New Trend of Software Engineering

Development of innovative bidirectional software

Productivity enhancement has long been a major issue in software development. Conventionally, update was undertaken by tracing back to the upstream processes when any change was made in the process of software development. This requires a tremendous amount of labor. In reality, the development works progress without tracing back to the previous processes, and discrepancies between the design and the implemented systems are often seen. Accordingly, Professor Hu and his team have developed an epoch-making system to synchronize the updates made in each stage of the processes, which could dramatically change the software development process. We interviewed the three researchers regarding the study of linguistic infrastructure technologies for bidirectional model transformation.

Zhenjiang Hu
Professor, Information Systems Architecture Research Division, NII

Soichiro Hidaka
Assistant Professor, Information Systems Architecture Research Division, NII

From unidirectional to bidirectional software development

Hu: NII has addressed various difficult issues to be tackled by next-generation informatics in the Grand Challenge Project Program. Our Study of the Linguistic Infrastructure Technologies for Bidirectional Model Transformation is one such Project. It is an international project organized mainly by NII researchers, including myself, with the participation of researchers from diverse organizations including the University of Tokyo, the University of Electro-Communications, the Shibaura Institute of Technology, Peking University, and Shanghai Jiao Tong University.

The keyword for the project is "evolution." Instead of simply developing software, the research aims to develop software that can cope with a society that is undergoing changes every minute, and make the software evolve along with the changes. For this purpose, the core technology would be "bidirectional." In other words, we are aiming to develop technologies with feedback mechanism in the process of software development.

For example, if you prepared a report and submitted it to your boss, he/she would ask you to make corrections, such as enlarging the title or using different wording. You would then make the corrections and submit the report again. We wanted to do the same in the world of programming. Conventionally, software development proceeded in one direction, from upstream to downstream. In contrast, the bidirectional approach plans to form a loop. If a correction is made in a downstream process, automatic corrections are made in all the other processes, including the design stage in the upstream. We hope to evolve programs in this way.

In conventional programming, directives flow in one direction only. If you need to trace back upstream, it is necessary to write another program to return. However, it is very difficult to formulate two programs while maintaining consistency between the two directions, or to make changes by tracing back manually. We hope to build the infrastructure technologies for developing software that can evolve and apply them in various settings.

——What are the concepts behind the technology?

Hidaka: We use programming languages to develop software. Most of the conventional languages, like Java, are one-directional. We plan to create a new language enabling bidirectional computation. Essentially programming languages represent computations. Numerous small-scale computations are combined to create large-scale computations, and commands are given to computers. We decided that we could perhaps create a framework for bidirectional communications if each of the small-scale computation components were given bidirectional properties and then combined. For this purpose, it would be important to divide the programs neatly into small components, and provide easy-to-understand descriptions for them.

Kato: To be honest, the importance of bidirectional software transformation was highlighted in the 1980s. Trial and error was undertaken during this decade, with no specific methodologies

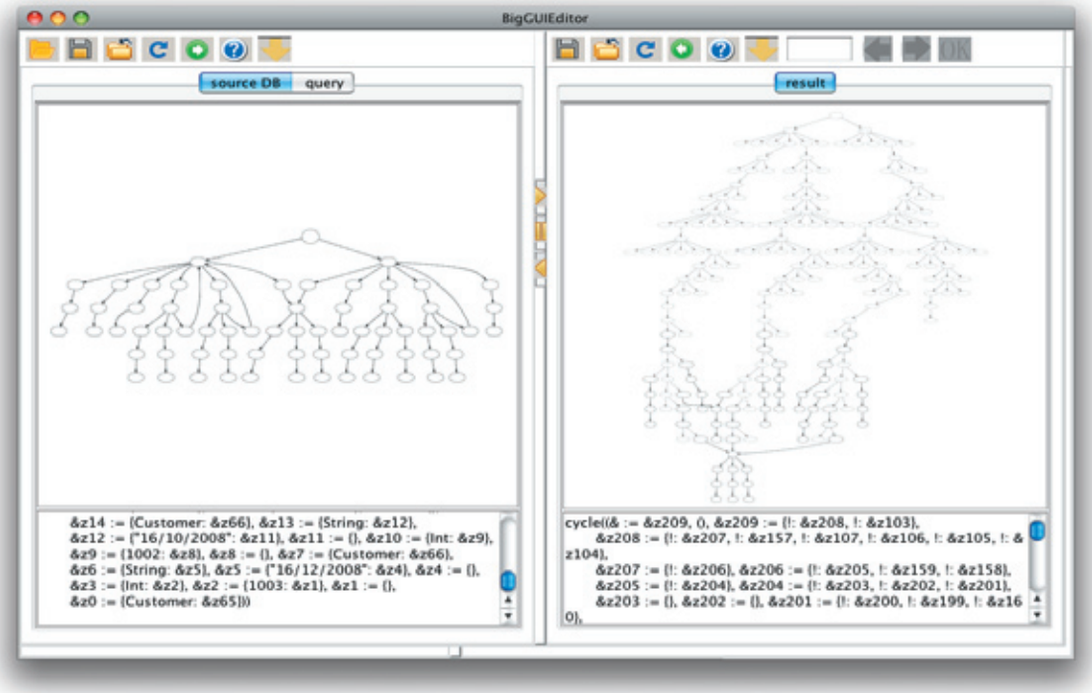

Fig.1: A personal computer screen showing examples of complicated graph transformations; the graph on the right shows the transformed version of the graph on the left.

able to be found. Among the results obtained during the 1980s was the development of a method called "View updating," which enables the cutting out of the portions to be modified and the provision of modification when making changes to a large database. Full-scale research into bidirectional technologies has been pursued since 2000. For instance, researchers from the University of Pennsylvania in the United States developed languages for synchronization, which enable, for example, the change made to a schedule book on the website to be reflected to data in different devices, such as personal computers, PDAs, and mobile phones.

Hu: I transferred to NII from the University of Tokyo two years ago, and the Takeichi Research Group, which I joined, was also trying to develop bidirectional languages for documents (specifications prepared when developing programs, etc.) It is a mechanism that can be used for website maintenance and other applications. If a website is required to be changed, it would be convenient if anyone could easily make the corrections without relying on expert service providers. Bidirectional transformation systems can achieve this too.

By the way, the majority of conventional programming languages deal with sequences and trees efficiently. The tree structure refers to a hierarchical structure without a loop. And most of the bidirectional transformation systems developed to date address computations on tree structures. Looking at the actual situation in society, however, we see mostly network structures, be it the relationship between a worker and his/her boss, or railway routes. As a result, we decided to address graphs. The graphs we are talking about comprise points (nodes) and lines (edges) that connect them, rather than the line charts or radar charts used with statistical data. Unlike tree structures, graphs sometimes share nodes or form loops. We are the first researchers to address graphs for bidirectional transformation mechanisms, which makes us pioneers in this area of research.

Development of transformation languages for achieving bidirectional mechanisms

——Specifically, what methods are used for driving the mechanisms?

Hu: In software development, executable instructions are developed step by step. The software artifact for each step (the software specifications/design drawings) is represented using a "model." The model refers to something that represents the essential nature of the object to be understood or manipulated according to a predetermined method. The Unified Modeling Language (UML) is used to represent models. Using UML, a model that uses graphical descriptions to express the abstract system is generated (Fig.1). In other words, the model we are talking about is essentially a graph.

Furthermore, transformation takes place from one model to the next, and so on. Ultimately, the codes (in the form of a graph) for executing the software can be generated. What we developed recently was the bidirectional transformation language UnQL+, which enables transformation in the reverse direction to provide bidirectional properties to the series of models.

Hiroyuki Kato
Assistant Professor, Digital Content and Media Sciences Research Division, NII

New Trend of Software Engineering

Kato: The language is an extended version of the existing query languages (languages for making queries on computer data). In other words, the language for extracting graph data has been extended to a language for transforming one graph into another. Grammatically, the syntax is very close to that of SQL (one of the query languages).

By using the bidirectional transformation language, automatic and consistent update would be possible by tracing the processes back to design or customer demand analysis, even when a change has been made in the implementation phase. This way, evolution of the software would be possible. The software design is separated from the actual architecture, because models are used. For this reason, the models can be reused even if the implementation technologies and architecture have been changed.

Hidaka: Let me now give a demonstration (Fig.2). You can see the models on the left- and right-hand sides of the screen. If you click the portion you want to update on the model on the right and make an update, the corresponding portions of the model on the left to be changed are displayed in red. You can see at a glance which portions have been changed and how. In actual programs, the size of models described is huge. It is therefore extremely significant that the portions that have been changed are displayed and made visible in synchrony. By the way, the text seen at the bottom is the representation of the diagram above in bidirectional transformation language. The transformation occurred instantly, and the speed is 100 times faster than the system we initially developed.

——Specifically, to what scenarios can the language be applied?

Kato: For example, when we develop software, it is tested before completion. If a defect is found, it needs to be corrected. If the correction is made in each process tracing back to the design drawings, the design (model) can be reused for other programs. It is very important to share information on the specifications of updated portions after testing by tracing back to the design phase, be it an automobile design or semiconductor design. Until now, there was no choice but to do this manually. Particularly in the case of the development of large-scale software, the design and implementation are undertaken by different people in many cases. As a result, a mechanism for automatic and bidirectional synchronization is needed to share information and reuse the design.

Hu: Speaking of bidirectional, there are many phases to it. For example, spotting the cause of a defect by tracing it back to the upstream processes, or the traceability mechanism, is one of them. Another is the mechanism that provides clues as to how to make a change when you wish to change some portions, and that makes the change by automatically achieving consistency and synchronicity. By using our mechanism, it would be possible to achieve bidirectional properties at various levels.

Hidaka: When we develop a program, it is rare for a genius programmer to begin writing it from scratch. Normally, programming proceeds from design to implementation phases in steps, according to a common method that everyone involved in the programming can understand. Since there are countless similar programs available in the world, it would be possible to reuse the software development processes if the consistency of the system were guaranteed by bidirectional transformation. We believe this would help enhance reliability and productivity.

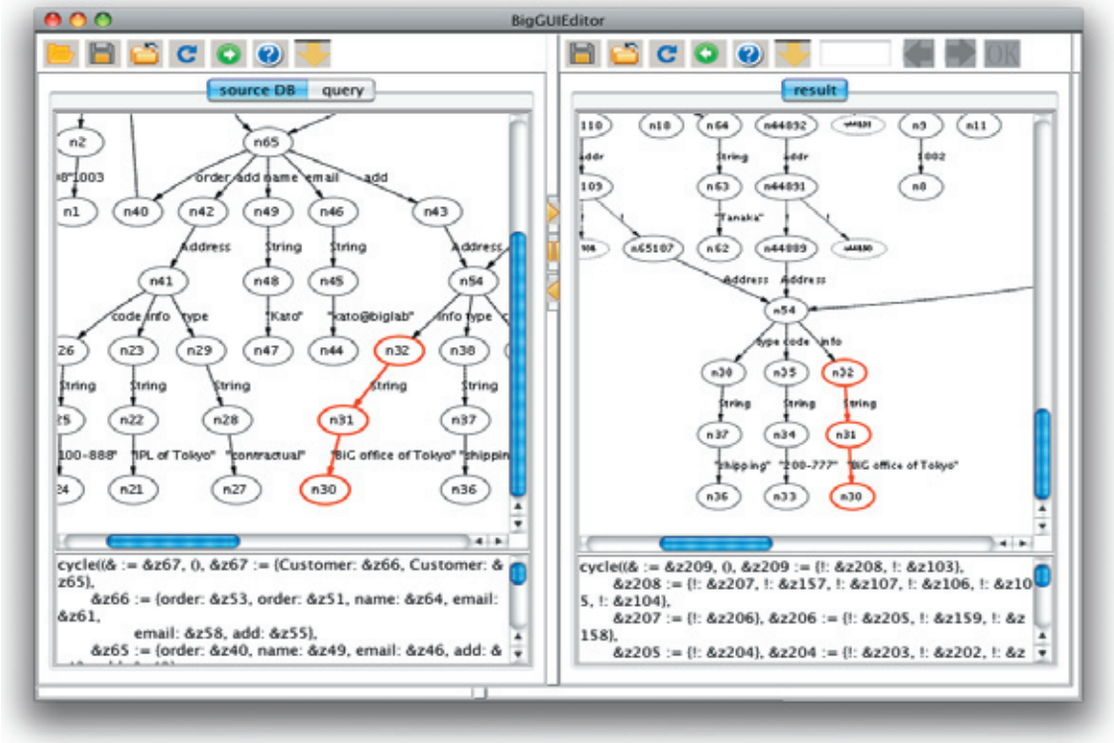

Fig.2: The node correspondence is shown in red; when an update is made on the right-hand screen (the red portion), the corresponding locations on the left-hand screen are displayed in red; the locations to be updated are identified at a glance.

Bidirectional transformation system under the spotlight

Hu: Our research has attracted a great deal of attention through the presentation made at the world's most authoritative academic society of software engineering [ICSE2009 (New Ideas and Emerging Results Track), FSE2009], and a paper chosen as the best paper at the Runtime Model Workshop, written by a student from Peking University who has worked at NII as an intern. In 2008, we organized the GRACE International Meeting on Bidirectional Transformations sponsored by the Center for Global Research in Advanced Software Science and Technology (GRACE) in NII, and held it at Shonan Village Center. The Meeting provided an opportunity for around 30 leading researchers on bidirectional transformation from numerous countries to gather and hold discussions. As can be seen, it is also very important that we have contributed to establish the new communities.

Moreover, to make our studies be widely recognized by many researchers, our project website (http://www.biglab.org) provides all papers we have published, the system and its source codes. In addition, we have on hand a demonstration system on which the bidirectional transformation can actually be performed, so that anyone can easily test the system. Recently, research institutes and enterprises have contacted us about the system. Discussions have commenced with an automotive parts manufacturer, as well as applications to sensor network research.

——What are the future prospects?

Hidaka: It was very helpful that each time we submitted and presented a paper at international conferences, numerous researchers expressed their opinions and provided criticism and advice, which led to improvements to the system. I hope to reason about the well-behavedness of the system in more detail and determine its properties.

Kato: At the same time as making improvements so that the system can be operated even more efficiently at greater speed, I think we should present what the system is suited for and also its drawbacks, rather than claiming that it can do everything. By doing so, I believe we can develop a truly useful system.

Hu: Software maintenance is a perpetual theme, and is the most costly aspect. Furthermore, it is extremely difficult to add new functions to software once it has been developed. Our research team has worked on the development and optimization of programming languages, along with theories on program development. It is built on a grand scheme that intends to develop the language itself from scratch and open up a new area of software engineering. It is extremely challenging, and that is the fun of it.

The members are having tough time staying up all night for many nights in a row, however, I would be very happy to see a greater number of application examples and persuade the world of the system's utility. (Written by Madoka Tainaka)