issued by the Open-POS Technical Council (OPOS document) are available for POS systems with various specifications. Using the OPOS document, we performed an experiment to detect expressions of functions.

First, we selected 18 chapters from the OPOS document, treating these as smaller documents for designing each of the system modules: a line display, a scanner, a coin dispenser, and so on. From one chapter, we manually built a function graph. Then, we created and updated classification models with the learning data, applied the model to another module document, extracting that document's function graph, checked and corrected it, and finally added learning data and updated the models. We repeated this sequence for different orders of presentation of the modules.

## 4 Result

Figure 4 shows that the average AUC score changes as the amount of learning data is increased. In one case, data are added according to the modules defined by the OPOS document, in the other case, data are added from various modules after mixed and equally divided. There was no notable difference in AUC score between these cases. This likely reflects that each document among the modules includes important features for extraction of the function graph. The performance was roughly similar among all modules, which is expected when the writing styles are similar between documents.

When the data were presented by module, the precision and recall scores for detecting expressions of functions were 0.68 and 0.72, respectively when averaged across the whole process and 0.72 and 0.81, again respectively, for the last module. We conclude from this that semi-automatic detection of functions is possible.

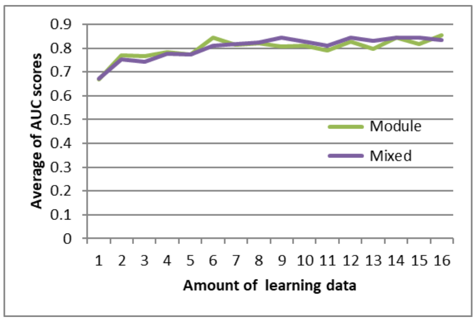

**Fig. 4.** Average AUC Scores Changing as the Amount of Learning Data

## 5 Conclusion

In this paper, we introduced a prospective approach for extracting a functional model, which represents the functional structure of a target system, from system design documents. This should improve the reusability of development aspects. To show the working of this approach, we described an experiment in which expressions of functions were detected from documents. In future work, we intend to develop an entire framework for extracting and using functional graphs, increase the application's generality, and evaluate it.

## References

1. SUNAGAWA, E., NAGANO, S.: An Approach to Extract Functional Model in System Design Documents, SIG-SWO 044-01 (2018) (in Japanese)
2. Apel, S., Kästner, C.: An Overview of Feature-Oriented Software Development. Journal of Object Technology 8(5), 49-84 (2009)
3. Kitamura, Y., et al.: Deployment of an ontological framework of functional design knowledge. Advanced Engineering Informatics 18(2), 115-127 (2004)