Model-Driven Software Development Platforms Reviews

Ftoon Kedwan
School of Computing
Queen's University
Goodwin Hall, Kingston
Ontario, Canada

Chanderdhar Sharma
School of Computing
Queen's University
Goodwin Hall, Kingston
Ontario, Canada

ABSTRACT
The Model-Driven Software Development Systems (MDSDS) were initially developed as an attempt to increase software development productivity and quality. This is because focusing on the logical solution abstract is more important than focusing on the pure infrastructure technicalities. Developers discovered the abstracted modelling technique that includes both programming and platform tools in the same time, which is now referred to as MDSDS.

Nowadays, there are plenty of modeling software applications that almost achieve the same work, yet, the user might not be aware of the detailed nuances between them. This paper aims to discover the distinguishing features between four of the most commonly used MDSDS including; YAKINDU, Papyrus-RT, Rhapsody, and The State Machine Compiler (SMC). Analysis of the suitability of those platforms for modeling structural and behavioral domain specific software will be investigated.

The same model will be built using the four MDSDSs. Then, main differences, obstacles, observations, and overall experience quality using those four environments will be discussed. Some of the common distinguishing features to be explored is GUI intuitivism, user friendliness, clarity of commands and tools, tool learning time needed and learning curve, model building time consumed, etc.

General Terms
Model-Driven Software Development Systems, YAKINDU, Papyrus-RT, Rhapsody, The State Machine Compiler-SMC, Model-Driven Engineering

Keywords
Software Development Systems, Model-Driven Platforms, State Machine Engineering, YAKINDU, Papyrus-RT, Rhapsody, State Machine Compiler-SMC.

1. INTRODUCTION
The MDSDSs were initially developed as an attempt to increase software development productivity and overall quality. This is achieved through including only the important characteristics of a system in the resulted software solution with some degrees of abstraction [1].

Instead of using complex programming languages and machine codes, developers discovered the abstracted modelling technique that includes both programming languages and platform technologies in the same time [2]. This approach helped them design programs according to their logical solution abstract rather than focusing on pure technicalities. Among those MDSDSs is the SMC.

SMC is an event driven Java application, that uses state patterns to combine Finite State Machines (FSM) with their Objects. SMC enables the definition of default transitions that allows objects to handle unexpected events, instead of crashing and reporting an error. eBus is responsible for exchanging messages between SMC transitions and FSM objects, thus, it is considered as a Java middleware. This automatic message exchange and generation technique eliminated the need for manual transition matrices, state transition arrays, or the scattered switch statements. Instead, state diagrams are directly coded into the SMC Language. State diagrams in SMC can be easily imported into a file in order for State Pattern classes to be automatically generated [3].

YAKINDU Statechart (part of state machine formalism) is an integrated modelling environment that uses Statecharts to develop interactive event-driven systems. Systems that contain Components (Statecharts) and Instances of those Components who have Interfaces containing Ports of exchange. YAKINDU adopts both graphical (e.g. States, Transitions) and textual (e.g. Declarations, Actions) notations to model its state machines and diagrams.

It includes syntax and semantic checks to validate its models live whilst editing. It also employs simulation techniques for its state machine models to check their dynamic semantics to save the time consumed for repetitively debugging the models. YAKINDU's state machines code generators use Java, C and C++ to generate the codes [4].

Papyrus-RT is another MDSDS that is an open source toolkit for a cohesive modeling environment used for the development of complex real-time systems. It is based on the Unified Modelling Language, together with its Real-Time services (UML-RT) which consists of capsules made of active classes and their composite structures.

Those capsules can communicate through ports using protocols that specify the messages that can be swapped [5]. It also provides specific tools that facilitates the development and validation of UML-RT models. Papyrus-RT programmatic behavior is provided as a C++, C, or Java action code. The embedded code generator translates the UML-RT model's structural elements and the behavior provided by each capsule's state machine into C++ code [6].

Rhapsody is also a visual model-driven development environment based on UML-RT and used to create real-time systems. Software applications are built using graphical models programmed in many languages such as C, C++, Ada, Java and C#.

Rational rhapsody helps system engineers and software developers to automate the software development lifecycle and create model designs by understanding systems' requirements, validate functionality and identify defects and design errors beforehand, and finally delivers highly structured software. Therefore, it gives you the flexibility to work in your specific domain [7].

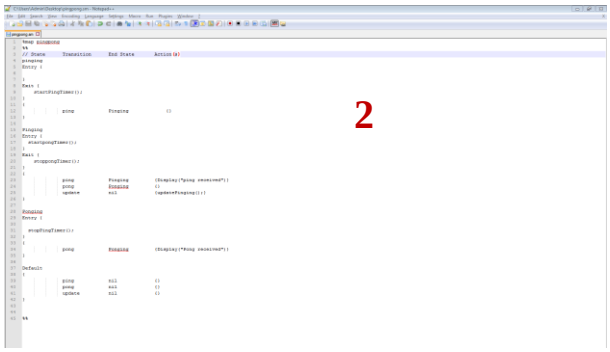

Figure 1 Ping-Pong Model Built in SMC Modelling Environment

Rhapsody was frequently investigated and tested by researches due to its remarkable capabilities and features in composite and orthogonal states, condition and junction connectors, and inter-level transitions [8].

From the above MDSDSs discerptions, it is clear how similar they are in structure, functionality, and the sought for deliverables. However, this was the core reason of this paper, which is to find out the little nuances between them to, help the user decide which MDSDS is going to provide them the ultimate satisfactory based on their specific domain and desired output. In the coming sections, the four MDSDSs are going to be explored through active experiments, and then, observations will be documented. The differences and similarities will be highlighted in the comparison table, and the findings will be discussed in further detail in section four. Finally, conclusion with general advices will take place, besides the provision of future work suggestions.

2. RELATED WORK

Researchers have been discussing MSDSs and UML-RT models in terms of their symbolic execution trees analysis, features modularity, besides the ability of reusing their analysis results. The Symbolic Analysis of UML-RT Models (SAUML) was also investigated as an effort to enhance the current practice of MDSDS with the analyses of UML-RT models 9], or for developing UML-RT modeling design rules and guidelines to find out the ideal implementation of complex real-time embedded software systems, and more importantly, to help improve the quality of those models by knowing what to avoid during software models building process [10], [11], [12].

Moreover, Model Driven Development tools that are known for their hierarchical organization and asynchronous transitions and communications have been extensively analyzed in terms of their semantics and syntax characteristics as an attempt to enhance their behavioral modeling capabilities (via enhancing state machine diagrams or statecharts), dimensions, and their final results quality [13], [14]. One drawback of UML-RT is that there is no significant support for models' debugging at model-level which is an important limitation for MDSD systems [15] This barrier leads model-driven software developers towards platform-independent solutions adding an extra dimension of complexity.

MDSDS capabilities and features are not the only things needed to be known to consider the selection of any particular modelling tool. What is more important is the availability of the high-quality supporting tools to those modeling software.

In 2017, a workshop was held specifically to discuss what effective supporting tools, materials, and documentations (e.g. Video tutorials) are available, and what are needed to be implemented and discovered. Research communities have realized that supporting MDSDSs software will significantly increase the chances of success and user-adoption for any new model development system [16].

There is no doubt that any beginner user will need supporting illustrative materials to understand the new system. Model driven development use cases proved to be highly beneficial, especially for scenario-based models [17], [18]. In fact, more evaluative and publicized MDSDS-based research results such as this paper should be accomplished more often for the sake of familiarizing the Model Driven Engineering (MDE) community with the available tools capabilities, especially the open-source kind of them. Such studies are useful to detect effective resolutions how to improve the users experience.

Potential resolutions might include changes to the tool itself or to supporting materials. For example, increasing GUI friendliness, adding facilitating features, or broadening the inputs/outputs formats. According to Whittle et al., most users' problems range around plug-ins issues, lack of online tutorials, or interoperability issues (i.e. importing/exporting files), among other issues as well [19]. Likewise, the more diverse modelling case scenarios the bigger the chances users are going to use these tools for various projects after realizing their suitability for their domain specific tasks [16].

Eclipse ecosystem is one of the first and most significant modeling tools supporters, while other supporters tend to be far more complex and immature. Eclipse offers an extensive variety of supportive tools, repositories, and information and documentation sources. It also supports several other aspects of modeling and MDE (i.e. model transformation) [20].

3. EXPERIMENT ANDMETHODOLOGY

3.1 SMC

The SMC is an open-source tool designed by JAVA, used to compile FSM projects. SMC is based on a robust and solid theoretical base, yet, it is not considered to be a new technology. It is known for its effective handling of Reactive and Transformational systems. Some of its famous applications include automobiles, avionics systems, and man-machine interfaces. The SMC code is generated to support the targeted application software. However, certain changes need to be made to the project's original code such as adding the SMC class definitions and source file into the targeted project file [3].

SMC is considered one of the simple modeling tools. It is mostly used in the industry to solve repetitive and tedious software development problems [21]. For example, it proved efficient performance in detecting patterns in a stream of characters, or finding tokens (e.g. words, numbers) inside the sentences. While writing the state classes for those problems would be long and frustrating, SMC views and edits the entire FSM problem logic easily in one single file.

The good thing about SMC is that it does not require any code logic change, making the combination of FSM objects and SMC codes easy and almost errorless. It also does not require any inheritance of SMC classes to the original project class. SMC provides high tolerance for unexpected events which supports the development of robust applications.

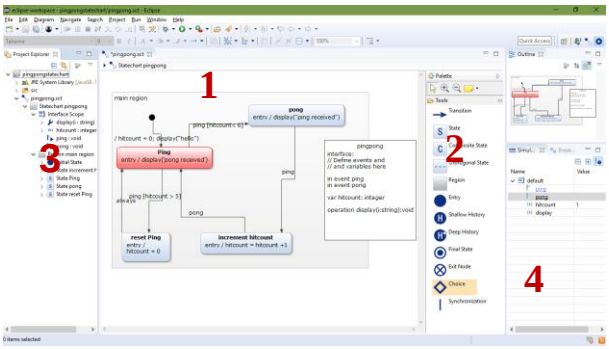

Figure 2 Ping-Pong Model Built in YAKINDU Modelling Environment

Though, the default transition within a state is considered to be one of the SMC shortcomings. That is because if a transition is not previously defined, it follows the saved default transition rules in the default state. This automatic resolution of missing transition definition might cause some problems if it conflicts with the overall project workflow.

From Figure 1, which is a high-level description of the SMC modeling environment, the following is the main windows descriptions:

1) Text editor building tools used to create an sm. type of file.

2) The text editing area where the C++ code is written.

3.1.1 Modeling Steps

The SMC tool version used is version v3.2. To generate a code using SMC compiler, a NotePad++ was used to write the SMC code. Java language output must be specified, then, specifying the target directory, and the SM file to be processed. The second step is writing the application classes.

Instances of FSM context classes must be defined. In this stage, it is possible to define the "Action Codes" or methods, their implementation behavior, and the transitions from/to different parts of the application code. The FSM context class executes the action codes in the application classes, and vice versa.

In a nutshell, the state diagram code is first written into an sm. file. Then, the SMC tool is ran, which will generate the state pattern code. After that, the action codes are written and executed. Finally, the FSM is interacted with through exchanging the transitions and the methods of execution.

3.1.2 Output Results

An sm. file was generated after writing the Ping-Pong C++ code in a notepad text editor. The code included many classes of state patterns describing the Pinger, Ponger, and the Referee action plans. The resulted sm. file can be now executed in an FSM context to compile FSM objects.

3.2 YAKINDU

YAKINDU is an open-source compositional modeling language. Essentially, it is a statechart editor with source code generation capabilities from Statecharts to a rich variety of languages including C++ which is the unified coding language in this paper experiment.

YAKINDU has a rich graphical editor that supports the development of compound hierarchical statecharts with validation and simulation features, in addition to supporting auxiliary variables, concurrency, and state refinement. Probably one of the most recent updates of Yakindu is introducing transition priority concept by specifying a total ordering of the allocated transitions that gives each outgoing transition a certain priority in case of a "race condition" [22].

However, YAKINDU lacks some advanced features such as buffering events, message queues, or compositional modeling that is essential to handle model's design complexity especially in the case of embedded reactive systems where Statecharts need to be composed into a component-based model. Yet, the comparison can still be managed by composing YAKINDU's individual statechart components by connecting its ports to form a single composite system [22], [23] The following is a high-level description of the YAKINDU platform:

From Figure 2, which is a high-level description of the YAKINDU modeling environment, the following is the main windows descriptions:

1) This area shows our editable Ping-Pong model in a class diagram format. This part includes the structuring elements. They are used to build the model.

2) This side shows the UML-RT elements used to build our diagram.

3) This side shows the Project and model explorer that keeps track of the model's build-up. It also shows the tree view of our model's elements synchronously as editing is presumed. The project explorer keeps track of all projects and the codes generated from the model explorer. The project also explores the projects space, and folders. The project explorer can also be used to inspect the structure of the state model.

4) This is the simulation explorer. It is used to select a certain simulation at a particular state. Selecting a particular transition will show you what state is going to be next. It also shows the outline of the ping pong-model.

3.2.1 Modeling Steps

To create and validate a Ping-Pong model using YAKINDU, a new project folder should be created, and then, a sub Yakindu statechart diagram model file should be created under the main project folder. On the model file, it is possible to import the C++ code into the model and have the Ping-Pong model statecharts built up automatically or drag and drop statecharts and their transitions from the elements panel.

In the latter option, action codes or "events" must be defined in order for the model to execute. When the model is ready, it can be running by simulating it. The simulation take place by manually triggering the events from the simulation explorer window.

3.2.1 Output Results

Once the events are triggered and the transitions are fired, the model gets executed and the Pinger and Ponger communicate by sending and receiving the pongs which is the output of this interactive event-driven Development systems.

3.3 Papyrus-RT

Papyrus-RT is a modelling development environment for UML-RT systems. It is an open-source application based on Eclipse platform and supported by Run-Time Service (RTS) ibrary [24]. Papyrus-RT will be used to generate and represent

a complete Ping/Pong model [5], and then, an executable and customizable code will be generated from this model.

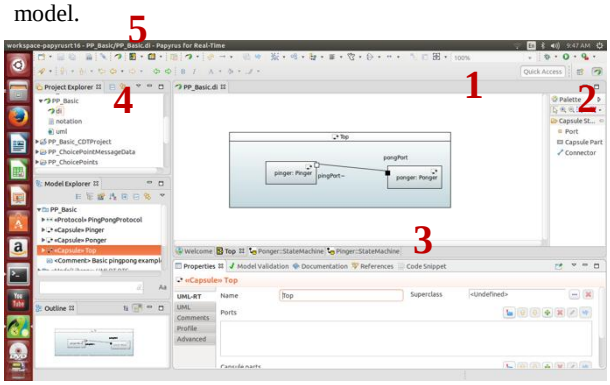

Figure 3 Ping-Pong Model Built in Papyrus-RT Modelling Environment

Papyrus-RT is well known for its mixed graphical/textual notations and other advanced features (e.g. import capabilities, defer/recall concurrency control, incremental code generation, and message-pools/buffers), yet, it does not support behavioral inheritance [25], [26], [27] therefore, composite states will be having to be used for comparison purposes.

However, in this paper9, only common features among the 4 MSDSs will be discussed to facilitate the comparison of tools. The following is a high-level description of the Papyrus-RT platform:

From Figure 3, which is a high-level description of the Papyrus-RT modeling environment, the following is the main windows descriptions:

1) This area shows our editable Ping-Pong model in a class diagrams format.
2) This side shows the UML-RT elements used to build our diagram.
3) At the bottom lies the property editing area where model's elements properties are specified.
4) This side shows the model explorer that keeps track of the model's build-up. It also shows the tree view of our model's elements synchronously as editing is presumed.
5) The project explorer keeps track of all projects and the codes generated from the model explorer.

3.3.1 Modeling Steps

To start building the Ping-Pong model, first build two capsules "active classes", Pinger and Ponger. Those two players will communicate to hit the ball, each in their assigned turn. Next, "Protocols" are created.

Protocols play the role of messages that keep track of who's turn it is to send the Pong, and how? A unique feature about Papyrus-RT is the port logs. They allow the code to print out messages to the user, possibly to assure them that the code is up and running.

Once the state machines, the state machines attributes and transitions, transitions triggers and guards, and their action codes are ready for execution, the model code is generated.

The project then can be running through the terminal window, and by using the "make" and then "./TopMain" commands will compile and run the C++ code generated from the model.

3.3.2 Output Results

The model results in the Pinger and the Ponger being able to communicate by exchanging the Pong through the Referee which is keeping trach of the number of Pongs sent and received until the 5th round.

3.4 Rhapsody

IBM Rational Rhapsody Developer MDE Tool was used for the modeling which is considered to be one of the most successful tools and frameworks that is widely used for its capabilities to support model-driven development [28]. After the modeling step, a C++ code can be automatically generated. Although Rhapsody has a wide variety of domain specific diagrams, Statecharts and Object Model Diagram (similar to UML's Class Diagrams) will be used to describe the software system.

The system structure and the Statecharts relationships will be specified between the classes and objects in the constructed model. The model's top-level object (root class) is a composite class called Ping/Bong Class. The default semantics of the Rhapsody Statecharts was kept as is to develop the model.

The model's transitions are labelled by action triggers and guards based on special events similarly as in the other MSDSs models. However, Rhapsody has special features (e.g. composite states, concurrent states, actions for entering or exiting states, OR-connectors) that don't exist in the other MSDSs discussed in this paper. Hence, any feature that is uniquely existing in only one single MSDS but not the others will not be used.

From Figure 4, which is a high-level description of the Rhapsody modeling environment, the following is the main windows descriptions:

1) This area shows our editable Ping-Pong model in a class diagrams format.
2) This side shows the UML-RT elements used to build our diagram.
3) This side shows the Project and model explorer that keeps track of the model's build-up. It also shows the tree view of our model's elements synchronously as editing is presumed. The project explorer keeps track of all projects and the codes generated from the model explorer.
4) The code debugging and error viewing side. Also, the code execution output is viewed from this window.

3.4.1 Modeling Steps

IBM Rational Rhapsody Architect was used to build the Ping-Pong model. First, a new C++ project folder is created and its default properties (e.g. Display Options) are fixed. Then, the Ping-Pong C++ code is imported and bound with a model. This model is now called the Ping-Pong model. The classes are directly built in the model and it is able to make structural changes at this stage in case the right classes weren't imported properly. The structural changes can be done by dragging and dropping the classes and their transitions, for instance, from the elements panel. Whenever the model is ready, the model can be built and ran.

3.4.2 Output Results

The output of this Rhapsody model is a structured Ping-Pong model with Pinger-Ponger classes communication capability.

4. RESULTS AND DISCUSSION

It is important to highlight here that this paper will only cover the comparison of the common features among the four MDSDSs for validation purposes. Yet, the unique features will be mentioned as well, but outside the comparison zone.

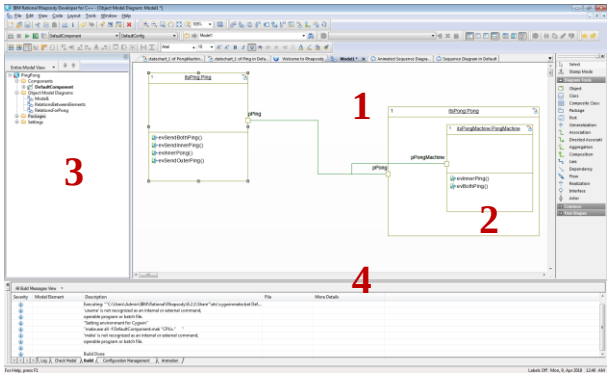

Figure 4 Ping-Pong Model Built in Rhapsody Modelling Environment

4.1 Similarities

The four MSDSs are similar in that they all employ state machines to visualize systems as a state-based formalism, showing the system behavior in an interactive way. They all support event-driven real-time software applications as well. However, they also include some unique features that differentiate each of them from the others.

4.2 Differences

The experiments' results will be presented in a tabular format (see Table 1) for easier comprehension. Each column represents one of the MDSDSs, and each raw is a unique feature. Across the table, there are the developer's notes, opinions, observations, and experience documentation. Based on that, the table includes a rough estimate of the best application areas suited for each MDSDS based on the results and discussion. This feature comparison distribution fashion is believed to make it easier for the targeted readers (Beginner Developers) to navigate through the table based on the criteria.

5. CONTRIBUTION

While most literature concentrated on the technicality of the model-driven development software, this research focus on the user-friendliness, easiness and the comprehensibility of those software. The original motivation of this research is to benefit those who are newly introduces to MSDSs, or just beginning to specialize on them, this paper will enlighten them of what sort of software is available and what are some of the general similarities and differences between them.

6. CONCLUSIONS

Among the most commonly used MDSDSs, Papyrus-RT, Rhapsody, YAKINDU, MSC. The first choice a beginner modeler needs to make is which modeling tool they should start with, and why? In general, all modeling tools enable you to model, but the question is which tool is best for my specific domain or precise resolution. This paper explored some of the high-level differences that was discovered while implementing the same Ping-Pong model in all the modeling tools.

However, even if the modeling purpose changed in the future, or the selected MDSDS turned to be limited in a certain scope, it is always possible to change the modeling tool to another. Nonetheless, due to some difference problems, it would be necessary to adjust the generated code from the older model according to the new modeling tool translation guidelines in order to obtain the same model execution behavior.

While most literature focus on technical differences and limitations, this paper covered the discussion about couple of modeling tools differences and similarities. This is because the authors believe that the social factor in the modeling tool development and success is just as significant as the effectiveness of the tool itself.

Furthermore, this paper presents the results of an informal comparative study of four MDSDSs with real illustrative modelling examples. Main classifications of feature differences were discussed according to the developer's experiences, in addition to careful examination of other academic researches and experiments.

As for future work, additional MDSDS competitors could be included in the comparison study (e.g. Simulink, MetaEdit+, objectiF). Furthermore, comparison of detailed syntactic and semantic differences (e.g. conjunction transitions, composite statecharts) could be included since only high-level comparison of those two features was covered.

7. THREATS OF VALIDITY

This experiment has been accomplished by one model developer only on Linux environment. User experience might differ according to the computer machine specifications (e.g. hard drive capacity, RAM size, etc.), the operating system installed, and the user's depth of knowledge and experience with the modelling tools in general. In that regard, opinions and experience quality might fall under the gray area, and not be absolutely accurate. However, technical and modelling comparison were based on sound model development processes and not related to a user opinion, which makes it valid and reliable to the best of our knowledge.

8. REFERENCES

[1] Sendall, S., & Kozaczynski, W. (2003). Model transformation: The heart and soul of model-driven software development. IEEE software, 20(5), 42-45.

[2] Schmidt, D. C. (2006). Model-driven engineering. COMPUTER-IEEE COMPUTER SOCIETY-, 39(2), 25.

[3] Home of SMC, The State Machine Compiler, http://smc.sourceforge.net/, accessed on April 12th, 2018.

[4] Selic, B. (1998). Using UML for modeling complex real-time systems. In Languages, compilers, and tools for embedded systems (pp. 250-260). Springer, Berlin, Heidelberg.

[5] Papyrus-RT/User/User Guide/Getting Started, Getting Started with Papyrus for Real Time v1.0, https://wiki.eclipse.org/Papyrus-RT/User/User_Guide/Getting_Started#Our_project:_PingPong, last accessed April 15th, 2018.

[6] Kahani, N., Hili, N., Cordy, J. R., & Dingel, J. (2017, May). Evaluation of UML-RT and Papyrus-RT for modelling self-adaptive systems. In Modelling in Software Engineering (MiSE), 2017 IEEE/ACM 9th International Workshop on (pp. 12-18). IEEE.

[7] Gery, E., Harel, D., & Palachi, E. (2002, May). Rhapsody: A complete life-cycle model-based development system. In International Conference on Integrated Formal Methods (pp. 1-10). Springer, Berlin, Heidelberg.

[8] Khalil, A., & Dingel, J. (2017). Optimizing the Symbolic Execution of Evolving Rhapsody Statecharts. Advances in Computers.

[9] Zurowska, K., & Dingel, J. (2011, November). SAUML: A tool for symbolic analysis of UML-RT models. In Proceedings of the 2011 26th IEEE/ACM International Conference on Automated Software Engineering (pp. 604-607). IEEE Computer Society.

[10] Das, T. K., & Dingel, J. (2016). Model development guidelines for UML-RT: conventions, patterns and antipatterns. Software & Systems Modeling, 1-36.

[11] Das, T. K., & Dingel, J. (2015, September). State machine antipatterns for UML-RT. In Model Driven Engineering Languages and Systems (MODELS), 2015 ACM/IEEE 18th International Conference on (pp. 54-63). IEEE.

[12] Das, T. K., & Dingel, J. (2016). Model Development Guidelines for UML-RT.

[13] Dingel, J., Paen, E., Posse, E., Rahman, R. R., & Zurowska, K. (2010, June). Definition and implementation of a semantic mapping for UML-RT using a timed pi-calculus. In Proceedings of the Second International Workshop on Behaviour Modelling: Foundation and Applications (p. 1). ACM.

[14] Crane, M. L., & Dingel, J. (2005, October). UML vs. classical vs. Rhapsody statecharts: Not all models are created equal. In International Conference on Model Driven Engineering Languages and Systems (pp. 97-112). Springer, Berlin, Heidelberg.

[15] Bagherzadeh, M., Hili, N., & Dingel, J. (2017, August). Model-level, platform-independent debugging in the context of the model-driven development of real-time systems. In Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering (pp. 419-430). ACM.

[16] Bagherzadeh, M., Bordeleau, F., Bruel, J. M., Dingel, J., Gérard, S., Hili, N., & Voss, S. Summary of Workshop on Model-Driven Engineering Tools (MDETools' 17).

[17] Liang, H., Dingel, J., & Diskin, Z. (2006, May). A comparative survey of scenario-based to state-based model synthesis approaches. In Proceedings of the 2006 international workshop on Scenarios and state machines: models, algorithms, and tools (pp. 5-12). ACM.

[18] Dingel, J., & Filkorn, T. (1995, July). Model checking for infinite state systems using data abstraction, assumption-commitment style reasoning and theorem proving. In International Conference on Computer Aided Verification (pp. 54-69). Springer, Berlin, Heidelberg.

[19] Whittle, J., Hutchinson, J., Rouncefield, M., Burden, H., & Heldal, R. (2013, September). Industrial adoption of model-driven engineering: Are the tools really the problem? In International Conference on Model Driven Engineering Languages and Systems (pp. 1-17). Springer, Berlin, Heidelberg.

[20] Kahani, N., Bagherzadeh, M., Dingel, J., & Cordy, J. R. (2016, October). The problems with eclipse modeling tools: a topic analysis of eclipse forums. In Proceedings of the ACM/IEEE 19th International Conference on Model Driven Engineering Languages and Systems (pp. 227-237). ACM.

[21] SMC Home Page, The State Machine Compiler, http://smc.sourceforge.net/, last accessed April 14th, 2018.

[22] Graics, B., & Molnár, V. (2017). Formal Compositional Semantics for Yakindu Statecharts.

[23] Yakindu Statechart Tools, http://statecharts.org/, last accessed April 15th, 2018.

[24] Hili, N., Dingel, J., & Beaulieu, A. (2017, May). Modelling and code generation for real-time embedded systems with UML-RT and papyrus-RT. In Software Engineering Companion (ICSE-C), 2017 IEEE/ACM 39th International Conference on (pp. 509-510). IEEE.

[25] Kahani, N., Hili, N., Cordy, J. R., & Dingel, J. (2017, May). Evaluation of UML-RT and Papyrus-RT for modelling self-adaptive systems. In Modelling in Software Engineering (MiSE), 2017 IEEE/ACM 9th International Workshop on (pp. 12-18). IEEE.

[26] Papyrus for real time (Papyrus-RT), https://www.eclipse.org/papyrus-rt, accessed: 2018-03-25

[27] Posse, E. (2015). PapyrusRT: modelling and code generation. In Workshop on Open Source for Model Driven Engineering (OSS4MDE'15).

[28] IBM Rational. Rational Rhapsody Developer, http://www03.ibm.com/software/products/en/ratirhap/.

9. APPENDIX

Table 1 MDSDSs Features Comparison Table

#	Features	Papyrus-RT	YAKINDU	Rhapsody	MSC
1	Availability	Open source	Open source	Not an open source program.	Open source
2	Clarity of Commands	Moderately clear	Very clear	Complex	Very clear
3	GUI Intuitivism	Moderately intuitive	Very intuitive	Moderately intuitive	Not applicable
4	User Friendliness	Very user friendly	Very user friendly	Complex	Moderately complex
5	Overall Experience Quality	Productive	Satisfying	Comparatively Struggling	Positively productive
6	Tool Learning Time Needed	1 week	5 days	10 days	4 days
7	Model Building Time Consumed	2 hours	2:30 hours	4 hours	1:30 hours
8	Learning Curve	The learning curve in Papyrus-RT is not so fast, because the user must familiarize themselves with every process they are about to implement, and every tool they are about to use.	The learning curve in YAKINDU is proportional to the time spent learning it.	The learning curve in Rhapsody is negatively proportional to the time spent using the tool due to its broad and wide options which the user can use from. This could help some users but could also be confusing to others.	The learning curve in SMC is relatively fast if the user knew the programming language used to code. Using the SMC tool is simple and bound to writing the code and generating the state chart patterns.
9	Semantic Differences	Its RTS library lacks support for implementing the different phases of the control loop usually implemented for monitoring and triggering the different adaptation of SAS systems.	It employs simulation techniques for its state machine models to check their dynamic semantics to save the time consumed for repetitively debugging the models.	It allows the users to make their own design decisions and support powerful execution semantics to code generation capabilities. It can scale up to handle larger systems.	SMC is based on a robust and solid theoretical base. State diagrams are directly coded into the SMC Language. State diagrams in SMC can be easily imported into a file in order for State Pattern classes to be automatically generated. In addition, it has effective handling of Reactive and Transformational systems.
10	Syntactic Differences	Mixed graphical/textual notations and other advanced features (e.g. import capabilities, defer/recall concurrency control, incremental code generation, and message- pools/buffers). Does not support behavioral inheritance.	Composed statecharts, it lacks the ability to compose statecharts into component-based model, but it is a rich language to model a single hierarchical statecharts. Does not support buffering events, message queues, or compositional modeling. Compound hierarchical	Statecharts and Object Model Diagrams. Special syntactic features include composite states, concurrent states, actions for entering or exiting states, and ORconnectors.	The default transition within a state is considered to be one of the SMC shortcomings. That is because if a transition is not previously defined, it follows the saved default transition rules in the default state. This automatic resolution of missing transition definition might cause some problems if it conflicts with the overall project workflow. Also, guarded transitions have

			statecharts with validation and simulation features. It supports auxiliary variables, concurrency, and state refinement.		higher priority of execution than unguarded ones.
11	Code Interpretation	Direct & clear interpretation	Basic interpretation	Embedded interpretation	Direct & clear interpretation
12	Code Generation Ability	C++	Java, C and C++	C, C++, Ada, Java and C#	C++, Java, TCL, VB, CSHARP
13	Installation Requirements	JDK8, JRE, and Linux OS for easy installation	It is a plug in that could be added on top of Eclipse from the "Help" tap and selecting "Install New Software". So, its installation requirements are the same as the requirements for installing Eclipse, JDK8 and JRE.	It is based on Eclipse, we need CDT, and then install the Rhapsody.	Java 8.0 (JDK 8), SMC.JAR. (SMC Compiler), text editor (e.g. notepad++)
14	Constructing Elements	State diagrams' elements. Consists of capsules made of active classes and their composite structures. Those capsules can communicate through ports using protocols that specify the messages that can be swapped.	Systems containing Components (Statecharts) and Instances of those Components who have Interfaces containing Ports of exchange. It adopts both graphical (e.g. states, transitions) and textual (e.g. declarations, actions) notations to model its state machines and diagrams.	Sequence diagrams, use-case diagrams, and State diagrams. Contains composite and orthogonal states, condition and junction connectors, and inter-level transitions.	State diagrams and State Pattern classes. eBus (a Java middleware) is responsible for exchanging messages between SMC transitions and FSM objects. SMMC supports disconnected, connecting, connected, and disconnecting type of states. For the file transfer process, it connects /disconnects with transitions for sending the data.
15	Behavior Executability	State diagram execution can be seen at log port. The embedded code generator translates the UML-RT model's structural elements and the behavior provided by each capsule's state machine into C++ code	The highlighted state and highlighted transition show the execution of statechart diagram. Variables can be seen on the simulation tap.	Execution of state diagram consists of highlighting the state as well as showing the events in the events section when they are triggered.	A default behavior can be defined in the state and be executed in higher priority than the default behavior definitions in locked or unlocked states. The logic of the state diagram is distributed across the classes That makes seeing all of the state diagram can't be seen from a single point. Working around this issue

					is a tedious coding.
16	Execution Style	Its models' applications can be executed on an embedded platform, or selectively logging parts of the execution. In general, Linux executes Papyrus-RT better than if models were built in Windows or MacOS.	Transition priority concept by specifying a total ordering of the allocated transitions that gives each outgoing transition a certain priority in case of a "race condition".	The test execution and verification engine execute test cases defined by sequence diagrams, flow charts, statecharts, and source code. During execution, it verifies the results against the defined requirements. It simulates the user's activity during test execution by automatically sending the message to the debugger to provocate a reaction.	The automatic message exchange and generation technique eliminated the need for manual transition matrices, state transition arrays, or the scattered switch statements. To generate a code using SMC compiler, Java language output must be specified, then, specifying the target directory, and the SM file to be processed.
17	Object Creation/ Destruction	Doesn't show error on a new state even though it is not reachable.	It shows the error while building the model.	Complex to create elements because they are all connected. It doesn't delete anything recursively.	SMC provides high tolerance for unexpected events which supports the development of robust applications.
18	Bugs Identification & Debugging	It provides specific tools that facilitates the development and validation of UML-RT models. You have to regenerate the model to remake the file and get the potential errors.	It adopts dynamic debugging. It shows errors instantly. It includes syntax and semantic checks to validate its models live whilst editing.	Automatic debugging while editing.	By combining virtual methods with the state pattern, SMC enables the definition of default transitions that allows objects to handle unexpected events, recover and continue providing service, instead of crashing.
19	Output	To create models that fully and correctly generate code for complex embedded and real-time applications.	Development of interactive event-driven systems.	Rational rhapsody helps system engineers and software developers to automate the software development lifecycle and create real-time model designs by understanding systems' requirements, validate functionality and identify defects and design errors beforehand, and finally delivers highly structured software.	SMC is an event driven application, that uses state patterns to combine FSMs with their Objects. Hence, it is used to compile FSM projects. Although only one file is produced, the actual code contains many classes of state patterns. The generated table produces HTML table illustration of the .sm file. It also outputs a graph in a GraphViz .dot file format containing the state machine logic.

20	Usages	Small Industry Education/Academy Workflow Prototyping UML-RT modelling Spiral development style	Small Industry Workflow Prototyping Agile development style	Large Industry Beginners/Intermediate developers Education/Academy DSL modelling Waterfall development style	Education/Academy Small Industry Agile development style
21	Level of Model Abstraction	Moderately abstracted.	Moderately abstracted.	Highly abstracted.	Barely abstracted.
22	Level of Model Automation	Composite states construct automatically by double clicking any state diagram. Generally, the user has to initiate every task to take place.	One of the automatic actions is error detection. Entry/exit transitions of the state are automated and predefined in the state chart. Interfaces (protocols) like "always" and "never" are created automatically. Generally, the user can use the automated generated protocol in statecharts without defining them.	Double clicking any class diagram will automatically generate a composite class. Generally, the user has to initiate every task to take place.	State diagrams are coded into Java language, making the code importing into a file easy and direct. State Pattern classes will then be automatically generated in the other MDSDS tool.
23	Proper Tooling	Papyrus-RT is generally properly tooled. However, tools for sequence diagrams and class diagrams need to be made available.	Yakindu has proper tools for its statecharts, but no plug-in tools are available for sequence diagrams or class diagrams.	Rhapsody is generally properly tooled.	SMC tools support state diagrams, but additional tools are needed to create class diagrams and sequence diagrams.