Modeling Data Races Using UML/MARTE Profile

By

Akshay K.C.
Dept. of I&CT
MIT, Manipal

Ashalatha Nayak
Dept. Of CSE
MIT, Manipal

Balachandra
Dept. of I&CT
MIT, Manipal

Presentation Outline

- Introduction
- Summary of Related Work
- Objectives
- Background
- Methodology
- Conclusion and Future Work
- References

Introduction I

- Model is an abstract view of the system to be developed.
- What is a Model-based software Development?
- It is an efficient method to develop software based on the models.
- It helps in identifying the missing requirements, especially while developing the creative or innovative products.

Introduction II

- Unified Modeling Language(UML) is the standardized modeling language in the field of Object-Oriented Software Development.
- UML is extensible with two mechanisms: *Profiles* and *Stereotypes*
- Profiles are special kind of packages which provide extended modeling mechanism, defined through Stereotypes.
- Stereotype is one of the extensibility mechanism which allows the designers to extend the vocabulary of the UML in order to give new model elements.

Introduction III

- UML diagrams are used to model both static and dynamic natures of the system.
- Sequence Diagram is of much more concern as it represents the dynamic nature of the system in an interactive way.
- Sequence diagram can be interpreted in a self-explanatory way in two dimensions as below:
  - *Vertical:* It shows the messages passed between the objects in the time order in which they occur.
  - *Horizontal:* It provides the object instances between which the message transfer takes place.

Introduction IV

- Concurrency is a property of systems where more than one activities or computations are being carried out simultaneously.
- Problems in Concurrent Systems: Data Race, Deadlocks or Starvation.
- **Data Race:** A race exists between two events if they conflict i.e one reads and other writes into the memory location.
- Identification of Data race is very important! Else it might lead to any inconsistent state.

Summery of Related Work I

- Works exist that are related to monitoring temporal properties of concurrent systems which either make use of
  - The implementation/source code of the system under test by Chen et.al, Flanagan et al, Lei and team [1, 2, 3]
  - Formal languages[4]

- In the existing works by Chen et.al[1] and model checkers such as Java Path Finder[4], Data races are not identified early in the development phase.

- The elements or variables which result in data race scenarios are not preserved any time which might result in loss of data or information about the components used.

Summery of Related Work II

- Most of the works like[1, 4, 3] are static in nature and has manual task of detecting the data race.
- The dynamic nature of the system is not modeled explicitly by making use of any behavioural diagrams like sequence diagram.
- Concurrently executing methods are not properly handled by specifying the valid and non valid execution paths until the testing phase.

Summery of Related Work III

- Demathieu et.al[6] probe how the UML/MARTE profile can be used for representing the real time concepts in software engineering by taking a case study of a fictive system Josefil Challenge.
- A framework for executing the UML models and its underlying model transformations with respect to MARTE standards has been discussed by Mraidha et. al [5].
- Using this iterative method the authors have tried to implement the MARTE profile for the real time and embedded system like cruise control system.

Summery of Related Work IV

- Deadlocks and Starvation by Shousha et al[7].
- Information is extracted and fed into the Genetic Algorithm with a certain fitness function tailored to deadlock and starvation detection

Objectives

- To detect the data race in a system early in the development phase by making necessary specifications using the MARTE profile.
- To parse the model elements or the objects of the systems and to be preserved for the future usage.
- To automate and reduce the manual task of detecting data race.
- To use the behavioural diagrams like Sequence Diagrams for modeling the dynamic nature of the system.
- To detect the valid and invalid scenarios.

UML/MARTE Profile I

- The UML profile for MARTE is the specialization of UML to model the real-time applications and platforms[5].
- Profiles introduces new semantics to the models by extending the classes and their attributes.
- The UML profile for MARTE extends the capabilities of UML for real-time and embedded systems development for design and analysis purpose.

UML/MARTE Profile II

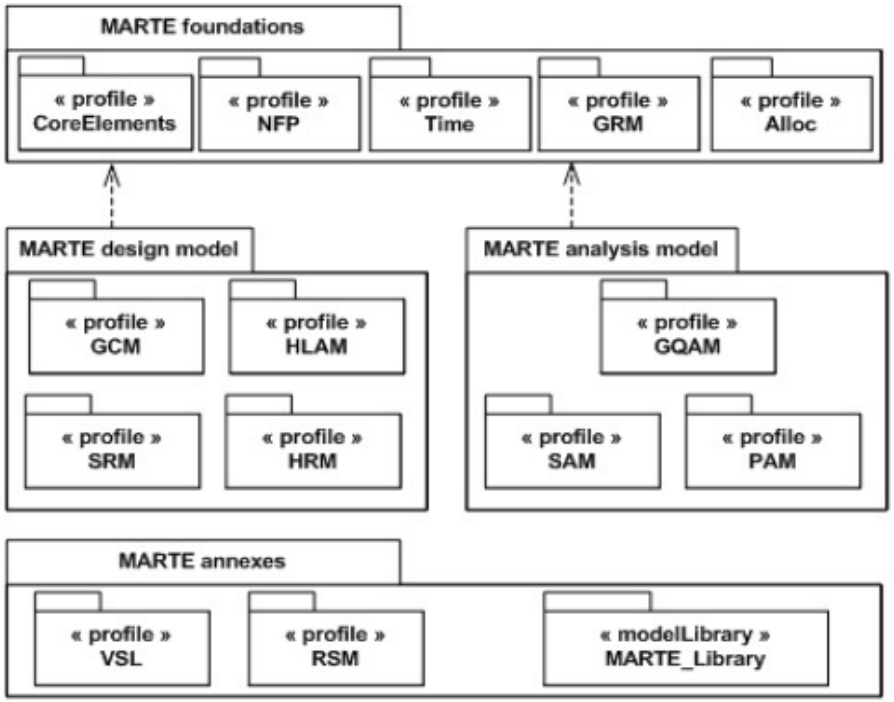


Figure: MARTE Architecture

MARTE Architecture

a.) MARTE foundation which contains the basis for real-time and embedded system modeling. It defines time concepts and use of concurrent resources.

b.) MARTE design model specializes the foundation, allowing modeling of various features of real-time and embedded systems.

c.) MARTE analysis model allows the annotation of models for system analysis purposes.

UML/MARTE Sub-Profiles I

Table: *SW_Concurrency* package and its attributes

		Description
Package	*SW_Concurrency*	Describes the entities for Concurrency
Stereotype	≪*SwConcurrentResource*≫	Represents the concurrently executing entities
Attribute	priorityElements	Determines the priority of the associated thread

UML/MARTE Sub-Profiles II

Table: Generic Quantitative Analysis Modeling sub-profile and its attributes

		Description
Sub profile	GQAM	factorizes common constructs
Stereotype	≪*gaStep*≫	used when decisions about the allocation of system resources is made
Attribute	*priority*	Determines the priority of the action on the host processor
	interOccTime	interval between multiple initiations of the action
	execTime	the execution time of the action

UML/MARTE Sub-Profiles III

Table: High-Level Application Modeling package and its attributes

		Description
Sub Profile	HLAM	Describes the services for real time constraints.
Stereotype	≪*RtServices*≫	Used to define real time constraints
Attribute	*concPolicy*	Determines the type of concurrency policy used for the real-time service
	Reader	Real-Time service for reading the data
	Writer	Real-Time service for Writing the data

UML/MARTE Sub-Profiles IV

Table: Sw_Interaction package and its attributes

		Description
Sub-Profile	SRM:SW_Interaction	Deals with communication and synchronization resources
Stereotype	≪*sharedDataComResource*≫	defines specific resources used to share the same area of memory among concurrent resources
Attribute	*waitingQueuePolicy*	Defines the algorithm to manage the resource waiting queue

Methodology I

- **Step 1:** In the initial step, the requirements for the scenario is gathered.
- **Step 2:** The scenario considered is modeled.
- **Step 3:** The sequence diagram is mapped on to the MARTE profile according to the scenario considered.
- **Step 4:** A <.uml> file is generated which contains the elements of the model.
- **Step 5:** Parse the <.uml> file.

Methodology II

- **Step 5.1:** If any Combined fragment is found during the parsing then the type of the combined fragment is identified.
- **Step 5.2:** If the identified combined fragment is par fragment then the number of operands in the par fragment is identified and also the messages in each operand of the par fragment is retrieved.
- **Step 5.2.1:** The valid scenarios are generated.
- **Step 5.3:** Check for the paths containing data race conditions.

Case Study I

- Assume a scenario in *withdraw_amount* usecase where two people are withdrawing amount from the same joint account simultaneously.
- The first deduction is a withdrawal from an ATM for $100 and that the second deduction is of $10 in another ATM.
- Assume the account has a total of $105 as the balance amount.
- Obviously, one of these transactions cannot correctly complete without sending the account into the low balance state.

Case Study II

1. *Sequential Transaction:* Consider bankClient2 transaction happens initially, $10 is subtracted to get a new balance of 95.
2. Then the ATM withdrawal of bankClient1 comes along and fails because $95 is less than $100.
3. *Simultaneous Transaction:* Both transactions verify that sufficient amount exists in the account i.e $105 is more than both $100 and $10.

Case Study III

4. bankClient1 withdrawal process subtracts $100 from $105, yielding $5.
5. The bankClient2 transaction then does the same, subtracting $10 from $105 and getting $95.
6. The withdrawal process of bankClient1 then updates the user's new total available funds to $5.
7. Now the amount transaction of bankClient2 also updates the new total, resulting in $95 providing the inconsistent balance amount.

Use case diagram for the scenario

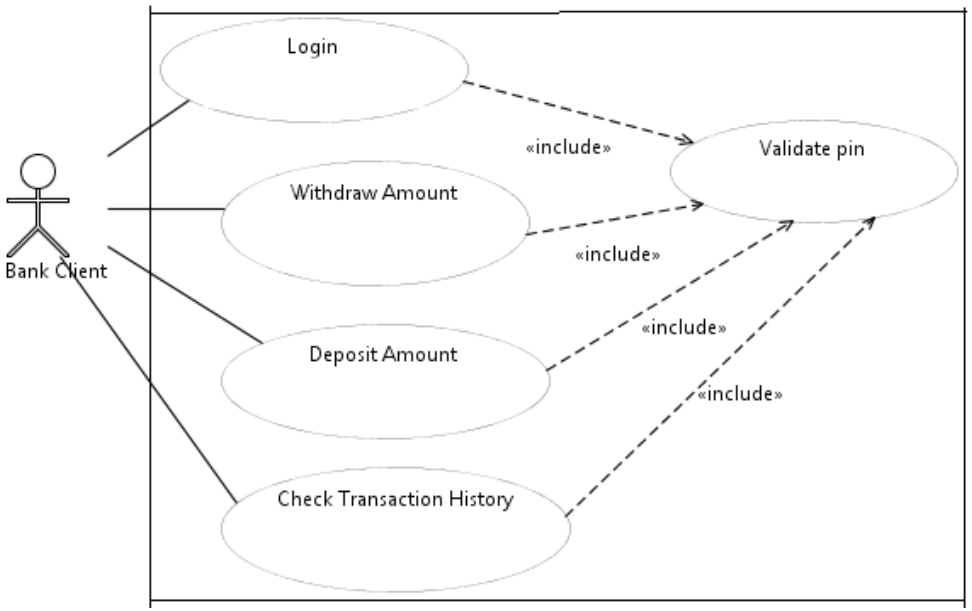
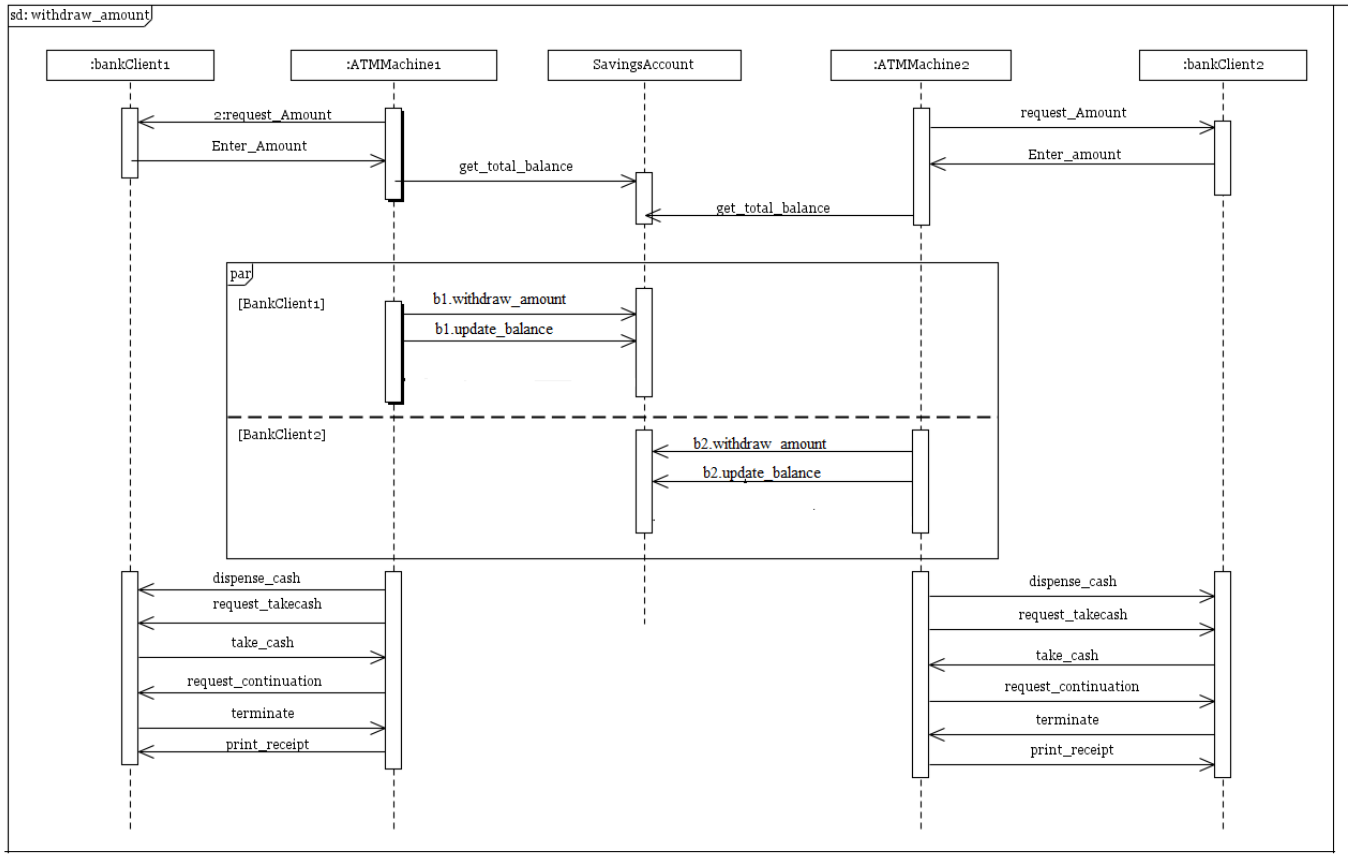
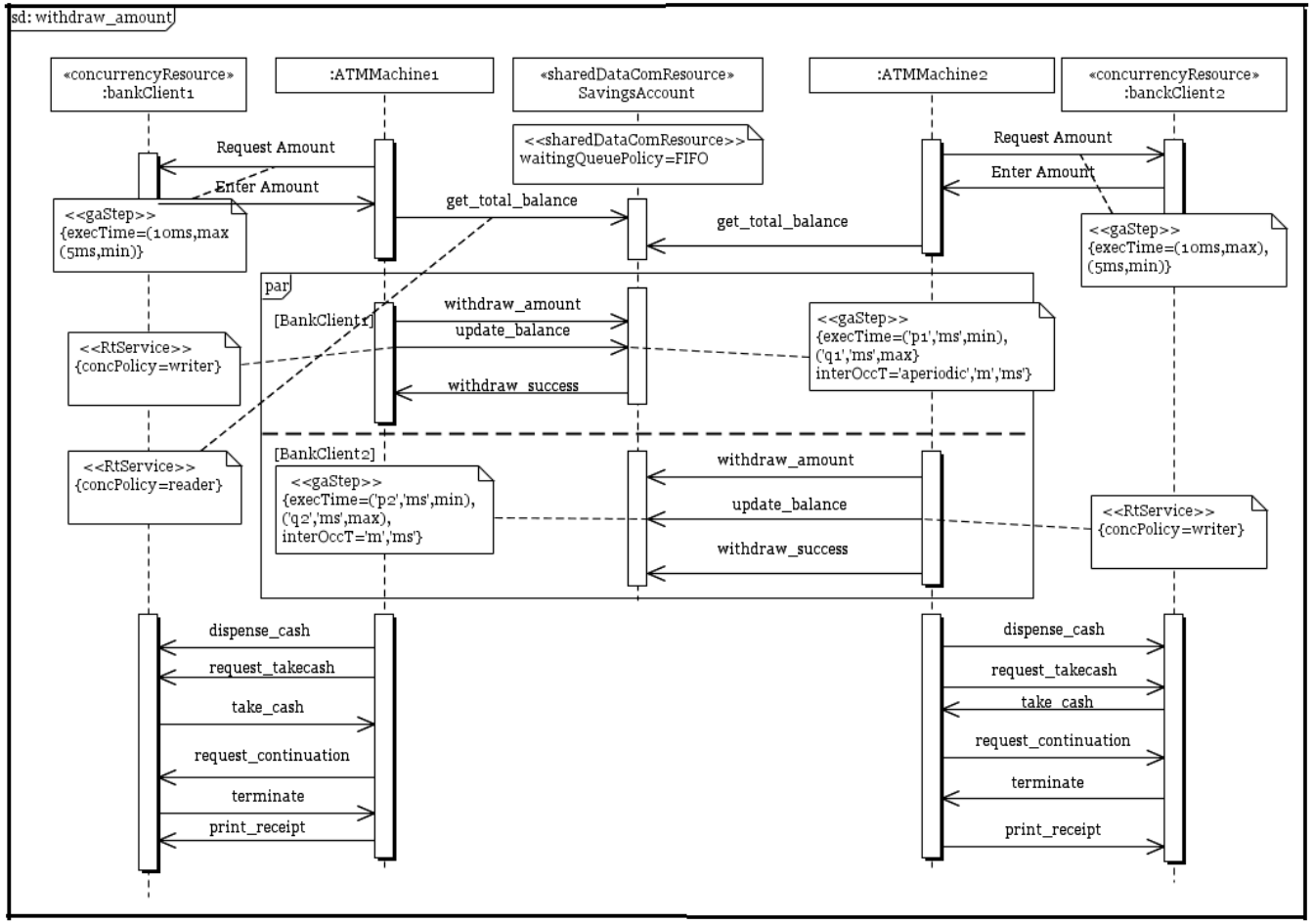


Figure: Usecase Diagram

Modeling the scenario using Sequence Diagram



Mapping MARTE profile to the UML Sequence diagram



Parsed Model Elements

- The auto generated <.uml> file is given as input to the parser.
- The algorithm begins to parse the elements of input file.

Reading the combined fragment

Algorithm 1 Combined Fragment parsing algorithm to identify and retrieve the messages from combined fragment

Precondition: The combined fragment is present in the model.
Postcondition: All the interaction operands and their respective messages are retrieved from the par fragment.
Input: Model elements of par fragment
Output: Array of all messages in par fragment
Local: *mid*, ▷ Denotes the message-id of j^{th} message in i^{th} interaction operand.
m[ ],▷ Denotes an array which contains the names of all the messages in i^{th} interaction operand.
msg[ ], ▷ Denotes an array which holds all messages of all the interaction operands in the par fragment.

1: Identify innermost *par* fragment and denote it as cfrag
2: Find the number of operands, N, in the cfrag
3: **for** $i \leftarrow 0$ to N in step of 1 **do**
4: Retrieve the oprname of each identified operand$_i$
5: **end for**
6: **for** $i \leftarrow 0$ to N in step of 1 **do**
7: Retrieve the message id, mid$_{ij}$ from each inner fragment of operand$_i$, j$\in$0 to M
8: **end for**
9: **for** $i \leftarrow 0$ to N in step of 1 **do**
10: **for** $j \leftarrow 0$ to M in step of 1 **do**
11: get message name for mid$_j$
12: store the name in an array, m$_i$[j]
13: msg[mcount++] $\leftarrow$ m$_i$[j] $\triangleright$ mcount gives the total number of messages in par fragment
14: **end for**
15: **end for**
16: call permutation algorithm

Table: Messages and their respective Operands

Operand Name	Message Name
InterctionOperand	b1.withdraw_amount
	b1.update_balance
InteractionOperand0	b2.withdraw_amount
	b2.update_balance

Finding Valid Scenarios

Algorithm 2 Permutation algorithm to find the valid scenarios of the messages retrieved from combined fragment

Precondition: The model element of the MARTE sequence diagram must be parsed and stored in an intermediate data structure.
Postcondition: Only the valid scenarios are retrieved from the par fragment.
Input: Model elements of par fragment
Output: List of all valid scenarios of the messages in par fragment **Local:** msg[ ], ▷ Denotes an array which holds all messages of all the interaction operands in the par fragment

1: **for** $i \leftarrow 0$ to mcount in step of 1 **do**
2: print msg[i]
3: **end for**
4: $i \leftarrow 0, j \leftarrow 0$ ▷ where i and j are counters or index values to access the messages of par fragment which are stored in msg[ ]
5: **while** i<N **do** ▷ where N is the total number of messages present in the par fragment.
6: **while** TRUE **do**
7: $j \leftarrow i + 1$
8: temp = msg[j] ▷ msg[i] depicts each message of different interaction operands in par fragment
9: msg[j] = msg[j+1]
10: msg[j+1] = temp
11: j++; i++

12: break
13: **end while** ▷ To print all the messages of msg[ ] after every single swap
14: **for** $k \leftarrow 0$ to mcount in step of 1 **do**
15: print msg[k]
16: **end for**
17: **end while** ▷ Loop ends when permutations are finished considering all the messages in the array

- Let M_{11}, M_{12} denote withdraw and update operation of bankClient1 respectively and M_{21}, M_{22} denote the corresponding operations of bankClient2 as shown below:

M_{11}: bankClient1.withdraw_amount(),
M_{12}: bankClient1.update_balance()
M_{21}: bankClient2.withdraw_amount(),
M_{22}: bankClient2.update_balance()

- Valid Scenarios:

$\{ M_{11}, M_{12}, M_{21}, M_{22} \}$,
$\{ M_{21}, M_{22}, M_{11}, M_{12} \}$,
$\{ M_{11}, M_{21}, M_{12}, M_{22} \}$,
$\{ M_{11}, M_{21}, M_{22}, M_{12} \}$,
$\{ M_{21}, M_{11}, M_{22}, M_{12} \}$,
$\{ M_{21}, M_{11}, M_{12}, M_{22} \}$

Analysing and Detecting the Data Race

Table: Description of attribute values for bankClient1 and bankClient2

Stereotype	Attributes	bankClient1 Time Period		bankClient2 Time Period		Remarks
		min	max	min	max	
≪gaStep≫	execTime	p1	q1	p2	q2	For p1<p2 and p2<q1, it is simultanoeus exectuion
						For p2>q1, it is sequential execution showing the ideal case.
	interOccT	m		M		Denotes the time within which the operation interleaving takes place
≪RtService≫	concPolicy	reader		reader		Denotes the read operation
		writer		writer		Denotes the write operation

Table: Test scenarios

Scenario-Id	Message format for scenario	Description
1	$\{M_{11}, M_{21}, M_{12}, M_{22}\}$	The scenario is inconsistent if p1<p2 and p2<q1
		The scenario is consistent if p1<p2 and p2>q1
2	$\{M_{11}, M_{21}, M_{22}, M_{12}\}$	The scenario is inconsistent if p2<p1 and p1<q1
		The scenario is consistent if p1<p2 and p2>q1
3	$\{M_{21}, M_{11}, M_{22}, M_{12}\}$	The scenario is inconsistent if p2<p1 and p1<q2
		The scenario is sequential if p2<p1 and p1>q2
4	$\{M_{21}, M_{11}, M_{12}, M_{22}\}$	The scenario is inconsistent if p1<p2 and p2<q1
		The scenario is consistent if p1<p2 and p2>q1
5	$\{M_{11}, M_{12}, M_{21}, M_{22}\}$	Sequential but inconsistent if p1<p2 and p2<q1.
6	$\{M_{21}, M_{22}, M_{11}, M_{12}\}$	Sequential but inconsistent if p2<p1 and p1<q2

Conclusion

- Concurrently executing methods can be properly handled early in the development phase.
- The modeling of the real time systems using MARTE is promising as it is possible to explicitly set the required attributes and properties of the participating concurrent objects.

Future Work

- The proposed approach can also be used to make the application more secure by identifying the inconsistent temporal constraints.
- For describing multiple scenarios it is required to extend the approach with complete system specifications that includes interaction overview diagram which focuses on the flow of control with each node as a sequence diagram.

References I

[1] Y. Chen, Y. H. Lee, W. Wong, and D. Guo, "A Race Cndition Graph for Concurrent Program Behaviour," *Proceedings of 3rd International Conference on Intelligent System and Knowledg Engineering*, pp. 662–667, 2008.

[2] C. Flanagan and S. N. Freund, "Detecting Race Conditions in Large Programs," Compaq System Research Center, Tech. Rep., 2001.

[3] B. Lei, L. Wang, and X. Li, "UML activty diagram based testing of Java Concurrent Programs for Data Race and Inconsistency," *Proceedings of 1st International Conference on Software Testing, Verification and Validation*, pp. 200–209, April 2008.

[4] S. Kulikov, N. Shafiei, F. van Breugel, and W. Visser. Detecting Data Races with Java Path Finder. [Online]. Available: www.cse.yorku.ca/~franck/research/drafts/race.pdf

[5] C. Mraaidha, Y. Tanguy, C. Jaouvray, F. Terrier, and S. Gerard, "An Execution Framework for MARTE-based Models," *13th IEEE International Conference on Engineering of Complex Computer Systems*, pp. 222–227, 2008.

References II

[6] S. Demathieu, F. Thomas, C. Andre, S. Gerard, and F. Terrier, "First Experiments using the UML profile for MARTE," *11th IEEE Symposium on Object Oriented Real-Time Distributed Computing(ISORC)*, pp. 50–57, 2008.

[7] M. Shousha, L. C. Briand, and Y. Labiche, "A UML/MARTE Model Analysis Method for Uncovering Scenarios Leading to Starvation and Deadlocks in Concurrent Systems," *IEEE transactions on Software Engineering*, vol. 38, no. 02, March/April 2012.

[8] L. C. Briand, Y. Labiche, and M. Shousha, "A UML/MARTE Model Analysis Method for Detection of Data Races in Concurrent Systems," *Model Driven Engineering Languages and Systems: Springer Publications*, vol. 5795, pp. 47–61, 2009.

Thank You!