

This item is the archived peer-reviewed author-version of:

DSML4CP: a Domain-specific Modeling Language for Concurrent Programming

Reference:

Azadi Marand Elaheh, Azadi Marand Elham, Challenger Moharram.- DSML4CP: a Domain-specific Modeling Language for Concurrent Programming
Computer languages, systems & structures - ISSN 1477-8424 - 44(2015), p. 319-341
Full text (Publisher's DOI): <https://doi.org/10.1016/J.CL.2015.09.002>
To cite this reference: <https://hdl.handle.net/10067/1710640151162165141>

DSML4CP: A Domain-specific Modeling Language for Concurrent Programming

Elaheh Azadi Marand, Elham Azadi Marand, Moharram Challenger*

Department of Computer Engineering, Shabestar Branch, Islamic Azad University, Iran
{elaheh.azadi, elham.azadi, challenger}@iaushab.ac.ir

Abstract: Nowadays, concurrent programs are an inevitable part of many software applications. They can increase the computation performance of the applications by parallelizing their computations. One of the approaches to realize the concurrency is using multi thread programming. However, these systems are structurally complex considering the control of the parallelism (such as thread synchronization and resource control) and also considering the interaction between their components. So, the design of these systems can be difficult and their implementation can be error-prone especially when the addressed system is big and complex. On the other hand, a Domain-specific Modeling Language (DSML) is one of the Model Driven Development (MDD) approaches which tackles this problem. Since DSMLs provide a higher abstraction level, they can lead to reduce the complexities of the concurrent programs. With increasing the abstraction level and generating the artifacts automatically, the performance of developing the software (both in design and implementation phases) is increased, and the efficiency is raised by reducing the probability of occurring errors. Thus, in this paper, a DSML is proposed for concurrent programs, called DSML4CP, to work in a higher level of abstraction than code level. To this end, the concepts of concurrent programs and their relationships are presented in a metamodel. The proposed metamodel provides a context for defining abstract syntax, and concrete syntax of the DSML4CP. This new language is supported by a graphical modeling tool which can visualize different instance models for domain problems. In order to clarify the expressions of the language; the static semantic controls are realized in the form of constraints. Finally, the architectural code generation is fulfilled via model transformation rules using the templates of the concurrent programs. To increase level of the DSML's leverage and to demonstrate the general support of concurrent programming by the DSML, the transformation mechanism of the tool supports two well-known and highly used programming languages for code generation; Java and C#. The performed experiments on two case studies indicate a high performance for proposed language. In this regard, the results show automatic generation of 79% of the final code and 86% of the functions/modules on average.

Keywords: Domain-specific Modeling Language; Metamodel; Code Generation; Constraint Control; Concurrent Programming.

1. Introduction

In the recent years, programmers' attention to parallel programming has significantly increased. A need for a high processing rate in software applications is an important motivation for the users that forms and develops concurrent programs [1]. Different problems exist in developing correct and efficient parallel and distributed applications in general and concurrent programs in specific. On the other hand, due to the performance issues reaching optimal efficiency (as a main goal of parallel and distributed computations) is difficult. Moreover, engagement of concurrent program developers with the issues such as coordinating the threads (in parallel and distributed form) and synchronizing them (such as controlling mutual exclusiveness and deadlock) [2] [3] adds even more complexity to such software. Thus, for concurrent programs, a resolution is needed to reduce the complexity of their development. Basically, such methods need to be independent from the programming context and language and in a higher level than code. One possible solution is to promote the abstraction level in the development of concurrent programs. This approach allows the programmer to work in a higher level than code and this abstraction level decreases involvement of developer with details during design and implementation. It is worth to mention that the targeted environment for the concurrent programming in this study is shared memory.

* Corresponding author: m.challenger@gmail.com, Tel: +90 541 918 8836.

Model-driven Development (MDD) is one of the approaches to promote abstraction level in software development with the aim of changing programming focus from code to model. In this approach, models are created with required details; then, codes are generated automatically based on those models. This increases productivity, reuse (through reusing standardized models), simplifying design process, and promoting team work capability. It must be considered that a modeling paradigm is effective when its models are understandable from a domain user's viewpoint [4] [5]. This can be realized by using the domain terms and concepts along with their relations as the main elements of the new language during modeling the problem.

One of the typical ways of applying a Domain-specific Language (DSL) [6][7][8] is that a developer designs his/her own model in a language specific for the applied domain and the language produces the architectural code for the program model. In this way, instead of dealing with coding details, the programmer works with a model of the program. Also, instead of working with a General Purpose Language (GPL), the developer works with a specific language of the domain; thus, keywords, relations, and concepts will be very close to the ones in the application domain. One type of DSLs is Domain-specific Modeling Language (DSML) [9] [10], in which graphical elements and models are used for designing programs, giving a better insight about software elements and relations. This methodology is applied in many other cases [11] [12]. In [13], we examined different approaches of designing and implementing DSMLs by using available tools for developing these languages which paved the way for development of DSML for this study. Also, in [19], we presented a metamodel for concurrent programs along with a textual tool for their development. In this paper, concurrent programs are addressed as the domain to apply DSML techniques and benefits from MDA methodology. We propose a new language for increasing the abstraction level and reducing development complexities of the domain. We call this language as Domain-specific Modeling Language for Concurrent Programming, DSML4CP. To increase level of the DSML's leverage and to demonstrate the general support of concurrent programming by the DSML, the transformation mechanism of the tool supports two well-known and highly used programming languages for code generation; Java and C#.

The remainder of this paper is organized as follows: in section 2 the language syntax including abstract syntax and concrete syntax is discussed. Section 3 discusses static semantic controls and code generation of the language. In section 4, the language is utilized in two case studies. In section 5, we evaluate, analyze and discuss DSML4CP. Related work and comparison between the new language and the previous studies are discussed in section 6. Finally, section 7 concludes the paper and states the challenges and future work.

2. Syntax of the Language

In computer science, language syntax is a set of rules which define the combination of the symbols and describe the structure of the language. Similarly, based on MDD principles, a DSML includes abstract and concrete syntaxes. So, accordingly we provide these two syntaxes for DSLM4CP in this section.

2.1 Abstract Syntax

In recent years, different metamodels for various domains have been proposed which are used as abstract syntaxes of DSLs [14] [15]. Besides, Object Management Group (OMG)¹ has played an important role in applying them in a standard way. With the advent of Model-Driven Architecture (MDA) [16] [17] by OMG and with the increasing need of standardization, applications of metamodels have been growing steadily. For development of a DSML, an abstract syntax can appear as a metamodel which can be implemented in a metamodeling tools such as Ecore². Abstract syntax of a language specifies domain terms and identifies how they can be combined for creating instance

¹ Object Management Group, <http://www.omg.org>

² EMF Ecore, <http://eclipse.org/modeling/emf/>

models [18] [5]. In this part, we present an updated version of the metamodel as the abstract syntax for concurrent systems. The base metamodel was presented in our previous study which can be accessed in [19]. To have thorough comprehension and efficient use of the metamodel, we imagine various meta-elements (as illustrated in Figure 1) in the metamodel which are explained in this section.

Concurrent programs are lists of instructions to be executed in parallel. Since each program has logic, they can realize special tasks. Thus, the “Program” meta-element on our metamodel is used for determining the intended program which is modeled. In concurrent programming, a thread plays the main role which allows the users to do tasks in parallel. So, a meta-element, called “Thread” is considered in the metamodel which is connected to the “Program” with a composition relationship indicating that a program may have several threads. Considering the fact that a thread is able to create another thread within a program, we have considered this feature for the “Thread” meta-element and a thread can make any number of other threads. Threads require including a start point to kick off the tasks from. The “StartResumePoint” meta-element in the proposed metamodel is representing start point. Aside from this, a start point can be also within a task which leads to a resume case. A stop point is also necessary to show when the task of a thread has ended and when it is going to deliver execution order to another thread. This element is called as the “StopPoint” meta-element.

Threads are specifically useful when it is required that the program wait for the response of another part of it which ends up overall speedup. Therefore, the “WaitPoint” meta-element is needed for the threads to pause. In general case, execution of a thread stops at a point, the thread goes into waiting mode and resumes later. The “StartResumePoint” meta-element also provides this capability. Besides, a program with a multi threads which can meet in a point, necessitates the “MeetingPoint” element. Every task consists of one or more threads and each task can have some sub tasks. So, we consider a meta-element called “Task”. This meta-element is connected to the “Thread” meta-element by an association link. This support a thread to have one or more tasks which can be constituted of subtasks. In addition to the “Thread” element, a “ThreadStore” meta-element can be regarded. It acts like a container in which threads can enter. Also, a link exists between “ThreadStore” and “Thread”, since a thread can be placed inside a “ThreadStore” or “Thread”. In addition, we specify a meta-element, called “CLASS”, which includes a class with some fields and methods to work on the data and offer services for requestors.

The use of a thread can be realized by instantiating an object from its specifying class. “Thread” and “CLASS” meta-elements represent these concepts and we should be able to create instances from them. If we imagine classes as drawn house plans, we will be able to make some objects based on a class as we can build houses based on plans. Creating instances from these meta-elements, leads to “ThreadObject” and “GeneralObject” meta-elements with a link in between; because a thread can hold other objects in itself. The instances created as data member or function member can be transferred among threads. So, these elements identify the point that the data or function members can be shared or locally used. Inheritance can be used in these meta-elements with which a class can inherit the attributes of another class. This important capability makes the hierarchical classification possible.

Synchronization has a great importance in concurrent programs, since it is used for coordination of competition among threads. When several threads run in the memory on a shared object, unpredictable results may occur, if they are not controlled. Therefore, multithread program behavior cannot be trusted without proper synchronization and coordination, needed to prevent problems from occurring runtime. Generally, when a coordination issue is regarded, Mutual Exclusion and Deadlock [20] are mainly concerned which play an important role in our metamodel as well. Therefore, we have included “MutualExclusion”, “Deadlock” and “Synchronization” meta-elements in the proposed metamodel. The “Mutualexclusion” meta-element represents the techniques to prevent problems from occurring during coordination, giving exclusive access permission to each thread working on a common object. While a thread is working on an object, the other threads trying to access that object need to wait. When the thread has performed its task with exclusive access, a waiting thread gains

access permission to the shared object. In this way, when a thread accesses a shared object, the other threads are prevented from concurrent access to the shared object. To realize exclusive access to the resources, concurrent techniques and algorithms such as Semaphore, Monitor and Lock can be used [21]. We represented them in the proposed metamodel for managing exclusive access to resources and coordinating threads. The representing meta-elements in the metamodel are called “Semaphore”, “Monitor” and “Lock”.

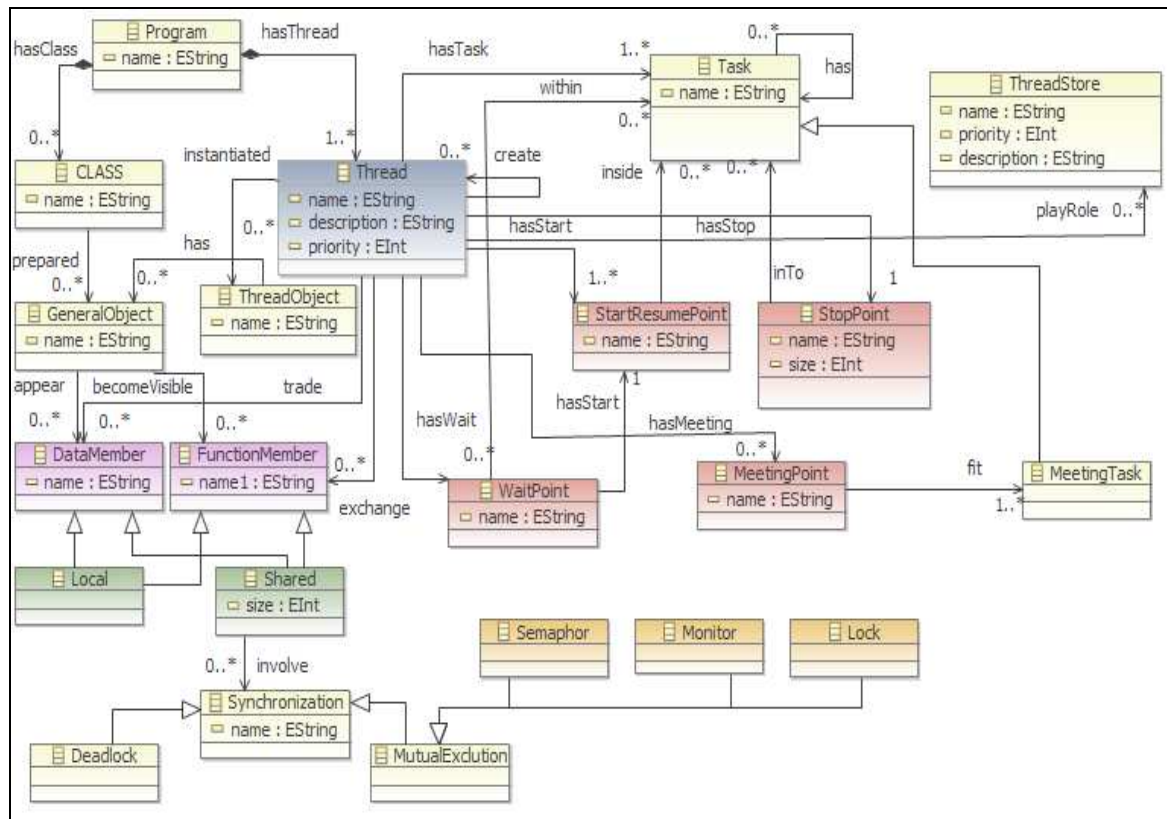


Figure 1. Metamodel for concurrent programs [19]

2.2 Concrete Syntax

Languages have notations for representing their concepts and have constraints for their syntaxes. These notations are known as concrete syntax of those languages. The concrete syntax has two main types: Graphical Concrete Syntax (GCS) and Textual Concrete Syntax (TCS), used by different languages [22] [68]. In this paper the main concrete syntax for DSML4CP is a GCS for the concurrent programs. In this study, the concepts of concurrent programs have been defined in an Ecore file as a metamodel. To design the Ecore model, we have used the above mentioned metamodel for which validation has been examined in Epsilon [23], and then we developed EMF¹ file in EMFatic language using EuGENia tool. Listing 1 shows a part of the code for concrete syntax of DSML4CP in EMFatic language.

EuGENia provides a set of high-level annotations in EMFatic language with which shield the developer from the complexity of Graphical Modeling Framework (GMF²) and removes the initial barrier for creating the GMF editor. To this end, in Listing 1, we have added various annotations before each meta-element to define the notations. For example, in Line 2 “@gmf.diagram” annotation defines the class as a root element where all of the other meta-elements are located within its class using “val” keyword (Lines 3-14). Also by adding “@gmf.node” annotation in Line 15, this capability is provided for “Program” meta-element which appears as a node on the palette.

¹ Eclipse Modeling Framework (EMF): <http://www.eclipse.org/modeling/emf>

² Graphical Modeling Framework (GMF): <http://www.eclipse.org/modeling/gmf/>

In addition, in Line 17, “@gmf.compartment” annotation defines a containment relation which creates a compartment for the model elements. Lines 18 and 21 are examples for this case and they make it clear that the “Thread” and “CLASS” meta-elements are inside the “Program” meta-element. It is worth mentioning that for showing links on the palette, we used “@gmf.link” annotation, e.g. Lines 28 and 34. In Line 29 the “ref” keyword indicates that between the “Thread” and “Task” meta-elements, there is a link named “hasTask”. Also, it is remarkable that, the “attr” keyword provides attributes for meta-elements. In Lines 25 to 27, “name”, “description” and “priority” attributes are defined for “Thread” meta-element.

```

01 package defaultname;
02 @gmf.diagram (foo="bar")
03 class concurrent {
04     val Program[*] programs;
05     val Thread[*] threads;
06     val CLASS[*] homes;
07     val Monitor[*] monitors;
08     val Lock[*] locks;
09     val MutualExclusion[*] mutexs;
10     ...
11     val Deadlock[*] deadlocks;
12     val ThreadStore[*] threadStores;
13     val Synchronization[*] sync;
14 }
15 @gmf.node (label="name")
16 class Program {
17     @gmf.compartment (foo="bar")
18     val Thread[+] hasThread;
19     attr String name;
20     @gmf.compartment (foo="bar")
21     val CLASS[*] hasClass;
22 }
23 @gmf.node (label="name")
24 class Thread {
25     attr String name;
26     attr String description;
27     attr int priority;
28     @gmf.link (label = "hasTask", style = "dot", width = "2")
29     ref Task[+] hasTask;
30     ...
31 }
32 @gmf.node (label="name")
33 Class ThreadObject {
34     @gmf.link (label = "has", style = "dot", width = "2")
35     ref GeneralObject[*] has;
36     attr String name;
37 }

```























Listing 1. Part of the code for concrete syntax of DSML4CP in EMFatic language

As a result, a set of graphical notations for the meta-elements in the metamodel has been selected for the concepts and relations to be added to the tool. Table 1 shows the graphical symbols for some of the concepts in the concrete syntax of concurrent programs. After selecting graphical symbols, Eclipse GMF was used for relating domain concept in Ecore format and their symbols. Resultant structure is a graphic editor developed based on the concrete syntax for concurrent programs and enables users to design concurrent programs on the modeling framework.

Although the main concrete syntax of DSML4CP is a graphical one, we also developed a textual concrete syntax [19] using Xtext [24] in order to increase the number of the users, since some

developers are interested in textual programming. In addition, a textual language has the advantage of scalability comparing to graphical languages.

Table 1. Some of the concepts and their notations for concrete syntax of concurrent programs

Concept	Notation	Concept	Notation	Concept	Notation
Program		MeetingTask		FunctionMember	
Thread		WaitPoint		Local	
Task		CLASS		Shared	
StartResumePoint		GeneralObject		MutualExclusion	
StopPoint		ThreadObject		Semaphore	
MeetingPoint		ThreadStore		Monitor	
Synchronization		DataMember		Lock	
Deadlock					

3. Semantics of the Language

The concepts of a language have a critical importance and providing them in a language is one of the main issues in language development process. However, the meanings of the concepts are as important as their syntax. With exact understanding of the concepts' meaning in a language, the ambiguity of the language is resolved and the artifacts can be easily generated for the target platforms. Because of the importance of this resolution, focusing on semantics and its definitions are inevitable for a language [25] [67]. Hence, in this section, the semantics of DSML4CP is stated in the form of static semantics using constraint checks and operational semantics using model transformations.

3.1 Static Semantics with Constraint Checks

Since abstract syntax and concrete syntax gives little information about language meaning, we need further controls to describe the language concisely. In this case, language concepts can be limited by constraints and can be described by the concepts of the other languages. These approaches are called constraint check and model transformation, respectively. These are necessary for an executable language to enable the language to operate accurately [22].

There are different tools for constraint check/control of language (e.g. Object Constraint Language (OCL)¹ [26] and Epsilon Validation Language (EVL)² [27]) from which necessary constraints can be applied on metamodels. In this section, static semantics are applied in DSML4CP, for the proposed metamodel and its corresponding elements and links, by providing different constraint controls including character/string constraints, "a number of elements and relations" constraints, and "a sequence of the relations" constraints using EVL. These constraints will be applied on models to limit the user when he/she designs the instance model based on the metamodel in DSML4CP framework. Each of these constraint controls is elaborated as follows:

Character and string constraints:

For each meta-element of metamodel in Figure 1, some character/string constraints have been added. One of these constraints is named "NameStartsWithCapital" and requires the initial character of each element to be capitally named, while designing, on the palette. Unless this happens, there will be a warning saying each element should be named with a capital letter. Listing 2 shows this constraint for the "CLASS" meta-element as an example. As another example, the "HasName"

¹ Object Constraint Language (OCL): <http://www.omg.org/spec/OCL/>

² Epsilon Validation Language (EVL): <http://www.eclipse.org/epsilon/doc/evl/>

constraint is considered for some of the meta-elements that contain the “name” attribute. This constraint limits the elements to have a name while designing on the palette unless; a “not allowed” message is displayed for the developer. Listing 2 shows this constraint for the “Program” meta-element.

```

01 context CLASS {
02     critique NameStartsWithCapital {
03         guard: self.satisfies ('HasName')
04         check: self.name.firstToUpperCase() = self.name
05         message: 'CLASS ' + self.name + 'should start with an upper-case letter'
06         fix {
07             title: 'Rename to ' + self.name.firstToUpperCase()
08             do { self.name := self.name.firstToUpperCase() }
09         }
10     }
11 }
12 context Program {
13     constraint HasName {
14         check : self.name.isDefined()
15         message : 'Unnamed ' + self.eClass().name + ' not allowed'
16     }
17 }
18 Context ThreadStore {
19     constraint BannedCharactersForName {
20         guard: self.satisfies ('HasName')
21         check {
22             if (self.name.toCharSequence().includes("*")) return false;
23             else if (self.name.toCharSequence().includes("/")) return false;
24             else if (self.name.toCharSequence().includes(":")) return false;
25             ...
26             else if (self.name.toCharSequence().includes("")) return false;
27             return true;
28         }
29         message: 'ThreadStore' + self.name + 'Must not have (/ , * , : , ? , < , > , | , \\ , ") characters!'
30         fix {
31             title: 'Remove banned character from name'
32             do {
33                 self.name := self.name.replace("\\*", "");
34                 self.name := self.name.replace("/", "");
35                 ...
36                 self.name := self.name.replace("", "");
37             }
38         }
39     }
40 }
41 Context StopPoint{
42     critique NumberOfElements{
43         check : self.name.size() < 1
44         message : "Every program should have one stop point"
45     }
46 }
47 context Thread {
48     constraint RalationBetweenElements {
49         guard: self.satisfies ("HasName")
50         check: self.source.eClass () and self.target.eClass ()
51         message: "Link is forgotten"
52     }
53 }

```

Listing 2. Part of constraints for DSML4CP in EVL language

In addition to these constraints, the “BannedCharactersForName” constraint is introduced in a way that, while naming, none of the special characters (* , / , : , ? , > , < , | , \ , “) can be used for meta-elements; because starting with these characters will lead to an error. Listing 2 shows this constraint over the “ThreadStore” meta-element in Lines 18-40.

Constraint for number of meta-elements:

For some meta-elements in the proposed metamodel, the number of elements constraint is added which is called “NumberOfElements”. In Listing 2, we considered this constraint for the “StopPoint” element. Based on this constraint, the stop points for each program are checked in Line 43. If the stop point is more than one, there should be a warning saying that “every program should have one stop point”. A similar constraint has been specified to the “Shared” element.

Relation constraint:

There are other kinds of constraints provided for elements’ relations. One of these constraints is “RelationBetweenElement”. Listing 2 shows this constraint for “Thread” element. As it can be seen, first, the “guard” keyword in Line 49 guarantees that each link has a name. Then, in Line 50, the source and target meta-elements are controlled for the thread. If a link is forgotten to be designed between the meta-elements in the metamodel, there will be a message displayed saying that “Link is forgotten”.

In addition to the abovementioned constraints, DSML4CP provides some other constraint controls for the users which are implemented using the Eclipse framework capabilities. The produced tool in Eclipse has control facilities that check the following constraints:

Compartment constraint:

The composition relationship among the meta-elements in the Ecore is transformed to a relationship in a way that an element can contain another element. The elements which do not have such a relationship have no possibility to be modeled in the form of a nested compartment. For example, this capability can be seen in the “Thread” and “CLASS” meta-elements as both of them are located in the “Program” meta-element.

Number of relationships constraint:

Based on one-to-one, one-to-many, and many-to-many relationships in the metamodel, a control is done on the number of relationships among the elements in the models. For instance, this relationship is considered between the “Thread” and “StopPoint” meta-elements, meaning that every thread can only have one endpoint.

Source and destination constraint:

The direction of the relationship defines the source and destination of that relationship. This constraint is defined at the Ecore level to control this direction. For example, based on our constraints, relationship between “Task” and “Synchronization” meta-elements cannot be established at the time of designing models.

Inheritance constraint:

The inheritance relationships defined in DSML4CP bring some more constraints when modeling the language tools. Naturally, a subclass in a model includes all of the attributes and relationships of its superclass. “Semaphore”, “Monitor” and “Lock” meta-elements are examples of this case.

The constraints mentioned above show the intelligence of a DSM which forces the user to model accurately and reduce error occurrence in the level of modeling. This also leads to a code with fewer errors, after the code generation step.

3.2 Operational Semantics by Code Generation

It is not enough to develop a DSML simply with conceptualization of terms and relationships on a graphical tool. A complete definition of a DSML requires code generation on validated models. For code generation, there are different tools to implement the model-to-code transformations, such as, MOFScript [28], Acceleo [29], JET [30], and Xpand [31]. In this study, we selected Xpand due to its advantages such as simplicity, integrity with Eclipse, and working with EMF based models.

```
01 «IMPORT defaultname»
02 «DEFINE main FOR concurrent»
03   «EXPAND javaClass FOREACH this.programs»
04   «EXPAND javaClass FOREACH this.tasks»
05   ...
06   «EXPAND javaClass FOREACH this.sync»
07   «EXPAND javaClass FOREACH this.threadStores»
08 «ENDDEFINE»
09 ...
10 «DEFINE javaClass FOR Program»
11   «FILE name+".java"»
12   public class «name» {
13     // The entry main method
14     public static void main() {
15       «EXPAND InstanceVar FOREACH this.hasThread»
16       «EXPAND InstanceVar1 FOREACH this.hasThread»
17       «EXPAND javaClass1 FOREACH this.hasClass»
18       «EXPAND javaClass FOREACH this.hasThread»
19     }
20   }
21 «ENDFILE»
22 «ENDDEFINE»
23 «DEFINE InstanceVar FOR Thread»
24   //Create our own Thread
25   «this.name» myThread «this.name» = new «this.name» ();
26   //Starting Threads
27   myThread «this.name».Start ();
28   //Wait for the threads to finish
29   myThread «this.name».join ();
30 «ENDDEFINE»
31 ...
32 «DEFINE javaClass FOR Thread»
33   «FILE name + ".java"»
34   public class «name» {
35     class thread «this.name» implements Runnable {
36       // override the run() method
37       @Override
38       public void run() {
39         /* Implementation of the thread goes here*/
40       }
41     }
42   }
43 «ENDFILE»
44 «ENDDEFINE»
```

Listing 3. Part of template rules for code generation in DSML4CP

For code generation in the solution domain, we need to access metamodel of the domain (Ecore file), and the problem instance model (XMI file). To realize the code generation, we need to provide some rules based on the architecture of the selected domain using the Xpand tool (XPT file). Eventually, we can generate the artifacts, which are Java code in this case, by running these rules.

Listing 3 shows part of the rules for code generation by Xpand tool. In line 1 of Listing 3, the metamodel namespace is imported in order to make the meta-types known to the editor. Next, the main template is created. Xpand’s keywords and meta-type references are always enclosed in “«” and “»” characters. In this Listing, the “javaClass” function is invoked for “this.programs”, “this.tasks”, “this.sync” and “this.threadStore” in lines 3-7 to call the other sub-templates. In Xpand, it is also possible to invoke a function for each of the elements. For example, function call for “Program” meta-element is shown in Listing 3. As it can be seen in Line 10 of Listing 3, the “javaClass” function is defined for the “Program” element. In Lines 15-18, by receiving the program name, there are invocations of “InstanceVar” and “javaClass” functions for each “this.hasThread”; and also, invocations of “InstanceVar1” and “javaClass” functions for each “this.hasClass” to fill the rest of patterns using sub-patterns. Later in Line 23 of this Listing, the “InstanceVar” function is defined for the “Thread” meta-element. This function gets thread names and makes objects for each thread class and then, starts the thread. Additionally, we have defined “javaClass” function for each “Thread” in Line 32. Inside this function, the class is implemented for the “Thread” element which overrides the run method.

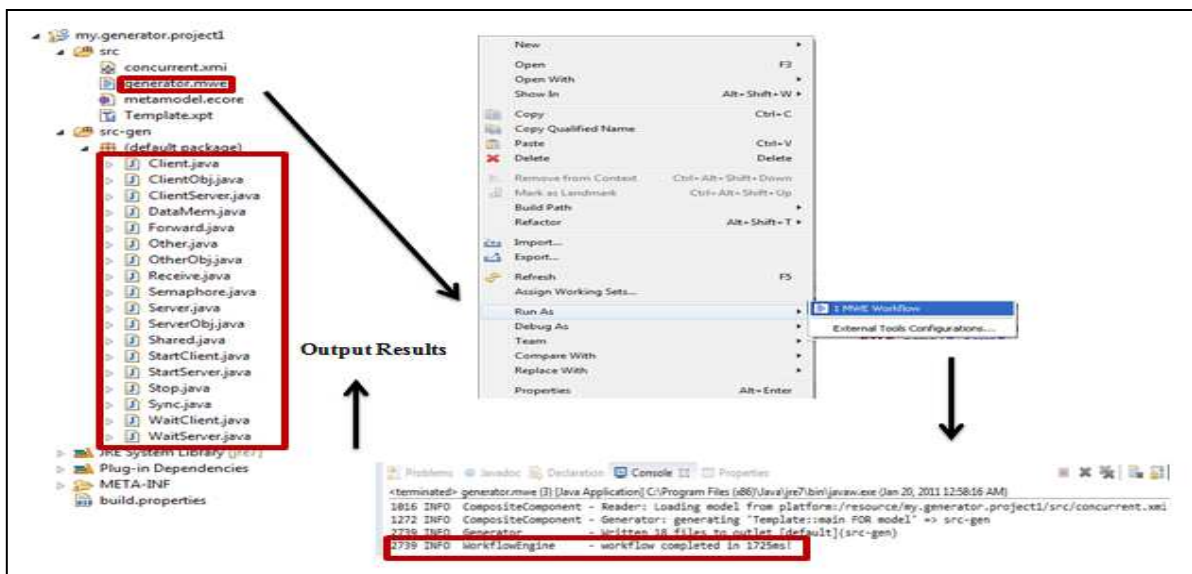


Figure 2. Generation procedure for DSML4CP with Xpand

It should be noted that, for the other meta-elements the “javaClass” function is defined similarly. For example, we examine the rules for the “Program” meta-element. To get the output, we should run generator.mwe file in Xpand tool, as it is shown in Figure 2. After completion of the workflow, the output files, in the form of Java codes, will be generated based on the provided rules for each element.

The left side in the Figure 2 shows that for each of the elements (XMI file), results have been generated automatically as a set of files including Java codes. Listing 4 shows a sample of codes generated for the “Program” element which is named “ClientServer”.

As it can be seen in Listing 4, an object is created for each thread in the main method. In addition, with having another class named Buffer, a separated object is created for this class to complete the task of the thread. Incidentally, all output files are generated likewise.

It is worth mentioning that the rules written for generating C# code is structurally similar with those for Java, see Listing 3, with only different templates. So, the generating rules for C# are not presented in the paper to avoid repeating. However, in the second case study (Parallel Matrix Multiplication Problem), the code generation for C# is used and the results are demonstrated.

```

01 public class CilentServer{
02     public static void main() {
03         //Create our own Thread
04         Client myThreadProducer = new Client();
05         //Starting Threads
06         myThreadClient.Start();
07         //Wait for the threads to finish
08         myThreadClient.join();
09         //Create our own Thread
10         Server myThreadServer = new Server();
11         //Starting Threads
12         myThreadServer.Start();
13         //Wait for the threads to finish
14         myThreadServer.join();
15         new Buffer();
16     }
17 }

```

Listing 4. Sample Output file for “Program” element

4. Case Studies

The syntax and semantics of the DSML4CP language is explained in section 2 and 3. In this section, we are going to explain the performance of the developed language over two examples. We selected Producer/Consumer and Parallel Matrix Multiplication Problem as they are among the most well-known problems in parallel processing. Also, they have enough complexity in coding to be modeled in DSML4CP.

4.1 Producer/Consumer Problem

By considering the definition of the Producer/Consumer problem, we can see that there are two main actors in this problem; the producer and the consumer which access a shared memory with specific capacity as a buffer. The producer’s task is to make data items and put them in the shared memory. At the same time, the consumer can take data items and consume them. The producer and the consumer can produce or consume just one data item in each turn. These two processes can run simultaneously.

The problem is that we should make sure that a producer does not try to put a data item in a full memory. Similarly, a consumer should not try to get a data item out of empty memory. Whenever the memory is full and producer tries to save an item in, it should sleep until a consumer gets an item out and awakes it, similarly, whenever a consumer wants to get an item out of empty memory, it should sleep until a producer saves an item in and awakes the consumer. An unwanted situation is when both of the processes go to sleep and nothing can awake them. We can prevent this problem with semaphore and monitor algorithms [32] [33].

To apply the development methodology proposed by DSML4CP on Producer/Consumer problem, we need to first model the solution in DSML4CP’s graphical language. Based on concepts discussed in section 2, the instance model of the concurrent program is implemented in a graphical editor which models required instance elements. The model for the Producer/Consumer problem is designed in the tool and its snapshot is illustrated in Figure 3.

In this instance model, each producer and consumer is regarded as a thread class including starting, waiting, and stopping points. Threads start their process at start point, wait in waiting point, and stop at stop point. Each point is a task and the tasks should be connected to a thread, as can be seen in Figure 3. When a producer produces data, it must be stored in some place of the memory. In our

metamodel, for storing data, a memory block is considered which is shared between producer and consumer.

The producer stores its produced data in this memory and the consumer has access to this data to take the produced data and remove it to use. If there is no data in the memory, a delay occurs between production and consumption. Then, the consumer waits to get produced data and remove it.

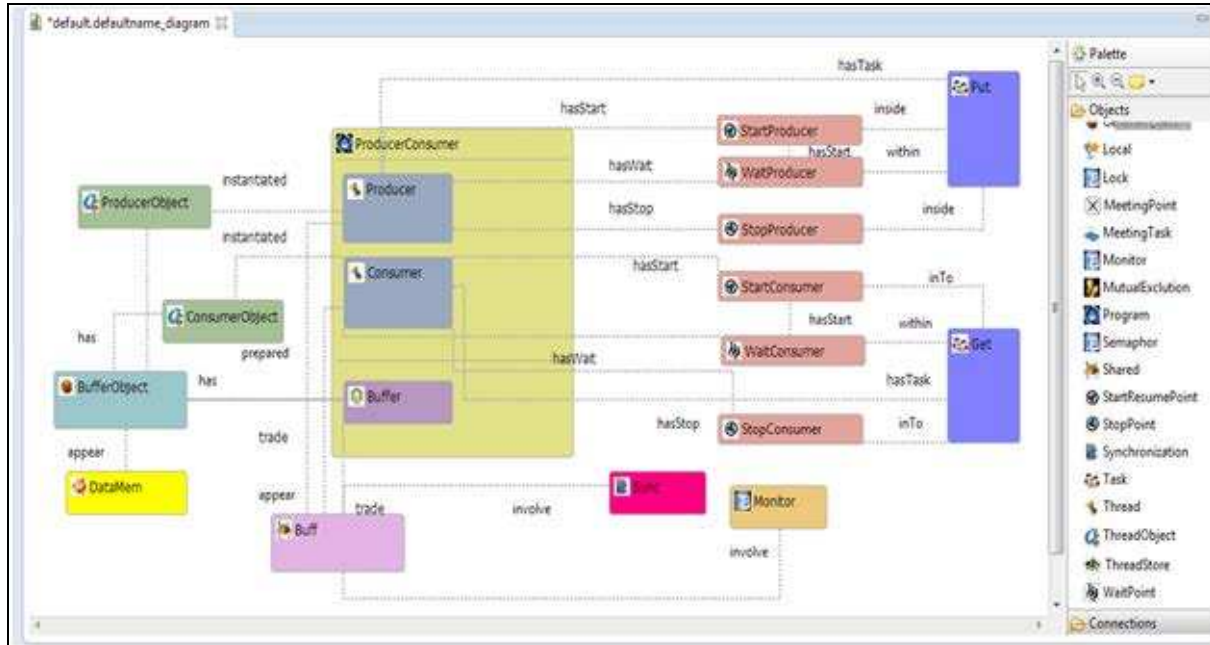


Figure 3. Modeling for Producer/Consumer problem

On the other hand, produced data should be accessed by only one consumer. To realize this, coordination is needed to make sure that produced values are consumed properly, e.g. mutual exclusiveness, which allows the program to coordinate threads. For managing the exclusive access to resources, techniques such as semaphore, monitor, and lock can be used.

During the modeling of the instance model for Producer/Consumer problem, the tool provided the semantic controls where needed. By adapting to the proposed constraints for the meta-elements discussed in section 3, in this section, the Producer/Consumer problem is modeled in the provided tool and checked in accordance with those constraints.

As it can be seen in Figure 4, the “HasName” constraint was not true for “Task”, “Program”, and “Thread” elements during the model design, so the language supposed it as an error and generated a relevant message for each of the elements separately. Also, the “NameStartsWithCapital” constraint was not true over “dataMember” and “consumer” elements, since the user has initialed them with small letters, so a warning is shown to the user meaning that a capital letter should be used for the initial letters. In addition, the connection between “WaitConsumer” and “StartConsumer” elements has been missed, so an error telling “the connection is missed” is given to the developer.

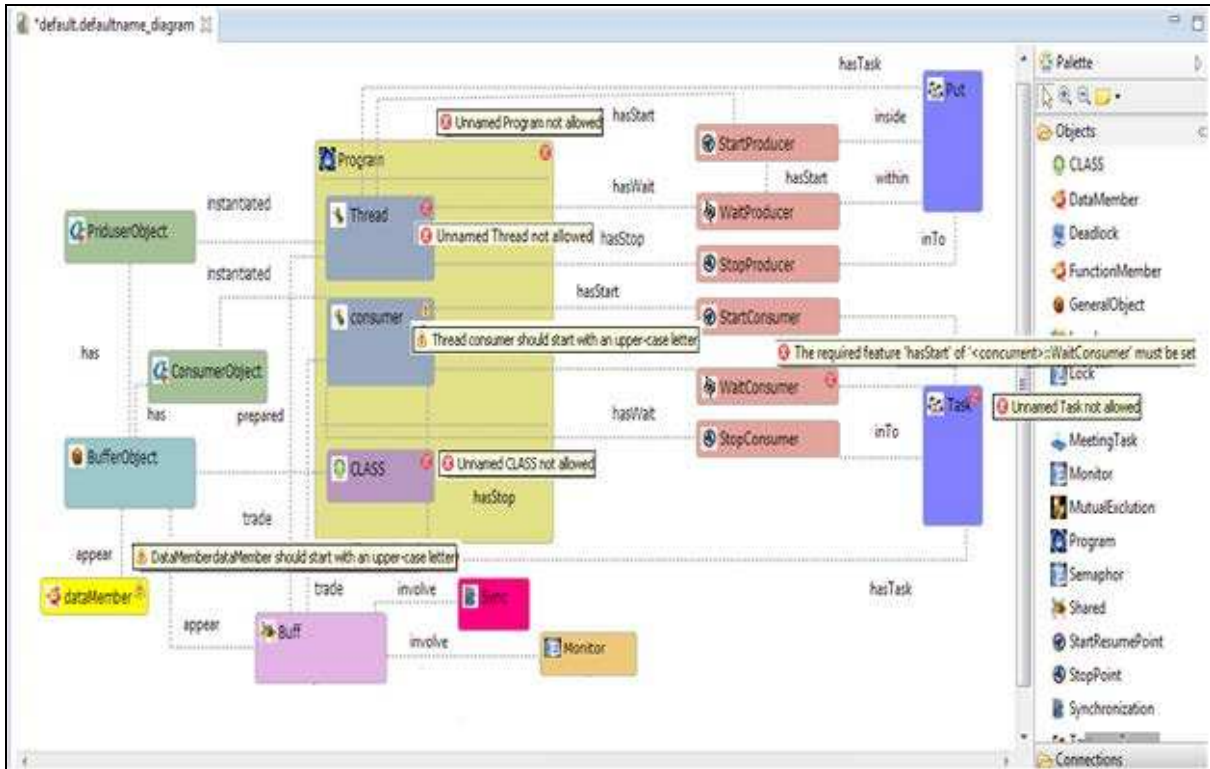


Figure 4. An illustration of constraints over Producer/Consumer problem

As the last step of using DSML4CP for Producer/Consumer problem, the code generation is realized. To this end, in this part, we describe how to generate the executable architectural code for this problem based on discussed concepts in section 3. For this purpose, we applied the steps outlined in Figure 2 and the output files were obtained as some Java codes for this case study. Figure 5 shows a list of output files for the Producer/Consumer problem. Altogether, 18 files were generated which constitutes the program of the model.

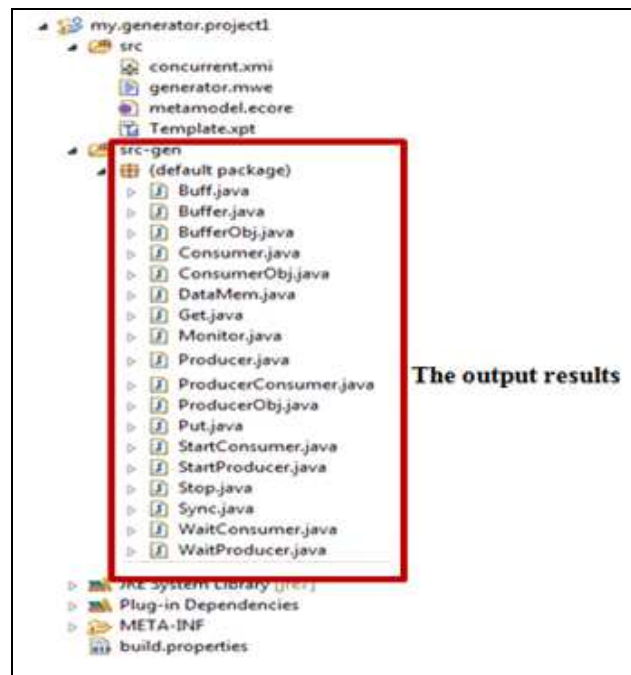


Figure 5. An illustration of output files for Producer/Consumer problem

```

01 public class Producer {
02     private int n;
03     private Buffer ProducerBuf;
04     public Buffer getProducerBuf () { return ProducerBuf; }
05     public void setProducerBuf (Buffer ProducerBuf) { this.ProducerBuf = ProducerBuf; }
06     public int getN () { return n;}
07     public void setN (int n) { this.n = n;}
08     @SuppressWarnings ("unused")
09     public Producer() {
10         int m = 0;
11         n = m;
12         Object buf = null;
13         Object Producerbuff = buf;
14     }
15     class threadProducer implements Runnable {
16         @SuppressWarnings ({"unused", "null"})
17         public void run() {
18             for (int i = 0; i < n; i++) {
19                 // sleep for a randomly chosen time
20                 try {Thread.sleep ( (int) Math.random() * 100); }
21                 catch (InterruptedException e) { return;}
22                 try {
23                     Buffer Producerbuff = null;
24                     Producerbuff.put (i + 1); //starting from 1, not 0
25                 } catch (InterruptedException e) { return;}
26             }
27         }
28     }
29 }
30 Public class Consumer {
31     private int n;
32     private Buffer ConsumerBuf;
33     public Buffer getConsumerBuf () {return ConsumerBuf;}
34     public void setConsumerBuf (Buffer ConsumerBuf) {this.ConsumerBuf = ConsumerBuf;}
35     public intgetN () {return n;}
36     public void setN (int n) {this.n = n;}
37     @SuppressWarnings ("unused")
38     public Consumer () {
39         int m = 0;
40         n = m;
41         Object buf = null;
42         Object Consumerbuff = buf;
43     }
44     class threadConsumer implements Runnable {
45         @SuppressWarnings ({"unused", "null"})
46         public void run() {
47             int value;
48             for (int i = 0; i < n; i++) {
49                 value = Consumer.get ();
50                 // sleep for a random time
51                 try { Thread.sleep ( (int) Math.random () * 100); }
52                 catch (InterruptedException e) {return;}
53             }
54         }
55     }
56 }

```

Listing 5. Output file for “Thread” in Producer and Consumer elements

It is worth mentioning that the generated code was completed with some manually added codes (developer’s delta codes) for each case study to become final executable programs. Some parts of the

final codes for thread classes, Thread in Consumer and Thread in Producer, are presented in Listing 5. The analyses of the generated and added codes are provided in the next section.

4.2 Parallel Matrix Multiplication Problem

In this section, we examine the Parallel Matrix Multiplication problem as one of the case studies in this paper. Parallel Matrix multiplication is one of the most prevalent operations in scientific computing and its modeling can represent complex examples of concurrent programming.

Multiplication operation of matrixes is costly and time consuming when size of two matrixes is very large. If we consider the complexity of its algorithm in mathematics, it needs at least $O(n^3)$ operations to multiple two $n*n$ matrixes, which is time consuming for big sizes of n . It is intended to decrease this time to as low as $O(n)$ by applying concurrent programming and using $n*n$ processing threads. As regards in the parallel matrix multiplication, the calculation of every element in the product matrix is result of vector multiplication for a row in the first matrix with a column in the second matrix and this calculation is independent of computing the other elements. Therefore, for multiplying two $n*n$ matrixes by having n^2 processors, multiplication of every rows and columns can be given to a processor which calculates one of the n^2 elements of product matrix [34].

For modeling the problem in a higher level, we design the Parallel Matrix Multiplication case study in the graphical editor of DSML4CP. Figure 6 shows the modeling of this problem based on the proposed metamodel in this study. In this model, we considered two classes of "Thread" namely "Main" and "Mul". The Main thread creates the other helper threads and distributes all the rows and columns of the matrix amongst those threads. Helper threads fulfil vector multiplication operations and return the results to the main thread which prints out the result matrix. As it can be seen in Figure 6, considering the instantiation of the threads, "Main" and "Mul", we have created "ObjMain" and "Obj1 to Obj16" objects from those classes and typed them as data. Given that, in this case study, we have considered the size of the matrixes as $4*4$ (to be able to represent the models in the paper), 16 thread objects have been created from "Mul" class and each of them does multiplication of a row and column in the matrixes.

On the other hand, for each "Thread" class, we have start, wait, and stop points. In this way, threads can start their tasks from the start points, wait for each other in the wait points and end their tasks in the stop points.

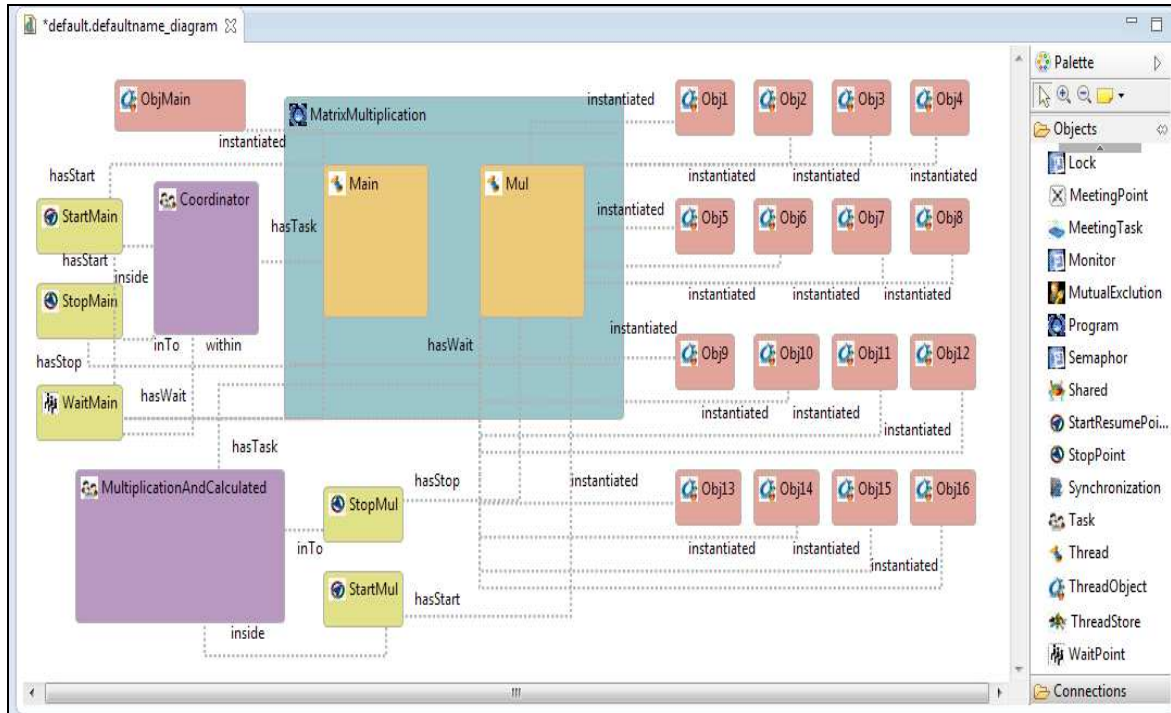


Figure 6. Modeling of the Parallel Matrix Multiplication problem

It should be mentioned that the main thread acts as a coordinator and after the distribution of rows and columns of the matrix between the threads, it waits at a waiting point for the results and then it prints out the result. Figure 7 shows the process of this computational model.

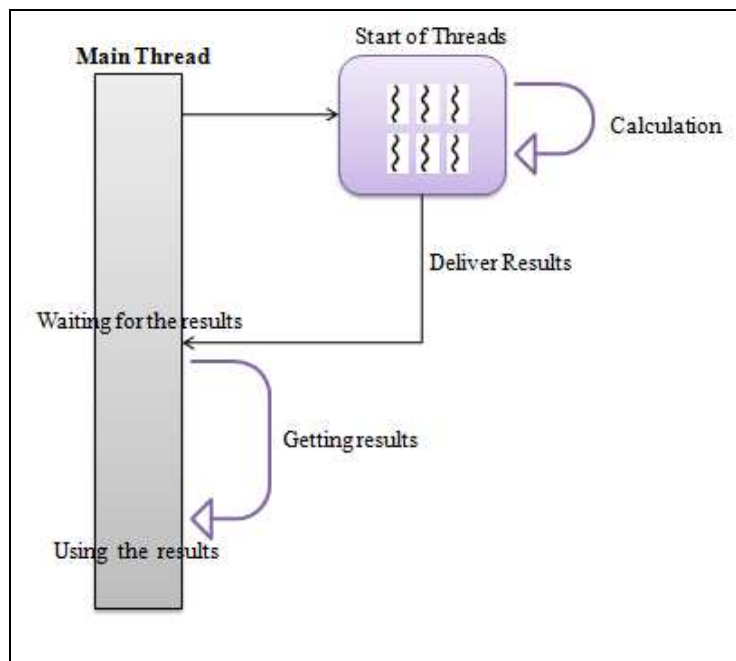


Figure 7. The process of Parallel Matrix Multiplication

For applying the semantic controls for the Parallel Matrix Multiplication problem based on given constraints in section 3, we examine the designed model in the graphical editor. As it is illustrated in Figure 8, some proposed constraints have been forgotten by the developer for the model, for which some errors and warnings have been appeared aside the elements.

As an example, “HasName” constraint is not considered for the “ThreadObject” and “Task” elements. So the language shows an error for these elements. Also, threads need to have a start point to begin their tasks from that point. Similarly, the connection between “Mul” and “StartMul” elements has been missed by the developer, so an error appears aside the element until the user adds the link between the elements.

In addition, the constraint rule of “BannedCharactersForName” is not observed by the developer for “Program” element. The developer should not use “*” as the first character for the name of this element, so there is an error stating that the name must not have (/ , * , : , ? , < , > , | , \ , ") characters. Also, “NameStartsWithCapital” constraint was not true for the “Task” and “ThreadObject” elements (named “multiplicationAndCalculated” and “obj10”) and the developer has used lowercase as the initial character for the name, so there is a warning stating that the name should start with a capital letter.

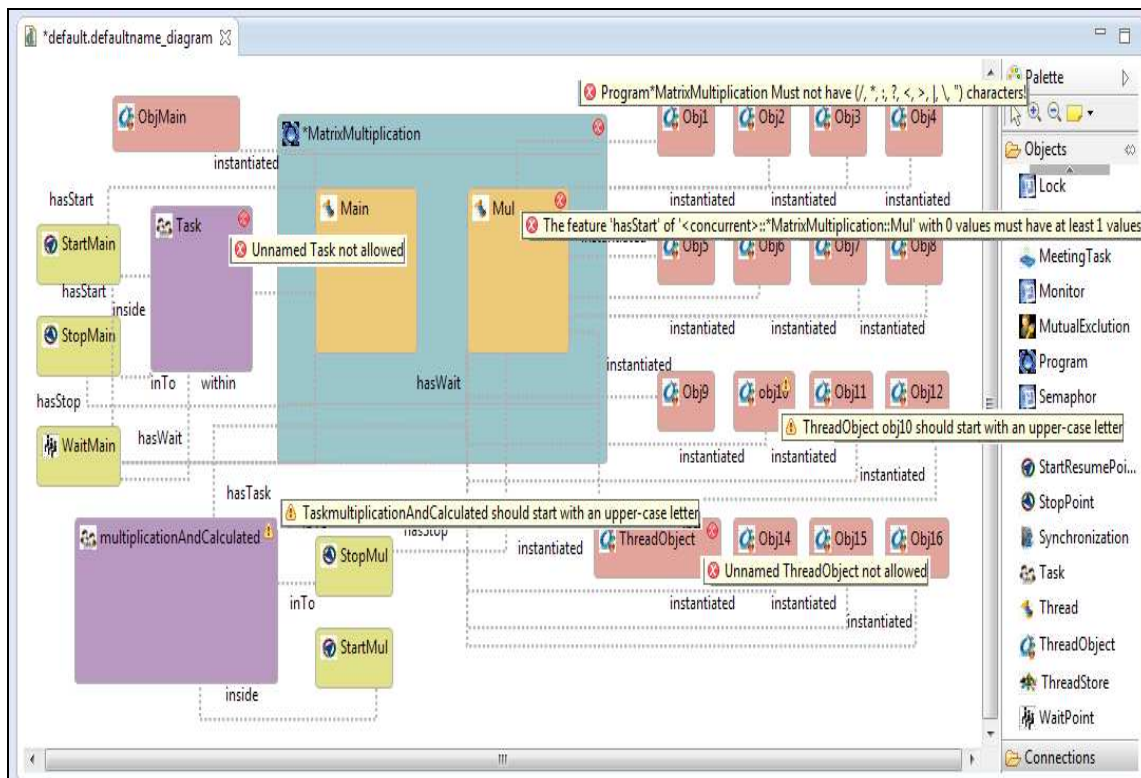


Figure 8. Applying constraints over the Readers/Writers problem model

Finally, to complete the procedure of development with DSML4CP, we need to generate architectural codes from the model for the Parallel Matrix Multiplication problem. The code generation for this case study is in C#. To this end, we have considered the model depicted in Figure 6 (in an XMI format) as the model for the Parallel Matrix Multiplication problem. With having the other input files (such as the metamodel), we performed generator.mwe in Xpand tool and achieved the results in C#, based on the rules given in Listing 3 for model transformation. In order not to have repetition, we have avoided demonstrating the rules here. Figure 9 illustrates generated artifacts for the Parallel Matrix Multiplication problem.

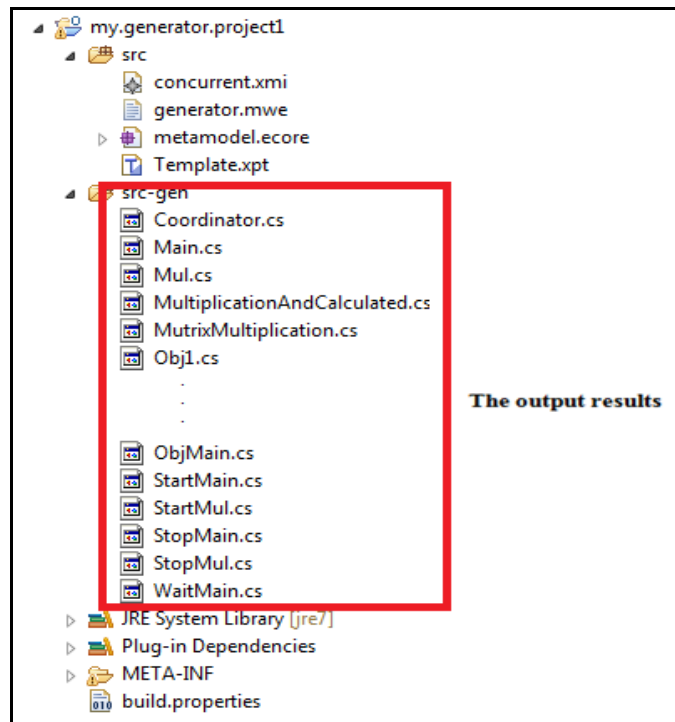


Figure 9. Generated artifacts for the Parallel Matrix Multiplication problem

It should be mentioned that to achieve the final running codes, the generated codes have been completed with manual codes (the developer's delta codes) which result in final codes for the case study. In Listing 6, part of final codes is shown for "Mul" element.

```

01 public class Mul : Thread
02 {
03     int[][] A; int[][] B; int[][] R;
04     int n, row, col;
05     public Mul(string s)
06     {
07         this.Name = s;
08     }
09     public void setMatrix (int[][] A, int[][] B, int[][] R, int n)
10     {
11         this.A=A;
12         this.B=B;
13         this.R=R;
14         this.n=n;
15     }
16     public void delay()
17     {
18         try
19         {
20             this.sleep(1000);
21         }
22         catch(Exception e)
23         {
24             Console.WriteLine(e.ToString());
25         }
26     }
27     public void setIndics (int r, int c)
28     {
29         row=r;
30         col=c;
31     }
32     public void run()
33     {
34         int i;
35         R[row][col]=0;
36         for(i=0;i<n;i++)
37             R[row][col]+=A[row][i]*B[i][col];
38         Console.WriteLine("[ "+int.ToString(row)+" ][ "+int.ToString(col)+" ] "+int.ToString(R[row][col]));
39     }
40 }

```

Listing 6. Output file for “Thread” in Mul element

It is worth to note that in the excerpt result presented in Listing 6, the Lines 1-3, 5-6, 16-26, 32-34, 36, and 39-40 are the codes that have been generated automatically and the Lines 4, 7, 9-15 and 27-30 have been added manually, as user delta code, to complete the development.

5. Evaluation and Analysis

In this section, we evaluate the DSML4CP language. To this end, we cover both evaluating the capabilities of DSML4CP and comparing it with other similar tools and languages. These are covered in this section and in the related work section respectively. In the following subsections, we show that developing a DSML increases the speed of development and decreases the error occurrence probability (in the domain) which leads to improve efficiency. We performed quantitative evaluation for DSML4CP, based on case studies with considering required time for development [69]. In addition, qualitative evaluation is realized for DSML4CP.

5.1 The Language Components Analysis

Quantitative analysis of DSML4CP components is performed in this section. For this purpose the abstract syntax, concrete syntax and steps of the code generation for DSML4CP are analyzed. Table 2 shows the specification of these parts. With considering the proposed metamodel in Figure 1, we can see that the metamodel has been created out of several meta-elements which are based on given issues in concurrency discussions. Each of these meta-elements symbolize a concept in a concurrent program. If a meta-element needs to be connected with another meta-element or with itself, there should be a link/relationship among them to let the connection happen. We have used different types of connections between meta-elements, such as association, composition, and inheritance. Table 2 shows the data for the above mentioned metamodel, editor, and code generation considering the number of meta-elements, links, attributes, and so on.

In short, all meta-elements and their links in the metamodel can be operated in the palette directly. By considering the results of quantitative analysis for language components, Table 2, it can be seen that, in DSML4CP, the number of the links in the palette has increased comparing to the metamodel. The reason is that we have used some inheritance relationships in the metamodel which make the possibility for all child meta-elements to inherit the links from their parent meta-elements. These child meta-elements have their own links when it comes to reflect the metamodel in the concrete syntax.

Table 2. Specification of Metamodel, Graphical editor and Code generation

Items	Metamodel			Graphical editor			Code generation	
	Meta-elements of the language	Attributes of the language	Links of the language	Nodes on the palette	Links on the palette	Attributes on the palette	Generated Lines of code	Generated Function/Module
Specification								
Number	22	29	33	22	43	29	260	21

5.2 Generation Evaluation

In section 4, we selected the Producer/Consumer and the Parallel Matrix Multiplication problems as our case studies and explained their implementation in details. In this part of the paper, we evaluate the language based on these two case studies to examine the generation performance of the language. It should be stated that all the data collected in the tables afterward are based on the average numbers provided from both cases. Also, the fraction numbers are rounded to make it easier for comparison.

To get the results of DSML4CP, we need some input files to complete the code generation step. Considering Figure 2, it can be seen that one of the input files is the XMI file in which we designed instance models in the graphical editor (Figures 3 and 6). Table 3 shows the specification instance models for the case studies.

The result of the code generation phase is an architectural code for the elements in several files. We have already shown some of these files in the case studies. Table 3 also illustrates the information for the generated artifacts. This table shows the number of generated files, the total lines of generated code, total generated functions, and the sum of the generated threads.

Table 3. Specification of input model, generated artifacts and added artifacts

Items	Input (Instance) Models					Generated Artifacts				Added (Delta) Artifacts		
Specification	Input model	Elements of the model	Links of the model	Attributes in the model	Logical errors	Files	Lines of Code	Functions	Threads	Lines of Code	Functions	Threads
Number	2	19	29	25	0	9	125	19	2	33	3	0

As it is mentioned earlier, to execute the artifacts, we need to add delta codes manually to the generated codes to make them functionally complete. Therefore, for both problems (Producer/Consumer and Parallel Matrix Multiplication), besides the generated codes for element, we have added a set of codes manually (developer’s delta code). Table 3 also shows the specification of these added codes. The data in Table 3 is demonstrated in Diagram 1 as a diagram to have a better understanding of the figures.

According to Diagram 1, considering both case studies, the results show that 79% of the final program code and 86% of the functions have been developed automatically, while just 21% of the codes have been written manually and 14% of functions have been added manually to the programs to be executable. All the threads in the models have been created automatically.

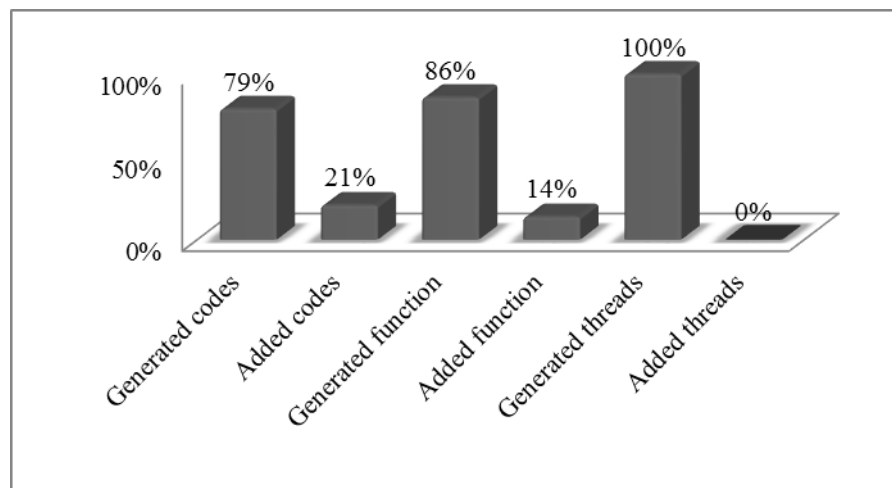


Diagram 1. Graphical representation of the performance for code generation of DSML4CP

By analyzing the information in Diagram 1, we can see that the majority of the codes have been generated automatically and a small amount of codes need to be written manually. This saves a significant amount of time in the development of the case studies which is examined in more details in the next subsection.

5.3 Development Time Evaluation

Given that the main purpose of this paper is the development of a DSML for reducing the complexity of concurrent programming and speeding up the development, we have evaluated the development times for the case studies both in the new language (automatically) and in the Java and C# programming languages (manually).

It worth to mention that we have considered two teams in the evaluation, one for using DSML4CP and the other one without using a DSML. Each of the teams is composed of four people to realize the evaluation. Evaluators of both teams were master students from software engineering department. They also had bachelor of computer engineering. They had passed “parallel computing (including

multithread topic)” and “advanced software engineering (including MDE)” among other courses. For the evaluation, three case studies are used: Producer/Consumer Problem, Parallel Matrix Multiplication problem, and Readers Writers problem. Members of the first team have developed the case studies using production tool for DSML4CP, and members of the second team have done the same task without using the modeling language and its framework. This team has developed the case studies in Java and C# environments.

Finally, the timing and coding information for both teams are collected and reported as the results. The results show that Average development times for the case studies are 23 minutes in DSML4CP and 39 minutes in Java and C#. Diagram 2 shows the development times of the artifacts in DSML4CP, Java and C# languages for the case studies.

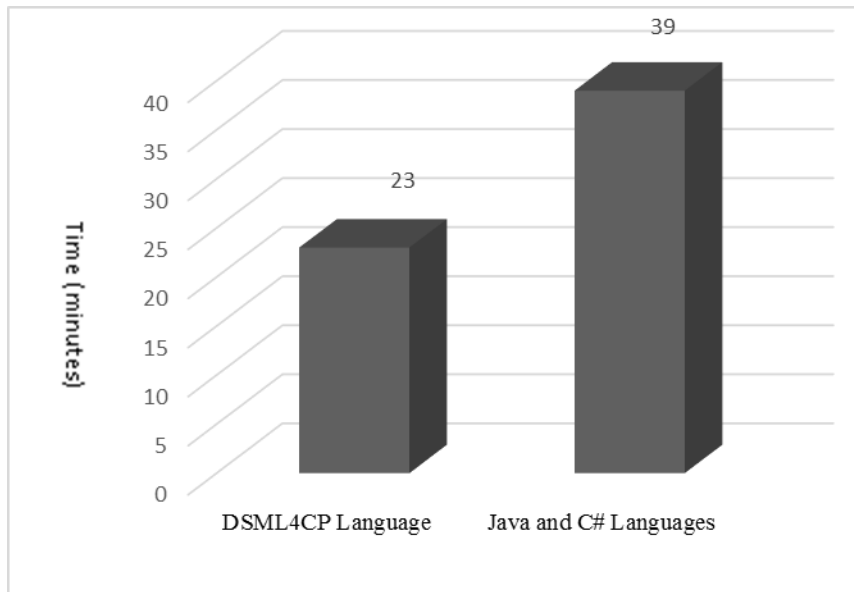


Diagram 2. Development times of the artifacts in DSML4CP, Java and C# languages for the case studies

As it can be seen in Diagram 2, development time in the new language, DSML4CP, is lower (almost by half) than of the one with Java and C# programming languages. This is because a big part of the code is generated automatically, based on Diagram 1. In some other studies also supporting results are presented. For example in [66] three controlled experiments are conducted to compare a DSL and a GPL and in all three experiments, the participants took less time to complete the DSL tests than they did to complete the GPL tests. As a result, the development using DSML4CP helps in increasing the speed of development for concurrent programs. Meanwhile, it decreases the probability of error occurrence due to automatic generation of the code which is error free and thus, leads to overall performance improvement.

5.4 Qualitative Evaluation

DSMLs brought changes to the software development with some key features. They are suitable for some specific domains for which they lead to increase development speed, decrease in the structural complexity, and improve the efficiency. Therefore, assessment of the quality of a modeling language is a significant part of this kind of research study. To this end, we evaluate the quality of DSML4CP by some of the success factors for the DSMLs suggested in [35]:

- Domain expertise

Development of DSMLs requires deep knowledge of the domain of interest. So, by focusing on that domain, we can achieve concepts and expressions (in addition to the rules and constraints of the system) with domain analysis [36]. After selecting a domain among possible choices, as the first step of developing the modeling language, we started to collect and process the information about that

domain. Information relevant to our domain is obtained from various sources such as domain experts, and technical documents which feed our metamodel.

- Targeting the domain

Determining a target for the domain is vital for developing a DSML. If the domain becomes so wide, there is no clear target for the domain. Also, if it is too narrow, the generation performance will be low. Certainly, these cases are not proper for developing DSML [36]. Therefore, based on the concepts that is considered for the concurrent programming domain, (such as synchronization, threads, mutual exclusion, deadlock, and so on), we have tried to develop a language to decrease the complexity of concurrent programs. In this manner, our target domain is clearly specified as concurrent programming in Java language.

- Effective metamodel

To develop a DSML, one should carefully provide the metamodel used for defining the domain's concepts and relations. Providing an appropriate metamodel requires that proper concepts to be extracted from the domain, e.g. from many sample programs in the target language or from the language documents. This is the vital pre-condition for the accurate artifact generation in any DSML. In contrast, an improper metamodel will lead to vague or incorrect specification which cannot be effective in the production for the domain [36]. In this case, we chose our metamodel based on existent concepts in concurrent programs from the experts and the texts, so that, each meta-element reflects a concept in concurrent programming.

- Effective supporting tools

Developing a DSML is a difficult task which needs expertise both in language engineering and in the targeted domain. Considering the earlier one, there is also a need to have technical knowledge of using some tools for development of a modeling language which can automatically do many tedious tasks of development process of a domain (for example, analyzing, designing, code generation, etc.). As indicated in the previous sections, we have also used several tools for developing our modeling language in order to facilitate the process of our domain [37]. With the help of these tools, we could create a graphical language and design various instances in it.

- High level of abstraction

For a DSML, high level of abstraction is a key feature, basically coming from the MDD definition, because it causes the improvement in the efficiency and quality, and reduces the level of complexity. Adoption of each instance model form its metamodel which is validated by the tools guarantees to have structurally accurate models for the domain. Applying the semantic control over the instance models help to have meaningfully accurate models [37]. As a result, a DSML, say DSML4CP, forces the designer to design accurate models which can help to generate accurate artifacts, with having right transformation rules. This leads to error free architectural codes which is result of working in a higher level of abstraction from code, namely model.

6. Related Work

In this section, we state the related studies. To this end, DSLs and tools for concurrent programming are examined in subsection 6.1. Also, a comparison is made between the new language and the previous studies in subsection 6.2.

6.1 Languages and Tools for Concurrent Programming

General purpose languages such as Java [38], C# [39], and Ada [40] support concurrent programming. They have structures for creating threads and their control. Also, there are worthwhile studies in which MDE [65] has been used for parallelism. For example, in [57], the authors use model driven approach to provide a framework, called PPMoel, for separating parallel blocks of a program, mapping them to different platforms, and executing the entire program in a parallel way. In [58], meta-programming is utilized for the domain of high performance computing (HPC) by introducing a

DSL, called SPOT, for HPC in FORTRAN. The goal is providing source-to-source translation of FORTRAN programs with the consideration of software maintenance and evolution in HPC systems. The study in [59] proposes a DSL to obtain the Application-Level Checkpoints (ALC) for distributed and parallel applications from end-users. Its aim is reengineering of existing applications to insert ALC mechanism in the program. Also, in the study of [60], integration of several modern software engineering tools and techniques, such as DSL and meta-programming, are used in an infrastructure called Hi-PaL to determine the methods that have the potential of reducing the parallelization complexity. They address explicit parallelization in their study. However, the goal of none of the above mentioned studies is modeling the problem domain of concurrent applications and generating the architectural code from the models using MDD tools and techniques.

On the other hand, there are specific concurrent programming languages (e.g. Alef [41], Axum [42] and Concurrent Pascal [43]) that provide programmers with more facilities in this regard.

Alef [41] is a concurrent programming language designed for system software. It provides exception handling, process management, and synchronization primitives. It supports both programming with shared variables and message passing. Statements in this language have a syntax similar to C, however the type system is different. Alef provides OOP using static inheritance and information hiding. Nevertheless, it does not support garbage collection, thus programs need to handle it.

Axum [42] is a DSL for concurrent programming. It is based on the Actor Model which was initially developed by Microsoft. Axum is an OO language based on the .NET CLR with a syntax similar to C. As a DSL, it is designed to develop parts of a software application which fits to concurrency. However, it has sufficient general-purpose structures which leave no need for the programmers to use a GPL such as C# for the sequential portions of the program. The main concept in Axum is an Agent/Actor which is an entity runnable simultaneously with other Agents. Objects in a domain are not directly accessible from another object. To resolve this issue, Axum offers channels which can be considered as directed pathways for communication among agents.

Concurrent Pascal Language [43] is also a concurrent language that is based on shared memory and is developed from Pascal language. It has standard characteristics which are used in parallel programs. Although this language is based on Pascal in which the processes are executed serially, but it provides the possibility of writing concurrent programs. Since this language is imperative, related simulators make the possibility of programming based on a shared memory and message passing.

Generally, the languages mentioned above give programmers the ability to produce concurrent programs. However, design and execution complexities are up to the programmer. As mentioned earlier, involvement with the problems of concurrent program's design and implementation such as thread coordination (in serial and parallel form) and their synchronization (e.g. mutual exclusion and deadlock control) in coding level is complex and erroneous, needing promotion of concurrent program's abstraction level. In this respect, some researchers tried to develop tools and methods for reducing such complexities.

For example, Convit [44] is a visualization tool based on Java provided for learning and understanding concurrent programming. This tool is a debugger and emulator that runs concurrent programs written in a simple pseudo-code. It is worth to note that the code is initially converted to Java code and then executed. Since Convit is based on Java applet, it can be used on any computer using a web browser. The syntax of this tool is similar to a combination of Pascal [45] and C [46] and can be understood by a person who has threshold knowledge of concurrent programming. Although Convit is a visualization tool, the tool has been proposed as a learning tool and is not a programming language. Therefore, it does not have the capabilities of a language (such as syntax and semantic controls).

Krystosik [47] used EMLAN language [48] and automatic features of DT-CSM [48] to evaluate concurrent program models. The aim was educational modeling and evaluation for concurrency and cooperation of tasks. As a result, concurrent program's error detection becomes possible though it is a difficult task. But despite using MDA, the aim of this approach is not producing code from model and decreasing structural complexities of concurrent programs.

Another work realized by Artho et al. [49] offers a method based on ordinal UML diagrams [50] to illustrate concurrent programs. This method facilitates analysis and design of concurrent programs, helping error detection. But, in this study, not only no language is offered for semantic control, but also, no tool is provided for using these models.

Visual language of VISO [51] is designed for making concurrent programs, based on OCCAM language [52] [53]. In this language, concurrent programs are designed in several abstract layers in modular form. VISO does not support shared memory and uses message passing for communication between the processors. Semantic structures of this language are provided in Petri net [54] and process calculus [55] [56]. But, this language does not support final code generation for other languages such as Java and C#.

In [61], Bull and Kambites has defined an OpenMP like interface for Java which enables a high level approach to shared memory parallel programming. As regards, OpenMP has been a relatively new industry standard for shared memory parallel programming, it is capable of increasing levels of support from both users and vendors in the HPC field. In addition, this standard defines a set of directives and library routines for both FORTRAN [62] and C/C++ [63], and provides a higher level of abstraction for programmers. In this study, although the possibility of writing parallel programs on a shared memory, which was originally difficult, became possible and implemented in Java. But in this study, not only no tool is offered to design the programs as graphical models; but also no facilities were considered for semantic control.

Java has provided a mechanism for concurrent programming implemented as language constructs, but it is too rudimentary for most programmers and includes a number of limitations. In research work [64], researchers have implemented Java4P, an extension of the Java language, which offers a simpler concurrency model and overcomes Java's limitations. In this study, although semantic controls have been taken into consideration, but they are not providing a tool to support using models.

As it is discussed earlier in this section, each of the tools and languages has its shortcomings comparing to DSMLs. Not only can a DSML make the possibility of graphical modeling for concurrent programs and controlling them while designing, but it can also promote semantic level of the language by turning one model into another peer model in another language and finally generate the code which is executable in the well-known target languages. In this case, the speed of development increases and error occurrence reduces, resulting in the improvement on the quality of the procedure and the product.

6.2 Comparison

Considering the tools and languages specific for concurrent programming which have been discussed in the previous subsection, it can be stated that although these languages give more facilities to the programmer to make use of them in creating a concurrent program, they only provide a textual syntax for the programmers without considering domain specific constraints. Thus, the complexity of the design and implementation of the program is the responsibility of the programmer.

Table 4 shows that DSML4CP provides an environment which reduces the complexity of concurrent programs, so that, the users are able to design and implement concurrent programs by creating domain models. In addition, as it can be seen in this table, DSML4CP has most of the capabilities which are provided by the other tools and languages in the domain with an extra capability of architectural code generation. Despite all the advantages provided by the DSML4CP, it

should be mentioned that it does not have the capability of controlling the program while the program is running which is an open research issue for the researchers.

Also, none of the above mentioned studies are based on MDA with the aim of code generation. On the contrary, DSML4CP not only can provide the possibility of modeling a concurrent program in a graphical form, but also it can do adequate controls during the design with which it improves semantics of the language. In addition, the speed of development increases by automatic generation and the efficiency rises by reduction of errors. A summary of discussed properties is given in Table 4 for the comparison between DSML4CP and the other tools and languages.

Table 4. Comparison of the new language with the previous studies

Studies Features	Previous studies									New DSML
	Study of Phil [41]	Study of Philips [42]	Study of Hansemp [43]	Study of Järvinen et al. [44]	Study of Krystosik [47]	Study of Artho et al. [49]	Study of Mulhem et al. [51]	Study of Bull et al. [61]	Study of Nugroho et al. [64]	
Language/Tool	Alef	Axum	Concurrent Pascal	Convit	-	-	VISO	JOMP	Java4P	DSML 4CP
Base language	C	C	Pascal	Pascal & C	EMLAN	UML	Occam	Java	Java	Java
Debugging and synchronization capabilities	+	+	+	+	+	+	+	+	+	+
Visualization capabilities	-	-	-	+	+	+	+	-	+	+
Capability of Control run time	+	-	-	-	+	+	-	+	-	-
Capability of Semantic Controls	-	-	-	-	+	-	+	-	+	+
Capabilities of code Generation	-	-	-	-	-	-	-	-	-	+

7. Conclusion

In this paper, concurrent programming has been addressed as a domain for developing a DSML, called DSML4CP. To this end, with regard to concurrency concepts and relations among them, a metamodel has been provided as the abstract syntax of the domain. Also, concrete syntax of the language has been prepared as a graphical tool, which provides the possibility of designing concurrent programs in the form of models in DSML4CP. It must be mentioned that for clarifying the meaning of this new language, static semantic controls for DSML4CP, in the frame of a series of constraints are considered, using EVL. Finally, for completing the language, code generation is fulfilled based on the abstract syntax, the instance model, and using the transformation rules in Xpand language. An evaluation based on case studies show a high efficiency forth proposed language. Regarding conducted evaluation on two case studies (Producer/Consumer problem and Parallel Matrix Multiplication problem), 79% of the final code and 86% of the functions are generated automatically and only 21% of the final code and 14% of the whole functions are added manually to have both case studies ready to run.

Although there are many advantages for DSML4CP and it is worth developing this DSML, there are some challenges in its development. These challenges are generic for DSMLs and DSML4CP is

not an exception in this regard. Underneath, we address three of them to convey our experiences and lessons learnt:

- High expenses of language production:

DSML development has different steps in which various technologies and tools are used. For this reason, developing a DSML is costly and time consuming. It should be considered that developing a DSML should return the investment at some point. Multi-thread programming is a very popular way of concurrent programming. This technique is supported by many generic languages such as Java which are used by many developers. So, considering the number of users, developing a DSML for concurrent programming can return the investment by reducing time and a number of errors for many developers.

- Need for high expertise:

To develop a DSML for a domain, such as concurrent programming, we need to master both the domain and DSML development. From one side, the language designer should be well informed of the domain details; from the concepts and their relations to the implantation and architectural structures. From the other side, the language developer should fully cover the knowledge of the modeling language development, in this case DSML development. Bringing these two expertise together is a difficult task, since it demands a high expertise.

- Expenses for learning a new language:

Developing a multi-thread program without using domain specific tools needs a limited level of expertise, since we can train a developer to do it in a single language. So, the developer who is going to implement a multi-thread program should only be trained in this field. However, if the developer is going to use a DSML (for instance DSML4CP) to use in development, he/she should not only get related training for multi-thread program, but they also must learn how to use domain specific tools. This requires some time and effort to start up a team for developing the programs using DSMLs. Although this time and effort will be retaliated in the later phases of development (e.g. problem domain analysis, design, and implementation), it will increase the learning curve of the language in the first development.

- Need for programming skills and domain knowledge:

To gain the benefits (e.g. performance) of the proposed DSML in this paper, the users need to have programming skills in general and parallel/multithread programming experience (i.e. domain knowledge in the context of this paper) specifically. This is because the automatically generated codes should be completed with the users to have final codes. In addition, to use the proposed DSML the users need to have familiarity with its tool which is discussed in the previous item.

Finally for the further studies, based on the proposed metamodel, it is planned to develop a DSML in Platform Independent Metamodel (PIMM) level; so that, the user will be able to generate codes for different programming languages from the same model. In addition, since our domain is a dynamic domain, we can apply dynamic controls and model checking (using formal methods) alongside static controls to increase the quality of instance models which increases the quality of the final generated artifacts.

Acknowledgment

This study is partially supported by Islamic Azad University of Shabestar, department of computer engineering in the scope of a Master thesis. The authors would like to acknowledge Vahid Khalilpour Akram for his helpful comments and suggestions for preparation of the domain concepts in this study. Also, Elaheh Azadi Marand and Elham Azadi Marand would like to acknowledge Moharram Challenger, their M.Sc thesis supervisor, for all his guidance and support during their thesis studies.

References

- [1] S. J. Hartly, "Concurrent Programming: The Java Programming Language", Oxford University Press, Inc., New York, 187 pages, 1998.
- [2] D. Lea, "Concurrent Programming in Java: Design Principles and Pattern", 2nd Edition, Addison-Wesley Professional, 1992.
- [3] R. N. Taylor, D. L. Levine and Ch. D. Kelly, "Structural Testing of Concurrent Programs", IEEE Transactions on Software Engineering, Vol. 18, No. 3, pp. 206- 215, 1992.
- [4] D.C. Schmidt, "Guest Editor's Introduction: Model-Driven Engineering," IEEE Computer, Vol. 39 No. 2, pp. 25-31, 2006.
- [5] M. Mernik, J. Heering, and A. Sloane, "When and how to develop domain-specific languages", ACM Computing Surveys, Vol. 37, No. 4, pp. 316-344, 2005.
- [6] J. Sprinkle, M. Mernik, J.P. Tolvanen and D. Spinellis, "Guest Editors' Introduction: What Kinds of Nails Need a Domain-Specific Hammer?", IEEE Software, Vol. 26, No. 4, pp. 15-18, 2009.
- [7] N. Oliveira, M. Joao, V. Pereira, P. R. Henriques and D. D. Cruz, "Domain-Specific Languages: A Theoretical Survey", The 3rd Compilers, Programming Languages, Related Technologies and Applications (CoRTA2009), Lisbon, Portugal, pp. 35-46, 2009.
- [8] M. Mernik, V. Žumer, "Incremental programming language development", Computer Languages, Systems & Structures, Vol. 31, No. 1, pp. 1–16, 2005.
- [9] J. H. Hill, "Measuring and Reducing Modeling Effort in Domain-specific Modeling Languages with Examples", The 18th IEEE International Conference and Workshops on Engineering of Computer-Based Systems (ECBS 2011), Las Vegas, USA, pp. 120-129, 2011.
- [10] S. Kelly and J. P. Tolvanen, "Domain-specific Modeling, Enabling Full Code Generation", IEEE Computer Society Publications, 446 pages, 2008.
- [11] M. Challenger, S. Getir, S. Demirkol, and G. Kardas, "A Domain Specific Metamodel for Semantic Web enabled Multi-agent Systems", Lecture Notes in Business Information Processing (LNBIP), Vol. 83, pp. 177-186, 2011.
- [12] S. Demirkol, M. Challenger, S. Getir, T. Kosar, G. Kardas and M. Mernik, "A DSL for the Development of Software Agents Working in the Semantic Web Environment", Computer Science and Information Systems, Vol. 10, No. 4, pp. 1525-1556, 2013.
- [13] E. Azadi Maran, M. Challenger and V. Khalilpour, "A survey on the methods and tools for development of Domain-specific Modeling Languages (DSMLs)", National Conference on Electrical and Computer Engineering, Islamic Azad University, Sarvestan Branch, Iran, pp. 1-6, (in Persian), 27 Feb 2013.
- [14] M. Challenger, S. Demirkol, S. Getir, S., M. Mernik, G. Kardas, G., T. Kosar, "On the use of a domain-specific modeling language in the development of multi-agent systems", Engineering Applications of Artificial Intelligence, Vol. 28, pp. 111–141, 2014.
- [15] H. B. Saritas, and G. Kardas, "A model driven architecture for the development of smart card software", Computer Languages, Systems & Structures, Vol. 40, No. 2, pp. 53-72, 2014.
- [16] D. S. Frankel, "Model Driven Architecture: Applying MDA to Enterprise Computing", Wiley Publishing, Inc., Canada, 355 pages, 2003.
- [17] Ch. Raistrick , P. Francis, J. Wright , C. Carter and I. Wilkie, "Model Driven Architecture with Executable UML", Cambridge University Press, 412 pages, 2004.
- [18] M. Voelter, S. Benz, C. Dietrich, B. Engelman, M. Helander, L. Kats, E. Visser and G. Wachsmuth, "DSL Engineering", 558 pages, Available online at: <http://dslbook.org>, 2013.
- [19] E. Azadi Marand, E. Azadi Marand, M. Challenger, "A Textual Tool for Concurrent Programming", International Research Journal of Applied and Basic Sciences, Vol. 8, No. 9, pp. 1271-1275, 2014.
- [20] F. Tang, I. You, S. Yu, Ch. L. Wang, M. Guo and W. Liu, "An efficient deadlock prevention approach for service oriented transaction processing", Computers and Mathematics with Applications, Vol. 63, No. 2, pp. 458-468, 2012.
- [21] W. Stallings , "Operating Systems Internals and Design Principles", 7th Edition, Pearson Education, 816 pages, 2011.
- [22] T. Clark, P. Sammut and J. Willans, "Applied Meta Modeling a Foundation for Language Driven Development", available online at: <http://www.ceteva.com>, 224 pages, 2008.
- [23] D. Kolovos, L. Rose, A. G. Dominguez and R. Paige, "The Epsilon Book", 194 pages, Available online at: <http://www.eclipse.org/epsilon/doc/book>, 2013.
- [24] Xtext, available online at: <https://eclipse.org/Xtext/> (last access, Aug. 2015).
- [25] S. Wurmbrand, "Syntactic vs. Semantic Control", The 15th Workshop on Comparative Germanic Syntax, pp. 1-1-7, 2001.

- [26] I. S. Bajwa, B. Bordbar and M. G. Lee, "OCL Constraints Generation from Natural Language Specification", The 14th IEEE International Enterprise Distributed Object Computing Conference, pp. 1-10, 2010.
- [27] L. M. Rose, A. Garcia-Dominguez, J. R. Williams, R. F. Paige and F. A. Polack, "Hello World with Epsilon", The 25th International Conference on Software Engineering, pp. 1-8, 2011.
- [28] J. Oldevik, T. Neple, R. Grønmo, J. Aagedal and A. J. Berre, "Toward Standardised Model to Text Transformations", The European Conference on Model Driven Architecture Foundations and Applications (ECMDA-FA), Nuremberg, Germany, pp. 1 -15, 2005.
- [29] Acceleo, available online at: <http://www.eclipse.org/acceleo/documentation/> (last access, Aug. 2015).
- [30] Java Emitter Template (JET): <https://www.eclipse.org/modeling/m2t/?project=jet/> (last access, Aug. 2015).
- [31] Xpand documentation, available online at: http://ditec.um.es/ssdd/xpand_reference.pdf/ (last access, Aug. 2015).
- [32] S. N. Mehmood, N. Haron, V. Akhtar and Y. Javed, "Implementation and Experimentation of Producer-Consumer Synchronization Problem", International Journal of Computer Applications, Vol. 14, No. 3, pp. 1-6, 2011.
- [33] H. Simonis and T. Cornelissens, "Modelling Producer/Consumer Constraints", The 1st International Conference on Principles and Practice of Constraint Programming (CP'95), 449-462, 1995.
- [34] T. Typou, V. Stefanids, P. Michailidis, and K. Margaritis, "Implementing Matrix Multiplication on An Cluster of Workstation", The 1st International Conference from Scientific Computing to Computational Engineering, Athens (IC-SCCE), Athens-Greece, 8-10 September, 2004.
- [35] J. P. Tolvanen and S. Kelly, "Defining Domain-Specific Modeling Languages to Automate Product Derivation: Collected Experiences", The 9th International Conference on Software Product Lines (SPLC 2005), pp. 198-209, 2005.
- [36] A. Kahlaoui, A. Abran and É. Lefebvre, "DSML Success Factors and Their Assessment Criteria", Metrics News, Vol. 13, No. 1, pp. 43-51., 2008.
- [37] H. Cho, "A demonstration-based approach for designing domain-specific modeling languages", The 26th Annual ACM SIGPLAN Conference on Object-Oriented Programming, System, Language, and Application (OOPSLA 2011), Portland, OR, USA, pp. 51-54, 2011.
- [38] D.J. Eck, "Introduction to Programming Using Java", Addison-Wesley Professional, 751 pages, 2011.
- [39] A. Hejlsberg, S.Wiltamuth and P. Golde, "The C# Programming Language", Addison-Wesley Professional, 644 pages, 2004.
- [40] M. Saaltink and S. Michell, "Ada 95 Trustworthiness Study: Analysis of Ada 95 for Critical Systems", ORA Canada, 304 pages, 1997.
- [41] W. Phil, "Alef Language Reference Manual", The Winter 1994 USENIX Conference, San Francisco, CA, USA, pp. 1-38, 1994.
- [42] Microsoft Corporation, "Axum programming language", available online at: <http://msdn.microsoft.com/enus/devlabs/dd795202.aspx/> (last access, Aug. 2015).
- [43] P. B. Hansen, "The Programming Language Concurrent Pascal", IEEE Transactions on Software Engineering, Vol. 2, pp. 199-207, 1975.
- [44] H. M. Järvinen, M. Tiusanen and A.T. Virtanen, "Convit, a Tool for Learning Concurrent Programming", In A. Rossett (Ed.), World Conference on E-Learning in Corporate, Government, Healthcare, and Higher Education, pp. 2220-2223, 2003.
- [45] M. V. Canneyt, "Reference guide for Free Pascal", Version 2.6.4, 2014.
- [46] T. Bailey, "An Introduction to the C Programming Language and Software Design", Addison-Wesley Professional, 153 pages, 2005.
- [47] A. Krystosik, "Model Checking in Concurrent Programming Teaching", The International Conference on Computer as a Tool, pp. 2390-2396, 2007.
- [48] A. Krystosik, "Embedded Systems Modeling Language", International Conference on Dependability of Computer Systems, PP. 27-34, Szklarska Poreba – Poland, May 25-27, 2006.
- [49] C. Artho, K. Havelund, and S. Honiden, "Visualization of concurrent program traces", Technical Report NII-2007-006E, National Institute of Informatics, Tokyo, Japan, pp. 1-6, 2007.
- [50] Ch. H. Kim, R. H. Weston, A. Hodgson and K.H. Lee, "The complementary use of IDEF and UML modelling approaches", Computers in Industry, Vol. 50, No. 1, pp. 35-56. 2003.
- [51] M. Mulhem and Sh. Ali, "Visual Occam: Syntax and Semantics", Computer Languages, Vol. 23, pp. 1-24, 1997.
- [52] M. Nosrati and R. Karimi, "Occam: A primary parallel programming language", World Applied Programming, Vol. 1, No. 1, pp. 85-88, 2011.
- [53] M. David, "Occam", Journal of ACM Transactions on Programming Languages and Systems, Vol. 18, No. 4, pp. 69-79, 1983.

- [54] M. K. Molloy, “A CAD tool for stochastic Petri nets”, ACM Fall joint computer conference, pp. 1082-1091, 1986.
- [55] J. C. M. Baeten and J. A. Bergstra, “Process Algebra with Signals and Conditions”, In M. Broy, eds., Summer School on Programming and Mathematical Methods, Mark Toberdorf, NATO ASI Series F88, pp. 273-323, 1991.
- [56] J.A. Bergstra, J.W. Klop, and J.V. Tucker, “Process Algebra with Asynchronous Communication Mechanisms”, In S.D. Brookes, A.W. Roscoe, and G. Winskel, eds., Seminar on Semantics of Concurrency, Pittsburgh, pp. 76-95, 1985.
- [57] F. Jacob, J. Gray, J.C. Carver, M. Mernik, P. Bangalore, “PPModel: a modeling tool for source code maintenance and optimization of parallel programs”, Journal of Supercomputing, Vol. 62, pp. 1560-1582, 2012.
- [58] S. Yue, J. Gray, “SPOT: A DSL for Extending FORTRAN Programs with Metaprogramming”, Advances in Software Engineering, vol. 2014, No. 917327, 2014.
- [59] R. Arora, P. Bangalore, M. Mernik, “A Technique for Non-invasive Application-Level Check pointing”, Journal of Supercomputing, Vol. 56, pp. 227-255, 2011.
- [60] R. Arora, P. Bangalore, M. Mernik, “Tools and Techniques for Non-invasive Explicit Parallelization”, Journal of Supercomputing, Vol. 62, pp. 1583-1608, 2012.
- [61] J. M. Bull and M. E. Kambites, “JOMP—an OpenMP like Interface for Java”, The ACM Conference on Java Grande, pp. 44-53, 2000.
- [62] OpenMp Architecture Review Board. OpenMP FORTRAN Application Program Interface, Version 1.1, Available online at: www.openmp.org (last access, Aug. 2015).
- [63] OpenMp Architecture Review Board. OpenMP C and C++ Application Program Interface, Version 1.0, Available online at: www.openmp.org (last access, Aug. 2015).
- [64] L. E. Nugroho and A.S.M. Sajeev, “Java4P: Java with High-Level Concurrency Constructs”, The 4th International Symposium on Source, pp. 1-6, 1999.
- [65] A. Rodrigues da Silva, “Model-driven engineering: A survey supported by the unified conceptual model”, Computer Languages, Systems & Structures, Vol. 43, pp. 139-155, 2015.
- [66] T. Kosar, M. Mernik, J. C. Carver, “Program comprehension of domain-specific and general-purpose languages: comparison using a family of experiments”, Empirical Software Engineering, Vol. 17, No. 3, pp. 276-304, 2012.
- [67] S. Getir, M. Challenger, and G. Kardas, “The formal semantics of a domain-specific modeling language for semantic web enabled multi-agent systems”, International Journal of Cooperative Information Systems, Vol. 23, No. 3, pp. 1-53, 2014.
- [68] M. Challenger, M. Mernik, G. Kardas, T. Kosar, “Declarative specifications for the Development of Multi-agent Systems”, Computer Standards & Interfaces, 2015, DOI: 10.1016/j.csi.2015.08.012 (in press).
- [69] M. Challenger, G. Kardas, B. Tekinerdogan, “A Systematic Approach on Evaluating Domain-specific Modeling Language Environments for Multi-agent Systems”, Software Quality Journal, 2015, DOI: 10.1007/s11219-015-9291-5 (in press).