An Eclipse-based Editor to Support LOTOS Newcomers

Giuseppe De Ruvo, Antonella Santone
Department of Engineering, University of Sannio, Benevento, Italy
e-mail: {gderuvo@unisannio.it, santone@unisannio.it}

***Abstract*—We present ELOTON, an Eclipse-based Editor to help people who want to approach the Language of Temporal Ordering Specification (LOTOS). LOTOS is a Formal Description Technique standardized by ISO for the design of concurrent and distributed systems, and in particular for OSI services and protocols. LOTOS has been widely used for the specification of large data communication systems. It is mathematically well-defined and expressive: it allows concurrency, non-determinism, synchronous and asynchronous communications. CADP is a popular toolbox that supports high-level descriptions written in LOTOS, among others. Unluckily, there not exists an user interface suitable for newcomers in formal methods. Thus, many people encounter many obstacles in using formal methods and in particular model checking due to the lacking of user-friendly tools. We argue that ELOTON, thanks to its rich text editor and visualization features might involve a major number of users coming from various disciplines. Our tool serves as a graphical front-end for CADP.**

***Keywords*-Model Checking, LOTOS, CADP, Formal Verification Tools, Eclipse RCP**

I. Introduction and Related Work

Formal methods are a collection of mathematical methods that allow one to verify correctness of the model of the system in early design phases. Among these methods, model checking [1] is a rather successful one. Basically, model checking allows to automatically prove whether a model of the system at the suitable level of abstraction satisfies a given specification. Before we can apply model checking, we have to formalize the model of the system and its specification, i.e. the requirements the system should satisfy. The process of model checking returns yes/no answers reflecting the correctness of the system being analysed. In more detail, in the case of negative answer, we can follow the bad trace, which does not satisfy the requirement. Therefore, model checking is also useful for correcting the model.

The development and application of formal methods and in particular model checking has been highlighted in connection with a variety of disciplines such as business process management [2], biology [3], collaborative knowledge [4], human automation interactions and aeronautics [5], among others.

On the one hand several formal specification languages exist, as for example CCS [6], LOTOS [7], Promela [8]. On the other hand, there are many model checkers developed in the world using the above specification languages (see next Section).

There are different works to improve the user experience with model checking tools. VIP [9], the Visual Interface to Promela (VIP) tool is a Java based graphical front end to the Promela specification language and the SPIN model checker. It provides a graphical v-Promela editor supporting point and click editing of v-Promela structure diagrams and hierarchically nested state machines. The editor incorporates syntax checking to warn the user about incorrect use of v-Promela graphical syntax. Storage and retrieval of models is made possible using Java serialization. The tool also has a fully integrated v-Promela compiler which generates Promela code. There also are other works on improving the support to Spin [10][11].

MCMAS [12] is an Eclipse-based tool which supports specifications based on CTL, epistemic logic (including operators of common and distributed knowledge), alternating Time Logic, and deontic modalities for correctness. The GUI guides the user to create and edit ISPL programs by performing dynamic syntax checking with an additional ANTLR parser. The GUI also provides outline view, text formatting, syntax highlighting, and content assist automatically. Morevoer, MSCMAS provides facilities to display counterexamples. The authors of MCMAS employed Graphviz[1] to represent graphs.

Tribastone et al. [13] designed an Eclipse plug-in to support the creation and analysis of performance models, from small-scale Markov models to large-scale simulation studies and differential equation systems. Performance Evaluation Process Algebra (PEPA) [14] is a concise formal language for high-level quantitative modelling. The tool enables Markovian steady-state analysis, stochastic simulation, and ODE analysis of PEPA models.

Yokoyama et al. [15] extend the functions of Java PathFinder [16], a software model checker to verify executable Java bytecode programs, and propose a graphical user interface with a high degree of usability. One of the key features of their GUI is a variable-value-based graph abstraction that allows users to focus upon an aspect they are interested in. It also has an intuitively easy-to-use interface for users to input linear temporal logic (LTL) formulae as a program specification based on the Specification Pattern System.

Arcaini et al. presented NuSeen [17], an eclipse-based environment for NuSMV [18], with the aim of helping NuSMV users. The tool mainly focuses in easing the use of the NuSMV tool by means of graphical elements. They also integrated a model advisor for NuSMV models to perform automatic review of NuSMV models, with the aim of determining if a

[1] http://www.graphviz.org

model is of sufficient quality, where quality is measured as the absence of certain faults [19].

Writing a complex model, it is not a trivial task when there is no assistance for the designer or "formalist". It may be even worse for newcomers in formal methods and in particular model checking.

In this paper, we are interested in investigating the kind of user interface that might help a non-expert user in taking advantage of formal specifications and verification techniques. Nowadays formal methods still encounter many obstacles in being widely used and accepted, and we believe that the kind of human interface they usually provide does play a main role in this difficulty.

CADP [20] is a popular toolbox for the design of asynchronous concurrent systems. It supports high-level descriptions written in various languages, mainly LOTOS.

Thus, our contribution is the implementation of ELOTON (Eclipse-based editor for LOTOs Newcomers), which provides a graphical front-end to CADP verification environment, making this tool usable to people who are not specialists in model checking. More precisely, our goal is two fold:

1) to help the user in writing LOTOS specification, the input specification language of CADP;
2) to help the user to pinpoint the error, when the verification fails, "exploding" on demand each node of the graph representing the behaviour of the system to be verified.

We have chosen CADP since, as stated in [20], it provides an unique combination of the following four features that no other tool presently offers:

- the tool supports not only model-checking but also equivalence checking which, beyond being standard practice in hardware verification, plays a crucial role for component-based systems and compositional verification.
- the tool supports distributed verification, i.e. it can use the computing power and memories of a cluster of machines, rather than a single machine;
- the modelling language of the tool support concurrency, i.e. it has some built-in notion of asynchronous parallel composition;
- the modelling language support user-defined (possibly unbounded) data types such as records, unions, lists, etc.

Furthermore, CADP toolbox is designed in a modular way and puts the emphasis on intermediate formats and programming interfaces (such as the BCG and OPEN/CAESAR software environments). In this fashion, the CADP tool can be combined with other tools and adapted to various specification languages.

The remainder of this paper is organized as follows. First, in the next section we give an overview of model checking tools. In Section III we present our tool, ELOTON. Finally, Section IV deals with conclusions and new ideas for further work.

II. Model checking and tools

As the size of software increases, it becomes increasingly difficult to perform testing. This is particularly important in systems where the reliability of the software cannot be compromised. Such concerns have motivated the software industry to consider alternative techniques for software verification, especially those based on formal proofs [21]. Thus, in the recent years large research effort has been devoted to develop formal approaches to the design and analysis of critical systems. To reason formally about real-world systems, tool support is necessary; consequently, a number of tools embodying various analysis have been developed. As stated in [20], there are many model checkers developed in the world. All provide a support for automatically answering the verification question: does a system sys satisfy a property φ? To implement such a tool, the verification question must be formulated more carefully by fixing:

1) a precise notation for defining systems;
2) a precise notation for defining properties; and
3) an algorithm to check if a system satisfies a property.

To cope with the first problem, several specification languages have been developed. Some tools use process algebras as, for example, CCS [6], LOTOS [7], CSP [22]. The second problem can be solved using a temporal logic as, for example, the mu-calculus logic [23], CTL [24]. For the last problem, several algorithms exist. The most used verification methodology is model checking [1]: the property that the system must satisfy, expressed in some temporal logic, is checked on a finite structure representing the behavior of the system. If we specify the system by means of a process algebra program, like LOTOS, this structure is a labeled transition system, i.e., an automaton whose transitions are labeled by event names. The transition system represents all possible executions of the program. To check a property, model checking explores every possible state that the system may reach during execution. If the system does not satisfy the property, a description of the execution sequence leading to the state is reported to the user. Many bugs such as deadlock and critical section violations may be found using this approach.

The main drawback of model checking is that it is not scalable to very large systems unless the model is very abstract. The number of states in the finite state representation increases exponentially with the number of variables ("the state explosion"). For this problem several approaches have been developed. Among them, reduction techniques based on symbolic model checking techniques [25], partial order techniques [26], [27], compositional techniques [28], [29], [30], abstraction approaches [31], [32] and heuristic-based approaches [33], [34], [35].

Another drawback of model checking is that it operates only on models. Thus, writing a model becomes one of the key problems in model checking. On the other hand, the main advantage of using model checking is that it is fully automatic. Furthermore, model checking tools give an error trace for each error path in the program. This can help in locating and correcting the error in the system. However, long counterexample prevents the comprehension of the fault. Therefore, in this paper, we are interested in investigating the kind of user interface that might help a non-expert user in taking advantage of both formal specifications and verification techniques.

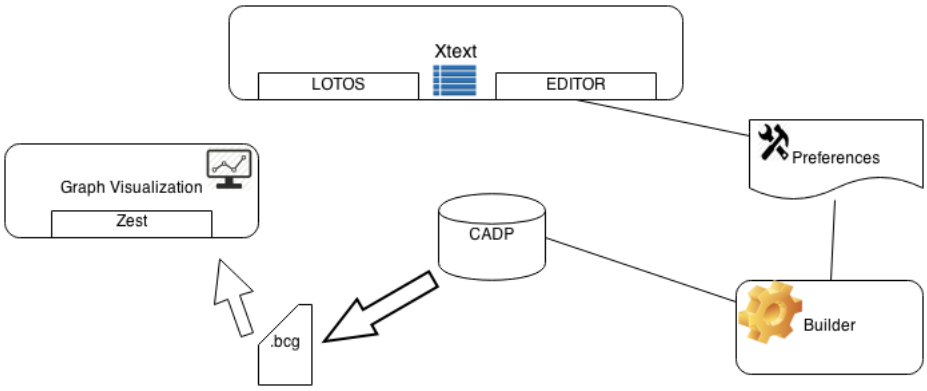

Fig. 1: ELOTON's components

One of the most popular formal verification environment is CADP [20], which includes LOTOS as specification language. Being a process algebra specification language CADP can also perform equivalence checking, i.e. process algebras can be used to describe both implementations of processes and specifications of their expected behaviors. Therefore, they support the so-called single language approach to process theory, that is, the approach in which a single language is used to describe both actual processes and their specifications. An important ingredient of these languages is therefore a notion of behavioral equivalence. One process description, say sys, may describe an implementation, and another, say $spec$, may describe a specification of the expected behavior. This approach to program verification is also sometimes called implementation verification. This is also a reason for choosing to make more user-friendly CADP.

III. ELOTON - Eclipse editor for LOTOS Newcomers

In this paper we present ELOTON, an Eclipse-based editor to support LOTOs Newcomers. Its main purpose is to help people who are not aware of formal methods and model checking to write specifications and easily proceed with verification. ELOTON is an Eclipse RCP application, as explained in the following. We did not develop an Eclipse-plugin, because we also do not want to "scare" people unfamiliar with Eclipse including its standard (and not required in this case) features.

A. Rich-client Application

Under the tooling platform of Eclipse [2] is the Eclipse RCP (Rich Client Platform). This is a generic platform for running applications. The Eclipse IDE happens to be one of such application. The Eclipse RCP addresses complex application scenarios that span the spectrum from thin to rich clients and from enterprise and business-oriented systems to scientific and data management scenarios.

Rich clients support a high-quality end-user experience for a particular domain by providing rich native User Interfaces (UIs) as well as high-speed local processing. Rich UIs support native desktop metaphors such as drag and drop, system clipboard, navigation, and customization. When done well, a rich client is almost transparent between end users and their work - fostering focus on the work and not the system. The Eclipse RCP platform is the best way to build rich-clients because such rich-feature is inherent in the Eclipse platform itself.

[2]http://www.eclipse.org

B. Features

Our tool is made up of an editor and a view to visualize automata generated by written specifications. The editor provides auto-completion, syntax highlighting and error marking features, supported by an Eclipse custom builder setting up on top of CADP's CAESAR compilers. Instead, the view adds new understanding capabilities in order to enhance the layout conveyed to the user. ELOTON's distinctive features are presented in the following with the aid of screenshots. Figure 1 shows ELOTON's main components. Component Xtext provides the editor and definitions for language LOTOS. Component Builder is basically a wrapper to call the external CADP's CAESAR compilers. It is necessary to set Preferences to locate CADP. Graphs can be depicted after the creation of one (or more) ".bcg" returned by CADP. Note that component Graph Visualization works using only the ".bcg" file. Thus, it does not share the same meta-model created by Xtext, as can be seen in the figure.

1) Syntax Highlighting and Auto-completion: formal specifications are text which have to be written in a file. Helpers such as code highlighting and auto-completion elevate the capabilities of textual editors significantly.

Almost any programming language has an editor with aforementioned capabilities or even an IDE with more support. Unfortunately, as far as we know, there not exists an IDE or just an editor for backing people who model in LOTOS. This is a main problem mostly for newcomers, where a newcomer in LOTOS might also be an expert software engineer or developer. Moreover, if the file is composed of hundreds lines, difficulties arise for experts too.

Figure 2 shows an example project with the popular ALTERNATING BIT PROTOCOL specification coming from CADP's demos. Keywords and comments are highlighted by the editor and the code is well formatted without additional effort. Looking at bottom-left of the figure is also possible

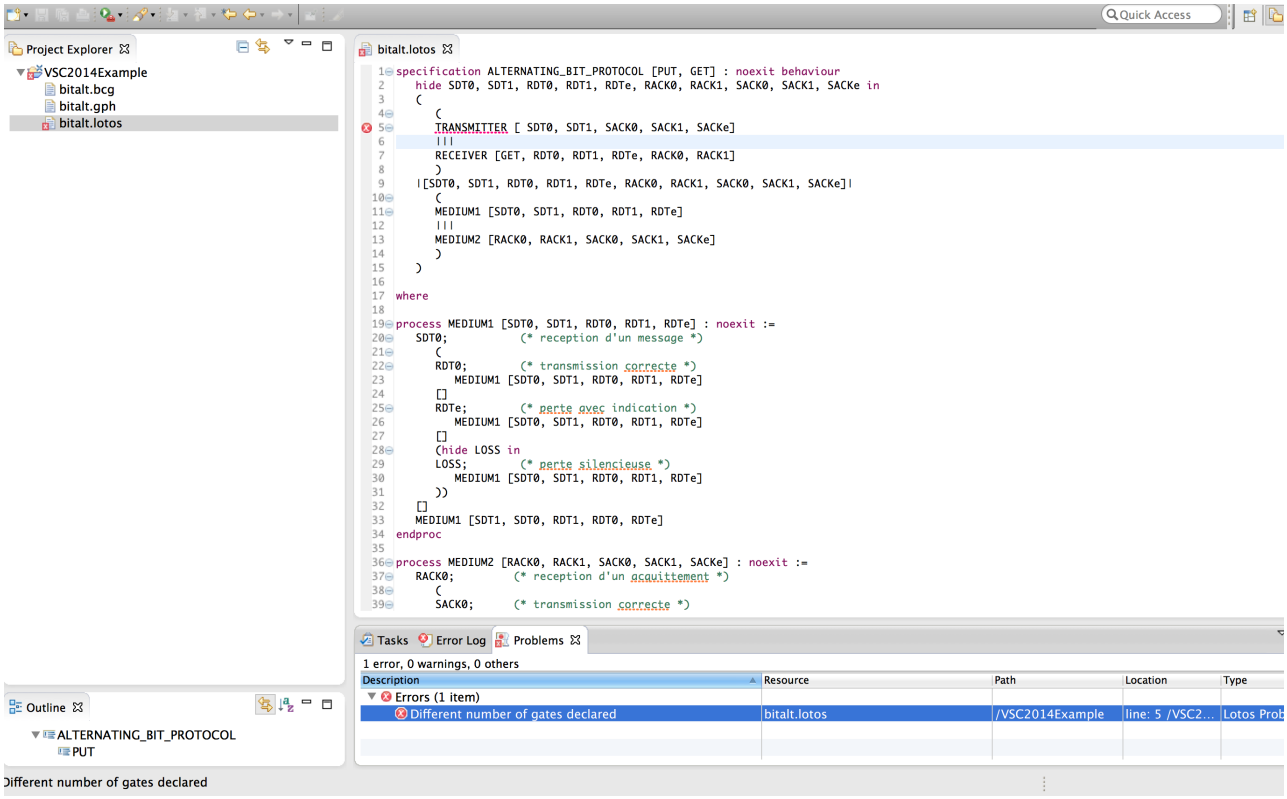

Fig. 2: ELOTON - syntax highlighting plus markers to show errors

to find an Outline to easily navigate trough processes. Instead, auto-completion capabilities assist the user proposing keywords (specification, behaviour, process, etc.), gates - if defined, process names and so on. Furthermore, there is a basic template for newcomers that provide initial specification - something like *specification something ...* with symbols, *process* and *endproc* and *endspec*.

Xtext[3] provides ELOTON's syntax highlighting, auto-completion and template capabilities. Xtext is a framework for development of programming languages and domain specific languages. It covers all aspects of a complete language infrastructure, from parsers, over linker, compiler or interpreter to exhaustive excellent Eclipse IDE integration.

2) Error Marking: another important feature of ELOTON is the error marking. Indeed, after setting CADP environment variable, it is possible to compile without using the Command Line Interface. Errors are given with useful markers and sorted in the Problems view to better understand what is going on. Figure 2 highlights a simple error: a different number of gates declared, i.e. process TRANSMITTER is missing gate PUT. Another error might arise when writing gates which do not exist, forgetting to close an important construct e.g. process, missing keywords and so on.

A marker is like a yellow sticky note stuck to a resource. On the marker we can record information about a problem (e.g., location, severity) or a task to be done. Or we can simply record a location for a marker as a bookmark. Users can quickly jump to the marked location within a resource. The Eclipse platform API itself defines methods for creating and managing markers, setting marker values, and extending the platform with new marker types.

3) Automata Visualization: ELOTON also offers visualization capabilities. BCG graphs representing automata generated by CADP are created in the Full Graph View of this RCP application. The user is able to view, zoom and arrange the generated graphs according to his own needs. There are four types of layout available, as shown in Figure 3. Notwithstanding, when there are too many nodes even this kind of visualization is useless. In fact, nodes overlap and actions are hard to understand and locate on the graph.

4) On Demand Creation of Automata: When the generated graph is puzzling automata cannot supply any kind of information due to confusion. This is a big matter when we do model checking and we want to analyse the counterexample to pinpoint the source of the error. Most of the time, long error paths can prevent an easy comprehension of the fault. ELOTON proposes the on demand creation of automata, allowing to generate nodes on specific paths. The graphs look prettier and clearer and provide enough information to the user. It is also possible to add a chosen node or action: ELOTON will add all the necessary nodes and arcs to accomplish that task. Figure 4 shows the on demand feature.

Automata Visualization features are made possible thanks to Zest. Eclipse Zest is a visualization toolkit for graphs. It is based on SWT/Draw2D. Zest supports the viewer concept from JFace Viewers and therefore allows to separate the model from the graphical representation of the model. Zest also provides a set of graph layout managers. A graph layout

[3]https://www.eclipse.org/Xtext

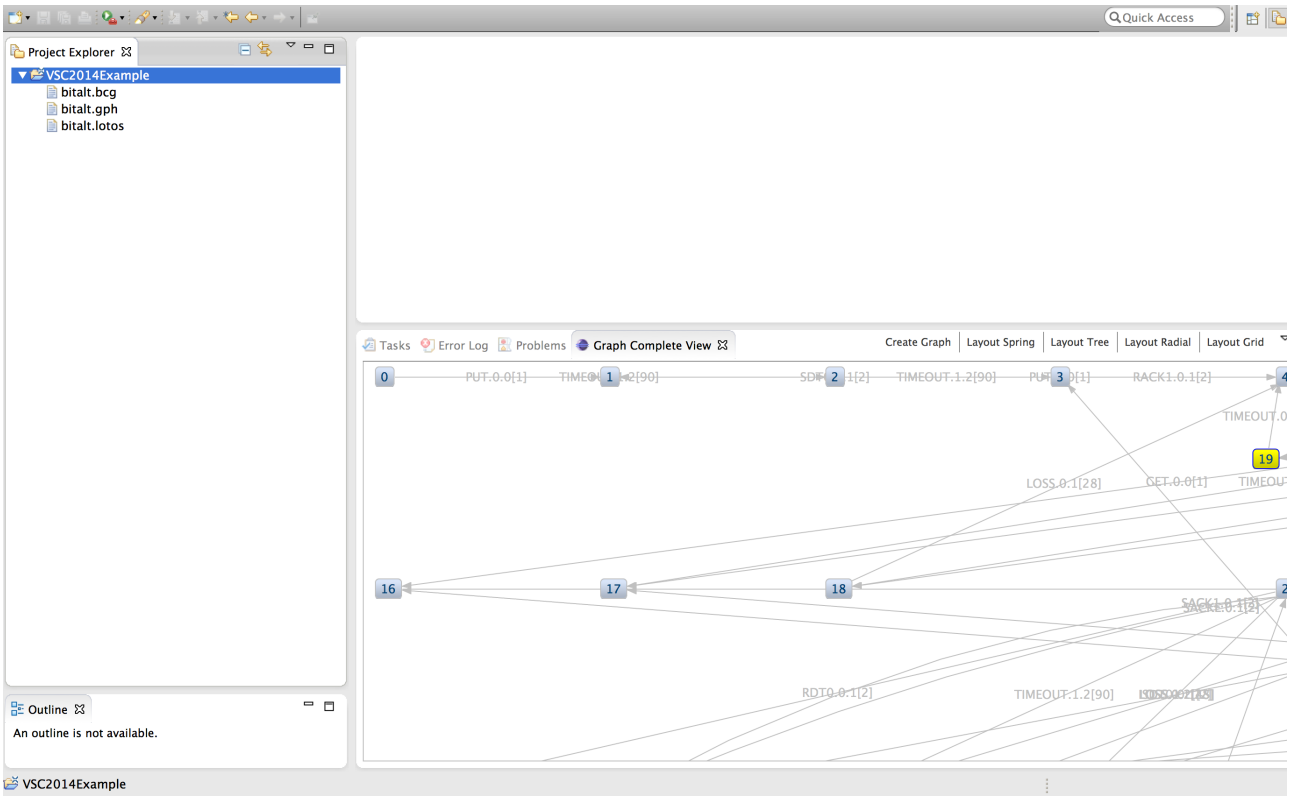

Fig. 3: ELOTON - visualization of automata

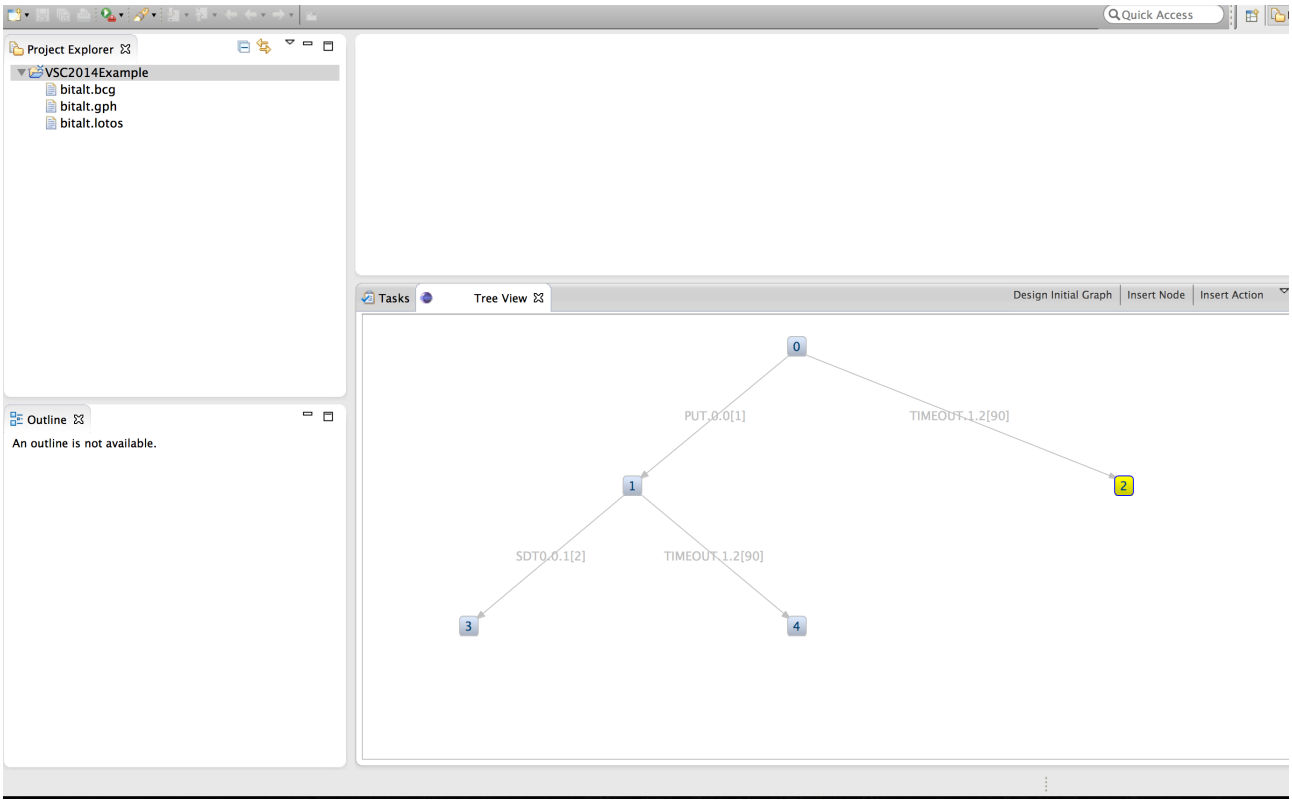

Fig. 4: ELOTON - on demand creation choosing a specific path

manager determines how the nodes (and the arrows) of a graph are arranged on the screen. Filters, to define which nodes and connections should be displayed, can be used too.

IV. Conclusions and Future Work

We have presented ELOTON, a tool to help people who write LOTOS formal specifications and perform verification with model checking.

Our tool is an Eclipse RCP application that uses frameworks Xtext and Zest, among others. We provide a rich text editor with syntax highlighting, auto-completion, templates and error marking capabilities. Furthermore, we supply two views to manage the creation of the automata from LOTOS models. On the one hand, users are able to visualize the generated graph choosing from four types of layouts. On the other hand, ELOTON offers on demand creation and insertion of nodes and actions to follow a specific path when the graph has too many nodes which prevent a clear representation. We argued that aforementioned facilities are a crucial point especially for newcomers and might also be a way to involve people using formal methods, in particular model checking. Many researchers or students would explore new interesting areas mixing various disciplines. We plan to improve ELOTON and add new features to enhance the user experience and add more semantic rules to improve the content assist before a public release. An editor to write properties to be verified is also needed. The possibility to do equivalence checking within ELOTON (using for example CADP's bisimulator) must be integrated in order to take advantage of our visualization environment.

References

[1] E. M. Clarke, O. Grumberg, and D. Peled, *Model checking*. MIT Press, 2001.

[2] A. Santone, V. Intilangelo, and D. Raucci, "Application of equivalence checking in a loan origination process in banking industry," in *Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE), 2013 IEEE 22nd International Workshop on*, 2013, pp. 292–297.

[3] A. Santone, L. Cerulo, and M. Ceccarelli, "Infer gene regulatory networks from time series data with formal methods," *2013 IEEE International Conference on Bioinformatics and Biomedicine*, vol. 0, pp. 115–120, 2013.

[4] G. De Ruvo and A. Santone, "A novel methodology based on formal methods for analysis and verification of wikis," in *Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE), 2014 IEEE 23nd International Workshop on*, June 2014.

[5] M. Bolton, R. Siminiceanu, and E. Bass, "A systematic approach to model checking human-automation interaction using task analytic models," *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on*, vol. 41, no. 5, pp. 961–976, Sept 2011.

[6] R. Milner, *Communication and concurrency*, ser. PHI Series in computer science. Prentice Hall, 1989.

[7] T. Bolognesi and E. Brinksma, "Introduction to the iso specification language lotos," *Computer Networks*, vol. 14, pp. 25–59, 1987.

[8] G. J. Holzmann, *The SPIN Model Checker - primer and reference manual*. Addison-Wesley, 2004.

[9] M. Kamel and S. Leue, *Vip: A visual editor and compiler for v-promela*. Springer, 2000.

[10] T. Kovše, B. Vlaovič, A. Vreže, and Z. Brezočnik, "Eclipse plug-in for spin and st2msc tools-tool presentation," in *Proceedings of the 16th International SPIN Workshop on Model Checking Software*. Berlin, Heidelberg: Springer-Verlag, 2009, pp. 143–147.

[11] B. De Vos, L. C. L. Kats, and C. Pronk, "Epispin: An eclipse plug-in for promela/spin using spoofax," in *Proceedings of the 18th International SPIN Conference on Model Checking Software*. Berlin, Heidelberg: Springer-Verlag, 2011, pp. 177–182.

[12] A. Lomuscio, H. Qu, and F. Raimondi, "Mcmas: A model checker for the verification of multi-agent systems," in *Computer Aided Verification*. Springer, 2009, pp. 682–688.

[13] M. Tribastone, A. Duguid, and S. Gilmore, "The pepa eclipse plugin," *SIGMETRICS Perform. Eval. Rev.*, vol. 36, no. 4, pp. 28–33, Mar. 2009.

[14] J. Hillston, *PEPA: Performance enhanced process algebra*. University of Edinburgh, Department of Computer Science, 1993.

[15] S. Yokoyama, H. Sato, and M. Kurihara, "User-friendly gui in software model checking," in *Systems, Man and Cybernetics, 2009. SMC 2009. IEEE International Conference on*, 2009, pp. 468–473.

[16] K. Havelund and T. Pressburger, "Model checking java programs using java pathfinder," *International Journal on Software Tools for Technology Transfer*, vol. 2, no. 4, pp. 366–381, 2000.

[17] P. Arcaini, A. Gargantini, and P. Vavassori, "Nuseen: an eclipse-based environment for the nusmv model checker," in *Eclipse-IT 2013: Proceedings of VIII Workshop of the Italian Eclipse Community*. Eclipse Italian Community, September 2013. [Online]. Available: http://arxiv.org/abs/1310.2464

[18] A. Cimatti, E. Clarke, F. Giunchiglia, and M. Roveri, "Nusmv: a new symbolic model checker," *International Journal on Software Tools for Technology Transfer*, vol. 2, no. 4, pp. 410–425, 2000.

[19] P. Arcaini, A. Gargantini, and E. Riccobene, "A model advisor for nusmv specifications," *Innovations in systems and software engineering*, vol. 7, no. 2, pp. 97–107, 2011.

[20] H. Garavel, F. Lang, R. Mateescu, and W. Serwe, "Cadp 2011: a toolbox for the construction and analysis of distributed processes," *STTT*, vol. 15, no. 2, pp. 89–107, 2013.

[21] A. Calvagna and E. Tramontana, "Combinatorial validation testing of java card byte code verifiers," in *Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE), 2013 IEEE 22nd International Workshop on*, June 2013, pp. 347–352.

[22] C. A. R. Hoare, "Communicating sequential processes," *Commun. ACM*, vol. 21, no. 8, pp. 666–677, Aug. 1978. [Online]. Available: http://doi.acm.org/10.1145/359576.359585

[23] C. Stirling and D. Walker, "Local model checking in the modal mu-calculus," *Theor. Comput. Sci.*, vol. 89, no. 1, pp. 161–177, 1991.

[24] J. Bohn, W. Damm, O. Grumberg, H. Hungar, and K. Laster, "First-order-ctl model checking," in *FSTTCS*, 1998, pp. 283–294.

[25] K. L. McMillan, *Symbolic model checking*. Kluwer, 1993.

[26] P. Godefroid, *Partial-Order Methods for the Verification of Concurrent Systems - An Approach to the State-Explosion Problem*, ser. Lecture Notes in Computer Science. Springer, 1996, vol. 1032.

[27] A. Santone and G. Vaglini, "Partial order interpretation of a mu-calculus-like temporal logic," in *ICSOFT*, 2013, pp. 233–238.

[28] E. M. Clarke, D. E. Long, and K. L. McMillan, "Compositional model checking," in *LICS*. IEEE Computer Society, 1989, pp. 353–362.

[29] A. Santone, "Automatic verification of concurrent systems using a formula-based compositional approach," *Acta Inf.*, vol. 38, no. 8, pp. 531–564, 2002.

[30] A. Santone, G. Vaglini, and M. L. Villani, "Incremental construction of systems: An efficient characterization of the lacking sub-system," *Sci. Comput. Program.*, vol. 78, no. 9, pp. 1346–1367, 2013.

[31] A. Santone and G. Vaglini, "Abstract reduction in directed model checking ccs processes," *Acta Inf.*, vol. 49, no. 5, pp. 313–341, 2012.

[32] E. M. Clarke, O. Grumberg, and D. E. Long, "Model checking and abstraction," *ACM Trans. Program. Lang. Syst.*, vol. 16, no. 5, pp. 1512–1542, 1994.

[33] A. Santone, "Heuristic search + local model checking in selective mu-calculus," *IEEE Trans. Software Eng.*, vol. 29, no. 6, pp. 510–523, 2003.

[34] S. Gradara, A. Santone, and M. L. Villani, "Delfin$^+$: An efficient deadlock detection tool for ccs processes," *J. Comput. Syst. Sci.*, vol. 72, no. 8, pp. 1397–1412, 2006.

[35] ——, "Using heuristic search for finding deadlocks in concurrent systems," *Inf. Comput.*, vol. 202, no. 2, pp. 191–226, 2005.