Нечеткое сравнение коллекций: семантический и алгоритмический аспекты

В.А. Семенов, С.В. Морозов, О.А. Тарлапан, И.В. Энкович[1]

Аннотация. Рассматривается задача нечеткого сравнения коллекций в приложениях реконсиляции. Задача возникает при оптимистической репликации структурированных данных и документов и имеет многочисленные приложения в таких актуальных областях, как управление конфигурацией программного обеспечения, управление мобильными базами данных, построение платформ и систем коллективной инженерии. В работе анализируются стандартные типы коллекций языков объектно-ориентированного моделирования, для которых описываются и обосновываются способы представления, журнализации, вычисления, принятия и согласования изменений. Для выделенных типов коллекций дается строгая, семантически содержательная интерпретация конфликтов и предлагаются конструктивные методы их идентификации и разрешения. Примеры иллюстрируют предлагаемые решения, а также служат мотивацией создания универсальной, основанной на модельном представлении оригинальной среды коллективной инженерии с развитыми возможностями семантически корректной и функционально содержательной реконсиляции дивергентных реплик данных.

1. Введение

Сравнение коллекций является одной из традиционных задач сопоставления версий и анализа изменений при оптимистической репликации структурированных данных и документов [1]. Программные средства сравнения, предоставляющие подобную функциональность, являются неотъемлемыми компонентами современных систем контроля версий. Подобные системы нашли применение в задачах управления конфигурацией программного обеспечения (software configuration management), управления мобильными базами данных (mobile database management), организации цифровых архивов, документооборота (document management), построения платформ и систем коллективной инженерии (collaboration workspaces, concurrent engineering environments), управления web–контентом (content management). Так, среди наиболее популярных решений коллективной

[1] Работа коллектива поддержана РФФИ (грант 07-01-00427)

программной инженерии следует отметить системы управления версиями ПО: CVS, Subversion, OpenCM, BitKeeper, Visual SourceSafe, Perforce, Synergy CM и т.п. [2].

Оформленные в виде программных утилит с пользовательским интерфейсом средства сравнения имеют многочисленные самостоятельные приложения. Например, утилиты командной строки cmp, diff, sdiff, diff3 известного проекта GNU [3] позволяют определить первый отличный байт для заданной пары файлов, вычислить минимальное множество отличий между двумя файлами, интерактивно объединить два файла в один общий, а также представить структуру изменений при одновременной модификации одного файла двумя пользователями или программами и сгенерировать файл, являющийся результатом слияния двух версий относительно базовой с предупреждениями о возникших конфликтах. GNU Diffutils работают преимущественно с текстовыми файлами, для бинарных файлов они могут лишь констатировать факт сходства или различия. Пакет JojoDiff включает набор утилит для сравнения бинарных файлов: jdiff — сравнивает два файла, jpatch — реконструирует второй файл из первого на основе списка изменений, сгенерированного jdiff, jsync — синхронизирует файлы или директории между двумя компьютерами. Приложения KDiff, Xxdiff, gtkdiff, tkdiff, Diffstat, ExamDiff предоставляют средства визуального сравнения произвольных текстовых файлов. Известная утилита UN*X dirdiff позволяет сравнивать деревья каталогов и синхронизировать изменения между ними. Развитые графические средства сравнения текстовых файлов, а также синхронизации директорий включает в себя популярный файловый менеджер TotalCommander.

Перечисленные программные средства производят сравнение произвольных текстовых или бинарных файлов без учета семантики их содержимого. Ряд программных утилит и приложений позволяет сравнивать документы в различных текстовых или бинарных форматах, HTML, XML, Multimedia данные. В частности, учитывая типизацию отдельных термов в текстовых данных, утилита spiff позволяет адекватно сравнивать числовые данные с плавающей точкой, а также исходные тексты на популярных языках программирования. Набор приложений CSDiff/CS-ExcelDiff/CS-HTMLDiff позволяет установить и отобразить изменения документов, представленных в форматах Microsoft Word, Microsoft Excel, HTML. Программы ExamDiff Pro, Compare Suite, Diff Doc поддерживают сравнение документов в различных форматах популярных офисных приложений, PDF, HTML, в том числе документов, хранящихся в архивах. Интегрированные пакеты Microsoft Office 2003 и 2007 имеют развитые встроенные средства сравнения и слияния версий документов. Наборы средств diffxml, 3dm, XMLComparator обеспечивают сравнение, коррекцию и слияние документов в XML разметке. Программы Image Comparer, ImageDupeless специализируются на сравнении графической, а Similarity — звуковой информации, хранимой в файлах наиболее распространенных форматов.

Средства сравнения реляционных данных позволяют определить изменения схем и содержимого популярных баз данных. В частности, программа DataDiff находит отличающиеся записи в альтернативных базах данных под управлением MySQL, утилита mysqldiff — находит отличия в определениях таблиц СУБД MySQL, утилита MDBDiff позволяет выявить структурные изменения в базах данных Microsoft Access, Oracle SchemaDiff — изменения в схемах Oracle, pgdiff — изменения в реляционных таблицах двух баз данных под управлением PostgreSQL с возможной генерацией команд конвертации схемы одной базы данных в другую. Универсальная программа с web-интерфейсом SQLDiff устанавливает и показывает отличия между двумя SQL таблицами для распространенных реляционных СУБД (Oracle, DB2, PostgreSQL).

Ряд программных компонентов сравнения обеспечивает работу с объектными моделями. В частности, библиотека difflib, реализованная на Python, позволяет вычислять изменения в объектных данных. Программный модуль UMLDiff [4], функционирующий в среде программной инженерии Eclipse, определяет структурные изменения статических UML моделей. Более подробные сведения о вышеперечисленных утилитах и библиотеках, а также об аналогичных программных средствах сравнения данных и документов можно найти в Интернет-обзорах и каталогах [5–8].

Несмотря на разнообразие приложений, в которых математические методы и программные средства сравнения находят содержательное применение, следует отметить строгую функциональную специализацию и определенную ограниченность перечисленных выше средств. Главной причиной этого является ориентация на конкретные классы приложений с предопределенной семантикой. Безусловно, подобная направленность делает данные средства чрезвычайно полезными при решении конкретных прикладных задач, однако не дает возможность распространить их на нетривиальные масштабные междисциплинарные информационные модели, описываемые сотнями (иногда тысячами) типами данных и ограничений.

Более того, часто и в простых случаях результаты сравнения оказываются неадекватными решаемой задаче. Например, сравнение документов в разметке XML позволяет идентифицировать изменения, связанные с появлением, удалением, модификацией элементов, однако не фиксирует изменения положения элементов относительно друг друга. Приложения, для которых порядок следования элементов данных существенен, не могут доверительно использовать результаты сравнения. Например, приложение, фиксирующее изменения в рейтинговом списке ученых, упорядоченном в соответствии с показателем результативности научной деятельности, не может корректно интерпретировать результаты сравнения и нуждается в более адекватном способе представления изменений в соответствии с семантикой модели данных.

В обобщенной постановке задача сравнения обычно не рассматривается, хотя все упомянутые выше виды документов, файлов, схем баз данных могли бы быть прозрачно представлены информационными моделями/метамоделями, и их сравнение могло бы быть проведено на более общей методологической и инструментальной основе. Подобная постановка является критично важной для приложений семантической реконсиляции, оперирующих произвольными типами данных при заданной прикладной модели с формально описанными структурой и алгебраическими ограничениями. В подходе, предложенном и развитом в наших работах [9–12], вычисление изменений данных рассматривается в качестве ключевого элемента реконструкции конкурентных транзакций и их семантически согласованной реконсиляции на основе заданной информационной модели. Учет семантики модели позволяет на формальной, математически строгой основе провести сравнительный анализ реплицируемых данных и выработать непротиворечивые (семантически корректные) и содержательные (обеспечивающие полноту итоговой транзакции) политики реконсиляции.

В определенном смысле подход следует важному доминирующему направлению информационных технологий, предполагающему активное использование моделей на всех этапах программной инженерии и анализа информации. Деятельность международных сообществ по разработке соответствующих информационных стандартов и многофакторных прикладных моделей, прежде всего, в рамках ISO 10303 STEP [13] и OMG MDA [14] также следует этой тенденции.

В настоящей работе обсуждаются проблемы сравнения коллекций — стандартных типов данных, поддерживаемых популярными языками моделирования и программирования, повсеместно используемых в приложениях и представляющих собой наиболее сложные и интересные конструкции для рассматриваемых задач реконсиляции. Главным лейтмотивом статьи является вопрос об адекватности способов представления изменений типам коллекций, а также о выборе эффективных алгоритмов нечеткого сравнения коллекций в соответствии с их семантикой.

2. Задача нечеткого сравнения в приложениях семантической реконсиляции

Обсудим особенности постановки задачи нечеткого сравнения коллекций в приложениях семантической реконсиляции. Напомним, что традиционная задача сравнения последовательностей обычно формулируется как задача отыскания минимального скрипта редактирования (последовательности элементарных команд, обеспечивающей преобразование исходной строки в заданную другую строку) [20, 21]. Множество найденных команд при соответствующей интерпретации может служить представлением изменений, внесенных в модифицированную версию коллекции относительно исходной. Отыскание наибольшей подпоследовательности также решает задачу

нечеткого сравнения и позволяет представить изменения модифицированной коллекции в виде добавленных и удаленных фрагментов строк, дополняющих найденную общую подпоследовательность до заданных строк.

В отличие от традиционных постановок задач сравнения последовательностей, приложения реконсиляции оперируют с произвольными типами коллекций, и для адекватной реконструкции изменений требуется детальный семантический анализ. Другой важной особенностью рассматриваемой постановки является необходимость декомпозиции долгих транзакций на группы операций, которые могли бы быть представлены и применены независимым друг от друга образом. Это означает, что в ходе реконсиляции одна часть выявленных изменений может быть принята в итоговом представлении при отмене другой без каких-либо дополнительных ограничений. Выбранный базис элементарных операций должен удовлетворять данному требованию.

Итак, пусть $X, X', X'' \in collection < T >$ — некоторые версии коллекции элементов типа *T*, причем X — базовая версия, а X', X'' — версии, полученные в результате ее одновременной модификации в двух параллельных ветвях. Задача реконсиляции в наиболее распространенной постановке заключается в вычислении соответствующих изменений модифицированных версий относительно базовой $\Delta' = Diff(X', X)$, $\Delta'' = Diff(X'', X)$ и в консолидации изменений $\Delta^* = Merge(\Delta', \Delta'')$ таким образом, чтобы сформировать итоговое представление коллекции $X^* = Apply(X, \Delta^*)$ как результат применения согласованных изменений к базовой версии. Это, так называемая, классическая “3-way Merge” схема реконсиляции в отличие от схем “2-way Merge” и “4-way Merge”, имеющих более узкое применение [15, 16].

Например, если при одновременной модификации текстового файла в ходе выполнения одной транзакции была вставлена новая строка, а в ходе выполнения другой транзакции одна из строк была удалена, результатом консолидации изменений должен стать итоговый документ с внесенными общими изменениями (см. рис. 1), дополняющий его существующие версии новым согласованным вариантом.

Корректная идентификация изменений и реализация функции исполнения предполагают, что соответствующие тождества $X' = Apply(X, \Delta')$, $X'' = Apply(X, \Delta'')$ необходимо удовлетворены.

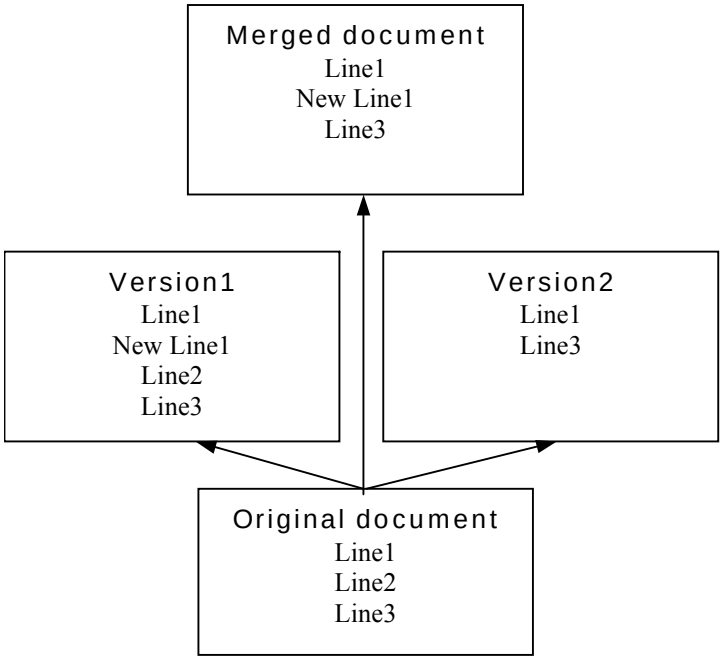

Рис. 1. Пример консолидации изменений в итоговом текстовом документе

Тривиальными случаями реализации функции реконсиляции являются $Merge(\Delta', \Delta'') = \varnothing$, $Merge(\Delta', \Delta'') = \Delta'$ и $Merge(\Delta', \Delta'') = \Delta''$, приводящие к уже известным версиям коллекции X, X' и X'' соответственно. Содержательная реализация функции реконсиляции $Merge(\Delta', \Delta'')$ нетривиальна, поскольку Δ', Δ'' могут представлять собой иерархически структурированные изменения $\Delta' = \{\delta'_1, \delta'_2, ...\delta'_{n'}\}$, $\Delta'' = \{\delta''_1, \delta''_2, ...\delta''_{n''}\}$, $\delta'_i = \{\delta'_{i,1}, \delta'_{i,2}, ...\delta'_{i,n'_i}\}$, $\delta''_i = \{\delta''_{i,1}, \delta''_{i,2}, ...\delta''_{i,n''_i}\}$ и т.д. По существу, дельты Δ', Δ'' являются реконструируемым представлением конкурентных транзакций, примененных к исходным данным X и имеющих результатом X' и X'' соответственно. В силу указанных причин реализация данной функции должна обеспечивать консолидацию как можно большего числа изменений при условии сохранения частичного порядка операций, индуцируемого семантикой приложений, и их бесконфликтности. В случаях, когда все конфликты разрешаются путем принятия изменений из первой версии, функция реконсиляции строится следующим образом:

$$Merge(\Delta', \Delta'') = \{\delta' \mid \delta' \in \Delta'\} \cup \{\delta'' \mid \delta'' \in \Delta'' \;\&\; \neg isConflict(\delta'', \delta'), \delta' \in \Delta'\},$$

где логическая функция $isConflict(\delta'', \delta')$ истинна в случае конфликта между изменениями δ', δ''. Является открытым вопрос о том, какие ситуации следует считать конфликтными, и каким образом они могут быть разрешены: отменой всех операций, выделением и принятием бесконфликтного подмножества операций, или путем их коррекции.

В дальнейшем под конфликтом будем понимать бинарное или множественное отношение между наборами или последовательностями операций в конкурентных транзакциях $\delta'_{i1}, \delta'_{i2}, ..., \delta'_{in'} \in \Delta'$, $\delta''_{j1}, \delta''_{j2}, ...\delta''_{jn''} \in \Delta''$, одновременное принятие которых приводит к некорректной интерпретации операций и неоднозначности результата, или к нарушению семантических ограничений, накладываемых на результирующее представление данных $X^* = Apply(X, \Delta^*)$, где $\Delta^* = \{\delta'_{i1}, \delta'_{i2}, ..., \delta'_{in'}, \delta''_{j1}, \delta''_{j2}, ...\delta''_{jn''}\}$.

Стандартный способ разрешения конфликта состоит в принятии одной из опций $Options = \{\varnothing; \delta'_{i1}, \delta'_{i2}, ..., \delta'_{in'}; \delta''_{j1}, \delta''_{j2}, ...\delta''_{jn''}\}$. При условии, что оригинальные транзакции приводят к корректным версиям X' и X'', а принимаемые операции не конфликтуют друг с другом, результирующая транзакция также будет корректна. В случае выявления конфликтов они разрешаются вплоть до достижения корректности итоговой транзакции. Заметим, что решение всегда существует, поскольку в качестве итоговой транзакции могут быть приняты $\varnothing$, Δ' или Δ''. Однако более содержательным была бы консолидация операций из обеих конкурентных транзакций.

Важной особенностью задачи нечеткого сравнения коллекций в приложениях реконсиляции является учет частичного порядка между операциями. Например, если при одновременном редактировании текста в одну и ту же позицию были добавлены отличные строки, то конфликт связан не с нарушением какого-либо ограничения для результирующего документа, а с неоднозначностью исполнения соответствующих операций δ', δ'' и с неопределенностью ожидаемого результата $X^* = Apply(X, Merge(\delta', \delta''))$. Возможным способом разрешения конфликта в данном случае являлась бы одна из опций: $Options = \{\varnothing; \delta'; \delta''; \delta' \angle \delta''; \delta'' \angle \delta'\}$, где символ $\angle$ означает отношение предшествования между исполняемыми операциями. Результатом может стать исходный документ, одна из его модифицированных версий либо документ с двумя возможными вариантами вставки строк (см. рис. 2). Заметим, что совпадение добавленных строк в обеих версиях модифицируемого документа не приводит к конфликту, и вариантами реконсиляции являются тривиальные решения $Options = \{\varnothing; \delta'; \delta''\}$, приводящие к уже существующим версиям документа.

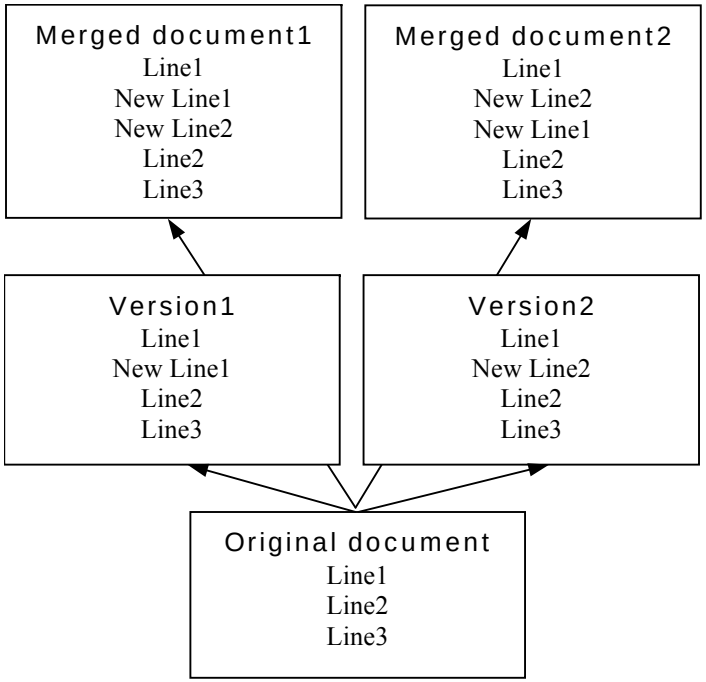

Рис. 2. Пример многовариантной реконсиляции с учетом частичного порядка операций редактирования текстового документа

Приведем вариант предыдущего примера редактирования текстового документа, иллюстрирующего важность учета семантики модели данных при дополнительном ограничении уникальности элементов коллекции. Предположим, что документ представляет собой персонофицируемый список имен, формируемый путем согласования альтернативных рейтинговых показателей в параллельных версиях (см. рис. 3).

В первой модифицируемой версии документа между персонами на первых двух позициях вставляется новое имя, после чего меняется порядок их следования. Дельта представляется следующим образом: $\Delta' = \{\delta'_1(insert, "NewPerson", 2), \delta'_2(transpose, 1, 3)\}$, где δ'_1 — операция вставки нового элемента в соответствующую позицию коллекции, а δ'_2 — операция транспозиции пары элементов в заданных позициях коллекции. Во второй версии документа то же самое имя вставляется на позицию между последними персонами, а также меняется порядок их следования: $\Delta' = \{\delta''_1(insert, "NewPerson", 3), \delta''_2(transpose, 2, 4)\}$. Заметим, что изменение порядка следования элементов коллекции и вставка новых элементов согласно репродуцируемой семантике множества не должна приводить к дублированию

имен в итоговом документе. Поэтому семантически корректными являются результаты $Options = \{\varnothing; \delta_1' \angle \delta_2'; \delta_1'' \angle \delta_2''; \delta_1'; \delta_2'; \delta_1''; \delta_2''; \delta_1' \angle \delta_2''; \delta_2'' \angle \delta_2'\}$, приводящие, соответственно, к оригинальным версиям *Original document*, *Version1*, *Version2*, версиям *Version1-1*, *Version1-2*, *Version2-1*, *Version2-2*, полученным частичным принятием операций одной из транзакций, и версиям документа *Merged document1*, *Merged document2*, полученным возможной консолидацией операций из двух конкурентных транзакций.

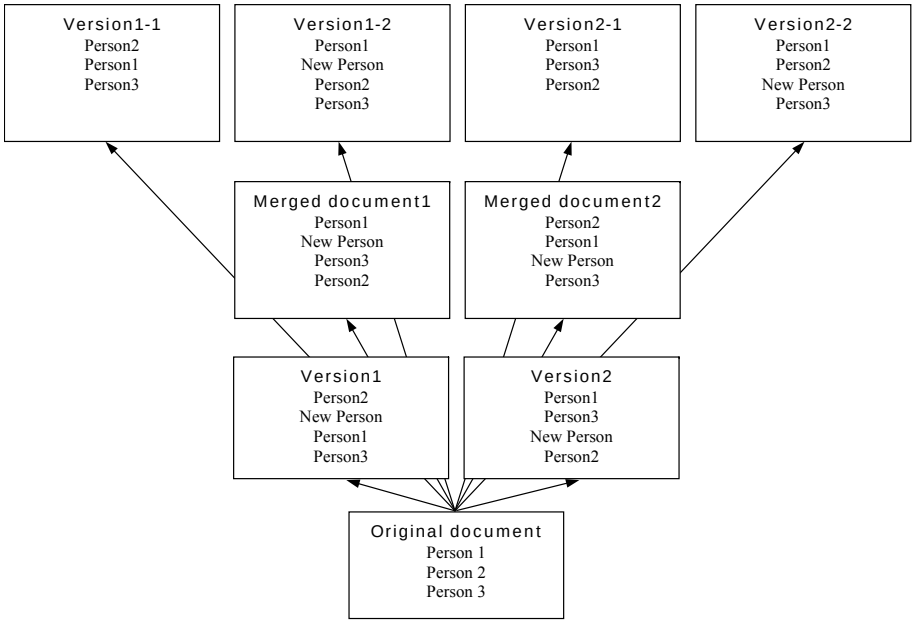

Рис. 3. Пример многовариантной реконсиляции с учетом семантики модели коллекции

3. Классификация коллекций

Коллекции — фундаментальный тип данных, поддерживаемый популярными языками моделирования и программирования для спецификации и реализации в приложениях агрегированных структурированных данных. Как иллюстрируют примеры предыдущего раздела, сравнение коллекций должно проходить в строгом соответствии с их семантикой. В противном случае результаты рискуют быть неадекватными исходной проблеме и теряют смысл для целевого приложения и пользователя.

Сравнение коллекций может рассматриваться в качестве частной задачи более общей проблемы семантического сопоставления (matching) и сравнения (differencing) расходящихся реплик структурированных данных, например, популяций объектов, заданных некоторой объектно-ориентированной моделью. Несмотря на многообразие частных типов коллекций, встречаемых в приложениях, можно выделить несколько фундаментальных свойств, в соответствии с которыми их анализ может проводиться содержательным образом. К таким свойствам мы относим уникальность элементов коллекции, упорядочение, возможную сортировку элементов коллекции, а также ограниченный размер коллекции.

Декларативные языки объектно-ориентированного моделирования, как правило, предоставляют некоторый набор абстрактных или конкретных типов коллекций с априори заданным набором семантических свойств. В производных пользовательских типах, определяемых на основе базовых, семантика коллекций может быть сохранена или уточнена. Однако выделенные фундаментальные свойства остаются принципиальными для содержательного анализа коллекций.

Так, декларативный язык ограничений OCL [17] определяет абстрактный интерфейс коллекций *Collection* и четыре конкретных класса *Bag*, *Set*, *Sequence* и *OrderedSet* для представления мультимножеств, множеств, последовательностей и упорядоченных множеств соответственно. Все виды коллекций задаются в обобщенном виде с возможностью параметризации типом элементов. *Set* — коллекция, по семантике соответствующая математическому понятию множества. Она не допускает дупликации элементов. *OrderedSet* — специализация данного типа для упорядоченных множеств. Ограничение уникальности необходимо поддерживается данным типом коллекции. *Bag* — мультимножество с возможным повторением элементов. *Sequence* — упорядоченное мультимножество или последовательность, допускающая повторение элементов. Таким образом, виды коллекций OCL могут быть классифицированы в соответствии с таблицей 1.

Язык моделирования EXPRESS [18] предоставляет иной набор типов коллекций, а именно: *Aggregate*, *Bag*, *Set*, *Array* и *List*. Абстрактный тип *Aggregate* определяет базовый набор методов оперирования с элементами коллекций. *Bag* — специализация данного типа для представления мультимножеств. *Set* — специализация типа *Aggregate* для произвольных множеств, исключающая дупликацию элементов и игнорирующая их порядок. Тип данных *List* применяется для представления последовательностей. Допустимое количество элементов в списках, множествах и мультимножествах задается дополнительными ограничениями. Коллекции имеют строго фиксированный размер в тех случаях, когда нижний и верхний пределы их размера совпадают. *Array* — специализация типа *Aggregate* для массивов фиксированной длины. С учетом индексации порядок элементов в

массивах имеет существенное значение. Возможна организация разреженных массивов с неустановленными значениями элементов при помощи специального спецификатора. Также возможно определение производных типов массивов и списков с наложенным ограничением уникальности элементов. Базовые типы коллекций, предоставляемые языком моделирования EXPRESS, приведены в таблице 1.

UNIQUE	ORDERED	SORTED	FIXED	Используемые сокращения	Коллекции OCL	Коллекции EXPRESS
-	-	-	-	BAG	BAG	BAG
+	-	-	-	SET	SET	SET
-	-	-	+	FIXED BAG		
+	-	-	+	FIXED SET		
-	+	-	-	LIST	SEQUENCE	LIST
+	+	-	-	ORDERED SET	ORDERED SET	UNIQUE LIST
-	+	-	+	ARRAY		ARRAY
+	+	-	+	UNIQUE ARRAY		UNIQUE ARRAY
-	+	+	-	SORTED LIST		
+	+	+	-	SORTED SET		
-	+	+	+	SORTED ARRAY		
+	+	+	+	SORTED UNIQUE ARRAY		

Таблица 1. Классификация базовых типов коллекций в языках моделирования EXPRESS и OCL

Базовые типы могут переопределяться пользователем с учетом семантики приложения путем задания дополнительных ограничений с использованием всего репертуара конструкций декларативных языков OCL и EXPRESS. В частности, может быть уточнено допустимое число элементов коллекции и способ их индексации, задан частичный или полный порядок на множестве элементов коллекции, определены свойства корреляции значений элементов и т.п.

Рассмотрим вопросы представления, вычисления изменений и исполнения соответствующих операций над коллекциями, следуя приведенной выше классификации в соответствии с выделенными семантическими свойствами.

4. Сравнение множеств

Начнем рассмотрение с наиболее простого типа коллекций $X \in set < T >$ — множества элементов типа *T*, предполагающего неявное задание и выполнение единственного семантического ограничения уникальности элементов $\forall x, y \in X \to x \neq y$. Дельта для двух версий множества $X, X' \in set < T >$ может быть естественным образом представлена в виде неупорядоченного набора операций добавления и удаления соответствующих элементов коллекции:

$$\Delta set < T > (X', X) = \{ins(x) \mid x \in (X' \setminus X)\} \cup \{del(x) \mid x \in (X \setminus X')\}$$

Корректное представление дельты $\Delta = \Delta set < T > (X', X)$ предполагает, что выполняется условие $\forall ins(x_1) \in \Delta \; \forall del(x_2) \in \Delta \to x_1 \neq x_2$, означающее, что один и тот же элемент не может одновременно участвовать в операции добавления и удаления. Более того, будем исключать повторение операций добавления и удаления с одним и тем же элементом, которое при исполнении операций противоречило бы определению множества: $\forall ins_1(x_1) \in \Delta \; \forall ins_2(x_2) \in \Delta \to x_1 \neq x_2$ и $\forall del_1(x_1) \in \Delta \; \forall del_2(x_2) \in \Delta \to x_1 \neq x_2$.

Применение операций, определяемых дельтой, довольно прозрачно. К заданному исходному множеству добавляются элементы, определяемые операциями *ins*, и удаляются элементы, определяемые операциями *del*. Тем самым, гарантируется тождественность условия $Apply(X, \Delta set < T > (X', X)) = X'$, в котором функция $Apply(X, \Delta)$ возвращает модифицированное представление коллекции, полученное в результате применения дельты к заданному представлению коллекции *X*.

Две конкурентные транзакции Δ', Δ'' могут оказаться конфликтными в случае $isConflict(ins'(x), del''(x)) = true$, когда в одной транзакции некоторый элемент добавляется в коллекцию, а в другой транзакции тот же самый элемент удаляется. Однако, если обе дельты вычислялись относительно общей версии коллекции в рамках распространенной схемы слияния “3-way Merge”: $\Delta' = \Delta set < T > (X', X)$, $\Delta'' = \Delta set < T > (X'', X)$, то подобные конфликты исключены в силу того, что удаляемый элемент обязан принадлежать базовой версии коллекции *X* и не может быть добавлен в нее повторно вследствие ограничения уникальности элементов множества. В случае иных схем слияния с участием нескольких базовых версий, например, схемы “4-way Merge”, подобные конфликты должны идентифицироваться и корректно разрешаться. Тривиальными способами разрешения являются следующие опции $Options = \{\varnothing; ins'(x); del''(x)\}$, предполагающие игнорирование обеих конфликтных операций или принятие одной из них.

Сложность вычисления дельты $\Delta set < T >$ определяется, прежде всего, способом представления исходных множеств (список, массив, сбалансированное дерево), а также алгоритмами поиска добавляемых и удаляемых элементов. Вычислительная сложность наивной реализации, основанной на простом поиске элемента в неупорядоченном списке и не требующей определения на элементах множеств полного порядка может быть оценена как $O(|X| \cdot |X'|)$. Сложность оптимальной реализации, основанной на предварительной сортировке исходных множеств, например, методом пирамидальной сортировки или методом слияния списков, можно оценить как $O(|X| \cdot \ln|X| + |X'| \cdot \ln|X'|)$. А в случае использования для работы с множеством сбалансированного дерева, хранящего элементы в уже отсортированном порядке, сложность операции построения дельты оценивается как $O(|X| + |X'|)$. Понятно, что последняя оценка не может рассматриваться как реальная оценка, поскольку в данном случае основная часть затрат перенесена на стадию формирования исходных множеств. Наиболее корректной оценкой в данном случае является также $O(|X| \cdot \ln|X| + |X'| \cdot \ln|X'|)$.

Ниже приведен пример сравнения двух версий множества натуральных чисел $X, X' \in set < N >$. Пусть X, X' — основная и модифицированная версии коллекции, содержащие следующие элементы: $X = \{1,2,3,6,8\}$, $X' = \{1,2,4,5,7,9\}$. Тогда дельта, вычисленная путем сравнения двух версий множества, представляется как
$\Delta(X', X) = \{ins(4), ins(5), ins(7), ins(9), del(3), del(6), del(8)\}$.

5. Сравнение мультимножеств

Перейдем к задаче сравнения мультимножеств $X \in multiset < T >$ с функцией $cardinality(x, X)$ для подсчета числа вхождений элемента $x \in T$ в коллекцию X. При модификации коллекции данного типа дельта представляется как неупорядоченный набор множественного добавления и удаления элементов. Пусть $X, X' \in multiset < T >$ — две версии мультимножества, тогда дельта может быть сформирована как

$$\Delta multiset\ < T > (X', X) = \begin{Bmatrix} card\ (x,n) \mid n = cardinalit\ y(x, X') - \\ cardinalit\ y(x, X), n \in Z, n \neq 0 \end{Bmatrix}$$

В используемых обозначениях Z — множество целых чисел. Положительное значение параметра n в операции изменения кардинальности *card(x,n)* указывает на добавление элемента в коллекцию соответствующее число раз, отрицательное значение — на удаление элемента из коллекции. Нулевое значение n не содержательно для представления дельты, поскольку означает, что количество экземпляров элемента в коллекции не изменилось. В

корректно сформированном представлении дельты $\Delta = \Delta multiset < T > (X', X)$ предполагается, что элемент мультимножества не может одновременно участвовать в нескольких операциях $\forall card_1(x_1, n_1) \in \Delta\ \forall card_2(x_2, n_2) \in \Delta \rightarrow x_1 \neq x_2$. В противном случае сравнение идентичных версий коллекции могло бы привести к неожиданному результату, отличному от пустого представления дельты, например: $\left\{card_i(x, n_i) \mid \sum n_i = 0\right\} \neq \varnothing$.

Применение базовой операции дельты *card(x,n)* состоит в кратном добавлении соответствующего элемента x при положительном значении параметра n и в его кратном удалении при отрицательном значении параметра. Это гарантирует выполнение необходимого тождества $Apply(X, \Delta multiset < T > (X', X)) = X'$.

Две операции $card'(x, n') \in \Delta'$, $card''(x, n'') \in \Delta''$ конфликтуют друг с другом, если параметры кратности перемещения экземпляров одного и того же элемента в конкурентных транзакциях отличаются друг от друга $n' \neq n''$. Тривиальные способы разрешения конфликта состоят в выборе одной из опций $Options = \{\varnothing; card'(x, n'); card''(x, n'')\}$. Вместе с тем, логичным представляется расширение возможных опций путем назначения параметру кратности всего интервала значений, порождающего конфликтную ситуацию: $Options = \{\varnothing; card(x, \min(n', n''));...; card(x, \max(n', n''))\}$. Это означает, что может быть выбрано любое число кратности для операции перемещения, лежащее в интервале между соответствующими параметрами конфликтных операций. В случаях, когда обе конфликтные операции являются родственными, конфликт целесообразно разрешать, исходя из промежуточных значений. В случаях, когда одна из конфликтных операций — операция добавления, а другая — операция удаления, допустимым является также весь диапазон значений параметра кратности, означая удаление или добавление числа элементов, не превышающего используемое значение соответствующей операции.

Сложность построения $\Delta multiset < T >$ определяется теми же факторами, что и сложность вычисления дельты обычного множества. Имеет место незначительное увеличение числа операций, однако это не влияет на асимптотическую оценку, которая в среднем выражается как $O(|X| \cdot \ln|X| + |X'| \cdot \ln|X'|)$.

В качестве примера рассмотрим процедуру сравнения двух версий мультимножества символов. Пусть $X, X' \in multiset < N >$ — основная и модифицированная версии коллекции, содержащие следующие натуральные элементы $X = \{g, a, b, c, c, f, b, c\}$, $X' = \{b, e, a, d, e, e\}$. Тогда дельта, вычисленная в результате сравнения двух версий коллекции, представляется

как
$\Delta(X', X) = \{card(b,-1), card(c,-3), card(d,1), card(e,3), card(f,-1), card(g,-1)\}$.

6. Сравнение списков

Для сравнения списков (или последовательностей) могут быть задействованы классические алгоритмы минимального редакторского расстояния (edit-distance (ed)) и алгоритмы нахождения наибольшей общей последовательности (longest-common-subsequence (lcs)), нашедшие применение в самых разных приложениях [20, 21]. Поскольку данные семейства алгоритмов достаточно хорошо проработаны и изучены, мы ограничимся вопросами их использования в общем контексте решения задач сравнения и слияния коллекций на основе семантики модели.

Пусть элементы списка $X \in list < T >$ предварительно последовательно пронумерованы, начиная с единицы, и каждый элемент $x(i) \in X, i = 1..|X|, x(i) \in T$ индексируется в соответствии с положением в списке. Далее $x(i)$ обозначается i-ый элемент коллекции, $|X|$ — число элементов коллекции и $x[i, j]$ — упорядоченное подмножество элементов коллекции X, начинающееся в позиции i и заканчивающееся в позиции $j \geq i$.

Тогда дельта, полученная в результате сравнения двух версий коллекции $X, X' \in list < T >$, может быть представлена множеством операций вставки новых элементов в соответствующие позиции исходного списка и удаления элементов с соответствующих позиций подобно тому, как это осуществляется в алгоритмах минимального редакторского расстояния:

$$\Delta list < T > (X', X) = \{ins(i, x'[k,l]) \mid i = 1..|X'| + 1, x'[k,l] \subseteq X'\}$$
$$\cup \{del(x[i, j]) \mid x[i, j] \subseteq X\} \cup$$
$$\{skip(x[i, j]) \mid x[i, j] \subseteq X, x'[k,l] \subseteq X', x[i, j] = x'[k,l]\}$$

Здесь операция $ins(i, x'[k,l])$ вставляет упорядоченный набор элементов модифицированного списка X' с индексами в отрезке $[k,l]$ в позицию i исходного списка X. Операция $del(x[i, j])$ удаляет элементы исходного списка с индексами, принадлежащими отрезку $[i, j]$, и, наконец, операция $skip(x[i, j])$ переносит элементы с индексами в отрезке $[i, j]$ в модифицируемый список без изменений. Последний тип операций избыточен при практической реализации, поскольку подмножество переносимых элементов может быть вычислено путем анализа интервалов индексов для удаляемых элементов. Тем не менее, здесь они используются с методической

целью. Все значения индексов позиций элементов рассчитываются относительно исходных версий коллекции.

Корректное представление дельты $\Delta = \Delta list < T > (X', X)$ предполагает, что выполняются следующие условия:

$$\forall ins_1(i_1, x'[k_1, l_1]) \in \Delta \; \forall ins_2(i_2, x'[k_2, l_2]) \in \Delta \rightarrow i_1 \neq i_2$$
$$\forall ins_1 \forall ins_2 \; ins_1(i_1, x'[k_1, l_1]), ins_2(i_2, x'[k_2, l_2]) \in \Delta list < T > (X', X) \rightarrow [k_1, l_1] \cap [k_2, l_2] = \varnothing$$
$$\forall del_1(x[i_1, j_1]) \in \Delta \; \forall del_2(x[i_2, j_2]) \in \Delta \rightarrow [i_1, j_1] \cap [i_2, j_2] = \varnothing$$

означающие, что индексы позиций вставки элементов не повторяются, а интервалы вставляемых и удаляемых элементов не пересекаются в разных операциях.

Будем считать также, что аналогичное условие выполняется для операций переноса элементов, индексы которых в исходном списке дополняют индексы удаляемых элементов:

$$\forall skip_1(x[i_1, j_1]) \in \Delta \; \forall skip_2(x[i_2, j_2]) \in \Delta \rightarrow [i_1, j_1] \cap [i_2, j_2] = \varnothing$$
$$\forall skip_1(x[i_1, j_1]) \in \Delta \; \forall del_2(x[i_2, j_2]) \in \Delta \rightarrow [i_1, j_1] \cap [i_2, j_2] = \varnothing$$
$$\forall i \in [1,|X|] \quad \exists del(x[i_1, j_1]) \in \Delta list < T > (X', X) \quad i \in [i_1, j_1] \wedge$$
$$\exists skip(x[i_1, j_1]) \in \Delta list < T > (X', X), \quad i \in [i_1, j_1]$$

Применение дельты предполагает предварительное упорядочение операций по индексам вставки и удаления элементов в исходной коллекции X таким образом, что при совпадении индексов операции вставки предшествуют соответствующим операциям удаления. Сами операции последовательно выполняются со значениями индексов, скорректированными с учетом предшествующих результатов.

При подобной интерпретации каждая операция вставки элементов $ins(i, x'[k,l])$ может рассматриваться в качестве композиции элементарных операций вставки отдельных элементов $ins(i, x'(k)) \angle ... \angle ins(i, x'(m)) \angle ... \angle ins(i, x'(l))$ с наложенными отношениями предшествования между ними. Используемый символ отношения $ins_1(i, x'_1) \angle ins_2(i, x'_2)$ означает, что операции $ins_1(i, x'_1)$, $ins_2(i, x'_2)$ должны выполняться таким образом, чтобы в результирующем представлении коллекции элемент x'_1 предшествовал элементу x'_2 при условии, что обе операции применяются. Последнее замечание существенно, поскольку одна из операций может быть не включена в результирующую транзакцию. Тем не менее, транзитивные отношения предшествования определяют частичный

порядок между операциями транзакции, который должен соблюдаться независимо от того, какие операции применяются, а какие — нет.

Таким образом, установленные отношения предшествования гарантируют, что элементы $x'[k,l]$ будут вставлены в результирующий список, не нарушая исходный порядок. Подобные отношения могут конструктивно использоваться при выполнении операций. Например, если операция реализуется как вставка в указанную позицию списка, то при условии $ins_1(i,x'_1) \angle ins_2(i,x'_2)$ операция $ins_2(i,x'_2)$ должна применяться до операции $ins_1(i,x'_1)$.

Две конкурентные операции с пересекающимися значениями интервалов индексов могут приводить к ситуациям, допускающим неоднозначное решение. Две операции удаления $del'(x[k',l']) \in \Delta'$ и $del''(x[k'',l'']) \in \Delta''$ с пересекающимися интервалами индексов $[k',l'] \cap [k'',l''] \neq \varnothing$ допускают консолидированное исполнение в виде $del(x[k',l'] \cup [k'',l''])$. Однако полный перечень опций применения определяется как множество всех простых сочетаний (сочетаний без повторений) операций удаления, включая пустое множество:
$Options = SimpleCombinations\{(del(x(\min(k',k'')),...,del(x(\max(l',l'')))\}$. Число возможных сочетаний может быть слишком велико для выбора необходимого варианта в ходе интерактивной сессии. Поэтому более естественным может оказаться представление конкурентных операций в более компактном виде с меньшим числом альтернатив выбора, а именно:

$$Options = SimpleCombinations\begin{Bmatrix}(del(x[k',l'] \cap [k'',l'']),\\ del(x[k',l'] \setminus (x[k',l'] \cap [k'',l''])),\\ del(x[k'',l''] \setminus (x[k',l'] \cap [k'',l'']))\end{Bmatrix}.$$

Отметим, что агрегированная операция удаления общих элементов $del(x[k',l'] \cap [k'',l''])$ может рассматриваться как консолидированное действие, не требующее в большинстве случаев дополнительного согласования. Альтернативы удаления $del(x[k',l'] \setminus (x[k',l'] \cap [k'',l'']))$ и $del(x[k'',l''] \setminus (x[k',l'] \cap [k'',l'']))$ дополняют общую операцию до соответствующих действий в каждой транзакции и могут приниматься в произвольном сочетании с двумя другими операциями.

Две конкурентные операции вставки и удаления элементов, пересекающиеся по индексам: $ins'(i',x'[k,l]) \in \Delta'$, $del''(x[i,j]) \in \Delta''$, где $i' \in [i,j]$, следует считать неконфликтными, поскольку операции удаления могут всегда быть корректно исполнены последовательно или вместе с соответствующими операциями вставки.

Для конфликтных операций вставки $ins'(i,x'[k',l']) \in \Delta'$ и $ins''(i,x''[k'',l'']) \in \Delta''$, добавляющих неэквивалентные списки элементов $x'[k',l'] \neq x''[k'',l'']$ в одну и ту же позицию i исходного списка, пользователь должен принять решение относительно способа формирования консолидированного списка вставляемых элементов. Потенциально, любое размещение элементов альтернативных списков может считаться допустимым при выполнении следующих двух условий: оригинальный порядок элементов не изменяется и исключается дублирование эквивалентных элементов из разных версий списка в смежных позициях результирующего списка. Таким образом, возможные варианты консолидации представляются следующим образом:

$$Options = Arrangements\begin{Bmatrix}ins'(i,x'(k')),...ins'(i,x'(l')),ins''(i,x''(k'')),...ins''(i,x''(l''))\\ |\ ins'(i,x'(k')) \angle ... \angle ins'(i,x'(l')),\\ |\ ins''(i,x''(k'')) \angle ... \angle ins''(i,x''(l'')),\\ |\ ins'(i,x'(m')) <> ins''(i,x''(m'')) \rightarrow x'(m') \neq x''(m'')\end{Bmatrix}$$

Первые два условия гарантируют, что исходный порядок элементов при консолидации не будет нарушен. Третье условие, связанное с непосредственным предшествованием операций $ins'(i,x'(m')) <> ins''(i,x''(m''))$ в итоговой транзакции, обеспечивает исключение тождественных элементов при вставке в соседние позиции из разных списков. Возможный способ реализовать подобную стратегию заключается в применении упомянутых выше методов сравнения к консолидируемым последовательностям.

Пусть вспомогательные списки $Y' \in list<T>$, $Y'' \in list<T>$ представляют собой соответствующие последовательности вставки элементов $Y' = x'[k',l']$ и $Y'' = x''[k'',l'']$. Тогда их дельта представима в виде:

$$\Delta list<T>(Y'',Y') = \{ins(i,y''[k,l]) \mid i = 1..|Y'|+1, y''[k,l] \subseteq Y''\} \cup$$
$$\{del(y'[i,j]) \mid y'[i,j] \subseteq Y'\} \cup$$
$$\{skip(y'[i,j]) \mid y'[i,j] \subseteq Y', y''[k,l] \subseteq Y'', y'[i,j] = y''[k,l]\}$$

Способы консолидации элементов из альтернативных списков задаются на основе $\Delta = \Delta list<T>(Y'',Y')$ следующими размещениями:

$$Options = Arrangements\begin{Bmatrix}ins(i,y'[k,l])\\ |\ \exists skip(y'[k,l]) \in \Delta \wedge \exists del(y'[k,l]) \in \Delta \wedge \exists ins(i,y''[k,l]) \in \Delta\\ |\ \forall ins_1(i_1,y'[k_1,l_1]) \angle ins_2(i_2,y'[k_2,l_2]) \rightarrow i_1 < i_2, l_1 < k_2\end{Bmatrix}$$

Размещения предполагают предшествование операций, определяемое естественным порядком позиций вставки и индексных интервалов в исходных операциях сформированной дельты. Тем самым, оперируя с альтернативными

последовательностями вставки и выделяя отличия между ними, удается определить конструктивный способ разрешения данного рода конфликтов.

Рассмотрим следующий пример. Пусть $X, X', X'' \in list < S >$ — основная и модифицированные версии некоторого списка символов: $X = \{a,b,b,c,c,d,e,h,j\}$, $X' = \{a,b,c,d,e,f,g,h,j\}$, $X'' = \{a,b,c,d,e,h,j,k,l\}$. Тогда дельты, вычисленные в результате сравнения соответствующих модифицированных версий с базовой, представляются как $\Delta'(X', X) = \{skip'(1,2), del'(3,4), skip'(5,7), ins'(8,[6,7]), skip'(8,9)\}$ и $\Delta''(X'', X) = \{skip''(1,2), del''(3,4), skip''(5,9), ins''(10,[8,9])\}$. В данном случае изменения не содержат конфликтов и могут быть консолидированы, приводя к результирующему представлению списка $X^* = \{a,b,c,d,e,f,g,h,j,k,l\}$.

Оценка вычислительной сложности классического алгоритма минимального редакторского расстояния с использованием метода динамического программирования составляет $O(|X| \cdot |X'|)$. Этой же оценкой определяется общая сложность формирования дельты для списков. При неоптимальном формировании дельты, например, путем нахождения наибольшей общей последовательности и определения дополняющих операций, оценка вычислительной сложности может быть улучшена до $O(|X| \cdot \ln|X| + |X'| \cdot \ln|X'|)$, однако количество элементарных операций в полученном представлении дельты может оказаться высоким. Более детальная систематизация алгоритмов и сравнительный анализ их вычислительной сложности приводятся в [20, 21].

7. Сравнение упорядоченных множеств

Упорядоченные множества являются одновременно специализацией и множеств, и списков, поэтому процедуры сравнения, рассмотренные выше, могли бы применяться и для этого случая. Однако в силу сочетания свойств упорядочения и уникальности элементов, видится более содержательный способ представления и расчета изменений для данного типа коллекций не только в виде операций вставки и удаления, но и операций перестановок.

Пусть $X, X' \in ordered\ set < T >$ — исходная и модифицированная версии упорядоченного множества. Подобно спискам предполагается, что элементы коллекции последовательно перенумерованы, начиная с единицы, и с каждым элементом ассоциирован соответствующий индекс позиции. Тогда дельта может быть представлена как множество операций циклических перестановок, вставок новых элементов и удалений существующих элементов:

$$\Delta ordered\ set < T > (X', X) = \{prm(i_1, i_2, ...i_n) | x(i_1), x(i_2), ..., x(i_n) \in (X \cap X')\} \cup$$
$$\{ins(i,[k,l]) \mid x[k,l] \in (X' \backslash X), i = 1..|X|+1\} \cup \{del([i,j]) | x[i,j] \in (X \setminus X')\}$$

Операция перестановки $prm(i_1, i_2, ...i_n)$ однократно циклически переставляет элементы исходного множества $x(i_1), x(i_2), ...x(i_n)$, приводя к следующему результату: $x(i_2), x(i_3), ...x(i_1)$. Операция $ins(i,[k,l])$ вставляет упорядоченный набор элементов модифицированного списка X' с индексами в отрезке $[k,l]$ в позицию i-ого элемента исходного множества X. Операция $del([i,j])$ удаляет элементы исходного множества X с индексами, принадлежащими отрезку $[i,j]$.

Определим более точно семантику операций, исходя из требований предшествования группы новых элементов элементу исходного множества, в позицию которого происходит вставка, сохранения порядка вставляемых элементов относительно друг друга в соответствии с их позициями, а также непрерывности их следования в результирующем представлении множества независимо от применения или неприменения всех других операций дельты.

Использование перестановок в представлении дельты упорядоченного множества позволяет изменить порядок элементов без их парных удалений и вставок, как это осуществлялось в случае списков. Более содержательный способ структуризации изменений, отражающий семантику данных, является принципиальным с учетом сложности приложений, оперирующих масштабными междисциплинарными информационными моделями.

Все значения индексов позиций указываются в представлении дельты относительно исходных версий коллекции. При выполнении дельты индексные параметры всех последующих операций должны корректироваться с учетом ранее примененных. Данная корректировка заключается в увеличении или уменьшении индексов элементов исходного множества на количество выполненных вставок или удалений. При выполнении перестановок корректировка состоит в циклическом распространении индекса для каждого последующего элемента группы перестановки.

Любое изменение порядка элементов в упорядоченном множестве может быть представлено композицией циклических перестановок. Причем при отсутствии пересечений по индексам перестановки удовлетворяют требованию коммутативности и могут применяться в произвольном порядке независимым друг от друга образом [19]. Тем самым удовлетворяется требование конструктивной декомпозиции и реконсиляции транзакций, связанное с возможностью независимого применения их отдельных операций. При этом наличие одного и того же индекса в разных группах перестановок дельты $\Delta = \Delta ordered\ set < T > (X', X)$ должно быть запрещено:

$$\forall prm_1(i_1, i_2, ..i_n) \in \Delta\ \forall prm_2(j_1, j_2, ..j_m) \in \Delta \rightarrow (i_1, i_2, ..i_n) \cap (j_1, j_2, ..j_m) = \varnothing$$

Данное условие не является обременительным, поскольку существует четкая методика разложения перестановки произвольного упорядоченного множества на группы циклических перестановок. Данная методика

приводится в [19] как доказательство теоремы о единственности специально заданного соединительного произведения перестановки линейно упорядоченного мультимножества.

Для корректного применения дельты $\Delta = \Delta\, ordered\ set < T > (X', X)$ операции могут быть частично упорядочены подобно тому, как это делалось для операций со списками. Предшествование операций циклической перестановки операциям вставки, а тех, в свою очередь, операциям удаления позволяет упростить реализацию применения дельты:

$\forall prm(i_1, i_2, ...i_n) \in \Delta\, \forall ins(i,[k,l]) \in \Delta\ \forall del([i,j]) \in \Delta \rightarrow$

$prm(i_1, i_2, ...i_n)\, \angle ins(i,[k,l]) \angle del([i,j])$

Данное требование связано с принятой семантикой операций, допускающей непосредственную индексацию элементов в исходных коллекциях. Вставка группы элементов неявно подразумевает задание ограничения предшествования добавляемых элементов элементу, в позицию которого осуществляется вставка. Однако следующая за вставкой операция перестановки может нарушить это условие. Для того чтобы удовлетворить условие и обеспечить неразрывность семантически связанных групп элементов, видятся два простых решения:

1. обобщение операции перестановки таким образом, чтобы обеспечить циклическую перестановку не отдельных элементов, а целых групп;
2. соблюдение условия предшествования операций перестановки операциям вставки.

На наш взгляд, второй способ более предпочтителен, поскольку контроль частичного порядка операций при выполнении не вызывает дополнительных сложностей в реализации в отличие от обобщенных перестановок.

Проиллюстрируем вычисление и применение дельты для упорядоченных множеств на следующем примере. Пусть $X, X' \in ordered\ set < S >$ — основная и модифицированная версии коллекции символов, представленные следующими последовательностями элементов: $X = \{a, b, c, d, e, f\}$, $X' = \{e, g, h, k, l, d, c, m, a\}$. Тогда дельта, вычисленная в соответствии с вышеописанной семантикой операций, представляется как $\Delta ordered set(X', X) = \{prm(1,5), prm(3,4), ins(4,[2,5]), ins(1,[8,8]), del([2,2]), del([6,6])\}$

В ходе применения дельты к основной версии X операции $prm(1,5), prm(3,4)$ переставляют элементы *a*, *e* и *c*, *d* исходного множества, приводя к промежуточным представлениям коллекции $\{e, b, c, d, a, f\}$ и $\{e, b, d, c, a, f\}$ соответственно. Операции $ins(4,[2,5]), ins(1,[8,8])$ добавляют элементы *g*, *h*, *k*, *l* перед *d* и элемент *m* перед *a*, формируя последовательности $\{e, b, g, h, k, l, d, c, a, f\}$ и $\{e, b, g, h, k, l, d, c, m, a, f\}$. Наконец, операции

$del([2,2])$, $del([6,6])$ удаляют элементы *b* и *f*, приводя к окончательному представлению модифицированной версии коллекции X'.

Опишем возможный способ формирования дельты двух упорядоченных множеств в соответствии с перечисленными выше условиями:

1. Поиск идентичных подмножеств $Y \subseteq X$ и $Y' \subseteq X'$ исходных множеств, $Y, X, Y', X' \in ordered\ set < T >$, сохраняющих оригинальный порядок следования элементов:

$$Y = \{y(i) \mid y(i) \in X \cap X', \forall y(i)\ \forall y(j)\ y(i) = x(k), y(j) = x(l), i < j \rightarrow k < l\},$$
$$Y' = \{y'(i) \mid y'(i) \in X \cap X', \forall y'(i)\ \forall y'(j)\ y'(i) = x'(k), y'(j) = x'(l), i < j \rightarrow k < l\}$$

   Возможная алгоритмическая реализация данного этапа состоит в предварительной сортировке элементов исходных множеств (при наличии отношения полного порядка) и последовательном просмотре и идентификации элементов как удаленных, вставленных или перенесенных без изменений. Альтернативный способ заключается в непосредственном использовании процедуры поиска для установления факта наличия или отсутствия элементов в исходных множествах и в параллельном формировании структур соответствия индексов элементов. Подобные структуры обеспечивают быстрое преобразование индексов, необходимое для эффективной реализации следующих этапов.
2. Поиск циклической перестановки элементов множества Y, приводящей к последовательности элементов множества Y'. Наиболее простым способом реализации данного этапа видится предварительная замена алфавита исходного множества Y на последовательность натуральных чисел $(1,2,...|Y|)$ и применение методики, применяемой в доказательстве теоремы о единственности специальной формы соединительного произведения перестановки линейно упорядоченного мультимножества [19].
3. Определение множества операторов вставки $\{ins(...)\}$, упорядоченного по индексам вставки элементов, используя структуры соответствия индексов элементов множеств X' и Y'.
4. Определение множества операторов удаления $\{del(...)\}$, упорядоченного по индексам удаляемых элементов, используя структуры соответствия индексов элементов множеств X и Y.
5. Формирование единого списка операций дельты в соответствии с принятым порядком исполнения: сначала следуют операции перестановок, затем — операции вставок и в конце — операции удаления.

Сформированная таким образом дельта состоит из множества независимых операций, каждая из которых может быть принята или отклонена в рамках результирующей транзакции независимо от статуса остальных операций.

Вычислительная сложность построения дельты определяется в первую очередь сложностью алгоритмов сортировки, применяемых в ходе реализации. Все остальные элементы, включая алгоритм разложения перестановки на множество циклических, имеют асимптотически линейную сложность при условии использования соответствующих структур быстрого преобразования индексов. Например, описанный метод формирования дельты с использованием сортировки на основе слияния списков имеет асимптотическую оценку вычислительной сложности $O(|X| \cdot \ln |X| + |X'| \cdot \ln |X'|)$.

Перечислим конфликтные ситуации, возникающие при реконсиляции конкурентных операций над упорядоченными множествами, а также возможные способы их разрешения. Основные конфликты сводятся к следующим содержательным случаям (опустим для краткости математическую нотацию, примеры и комментарии подобно тому, как это делалось в предыдущей главе):

1. Вставка тождественных элементов в разные позиции. Стандартным способом разрешения конфликта является принятие одной из операций или отмена обеих.
2. Вставка нетождественных последовательностей элементов в одну и ту же позицию. Варианты разрешения — те же самые, что и при сравнении списков.
3. Удаление элемента в одной транзакции при его перестановке в другой. Способ разрешения — стандартный.
4. Неэквивалентная перестановка одного и того же элемента в разных транзакциях. Принятие одной из циклических перестановок, в которых участвует элемент, является наиболее простым и очевидным способом разрешения подобного конфликта. Альтернативой ему может служить решение, основанное на декомпозиции конфликтных операций перестановки на элементарные транспозиции и выделение неэквивалентных цепочек транспозиций для заданного элемента. В этом случае разрешение конфликта сводится к отмене одной из конфликтных транспозиций и формированию итоговой консолидирующей перестановки.

Очевидно, что среда для согласования версий должна предусматривать возможность интерактивной работы для разрешения описанных видов конфликтов. При этом пользователю должны предлагаться уже сформированные, семантически корректные и наглядные варианты имеющихся альтернатив.

8. Сортированные последовательности

Наличие свойств сортировки для последовательностей элементов позволяет существенно ускорить процедуры сравнения коллекций $X, X' \in sorted\ list < T > \subset list < T >$ и применения соответствующих операций дельты $\Delta sorted\ list < T > (X', X)$. Способ представления дельты в этих случаях повторяет ранее описанный для произвольных списков, однако методы ее вычисления допускают оптимизацию с учетом свойств порядка. Вместо вычислительно сложных алгоритмов минимального редакторского расстояния и наибольшей общей последовательности может эффективно применяться алгоритм линейной сложности $O(|X| + |X'|)$, осуществляющий последовательный просмотр элементов версий коллекции в сортированном порядке и фиксирующий изменения сразу по ходу их просмотра. Аналогичным образом может быть оптимизирована процедура применения операций дельты, допускающая эффективный поиск элементов по индексам.

9. Последовательности фиксированной длины

Наличие у последовательности элементов статически фиксированной длины вносит коррективы в способ представления дельты, наиболее адекватно отражающий ее семантику. Коллекции данного типа будем называть статическими массивами. Для подобных случаев $X, X' \in array < T > \subset list < T >$ будем использовать следующее представление дельты:

$$\Delta array < T > (X', X) = \{alt(i, x'[i]) \mid i = 1..|X|, x'[i] \in X'\},$$

фиксирующее индексы измененных элементов. Здесь операция $alt(i, x'[i])$ заменяет значение элемента исходного массива X с индексом i значением соответствующего элемента модифицируемого массива X'.

Корректное представление дельты $\Delta = \Delta array < T > (X', X)$ предполагает, что индексы модифицируемых элементов не повторяются в разных операциях:

$$\forall alt_1(i_1, x'[i_1]) \in \Delta\ \forall alt_2(i_2, x'[i_2]) \in \Delta \rightarrow i_1 \neq i_2$$

Две конкурентные операции $alt'(i', x'[i']) \in \Delta'$ и $alt''(i'', x''[i'']) \in \Delta''$ в соответствующих транзакциях $\Delta' = \Delta array < T > (X', X)$ и $\Delta'' = \Delta array < T > (X'', X)$ оказываются конфликтными в тех случаях, когда присваивают разные значения элементу с одним и тем же индексом: $i' = i''$, $x'[i'] \neq x''[i'']$, $x'[i'] \subseteq X'$, $x''[i''] \subseteq X''$, $i', i'' = 1..|X|$. Способ разрешения конфликтов подобного рода тривиален и состоит в игнорировании обеих конкурентных операций или принятии одной из них: $Options = \{\varnothing; alt'(i, x'[i]); alt''(i, x''[i])\}$. В частном случае $x'[i] = x''[i]$ операции эквивалентны и не конфликтуют друг с другом.

Рассмотрим следующий пример согласования изменений в массиве. Пусть $X, X', X'' \in array < S >$ — базовая и модифицированные версии массива символов: $X = \{a,b,b,d,d\}$, $X' = \{a,b,c,h,i\}$, $X'' = \{a,b,c,d,e\}$. Тогда соответствующие дельты представляются как $\Delta'(X', X) = \{alt'(3,c), alt'(4,h), alt'(5,i)\}$, $\Delta''(X'', X) = \{alt''(3,c), alt''(5,e)\}$. В данном случае операции $alt'(3,c) \in \Delta'$ и $alt''(3,c) \in \Delta''$ эквивалентны и, следовательно, в результат включается одна из них. Операция $alt'(3,c) \in \Delta'$ переносится без изменений. Операции $alt'(5,i) \in \Delta'$ и $alt''(5,e) \in \Delta''$ конфликтны, поэтому лишь одна из них может быть включена в результирующую дельту. В случае принятия операции второй транзакции дельта приобретает вид $\Delta^* = \{alt'(3,c), alt'(4,h), alt''(5,e)\}$, а итоговый массив — $X^* = \{a,b,c,h,e\}$.

10. Коллекции с ограниченной мощностью

Наконец, особым образом должны реализовываться процедуры применения дельты для коллекций с ограниченной мощностью. В предположении, что исходная и модифицируемая версия коллекции были корректны и удовлетворяли необходимым семантическим ограничениям, частичное применение операций дельты также должно удовлетворять наложенным ограничениям мощности (размера коллекции). Способы представления и вычисления дельты при этом не меняются и определяются иными семантическими свойствами (см. предыдущие разделы), однако применение отдельных операций удаления и добавления элементов должно проводиться в соответствии с дополнительными логическими отношениями, индуцируемыми соответствующими ограничениями.

Пусть задано следующее ограничение мощности коллекции: $|X| \in [p,q]$, где $p,q \in N$. Частный случай $p = q$ соответствует коллекции фиксированной мощности. Если $\Delta_{ins} \subset \Delta$ и $\Delta_{del} \subset \Delta$ — соответствующие подмножества операций вставки и удаления исходного представления дельты $\Delta = \Delta collection < T > (X', X)$, то должно выполняться следующее дополнительное условие: $p \le |X| + |\Delta_{ins}| - |\Delta_{del}| \le q$.

Очевидно, что в случае фиксированной мощности $p = q$ количество операций вставки и удаления в транзакции должно совпадать. Для мультимножеств условие представления дельты $\Delta multiset < T > (X', X) = \{card_i(x_i, n_i)\}$ приобретает вид $p \le |X| + \sum n_i \le q$.

В случае конкурентных транзакций Δ', Δ'' приведенные выше условия должны выполняться для консолидированной дельты $\Delta^* = Merge(\Delta', \Delta'')$ и итогового представления коллекции $X^* = Apply(X, \Delta^*)$. В случае $|X^*| < p$ конфликт вызывает преобладание операций удаления над операциями вставки, в случае $|X^*| > q$ — преобладание операций вставки. Логичным способом разрешения подобных конфликтов является исключение такого количества преобладающих операций, чтобы мощность итоговой коллекции удовлетворяла наложенному ограничению: $|X^*| \in [p,q]$. Очевидно, что при выборе исключаемых операций следует учитывать и другие ограничения, наложенные на коллекцию. Например, в случае ограничения уникальности операции вставки и удаления элемента с одним и тем же значением должны быть включены или исключены совместно.

Рассмотрим следующий пример. Пусть $X, X', X'' \in multiset < S >$ — базовая и модифицированные версии мультимножества символов с ограниченной мощностью $|X| \in [2,6]$: $X = \{a,b,c\}$, $X' = \{a,b,d,e,e\}$, $X'' = \{a,b,e,e,e,e\}$. Тогда дельты, вычисленные путем сравнения модифицированных версий с базовой, представляются как $\Delta'(X', X) = \{card'(c,-1), card'(d,1), card'(e,2)\}$, $\Delta''(X'', X) = \{card''(c,-1), card''(e,4)\}$. Операции $card'(c,-1)$ и $card''(c,-1)$ эквивалентны, поэтому в результирующую дельту следует включить любую из них. Операция $card'(d,1)$ не конфликтует ни с одной другой операцией, поэтому переносится в результат без изменений. Наконец, операции $card'(e,2)$ и $card''(e,4)$ конфликтуют друг с другом. Согласно рассмотренным выше методам согласования изменений для мультимножеств, конфликт можно разрешить выбором кратности вхождения элемента *e* из интервала [2, 4]. Однако выбор значения кратности, равного 4, приводит к нарушению ограничения мощности результирующего мультимножества, в которое в таком случае войдет 7 элементов. Следовательно, допустимыми значениями кратности вхождения элемента *e* в итоговую коллекцию являются 2 и 3. В случае принятия второго значения результирующая дельта приобретает вид: $\Delta^* = \{card'(c,-1), card'(d,1), card(e,3)\}$, а итоговое семантически корректное представление мультимножества — $X^* = \{a,b,d,e,e,e\}$.

11. Коллекции прямых и инверсных ассоциаций

Коллекции могут использоваться для реализации множественных ассоциативных связей между объектными типами. В языках объектно-ориентированного моделирования имеется возможность для каждой прямой ассоциации объявить соответствующую инверсную, которая представляется, как правило, неупорядоченным множеством или мультимножеством

объектных ссылок и налагает дополнительные семантические ограничения (уникальности или мощности множественной ассоциации) на исходную модель.

Пусть C_1 и C_2 — объектные типы, $X \in collection < C_2 >$ — прямая множественная ассоциация, $Y \in collection < C_1 >$ — соответствующая ей инверсная. Будем считать, что в качестве прямой ассоциации может быть использована произвольная коллекция, в качестве инверсной — *set* или *multiset*. Поскольку модификация прямой ассоциации подразумевает симметричную коррекцию инверсной, операции установления и отмены ассоциативных отношений связаны логической эквивалентностью следующим образом: $ins(x) \sim ins(y)$, $del(x) \sim del(y)$, $x \in X$, $y \in Y$. Поэтому данные операции обязаны совместно участвовать в итоговой транзакции.

Если инверсная ассоциация представляется множеством, то дополнительно устанавливается ограничение уникальности инверсного отношения. При сочетании в качестве прямой и инверсных ассоциаций различных коллекций более строгое ограничение уникальности в итоге распространится на обе ассоциативные связи. Таким образом, возможны следующие варианты сочетания прямых и инверсных коллекций: «*set–set*», «*multiset–multiset*», «*list–multiset*», «*ordered set–set*».

Нарушение ограничения уникальности прямой ассоциации автоматически приводит к аналогичному нарушению на стороне инверсной. Поэтому наличие инверсного ассоциативного отношения с уникальными элементами не вносит дополнительных корректив в способы представления и формирования дельты, а также в методы разрешения конфликтов, описанные в предыдущих разделах.

Более интересным с этой точки зрения представляются ограничения мощности множественной ассоциации $| X | \in [m, n]$, $| Y | \in [p, q]$, $m, n, p, q \in N$.

В данном случае корректное представление дельты предполагает выполнение следующих условий:

$$m \le | X | + | \Delta_{ins(x)} | - | \Delta_{del(x)} | \le n$$

$$p \le | Y | + | \Delta_{ins(y)} | - | \Delta_{del(y)} | \le q$$

В силу отношений логической эквивалентности операций над прямыми и обратными ассоциациями, условия приобретают вид:

$$\max(m - | X |, p - | Y |) \le | \Delta_{ins(x)} | - | \Delta_{del(x)} | \le \min(n - | X |, q - | Y |)$$

В случае конкурентных транзакций Δ', Δ'' данные условия должны выполняться также для консолидированной дельты $\Delta^* = Merge(\Delta', \Delta'')$.

12. Заключение

Таким образом, рассмотрены основные типы коллекций, поддерживаемые популярными языками объектно-ориентированного моделирования. Для них определены способы представления, журнализации, вычисления, принятия и согласования изменений. Для каждого выделенного типа дается строгая, семантически состоятельная интерпретация конфликтов и предлагается конструктивный метод их идентификации и разрешения. Результаты предполагается использовать при создании универсальной, основанной на модельном представлении среды коллективной инженерии с развитыми возможностями семантически корректной и функционально содержательной реконсиляции дивергентных реплик данных.

Литература

[1] Y. Saito, M. Shapiro. Optimistic Replication // In ACM Computing Surveys, Vol. 37, No. 1, March 2005, pp. 42–81.

[2] Better SCM Initiative: Version Control System Comparison, http://better-scm.berlios.de/comparison/comparison.html

[3] Diffutils — GNU Project — Free Software Foundation (FSF), http://www.gnu.org/software/diffutils/diffutils.html

[4] Z. Xing, E. Stroulia. UMLDiff: an algorithm for object-oriented design differences. // Proceedings of the 20th IEEE/ACM international Conference on Automated software engineering, Long Beach, CA, USA, 2005, pp. 54–65.

[5] Open Testware Reviews — Data Comparator Survey, http://tejasconsulting.com/open-testware/feature/data-comparator-survey.html

[6] Comprehensive List of File and Folder Comparison and Synchronization Tools, http://www.foldermatch.com/fmcompetitors.htm

[7] Сравнение файлов — Soft Софт каталог, http://www.softsoft.ru/search/19709/index.htm

[8] Google directory — Computers > Software > File Management > File Comparison, http://www.google.com/Top/Computers/Software/File_Management/File_Comparison

[9] Semenov V.A., Karaulov A.A. Semantic-Based Decomposition of Long-Lived Transactions in Advanced Collaborative Environments. // Proceedings of 6 European Conference on product and process modeling, ECPPM 2006, Spain, Valencia, September 11–15, 2006, pp.223–232.

[10] Семенов В.А., Ерошкин С.Г., Караулов А.А., Энкович И.В. Семантическая реконсиляция прикладных данных на основе моделей. // Труды Института системного программирования: т. 13, ч. 2. / Под ред. В.П. Иванникова — М.: ИСП РАН, 2007, с. 141–164.

[11] Semenov V.A. Collaborative Software Engineering Using Metamodel-Driven Approach. // Proceedings 16th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises, WET ICE 2007, IEEE Computer Society Conference Publishing Services, 2007, pp. 178–179.

[12] Semenov V.A. Semantics-Based Reconciliation of Divergent Replicas in Advanced Concurrent Engineering Environments. // Complex Systems Concurrent Engineering: Collaboration, Technology Innovation and Sustainability, Springer-Verlag, 2007, pp. 557–564.

[13] ISO 10303: 1994, Industrial automation systems and integration — Product data representation and exchange.
[14] OMG. Model Driven Architecture: How systems will be built, http://www.omg.org/mda.
[15] T. Lindholm. XML three-way merge as a reconciliation engine for mobile data. // Proceedings of the 3d ACM international workshop on data engineering for wireless and mobile access, San Diego, CA, USA, 2003, pp. 93–97.
[16] J. Katajainen and J. L. Träff. A Meticulous Analysis of Mergesort Programs. // Lecture Notes In Computer Science, vol. 1203, 1997, pp. 217–228.
[17] Object Constraint Language Specification, Version 2.0, http://www.omg.org/technology/documents/formal/ocl.htm
[18] ISO 10303-11: 2004, Industrial automation systems and integration — Product data representation and exchange — Part 11: Description methods: The EXPRESS language reference manual. Edition 2.
[19] Д. Кнут. Искусство программирования, том 3. Сортировка и поиск, 2-е изд. — М.: Издательский дом «Вильямс», 2000.
[20] Д. Гасфилд. Строки, деревья и последовательности в алгоритмах: Информатика и вычислительная биология. — СПб.: Невский Диалект; БХВ-Петербург, 2003.
[21] G. Navarro. A Guided Tour to Approximate String Matching. // ACM Computing Surveys, vol. 33, no. 1, March 2001, pp. 31–88.