Technical Note

ASSE: A Support Environment for ADT Specification Based on Reuse of Similar ADT

PAIROJ TERMSINSUWAN,† ZIXUE CHENG†† and NORIO SHIRATORI†

To overcome difficulties in Abstract Data Types (ADTs) specification, we proposed an ADT Specification Support (ASS)[1], and an ADT Modification Algorithm (AMA)[2),3)]. However, there are still some remaining problems. To solve these problems and provide a more effective support method, in this paper we first extend our AMA, and then propose and implement an ADT Specification Support Environment (ASSE), which is an extension of ASS by the ADT Modification Editor (AME), designed on the basis of the extended algorithms. ASSE facilitates the specification, thus greatly reducing the specifier's load, since the specifier can avoid both specifying a required ADT from the beginning and modifying a similar ADT manually. Finally, the system is evaluated through an experiment with ADTs in the data communication field.

1. Introduction

Abstract Data Types (ADTs) provide a powerful formal technique for specifying communication services and protocols, but they are rarely used in practice[1]. To solve this problem, we proposed an ADT Specification Support (ASS)[1] method using case-based reasoning (CBR). ASS acquires the specifier's requirement and provides a similar ADT as an output. The remaining problem in ASS is that the specifier still has to modify a similar ADT to satisfy the requirement. As a first-step solution, we further proposed an ADT Modification Algorithm (AMA)[2),3)] to automatically modify the equation part of an ADT whenever a specifier inserts a new equation. However, AMA has not yet been incorporated into ASS and there are still some other types of modification (e.g., modifying functions and constructors) that AMA cannot support. To solve this problem and also to provide a more effective support method, we first extend our AMA, and then propose an ADT Specification Support Environment (ASSE), which is an extension of ASS by the ADT Modification Editor (AME) that provides four types of automatic modification of ADTs based on the extended algorithms. ASSE facilitates the specification, thus greatly reducing the specifier's load, since the specifier can avoid both specifying a required ADT from the beginning and modifying a similar ADT manually.

† Research Institute of Electrical Communication/ Graduate School of Information Sciences, Tohoku University

†† Department of Computer Software, The University of Aizu

2. ADT Specification Support Method

2.1 ADT specification

In this paper, an ADT $SPEC$ consists of sets of *sorts*, *operations* (further classified into constructors and functions) and *equations*, as described in[1),2)]. Syntactically, $SPEC$ is a triple

$$\langle S, OP_c \cup OP_f, E \rangle$$

where S is a set of sorts. OP_c and OP_f are a set of constructors and a set of functions, respectively. E is a set of equations describing the assignments of functions. Any equation is further classified as follows:

Let $c : S_1, \ldots, S_m \rightarrow S$ be a constructor, let $f : s_1, \ldots, s_n \rightarrow s$ be a function in $SPEC$, and let $f_k : s_1, \ldots, s_j, \ldots, s_n \rightarrow s$ be a function with some domain sorts (e.g. s_j) equal to S, which is the range sort of c.

An **equation related to function** f (equation describing the assignment of f) is defined as an equation of the form $f(t_1, \ldots, t_n) = t_s$, where each t_i and t_s are terms of sorts s_i and s, respectively.

An **equation related to function** f_k **with argument** c (equation related to f_k describing the assignment from at least one term of constructor c) is defined as an equation (having an instance) of the form $f_k(t_1, \ldots, c(T_1, \ldots, T_m), \ldots, t_n) = t_s$, where each t_i, T_i, and t_s are terms of sorts s_i, S_i, and s, respectively.

For convenience, we define E_f as a set of **all equations related to** f and $E_{f_k(c)}$ as a set of **all equations related to** f_k **with argument** c.

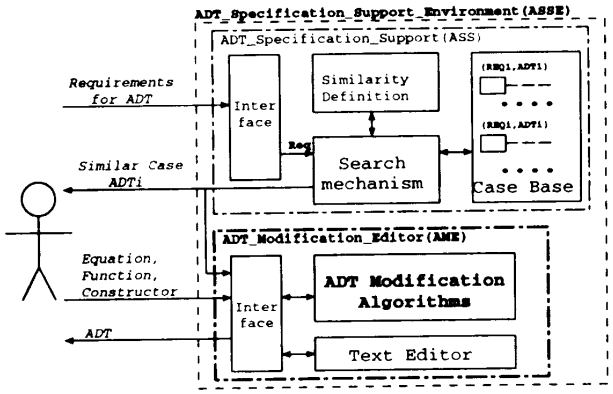


Fig. 1 Outline of ASSE.

2.2 Outline of ASS and the Remaining Problems in ASS

In order to provide a support system for specifying ADTs based on reuse of similar ADTs, we proposed ASS[1], which consists of the four parts shown in **Fig. 1**: (a) an **interface**, which consists of templates for acquiring the specifier's requirement, (b) a **case base**, in which ADT cases are kept to help the search mechanism look for similar ADTs, (c) a **similarity definition**, which calculates the similarity between two ADTs, and (d) a **search mechanism**, which receives the requirement from the interface, and searches for and retrieves a similar ADT from the case base.

The retrieved ADT is different from the required ADT in the sense that (1) some of the semantics of functions (equations) is different or (2) some of the functions or some of the constructors as well as the corresponding equations must be added or deleted.

However, these modifications, especially deletions and additions of equations, are difficult and may cause errors, (incompleteness and inconsistency), as stated in Termsinsuwan et al.[2].

3. ADT Specification Support Enviroment (ASSE)

3.1 Modification Algorithms of AME

As a first step to solve the remaining problems of modification support in ASS and AMA, we propose three new types of ADT modification algorithm based on AMA, which automatically modifies the equation part of an ADT whenever a specifier inserts a new equation without error. Each type of modification is defined as follows

Definition 3.1 (Deletion of a function)
Deletion of a function $f : s_1, \ldots, s_n \rightarrow s \in OP_f$ (as well as the corresponding equations to f) from $SPEC$ is a new ADT $DelFunc(SPEC, f)$ defined as $\langle S, OP_c \cup (OP_f - f), E - E_f \rangle$, where E_f is a set of all equations related to f □

Let c, f_k, and $E_{f_k(c)}$ be a constructor of sort S, a function with some domain sorts equal to S in $SPEC$, and a set of all equations related to f_k with argument c, respectively.

Definition 3.2 (Deletion of a constructor)
Deletion of a constructor $c \in OP_c$ (as well as the corresponding equations to c) from $SPEC$ is a new ADT $DelCons(SPEC, c)$ defined as $\langle S, (OP_c - c) \cup OP_f, E - All(E_{f_k(c)}) \rangle$ where $All(E_{f_k(c)})$ is $E_{f_k(c)}$ of all f_k □

Definition 3.3 (Addition of a constructor)
Addition of a new constructor c to $SPEC$ is a new ADT $AddCons(SPEC, c)$ defined as $\langle S, (OP_c \cup c) \cup OP_f, AddDummyEqn(E, c) \rangle$ where $AddDummyEqn(E, c)$ is E modified by, for each f_k, inserting $E_{f_k(c)}$ with a temporary constant $Dummy$ as RHS such that the set of modified equations is still complete and consistent. □

The modified ADT in this case is $SPEC$, including c and all new equations corresponding to c ($E_{f_k(c)}$) with $Dummy$ as RHS. The specifier can automatically modify these equations later by using our AMA to insert new equations corresponding to c until all $Dummy$ equations disappear. Algorithms corresponding to all modifications are also defined.

3.2 Design and Implementation of ASSE

In addition to ASS[3], we have designed AME, which consists of the following three components (Fig. 1):

(1) The ADT Modification Algorithms, consisting of the four algorithms mentioned in the previous section. Each algorithm gets a similar ADT from the main window as well as the modification information from a template of the Modify menu. As a result, the similar ADT is automatically modified and returned to the main window.

(2) The Interface, which is composed of a main window with a four-button menu bar, as shown in the upper middle window of **Fig. 2**. The main window gets a similar ADT from ASS. When a button inside the menu bar is clicked, one of the corresponding pulldown menus (File, Edit, Search, Modify) will be

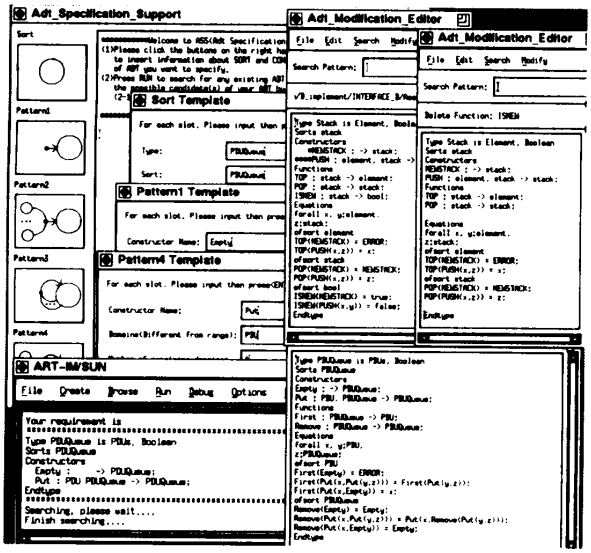


Fig. 2 An application example of ASSE.

posted. The File menu contains components for performing actions on the files (e.g., opening and saving). The Edit and Search menus are the interfaces to the Text editor. The Modify menu is a pulldown menu with four cascade buttons (Modify Equation, Delete Function, Delete Constructor, and Add Constructor), which will post one of the four templates corresponding to it.

(3) The Text Editor, which performs some helpful text-editing functions (e.g., text searching, text replacement, and cutting and pasting).

We have implemented a prototype of ASSE on a Sun workstation, using the Automated Reasoning Tool ART-IM as a search mechanism, Motif for the interfaces, and C for the remaining parts. The total program size is about 9,150 lines.

3.3 Application Example

The following is an example that involves specifying an ADT of PDU Queues in Sliding Window Protocol, as shown in Fig. 2. This ADT has two constructors (Empty and Put) and two Functions (First and Remove).

(1) From the ASS (upper left) window, the Requirement Acquisition Templates (Sort, Pattern 1–4 templates) acquire the requirements for ADT PDUQueue.

(2) The search mechanism finds Stack as the most similar ADT, as shown in the lower left ART-IM window, and sends it to AME (upper middle).

(3) To obtain PDUQueue from Stack, an unnecessary function (ISNEW) must be deleted and two new equations (TOP(PUSH(x, PUSH(y, z))) = TOP(PUSH(y, z)); and POP(PUSH(x, PUSH(y, z))) = PUSH(x, POP(PUSH(y, z)));) must be inserted. AME acquires this modification information from the specifier through the Delete Function and the Modify Equation templates. Then it automatically deletes ISNEW (the upper right window in Fig. 2) as well as the corresponding equations, and modifies the equation part of PDUQueue with the just-deleted ISNEW to satisfy the two properties of AMA. Finally, the specifier simply changes the names of Stack, Element, stack, element, NEWSTACK, PUSH, TOP, and POP to PDUQueue, PDUs, PDUQueue, PDU, Empty, Put, First, and Remove, respectively, to get the new ADT in the lower right window.

4. Experiments and Evaluations of ASSE

In order to evaluate the effectiveness of the system, we experimented with two groups of data types (Service and Protocol) of specifications in the data communication field. Details of the experimental ADTs and the preparation of the case base are given in other papers by the authors[1),3)].

4.1 Evaluation Method

ASSE is evaluated from the viewpoints of reuse of ADT specifications and modification support for the retrieved similar ADTs. For this purpose, we introduce the following three criteria:

Definition 4.1 (Full Support Rate (FSR), Specification Support Rate (SSR), No Support Rate (NSR))
In developing ADTs of any service or protocol,

- Let $j + k + n$ be the number of all ADTs in that service or protocol,
- let $j + k$ be the number of ADTs for which similar ADTs can be retrieved,
- let j be the number of retrieved similar ADTs that can be automatically modified by the system,
- and let k be the number of retrieved similar ADTs that specifiers have to modify manually.

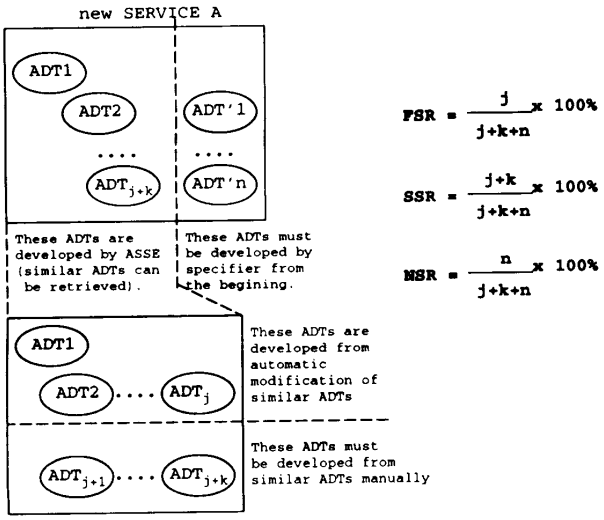
$$\mathrm{FSR} = \frac{j}{j+k+n} \times 100\%$$

$$\mathrm{SSR} = \frac{j+k}{j+k+n} \times 100\%$$

$$\mathrm{NSR} = \frac{n}{j+k+n} \times 100\% \qquad \square$$

4.2 Results and Evaluation

The results of experiments for both groups of



SERVICE NAME	FSR	SSR	NSR
Deamon	50%	50%	50%
Unreliable Medium	75%	75%	25%
Abracadabra Service	66.67%	83.3%	16.7%
Average	**63.89%**	**69.4%**	**30.6%**

PROTOCOL NAME	FSR	SSR	NSR
Sliding Window	63.6%	81.8%	18.2%
Abracadabra Protocol	87.5%	100%	0%
Average	**75.55%**	**90.9%**	**9.1%**

Fig. 3 Evaluation of ASSE.

ADTs are shown in **Fig. 3**. For example, among 11 ADTs in Sliding Window Protocol, there are 9 ADTs for which similar ADTs can be retrieved (SSAPs, SSDUs, SSPs, PDUs, MSAPs, MSPs, TimerSignal, PDUQUEUE, NatMinus). Seven ADTs (except PDUs and NatMinus) can be automatically modified by ASSE. Therefore FSR is (7/11) × 100% = 63.6%, SSR is (9/11) × 100% = 81.8% and NSR is (2/11) × 100% = 18.2%. FSR, SSR, and NSR for the remaining services and protocols are calculated in the same way.

We can see that in order to specify ADTs in data communication fields, from the viewpoint of the specifier's workload, a high percentage (69.4% and 90.9% respectively) of ADTs can be specified by ASSE (from SSR). Furthermore, FSR shows that a high percentage (63.89% and 75.55% respectively) of ADTs can be specified with full support. NSR also shows that the specifier's workloads of specifying all ADTs without support have been reduced quite significantly (from 100% to 30.6% and 9.1%, respectively).

5. Conclusion

To solve problems of modification support in ASS and provide a more effective support method, we have proposed ASSE, an extension of ASS, based on AMA. By using our system, a specifier can easily specify a required ADT by inputting modification information for each similar ADT retrieved from the case base. Therefore, the specifier's workloads in both (1) specifying the whole required ADT from the beginning and (2) modifying the retrieved similar ADT (comparing to ASS) are quite significantly reduced. Furthermore, in order to evaluate the effectiveness of the system, we experimented with data types in the data communication field. The evaluation confirmed the effectiveness of the system.

References

1) Termsinsuwan, P., Cheng, Z. and Shiratori, N.: A New Approach to ADT Specification Support Based on Reuse of Similar ADT by the Application of Case-Based Reasoning, Accepted for publication in *Information and Software Technology*.
2) Termsinsuwan, P., Cheng, Z. and Shiratori, N.: A New Approach to Reuse ADTs by Automatic Modification of Equation, Under Review by *Information Processing Letters*.
3) Termsinsuwan, P., Cheng, Z. and Shiratori, N.: A New Approach to Reuse ADTs and Its Application to ADT Specification Support, *Technical Report of IEICEJ*, IN94-140, pp.25–30 (Jan. 1995).
4) Manas, J.A. and Veiga, M.: Built-In and User Defined Data Types for LOTOS, Working Draft on Enhancements to LOTOS, ISO/IEC JTC1/SC21/WG1 N1349 (Oct. 1994).

(Received September 22, 1995)
(Accepted December 8, 1995)