Kivonat

Napjainkban a szoftverfejlesztés terén egyre nagyobb szerepet kap a modellvezérelt tervezés. Ennek a megközelítésnek nagy előnye, hogy a modellen automatikusan különféle tesztek és ellenőrzések sokaságát hajthatjuk végre automatikusan, még a kész szoftver előállítása előtt. Ezzel a célberendezésbe vagy végfelhasználóhoz kerülő szoftver megbízhatóságát, karbantarthatóságát, továbbfejleszthetőségét növelhetjük. A modellvezérelt fejlesztési folyamatok gyakorlati megvalósítása során gyakran előkerülő probléma, hogy a modellek felett bizonyos feltételeket szeretnénk megfogalmazni, deklaratív megközelítéssel lekérdezéseket futtatnánk a választott modellen. Erre az Eclipse EMF modellező rendszerében megjelenő igényre számos eszköz jött létre, többek között az Eclipse OCL [8] és a Méréstechnika és Információs Rendszerek tanszéken fejlesztett és az ipar által is használt Emf-IncQuery keretrendszer [18].

A lekérdezéseket megfogalmazhatjuk a gráf felett gráfmintákként is, melyek deklaratívan megfogalmazható lekérdezési szabályokkal specifikálják a lekérdezést. Ezeknek a modellben történő feltérképezése során számításigényes feladat az adott gráfminták illeszkedéseinek megkeresése a modellben. A mintaillesztésre többféle (különböző helyzetekben gyors és hatékony) megközelítés létezhet.

Az egyik lehetséges megközelítés az inkrementális illesztés. Alapötlete abban rejlik, hogy mikor egy mintának az összes illeszkedését kigyűjtötte az eljárás, eltárolja ezeket, hogy a későbbi lekérdezések során rendkívül gyorsan visszaadhassa azt. Ezzel jelentős gyorsulás érhető el ismételt lekérdezéskor, ellenben sok esetben nehézségeket, kihívásokat tartogathat. Egyfelől az illeszkedéshalmazok számosságuknál fogva jelentős memóriaigénnyel jelentkezhetnek. További probléma, hogy a modell módosulása miatt az illeszkedéshalmazokat frissíteni kell. Ehhez egyedi megoldás kell, valamint értesülni kell a modell változásairól. Az Emf-IncQuery keretrendszerben már létezik egy ilyen inkrementális illesztésen alapuló algoritmus, a témakörben népszerű Rete hálókon alapuló eljárás. Ennek sajátossága, hogy az illeszkedéshalmazokon kívül további információkat is eltárol a modell változása esetén bekövetkező frissítés gyorsítására. Ennek azonban következménye a megnövekedett memóriafogyasztás.

Egy másik lehetséges eljárás a tanulmányaim során általam megvizsgált Treat algoritmus [14]. Inkrementális tulajdonságán felül alapötlete az, hogy csak a lekérdezés végeredményét, az illeszkedéshalmazokat tárolja, így kisebb a memóriaigénye, viszont a frissítés művelete lassabb, összetettebb is lehet. Emiatt viszont szükség van egy olyan betétalgoritmusra, mely gyorsan és hatékonyan segít megválaszolni a megváltozott modell felett

a frissítés kérdését. Választásunk a korábbi félévek során a VIATRA2 VPM [19], majd az EMF modellreprezentációk felett általam megvalósított, tesztelt és mért keresőalgoritmusra esett esett, az előretekintő mintaillesztésre [16]. Az eljárás fő előnye, hogy más mélységi keresőalgoritmusokhoz képest hatékonyabban illeszt többféle helyzetben is: ha nagyobb a minta, csak egyetlen illesztésre van szükség, vagy a mintának egy része már illeszkedik. Utóbbi különösen előnyös a frissítés során.

Diplomám keretein belül megvalósítottam az előretekintő keresést az EMF-INCQUERY keretrendszerben, továbbá elkészítettem a TREAT alapú inkrementális illesztő logikát. A TREAT eljárás aktívan használja és támaszkodik a (keresés típusú) előretekintő mintaillesztőre, a kezdeti illesztéshez és a frissítéshez is ezt az algoritmust használja. Külön kihívást jelentett, hogy a minták egymásra hivatkozása a frissítést egy nagy kihívást jelentő, érdekes feladattá teszi. Az elért eredményeimet a Méréstechnika és Információs Rendszerek tanszéken fejlesztett és alkalmazott TrainBenchmark nevű tesztkörnyezetbe is integráltam. Végül összemértem a TREAT eljárás teljesítményét és memóriafogyasztását a jelenleg használt inkrementális, és kereső alapú megoldásokkal.

Abstract

Nowadays model driven development gains wider and wider adoption in software engineering. The powerful advantage of this paradigm is that we can automatically perform several kinds of analysis before implementing the final software product. Thus the quality, reliability, improvability and maintainability of the delivered product can be significantly improved. A frequently occuring challenge in model driven development is that the user requires extra conditions, well-formedness rules or custom model queries against models. Based on these challenges, many industrial and academic tools have arisen till today, including Eclipse OCL [8] and the EMF-IncQuery model query evaluation framework [18] developed at Department of Measurement and Information Systems, which has been applied in various research and public projects.

Model queries can be interpreted as declarative graph patterns specifying the model query. Evaluating these queries can be performance-critical, including computation performance and significant memory consumption. Many different methods are developed and many solutions exist to answer these challenges.

One possible solution to evaluate queries efficiently is the incremental approach. Its main idea is based on the cache of the query result. When a new execution arrives, the query result cache can be immediately returned. However, more problems appear: result caches can consume significant memory and changes of the model must be carefully examined and the query result sets must be updated. This requires, that the incremental solution is able to monitor the model changes and must implement the update process. There is an implementation currently in the EMF-IncQuery system, the popular and widely used Rete algorithm, which is based on the Rete network. This method does not only cache the results, also stores additional partial information about the queries to speed up the update process, when the model changes. However, this means even more memory consumption, which can be significant against bigger models.

Another possible solution is the Treat algorithm I examined during my studies. On top of its incremental nature, its base idea is to store only the final results to consume less memory. In contradiction, the performance of the update process might be worse and more complicated and challenging to solve. The approach needs an internal search algorithm to search for the results of the query, and quickly identify the changes happening to them, after the model is modified. Our choice was the Lookahead algorithm implemented by me earlier [16]. The Lookahead was written and tested over Viatra2's VPM model representation [19] and later over EMF models. The main advantage of the Lookahead

algorithm is that the query evaluation can be more efficient in many scenarios against other search algorithms. For example, when the query is bigger, or we need only one result (not the whole result set), and an another case is when part of the result is already known. The last one is essential when updating the result sets as part of the reaction of the model changes.

In my thesis I implemented the Lookahead search algorithm in the EMF-INCQUERY framework, and designed and implemented the incremental TREAT query evaluation approach. The TREAT actively relies on the Lookahead algorithm using it for inital evaluation and maintenance of the stored result sets. Other challenges were the query compositions, where different queries can refer to each other and updating the result sets requires attention and careful design. The results achieved were implemented into the TrainBenchmark benchmarking system developed also at Department of Measurement and Information Systems. After these steps I measured the performance and memory consumption of the TREAT algorithm against the other incremental and search based solutions.