Integrating the E4J editor to the JGOOSE tool

Leonardo Pereira Merlin[1], Alexandre Luiz de Borba Silva[1], Victor Francisco Araya Santander[1], Ivonei Freitas Da Silva[1], and Jaelson Castro[2]

[1] Universidade Estadual do Oeste do Paraná, UNIOESTE
Cascavel, PR, Brasil
leonardo.merlin@gmail.com,{alexandre.silva10,victor.santander,ivonei.silva}@unioeste.br
[2] Universidade Federal de Pernambuco, UFPE
Recife, PE, Brasil
jbc@cin.ufpe.br

Abstract. Requirements engineering is an important phase of software engineering. The integration of the various requirements models remains a major challenge. In previous work we proposed a process to generate UML Use Cases from i* (istar) organizational models. It is supported by the JGOOSE tool, which transforms i* models, produced by a third party tool, into Use Case models and descriptions. This dependence on an external tool, for the creation of the organizational models, was a major shortfall. Thereby, to eliminate this dependence, we present the E4J (an integrated Editor for JGOOSE tool).

Keywords: E4J, JGOOSE, i* Framework, Requirements Engineering.

1 Introduction

System development should occur in a context where organizational processes are well established. The primary reason for software system failure is the lack of proper understanding of the organization by the software developers [1]. Unfortunately, the dominant object oriented modeling technique, UML, is ill equipped for organizational requirement modeling. We need other techniques, such as i* (istar) [1] to represent organizational modes.The i* (iStar) framework is well suited suited to represent the organizational requirements that occur during the early requirements phase. It is capable of modeling variability and offers a rich set of modeling concepts, such as goal, softgoal, task and resource. These early activities can enable an understanding of how and why the requirements came about.

Nevertheless, organizational requirements must be related to functional requirements which are often represented as Use Cases [2]. However, Use Case development demands great experience of the requirements engineers. The heuristics presented in the literature to develop Use Cases are not sufficient to allow a systematic development. In this sense, in previous work [3] we proposed an approach to support the development of Use Cases from organizational models described in i*.

Our approach allows the elicitation and specification of requirements from the actors goal in relation to the system-to-be. Moreover, the relationships between system to be and its environment can also be expressed in terms of goal based relationships obtained from the organizational models. Hence, we can derive and map goals, intentions and motivations of organizational actors to main goals of Use Cases.

In order to support the mapping of the i* into Use Cases models we developed the JGOOSE tool (Java Goal Into Object Oriented Standard Extension) [4]. Using this tool we can derive Use Cases from the TELOS (.tel) [5] [6] files generated by OME (Organization Modelling Environment) [7] tool.

Users must use the OME to generate i* models, then they use JGOOSE to import them and, finally, generate Use Cases. However, there are shortfalls with this process: it is necessary to generate an intermediary file (.tel) and organizational models cannot be created or modified in the JGOOSE environment. Hence, it is not possible to change the i* models within the JGOOSE, neither perceive the impact on the generated Use Case. As a consequence productivity is decreased. In order to address these shortcomings, in this work we propose a new i* editor. The editor structure and its interface is designed to facilitate the Use Cases generation.

Therefore, in this paper we propose the E4J (Editor for JGOOSE). It is integrated the JGOOSE environment to support the creation of i* models. Hence, it now generates Use Cases on the same JGOOSE environment without intermediary files. Hence, this new version of the JGOOSE also supports direct edition of i* models. As a consequence, changes to organizational models are now easily mapped to new Use Case Models. Moreover, it is possible to save the i* models with .mxe extension and export them to the iStarML format [8]. The iStarML is an XML compliant format to represent i* diagrams. The goal of iStarML is to have a common format where the common conceptual framework of the main i* language variations is made explicit[3].

On the other hand, all JGOOSE functionalities [4] are preserved. For example, it continues to read generated files (.tel) by the OME tool. Hence, with the E4J we expect to motivate the academic [9] and industrial communities to use the JGOOSE tool. Furthermore, the architecture proposed can also be used to enable the development of further editors (for notations such as BPMN, USe Case, etc) to be integrated into JGOOSE.

This rest of this paper is organized as follows. Section 2 discusses related works. Section 3 introduces the concepts used by i* framework to represent organizational requirements and early requirements. In Section 4, we describe the JGOOSE tool considering the integration with E4J. Section 5 outlines the E4J project and its architecture. In Section 6 we present a brief example using the E4J. Section 7 concludes the paper.

[3] Is important to notice that, in the current E4J version, it is not possible to import iStarML files. Currently, only the original i*[1] is supported.

2 Related Works

There are several i* editors available. In [20] a comparison is provided based on the version of i* supported, the availability of syntax checking in i* model as well the technology used, such as if it is open source or not. We can use these criteria to investigate if a particular i* editor can be integrated into the JGOOSE tool. Given that the JGOOSE tool supports the derivation of Use Cases from only the original i*, if a given i* model is not adequate, an inconsistent use case can be generated. Hence, the editor needs to provide syntax checking. From an analysis of the i* editors described in [10] we concluded that the OME, OpenOME, J-PRiM, Redepend React and DesCARTES satisfy almost all criteria, except i* model checking. Thus, these tools are discarded.

In sequence, we analyzed the iStarTool that satisfy all criteria. However, it was developed using the Eclipse open-source platform. So, if we tried to integrate the iStarTool we would need to refactor the JGOOSE tool code to make it compliant with the Eclipse plataform. This would require a major effort. Hence, due to lack of experience of the developers with the Eclipse plataform and considering the time to understand the iStarTool structure, we preferred to develop our E4J solution, instead of customizing any other tool.

In [11] the eXtended GOOD, XGOOD and GOOSE tools are presented. They assist the mapping of i* diagrams into UML diagrams. However, these tools depend on an external tool to generate the i* models. In [8][12][13] the iStarML is proposed. It is a common format that enables the exchange of models which are built using i* tools based on different metamodels. Their main goal is to create an ecosystem of models and techniques associated to the i* community. Our E4J tool supports the iStarML format.

3 I* Framework

When developing systems, we usually need to have a broad understanding of the organizational environment and goals. The i* framework [1] provides understanding of the reasons (Why) that underlie system requirements. It focuses on strategic actor relationships.

i* allows the description of the intentions and motivations involving actors in an organizational environment. It offers two models to represent these aspects: The Strategic Dependency (SD) Model and the Strategic Rationale (SR) Model. The SD focuses on the intentional relationships among organizational actors. It consists of a set of nodes and links connecting them, where nodes represent actors and each link represents the dependency between actors. The depending actor is called Depender and the actor who is depended upon is called Dependee. The i* framework defines four types of dependencies among actors: goal, resource, task and softgoal. The SR model complements the SD model. Its supports the modeling of the reasons associated with each actor and their dependencies. SR model assists in requirements engineering by allowing process elements and the rationales behind them to be expressed. During early requirements engineering,

the SR model can be used to understand how systems are embedded in organizational actors routines, to generate alternatives, as well as to support the reasoning that goes along the choice of alternatives.

4 JGOOSE

The JGOOSE tool [4] supports the guidelines proposed in [3] for deriving Use Cases from organizational modeling. The results of the integration processes are Use Case diagrams for the intended system and scenario textual descriptions for each Use Case using the Cockburn template [14]. Use Cases are derived considering the intentions associated to the actors of the organizational environment. JGOOSE was developed in Java and depends on OME tool [7] to create the i* models in the Telos (.tel) [5][6] format. Files in this format are used as input to JGOOSE and the Use Cases are derived applying the guidelines. Therefore, JGOOSE, in its current version, depends on an external tool for the creation of the organizational models. In order to solve this problem we propose in this paper the E4J (Editor for JGOOSE). This editor provides autonomy and improves the i* models creation and modification as well as the Use Cases derivation by eliminating the dependence on an external tool. Note that the mapping of the SD and SR models to Use Cases is supported by JGOOSE tool while the E4J editor is used to edit the i* models during the construction of the organizational models.

5 E4J

The E4J is a graphical editor for the i* framework models. It supports the creation and modification of SD and SR models as well as the mapping of parts of these models to the JGOOSE tool internal components. This new extension to JGOOSE allows the derivation of the Use Cases without the need of any intermediary file (.tel).

According to [1], the SD and SR models are composed by graphs with vertices and edges. Therefore, we can resue a module or library that supports graph edition. In this sense, we relied on the JGraphX [15] Java library to represent the i* framework models. This library is briefly described in Section 5.1. In Section 5.2 we present the E4J architecture.

5.1 JGraphX

JGraphX [15] is a Java Swing diagramming (graph visualisation) library licensed under the BSD license [17]. It provides functionality for visualisation and interaction with node-edge graphs. Some example applications that you might write with it are a workflow editor, an organisational chart, a business process modelling tool, a UML tool, an electronic circuit diagrammer and network/telecoms visualization. We know that i* organizational models can be created by using

graphs [1] and the JGraphX library can be used for it. Note that the JGraphX is open source, has a public repository updated by the community, discussion forums and free documentation. Thus, this library is an essential component in the E4J construction.

5.2 E4J Architecture

The E4J architecture is represented by four views as follows: i* organizational modeling view, Use Cases diagram as the functional view, class and components diagrams as the structural view and activity diagram as the dynamic view. Due to space limitation, we do not describe all views in this work. All views are described in [16].

The SR model in Figure 1 represents the organizational modeling view. The external dependencies define the essential relationships between the stakeholders and the E4J. These relationships can be used for deriving functional requirements of the system-to-be. For example, analyzing the SR model we can observe that the E4J actor depends on JGOOSE actor to Generate Use Cases, i.e., the JGOOSE tool is responsible for achieving this goal. JGOOSE actor depends on E4J to structure the Use Cases. It is also important to note how the E4J is integrated to JGOOSE, i.e., both are connected by the is-part-of link.

On the other hand, according to Figure 1, the Create i* models (SD and SR) goal dependence is satisfied internally through the Create i* model task which is decomposed into the tasks Initialize graph structure and Modify i* model by using the task-decomposition link. These tasks are necessary to initialize the graph structure (vertices and edges) using the JGraphX library as well as to handle the i* models.

In Figure 2, we present the Use Cases diagram generated from the SR in Figure 1. This diagram shows the main E4J functionalities (Note that it was generated by the JGOOSE tool).

In order to enable the E4J and JGOOSE integration, an option to Open E4J Editor has been included in the JGOOSE tool graphic interface (Figure 3). It activates the E4J graphic interface for the user and the context switches to the E4J control, returning to JGOOSE control only in situations described in Figure 4. Note that before E4J development, the only option was to open the .tel file and to map their internal structure to JGOOSE object oriented structure and then generate the Use Cases in UML. With the E4J integration, this option still can be used, but additionally users can also open .mxe files or create a new i* model using the E4J.

In order to improve the understanding of E4J activities, in Figure 4 we present an activity diagram with partitions: E4J, JGraphX and API IStarML. They are used to explain the possible activities sequences and restriction using the E4J. In the sequel we describe each partition:

- E4J (Partition): This partition concentrates on the the main functionalities under the E4J responsibility. Initially, the idiom and logger configuration files are loaded following by the Load Shapes action. The shapes are graphical

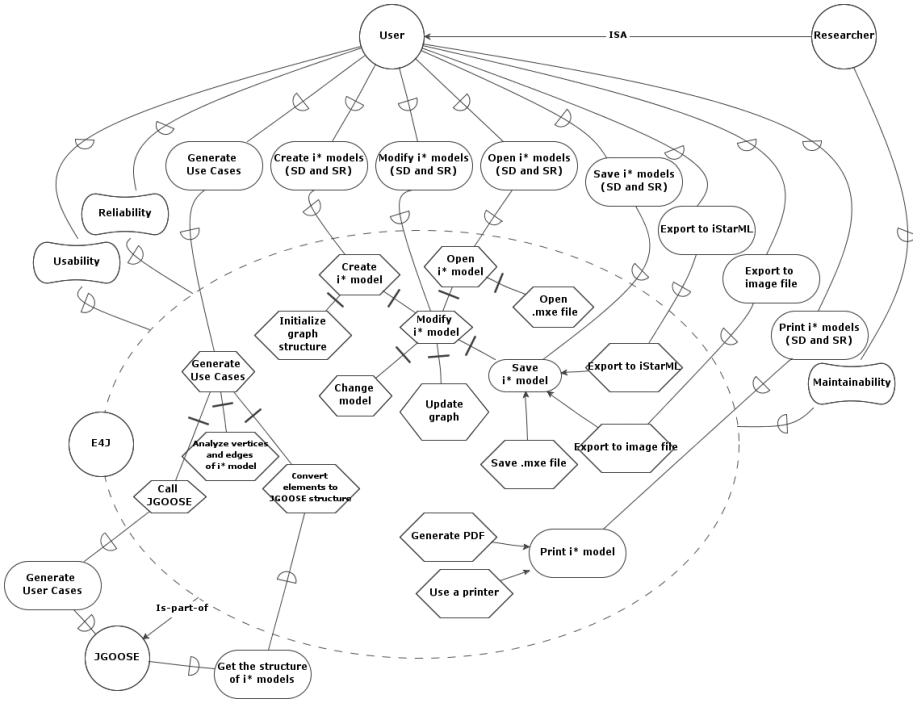


Fig. 1: SR model for E4J.

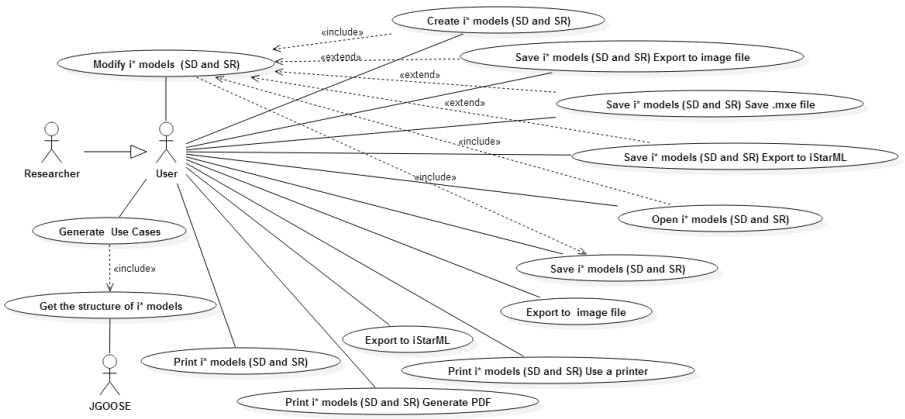


Fig. 2: Use Cases Diagram for E4J.

elements representations. The next action is Load Graphical User Interface, concluding the editor initialization process. In sequel, the graphical elements are showed to the user and users events can be receiving and sending. The connector (A) indicates that the flow is transferred to the JGraph partition

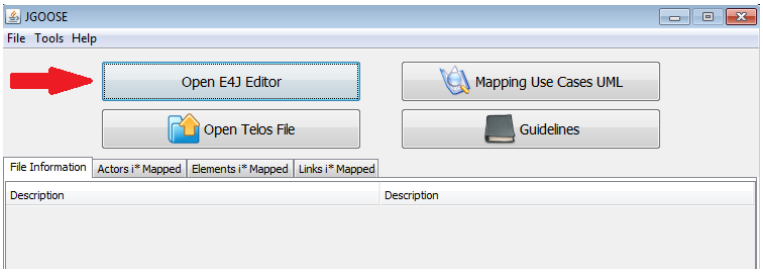


Fig. 3: JGOOSE Graphic Interface with Open E4J option

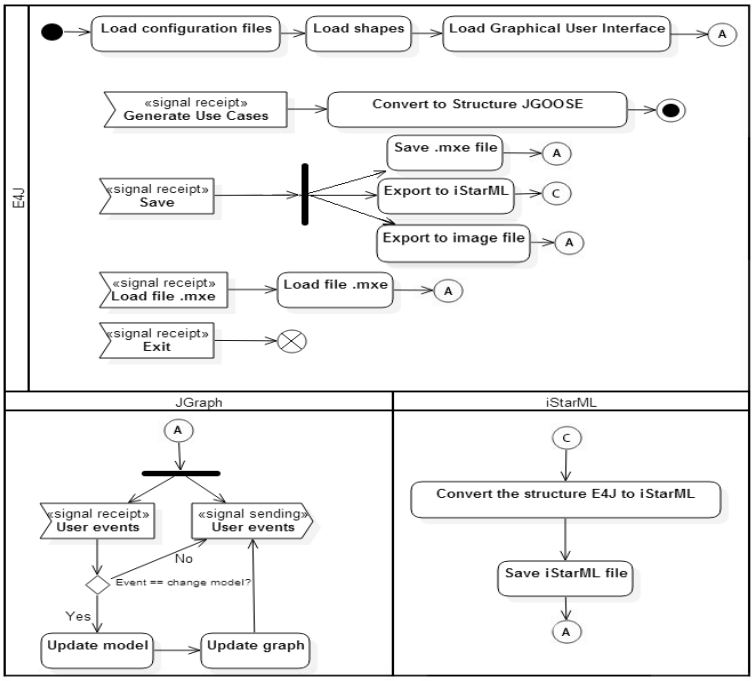


Fig. 4: Activity Diagram for E4J.

where user events are handled. The connector (C) indicates that the flow is transferred to the iStarML partition. In this partition there are four user events captured by E4J: Generate Use Cases, in which the user clicks in menu item Generate Use Cases (see Figure 10), and triggers a Convert to Structure JGOOSE action, which performs the mapping from the structure E4J to JGOOSE. Last but not least, the flow ceases the subcomponent E4J and passes to the mapping flow in JGOOSE; Save is the event generated to store the designed model in file. The file format options are the following: mxe, istarml, and image (png, jpg, bmp). The first option is a native format

of JgraphX, and after this action, E4J returns to handle the user events by connector (A). Second option is a iStarML structure. The flow is directed to the partition iStarML, which performs the necessary treatments and saves in the istarml file; Load .mxe file is an event that generates an action of same name. This action is the routine to load the file and to present it in the graphical interface. After this action, the editor returns to handle the user events; Exit is an event which user clicks in menu item File Exit or when the user presses the hotkeys Alt + F4. After this event, the execution flow of the application is over.

- JGraph (Partition): This partition is deals with the treatment (send and receive) of user events performed by JGraph, such as add, remove, and update elements. The user sends the events. These events are sent in whole application and are detected by JGraph. The main event showed in the diagram is the model updating. For example, the addition of an actor in the diagram is an update event of the model. In this flow, there is a condition that checks the type of event generated. If the event generated is an updating of the diagram event, the activity flow passes to the execution of the Update model action which updates the structure of the model according to the manipulation actions and generates the structure of the Updated graph, which is sent as an event signal to be treated in other parts of the application, such as the consistency checking of the final model. Continuing the flow, in situations where the generated event is not an updating of the model, the flow is transferred to the remaining application.
- iStarML (Partition): When E4J receives an exporting to iStarML event, the iStarML partition converts E4J structure to iStarML through the mapping routine (it considers edges and vertex of the graph). To finish the flow, the Save .istarml file routine is called, which has as input the iStarML structure and saves to iStarML file through the JAXB functionalities [18]. After finishing, the flow is returned to the treatment of graphic interface events of JGraph.

6 Using E4J

This Section relies on an example to describe the use of the tool. Note that the i* organizational models, developed in E4J, are mapped to the classes and objects of the JGOOSE tool. After this mapping, the user may generate the Use Cases from models developed in E4J.

E4J interacts with the user by means of its graphical view, which includes many resources for model manipulation and edition. The example in the sequel displays the screens, windows, and dialogues of the tool. We use the E4J editor to create the model of a generic organization, which needs an application to support its buying and selling process. This example was borrowed from [4].

The Graphical User Interface of E4J is described bellow. In order to simplify the presentation of the main screen (Figure 5), it was split in six areas as follows.

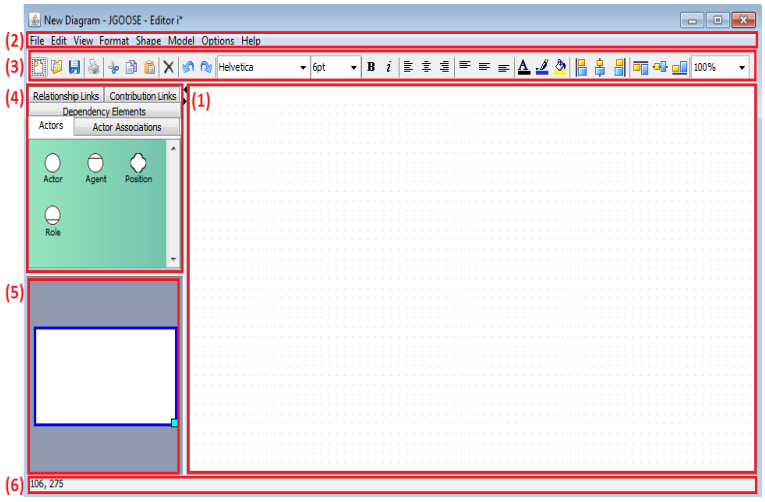


Fig. 5: Main screen in E4J.

1. Drawing Area: this is the user desktop, where the models are built. The User visualizes and handles the i* models (SD and SR) and its links;
2. Menu Bar: it provides access to all general actions on the application. All menus (File, Edit, View, Format, Shape, Model, Options and Help) are presented to provide the tool functionality, such as open/save files, zoom control, export to iStarML, and generate Use Cases;
3. Tool Bar: it is a set of shortcuts to the most common functions of a application. The ones frequently used during the construction or edition of diagrams;
4. Elements Palette: it contains five tabs that group the elements of the SD and SR models (vertices and edges). The elements are split in following tabs: Actor (Actor, Role, Agent and Position), Actor Associations (ISA, Is-part-of, Plays, Covers, Occupies and Instance of), Dependency Elements (Goal, Task, SoftGoal and Resource), Relationship Links (Dependency, Means-end and task-decomposition) and Contribution Links (Make, Some +, Help, Break, Some -, Hurt, Unknown, And and Or);
5. Mini-map: it is a miniature for the whole model. It aids the handling the big and complex models. It contains a blue rectangle representing a portion of the complete model corresponding to what is being exhibited in the drawing area;
6. Information Bar: it provides to the user the mouse position or some information about the desktop state.

The building of the SD and SR models can be summarized as addition of elements to the palette and linking among these elements. For example in order to create a SD model, you can begin adding an actor to the desktop. Hence, first select the tab associated to the target element in palette of elements (4) (Actors), then

select the element and drag it to the drawing area(1). By default, the created element is self-selected. Thus, we can use standard editors hotkeys such as CTRL + C (Copy), CTRL + V (Paste), Delete (Delete Element), and so on. The user can also rename the element.

In order to make links among elements, first it is necessary to select the appropriate link in the palette of elements tab. After the selection, it is required to click on the center of the source element and drag it to target element. Figure 6 shows this process by creating a dependence link (Employee Search ProductsSystem).

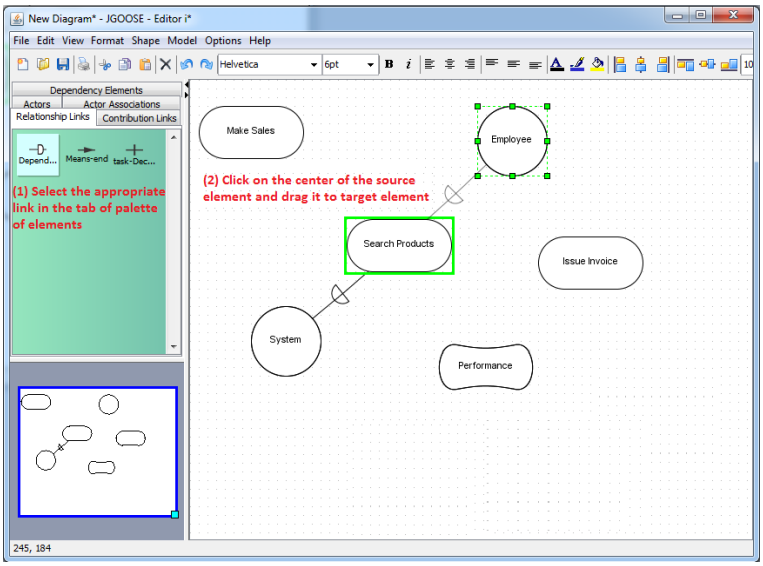


Fig. 6: Example: creating a dependence linking.

As mentioned before, the SR model is a refinement of the SD model. It details the SD model, describing the associated reasons of each actor [1]. Hence, the details are represented inside the boundary of each actor. In order to expand the boundary of an actor, it is enough to pick this actor, to click with right button, and to select the Add Boundary option. An example is presented in Figure 7, in which the System actor is expanded to detail how the dependencies that the actor.

It is worth mentioning that when an i* model is developed, some constraints and good practices regarding its construction must be considered. In E4J, all links have checking routines that analyze the source and target elements, and indicate if the link is valid or not. If the linking is invalid, a red rectangle is showed in target element and in case the creation persists, an error message is presented to the user. Thus, the syntax checking capability is much appreciated.

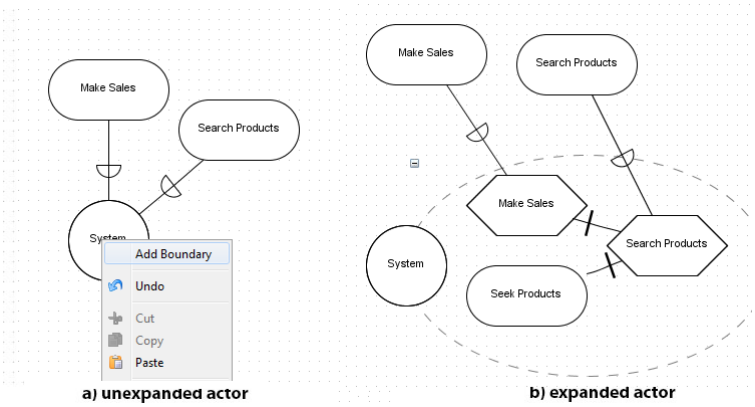


Fig. 7: Example: adding boundary to an actor.

After building the SD and SR models in E4J, we can save them in the .mxe format, export them to the iStarML format, as well as use them to generate the Use Cases in UML. For example, when exporting a SD or SR model to the iStarML format, it is enough to follow the steps indicated in Figure 8: (1) click the File menu; (2) click the Export iStarML submenu; (3) choose the folder and name of the file, and (4) click in Save option. The result is a file with .istarml extension structured in XML.

In order to generate the Use Cases from a E4J model, follow the steps indicated in Figure 9: (1) click the File menu and (2) select the Generate Use Cases submenu. After this, a JGOOSE interface will be activated.

When JGOOSE screen is enabled, the tool asks the user what is the actor that represents the system. For the given example, the user must select the System actor. After this selection, it is possible to visualize information about the opened file and existing actors, mapped elements (goals, resource, softgoal, task) and links (dependence, means-end, task-decomposition and ISA relation).

The next step is to generate the UML Use Cases. When the user clicks the Mapping Use Cases UML button (see Figure 3), it activates the screen showed in Figure 10 (a). In this example, it is possible to visualize that the unique actor mapped to the diagram is Employee and the Search Products, Make Sales and Issue Invoice Use Cases are associated to the actor, meaning that there are dependencies between these actors and the system (see Figure 9). Pressing the Diagram button, we obtain the diagram showed in Figure 10 (b). More details aboult the JGOOSE tool can be seen in [4].

7 Conclusion

In this paper, we presented the i* editor, named of E4J (editor for JGOOSE) which is integrated to JGOOSE tool. The main motivation for the editor development was the need to facilitate the process of creation and modification

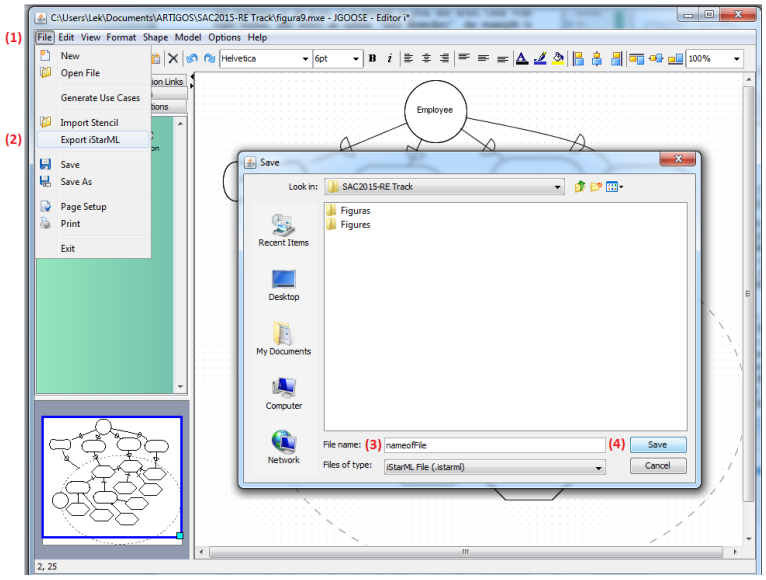


Fig. 8: Exporting diagram to .istaml.

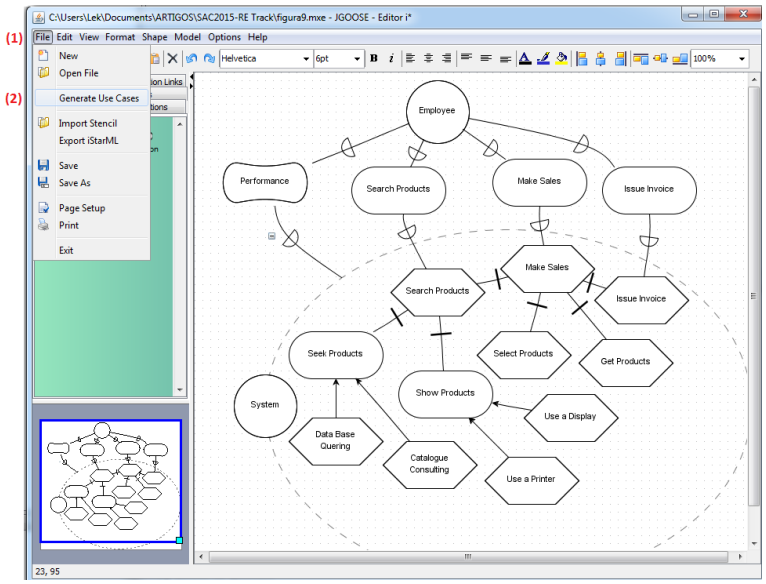
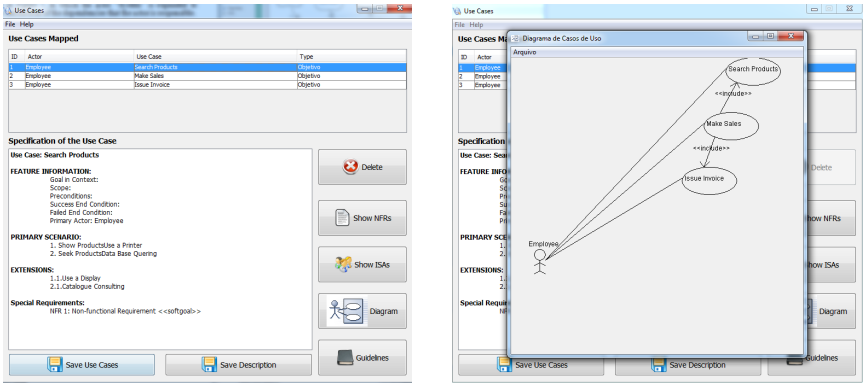


Fig. 9: Generate Use Cases. Steps (1) and (2).

of the i* organizational models and their mapping to UML Use Cases, which is currently supported by the JGOOSE tool. This integration turned JGOOSE into a standalone application, dispensing the need to have an external software



(a) Use Cases in JGOOSE. (b) Use Cases Diagram in JGOOSE.

Fig. 10: JGOOSE Screens

(extra tool) to create i* models. Hence, it also eliminates the dependence on the OME tool.

E4J supports the creation and edition of i* models. It was built to support the mapping of UML Use Cases implemented in JGOOSE tool. However, E4J runs autonomously with few adjustments.

Observing the i* tools listed in [20], we noted that most are plug-ins for other software, such as Eclipse [19]. Thus, even though the E4J development was a considerable effort, we decided to build it as an independent as possible, aiming to address our necessities of integration into JGOOSE, as well as the possibility of having a reduced size (few megabytes). This decision facilitated the inclusion, in JGOOSE, of other editors. For example, we are currently developing UML Use Cases editor and BPMN editor.

Moreover, we also considered the need for interoperability among the many modelling tools i*. Thus, E4J exports models to the iStarML [8] format.

As future works, we plan to improve the import routine that deals with iStarML structure. Thus, soon E4J will be able to import i* models in this format. Furthermore, we also want to create i* organizational models, which comply with the syntax and semantic constraints proposed in i* wiki [20]. Similarly, we intend to provide the changes in textual and graphical representation of Use Cases generated by JGOOSE to be reflected in the SD and SR models associated and built in E4J. Another future work involves the use of JGOOSE and E4J in more case studies, mainly in industrial context.

Finally, we emphasize that the tool is open-source code and is available for download [21].

References

1. Yu, E. S-K. Modelling Strategic Relationships for Process Reengineering. Phd Thesis - University of Toronto, 1995.

2. Booch, G., Rumbaugh, J., Jacobson, I. UML: User Guide, 2 Edition, Publisher Campus, 2005.
3. Santander, V. F. A., CASTRO, J. F. B. Deriving Use Cases from Organizational Modeling. In: IEEE Joint International Requirements Engineering Conference - RE, 2002.
4. Brischke, M., Santander, Victor F. A., SILVA, I. F. Melhorando a Ferramenta JGOOSE. In: 15th Workshop on Requirements Engineering, 2012, Buenos Aires, 24 a 27 de Abril. Anais do 15th Workshop on Requirements Engineering, 2012. In Portuguese.
5. Mylopoulos, J., Borgida, A., Jarke, M., Koubarakis, M. Telos: Representing knowledge about information systems. ACM Transactions on Information Systems (TOIS), ACM, v. 8, n. 4, p. 325362, 1990.
6. Koubarakis, M., Mylopoulos, J., Stanley, M., Borgida, A. Telos: Features and formalization. [S.l.]: Computer Science Institute, Foundation of Research and Technology, Hellas. 1989.
7. Yu, E., Yu, Y. Organization Modelling Environment. Available in: http://www.cs.toronto.edu/km/ome/. Last Access: February, 22 2015.
8. Cares, C., Franch, X., Perini, A., Susi, A. istarml: An xml-based model interchange format for i*. In: Proc. 3rd Int. i* Workshop, Recife, Brazil. [S.l.: s.n.], v. 322, p. 1316. 2008.
9. Santander, Victor F. A. Avaliando a utilização da Técnica i* no Processo de Ensino e Aprendizagem na Engenharia de Requisitos - Um Relato de Experiência. In: IV Fórum de Educação em Engenharia de Software, Evento integrante do XXV Simpósio Brasileiro de Engenharia de Software (SBES), São Paulo, 2011. In Portuguese.
10. Santos, B. S. IStar Tool-A proposal to support i* modeling. (Master Thesis) Centro de Informtica, UFPE. . 2008. In Portuguese.
11. Pedroza, F.P., Alencar, F.M.R., Castro, J., Silva, F.R.C., and Santander, V.F.A. Tools to the mapping of i* modeling into UML. In Proceedings of WER. 2004, 164-175. In Portuguese.
12. Colomer, D., López, L., Cares, C., Franch, X. Model interchange and tool interoperability in the i* framework: a proof of concept. In: Proc. of the 14th Workshop on Requirements Engineering. [S.l.: s.n.], p. 369381. 2011.
13. Cares, C., Franch, X., Perini, A., Susi, A. Towards interoperability of i* models using istarml. Computer Standards & Interfaces, Elsevier, v. 33, n. 1, p. 6979, 2011.
14. Cockburn, A. Writing effective Use Cases. [S.l.]: Addison-Wesley Reading. 2001.
15. Alder, G. Design and implementation of the JGraph swing component. Technical Report, v. 1, n. 6, 2002.
16. Merlin, L. P. E4J: i* Editor for JGOOSE. (monograph) UNIOESTE. 2013. In Portuguese. Available in: http://www.inf.unioeste.br.
17. BSD License Definition. Available in: http://www.linfo.org/bsdlicense.html. Last Access in: February, 22 2015.
18. Fialli, J., Vajjhala, S. The Java architecture for XML binding (JAXB). JSR, JCP, January, 2003.
19. Eclipse The Eclipse Foundation open source community website. Available in: https://www.eclipse.org/. Last Access in: February, 22 2015.
20. i* Wiki. Available in: http://istar.rwth-aachen.de/tiki-index.php. Last Access in: February, 22 2015.
21. Software Engineering Laboratory - UNIOESTE. Available in: http://inf.unioeste.br/les/. Last Access in: February, 22 2015.