Designing and synthesizing the synchronization of concurrent object-oriented systems

Theses of the Ph.D. Thesis

Szabolcs Hajdara

Supervisor:
László Kozma, C.Sc.

Ph.D. Program in Informatics
Ph.D. School of Informatics
Prof. János Demetrovics

Eötvös Loránd University
Faculty of Informatics
Department of Software Technology and Methodology

Budapest, 2007

Supported by GVOP-3.2.2-2004-07-005/3.0.

1 Introduction

Object-oriented programs [21], [23], etc. have several advantages, they can be maintained effectively and are easily reusable. The great improvement of the hardware resulted that software run in a hardware environment with more processors in more and more cases, and in many cases, connected computers are used. So it is very important to design the parallel working of – maybe distributed [19], [25], [26] – objects. One of the most important tasks is to design and implement [16], [18], [20], etc. the synchronization of parallel objects.

The synchronization code of programs can usually be separated from the computation code, so assuming that, we consider only the synchronization code of programs. The use of concurrently running objects which communicate with each other causes that making the synchronization code is more difficult and complex, and because of the complexity, it is hard to maintain.

The goal of this thesis is to develop a method which automatically generates the synchronization code of object-oriented programs, so that the synchronization is correct in the sense of a specification. Furthermore, we provide tools to connect the computation code to the synchronization code so that developing parallel object-oriented software can become safer and faster. We aspire to expand the group of the problems that can be handled with our method and to grow the usability of the method. We implement the synthesized abstract programs, so that they can be maintained easily, and we try to keep the simplicity of the generated code, furthermore, we give a method to improve their effectiveness. We examine, how the structure of the synchronization code and the computation code can be separated, and how the reusing of the synchronization code is possible. We consider only class based languages in the thesis.

I have done my researches together with Balázs Ugron, we have published our results together. After labouring the bases, Balázs Ugron continued his work with pipeline systems [29], while I continued my work in the area of the object-oriented parallel systems.

2 Bases

One of the methods of creating proven correct synchronization programs is the synthesis method [14], [17], which ensures that the synthesized program is correct in the sense of the specification. For synthesizing programs, we need a specification language and an algorithm, which generates the program based on the specification. We chose MPCTL* (Many-Processes Computational Tree Logic*) for specifying the synchronization, which is a special branching time temporal logic language [17].

We extended the method of Attie and Emerson [14] for synthesizing the synchronization code of programs. This method, the Many-Processes synthesis, is based on the method of Emerson and Clarke [17], which is a tableau based algorithm. The Many-Processes synthesis solves the state explosion problem by introducing the constraint of the similarity of processes.

We solved the problem of the automatic generation of the synchronization code in object-oriented environment. Because of this, we need to design some structural and logical behaviour of the system, so we chose UML (Unified Modeling Language) [27], [28] to do this.

For implementing abstract programs, we used a widespread tool of parallel programming, the semaphores [30], the idea of the locking via token capturing [24] and the bases of the "pass the baton" technique [13], which can be used for evaluating conditions in mutual exclusive mode.

We implemented our examples in Java [22], [25] by using the standard tools of the language.

3 The structure of the thesis

After considering the bases, we started the thesis with examining the synchronization connections of objects, and in Section 3.2.1, we designed the structure of the handling of the synchronization connections, which is our *new result.* In Section 3.2.2, we give a method for specifying the synchronization connections, which *was developed by us.* In Section 3.2.3, we show a method, which *was developed by us*, for implementing the interconnection re-

lation. In Section 3.2.4, *we defined* the class structure of the synchronization parts of objects, which contains classes that can be automatically generated by using our method. In Section 3.2.5, we introduce new constraints, which make the specification more handleable. In Section 3.2.6, we *introduce an algorithm* for implementing the abstract synchronization code. In Section 3.3, we give a *new approach* of the similarity of objects and we give a method for separating the structure of the synchronization and the computation, so the number of the states can be handled more effective. Based on this, we give a method for synthesizing the synchronization code of a system that consists of objects of several classes. In Section 3.4, we show a partial solution for a special case of state the partitioning inheritance anomaly, and this is our *new result*, furthermore, we apply the possibilities of the inheritance *in a new way*, for defining the dynamic changes of the synchronization behaviour. In Section 3.5, we give a method, which *was developed by us*, for specifying the synchronization of shared classes by tracing them back to the synchronization specification of parallel object-oriented systems. In Section 3.6, we consider the possibilities of connecting the synchronization code to the computation code by using extended state diagrams, which is our *new result*. In Section 4.1, we show an extension of the spatial operators and the interconnection relation for increasing the number of the solvable problems, and this is our *new result*. Section 4.2 shows an extension of MPCTL* language by *introducing* graphical elements. In Section 4.3, we consider the effectiveness of the generated code, and we *introduce methods* to improve it. The thesis contains the examination of the related works, followed by a summary and conclusion, and finally we consider our future works.

4 Theses

4.1 Synchronization connections of objects

By using the method of Attie and Emerson [14], we can define the synchronization of connected similar processes. In the case of object-oriented systems, we have to take care of the fact that objects can be created and de-

stroyed dynamically, therefore, we have to implement the dynamic creation and deletion of the connections of objects.

The method of Attie and Emerson uses an interconnection relation I [14], which contains the index pairs of the connected objects. We replaced the index pairs to pairs of object pointers, so we can easily implement the connections. We designed the structure of the classes that implement the maintenance of I, by using UML [27], [28]. The handling of I is implemented in a central class, in *SharedObject*, by using static methods, or in the case of languages that do not support static methods, a globally used instance can be applied, e.g. by using the pattern *Singleton* [15].

We gave a method for specifying the interconnection relation. We used the spatial operators of MPCTL* language [14] for defining the specification method of the interconnection relation, and we extended them with two type-parameters, so we can formulate the requirements of the connections based on the types. (The implementation of the specification also can be handled in a central way.)

The implementation of the interconnection relation is based on the implementation of the first-order logic formulae of the specification, by taking into consideration that all possible object-pairs have to be examined because of the spatial operators. Handling of quantifiers may lead to performance problems, so we have optimized the reaching of I. The main part of the program code that is needed to handle I can be implemented as constant code, so we can give an efficient implementation of it (because we can specialize it). The safe behaviour of the generated code that handles I is ensured by using the semaphore based technique that was introduced in [21]. We published an earlier method for handling the interconnection relation in [1].

Thesis 1 *We introduced a method for specifying the interconnection relation of objects, we gave the structure and the implementation of the program that can handle I safely, furthermore, we defined the steps of the implementation of the function that can decide the object connections.*

4.2 Specifying the synchronization

The behaviour of the connected objects has to be defined. In order that we can implement the specified behaviour, we designed the structure of the synchronization code. The objects have to gain information from the connected objects to decide, whether their work can be continued, so we use interfaces in the implementation.

If more classes are used, we have to join the state sets of the classes to make the objects similar from the point of view of the synchronization, so the method of Attie and Emerson [14] can be used. This increases the states of the system, but only at the level of classes and it is independent of the number of objects. The number of the states can be decreased by using two techniques. We can merge states of the large state set. Unfortunately, the state merging has to be done mainly manually, we can merge states automatically only in special cases (e.g. merging critical states with the same semantic). The other technique is to separate the structure of the synchronization code. This is based on the fact, that classes with different computation codes can have the same synchronization code. This decreases the number of the synchronization classes, so as the number of the states. By using the possibility of the inheritance, objects can change their synchronization behaviours by connecting themselves into the different points of the synchronization class hierarchy. For specifying the synchronization of the system, MPCTL* language can be used over the joint state set. We published our earlier results of specifying and implementing the synchronization code of classes in [1], [2] and [5].

By using parallel object-oriented programs, it can happen that the synchronization code of a class has to be modified, if an other class extends it. Using the state mapping technique, we can give a partial solution for this problem in that cases, when the new states of the descendant can be handled similarly to some states of the ascendant. In these cases, objects of the descendant can set a class level mapping structure of the ascendant, which effects that the synchronization code of the ascendant does not need to be redefined. This mapping can be defined at UML level, in the class diagram,

by using stereotypes.

Shared classes are special cases of shared resources. The designing of the behaviour of objects of the shared classes can be traced back to the synchronization specification of parallel object-oriented systems.

The connection of the synchronization and the computation class hierarchy can be defined in the state diagrams of the computation classes, construction of which was extended in order that the connection could be defined easily. An earlier version of this method is written in [11].

We use the method of Attie and Emersof for synthesizing the synchronization skeleton. To implement this abstract program (which is an automaton) we have to ensure, that the condition evaluation of the abstract program is done in mutual exclusive mode, so we build our algorithm on the concept of the token capturing [24]. As token capturing is used for handling the connected objects, its algorithm is implemented in class *SharedObject*. The synchronization code of the synchronization classes is implemented in a function *setState*, and the implementation is based on the automaton. An earlier version of implementing the synchronization code can be found in [1].

Thesis 2 *We provide a solution for designing the structure of object-oriented systems that consists of more classes, by separating the synchronization structure, so the synchronization of the system can be defined easily, and safe synchronization code can be generated automatically. We define the method of connecting the synchronization code to the computation code. Furthermore, we define a possible method for specifying the synchronization code of shared classes.*

4.3 Extensions

We introduce strong spatial operators to extend our method for more complex problems. The implementation of the behaviour that was defined by using strong spatial operators may block the running of the synchronization code of an object if the object has no connected pairs, with which it could communicate. By using strong spatial operators, we can describe the behaviour of objects that serve several other objects. Furthermore, by modifying the

interconnection relation to be antisymmetric, we can define the order of the data processing. Strong operators were introduced in [3], by using an other approach.

For improving the usability of the method, we introduced the synchronization diagrams, which can be an extension to UML. By using synchronization diagrams, graphical elements can describe the MPCTL* specification, so it is easy to use. We designed synchronization diagrams, so that the important feature that can be formulated in MPCTL* can be expressed by using synchronization diagrams. The synchronization diagrams define abstract elements, which can make the steps of the design faster, and the resulted design of synchronization more simple. An earlier version of the synchronization diagrams was introduced in [11].

Developers can increase the effectiveness of the synchronization code by using stereotypes. Optimized synchronization code parts belong to the stereotypes, so an optimizer can decide to use them (if they have no effect to the other parts of the synchronization). As the optimization of the synchronization code may depend on the interconnection relation, optimized programs can be used only if the interconnection relation of the stereotype is the same as the interconnection relation of the system.

The using of antisymmetric interconnection relations and strong spatial operators provide a well usable tool. However, this may lead to unnecessary and sometimes very expensive steps by handling I. So developers can specify, if some of the operations does not need to be executed. We can achieve further improvement of the running time of the synchronization code by reducing the number of the token capturing calls, which can be done with state setter functions that have an array parameter – an array of states. The concept of the optimization with modified state setter functions and stereotypes were introduced in [10].

Thesis 3 *We extended our method for a larger group of the problems by introducing strong spatial operators and antisymmetric interconnection relations, which provide further possibilities by defining the connections of objects. Introducing synchronization diagrams improves the usability of the method,*

and by using abstract graphical elements, developers can specify complex synchronization schemas easier. Developers can optimize the handling of the structures, and can influence the effectiveness of the synchronization code by using stereotypes.

Bibliography

List of Personal Publications

Referred journals

[1] Sz. Hajdara, L. Kozma and B. Ugron: *Synthesis of a system composed by many similar objects*, Annales Univ. Sci. Budapest., Sect. Comp. 22 (2003) 127-150

[2] B. Ugron, L. Kozma, Sz. Hajdara and L. Blum: *Implementations of synchronization of concurrent objects*, Annales Univ. Sci. Budapest., Sect. Comp. 23 (2004) 79-102

[3] Szabolcs Hajdara and Balázs Ugron: *An extension of synthesis of the synchronization of object-oriented pipeline systems*, Pure Mathematics and Applications (PU.M.A.), Vol. 15, No. 2-3 (2004), pp. 157-169

[b] 5th Joint Conference on Mathematics and Computer Science, Debrecen (2004)

[4] Balázs Ugron, Szabolcs Hajdara, and László Kozma: *Synthesis of the synchronization of general pipeline systems*, Acta Cybernetica 17 (2005) 123-151

Referred conference issues

[5] Szabolcs Hajdara, Balázs Ugron: *An Example of generating the synchronization code of a system composed by many similar objects*, 17th European Conference on Object-Oriented Programming (ECOOP), The 13th Workshop for PhD Students in Object-Oriented Systems, Darmstadt (2003), ECOOP 2003 Workshop Reader, LNCS 3013, Springer-Verlag, Germany (2004) 55

[6] Szabolcs Hajdara, Balázs Ugron: *Describing Semantics of Data Types in XML*, Proceedings of the 6th International Conference on Applied Informatics, Vol. I, B. V. B. Nyomda és Kiadó Kft., Eger (2004) pp. 173-180

[7] Balázs Ugron, Szabolcs Hajdara: *Synthesis of the synchronization of pipeline systems*, Proceedings of the 6th International Conference on Applied Informatics, Vol II, B. V. B. Nyomda és Kiadó Kft., Eger (2004) pp. 351-359

Conference presentations

[8] Hajdara, Sz., Kozma, L., Ugron, B.: *Párhuzamos programok szintézise és objektum elvű kiterjesztése*, V. Országos Objektum-orientált Konferencia, Dobogókő (2002)

[9] Ugron, B., Kozma, L., Hajdara, Sz.: *A strukturált programozástól az objektum elvű technológiáig*, V. Országos Objektum-orientált Konferencia, Dobogókő (2002)

[10] Szabolcs Hajdara, Balázs Ugron: *Analyzing the effectiveness of synthesized synchronization codes*, 6th Joint Conference on Mathematics and Computer Science, Pécs (2006)

Other scientific results

[11] Szabolcs Hajdara, Balázs Ugron, László Kozma: *A graphical interface for specifying the synchronization of a concurrent system*, Periodica Politechnica, Budapest(2005) – submitted

[12] Szabolcs Hajdara: *Objektum-elvű rendszerek szintézisproblémái*, Professional conference of BME, Department of Measurement and Information Systems (2007)

Other Bibliography

[13] G. R. Andrews: *A Method for Solving Synchronization Problems*, Science of Computer Programming 13 (1989/90), pp.1-21.

[14] P. C. Attie, E. A. Emerson: *Synthesis of Concurrent Systems with Many Similar Processes*, ACM TOPLAS Vol. 20, No. 1 (1998), pp. 51-115.

[15] J. W. Cooper: *Java Design Patterns: A Tutorial*, Addison-Wesley (2000), ISBN 020 148 539 7.

[16] M. Dwyer, J. Hatcliff, M. Mizuno, M. Neilsen, G. Singh: *Automatic Derivation, Integration and Verification of Synchronization Aspects in Object-Oriented*

Design Methods Report for DARPA Order K203/AFRL Contract F33615-00-C-3044 (2001).

[17] E. A. Emerson, E. M. Clarke: *Using branching time temporal logic to synthesize synchronization skeletons*, Science of Computer Programming, 2 (1982), pp. 241-266.

[18] C. Girault, R. Valk: *Petri Nets for Systems Engineering*, Springer (2007), ISBN 354 041 217 4.

[19] D. Harkey, R. Orfali: *Client/Server Programming with Java and CORBA, 2nd Edition*, John Wiley & Sons (1998), ISBN 047 124 578 X.

[20] S. Jagadish, R. K. Shyamasundar: *UML based approach to specify secured, fine-grained concurrent acces to shared variables*, Journal of Object Technology, Vol. 6, No. 1 (2007), pp. 107-119

[21] L. Kozma, L. Varga: *A szoftvertechnológia elméleti kérdései*, ELTE Eötvös Kiadó (2003).

[22] D. Liang: *Introduction to Java Programming-Comprehensive Version (6th Edition)*, Prentice Hall (2006), ISBN 013 222 158 6.

[23] B. Meyer: *Object-Oriented Software Construction, Second Edition*, Prentice Hall (1997), ISBN 013 629 155 4.

[24] T. Minoura, G. Wiederhold: *Resilient Extended True-Copy Token Scheme for a Distributed Database System*, IEEE Trans. Software Eng. 8(3) (1982), pp. 173-189.

[25] G. J. Nyékyné et al. ed.: *Java 2 útikalauz programozóknak 1.3 (Programmers' Guide to Java 2, version 1.3)*, ELTE TTK Hallgatói Alapítvány, Budapest, Hungary (2001).

[26] G. J. Nyékyné et al. ed.: *J2EE útikalauz programozóknak (Programmers' guide to J2EE)*, Kiskapu Kft., Budapest, Hungary (2002).

[27] T. Quatrani: *Visual Modeling with Rational Rose 2002 and UML*, Addison-Wesley (2003), ISBN 020 172 932 6.

[28] S. Sike, L. Varga: *Objektum elvű modellalkotás UML-ben – Példatár definíciókkal* –, ELTE, TTK, Informatikai Tanszékcsoport, Budapest (2001).

[29] B. Ugron: *Synthesis of the synchronization code of pipeline systems*, PhD Thesis, ELTE IK, Budapest (2007).

[30] A. S. Woodhull, A. S. Tanenbaum: *Operating Systems. Design and Implementation.*, Prentice Hall (1997)