Implementing a tool to Support KAOS-Beta Process Model Using EPF

Malihe Tabatabaie
Malihe.Tabatabaie@cs.york.ac.uk

Department of Computer Science
The University of York
United Kingdom

Eclipse Process Framework

March 2010

Contents

List of Figures

Chapter 1

Introduction

This chapter introduces the required elements for using EPF to develop Kestrel. Kestrel is a tool to support KAOS-β process model. More detail information about Kestrel and KAOS-β will be demonstrate in Chapter 2.

1.1 Process

Process is a well-known term in different fields of science. [6, P. 28] defines the software process as follow:

> A software process can be defined as the coherent set of policies, organizational structures, technologies, procedures, and artifacts that are needed to conceive, develop, deploy, and maintain a software product.

In this definition the aim of the process is to "conceive, develop, deploy, and maintain a software product". The middle steps are "coherent set of policies, organizational structures, technologies, procedures, and artifacts", however the start point in not clear. In cases like KAOS-β the start point could be implicit, for example the investigation within the environment of the final product.

[1, P. 76] from [5] defined process as follow:

> A set of partially ordered steps intended to reach a goal.

Further in their work authors defined process steps and process elements as follow:

"Any component of a process is a process element. A process step is an atomic action of a process that has no externally visible substructure"[1, P. 76].

Further research illustrates that [8, P. 6] in his software engineering book defined software process as follow:

> A set of activities whose goal is the development or evolution of software.

The common aspect between these definition is following a set of rules, activities, or steps that lead the user to a defined goal. In none of these definitions the source of the rules and the initial step is emphasised. As it was mentioned before they could be implicit but in our definition we make it explicit because we feel it is important for the designers of the process to have a clear idea of where and how to start developing their process.

We define the process in software and system engineering domain as follow:

A set of steps, rules, descriptions, and actions that initiate from a phase or step and continue towards a defined goal. Therefore, three elements of a process is the start point, end point, and the middle steps or rules that leads the users from the start point to the destination or aiming point.

In practice process definition includes the detail implementation that could differs from project to project such as allocating resources which is project dependent. In theory there is a demand to define a process that is not specific to individual projects therefore another concept is created to solve this problem. *Process model* which is "an abstract representation of an actual or proposed process" [1, P.76]. [8, P.6] in his book defined process model as "a simplified representation of a software process, presented from a specific perspective". This definition of process model uses the term simplified representation which compare to abstract version is not informative enough. It also limits the process model to a specific perspective. We consider the abstract term because in the abstract version the process model is complete even though it does not include the implementation detail. Therefore, we consider the first definition for process model and define KAOS-β as a process model based on this definition.

EPF is a framework or tool to develop processes, therefore it defined some elements for process that could be used as a starting point to evaluate the process at the same time as defining it. To the best of our knowledge Eclipse Process Framework (EPF) gives no definition for the process model. However it provides the following definition for the process concept:

> A process describes how a particular piece of work should be done. The work may have a relatively small scope, in which case it can be described as a capability pattern, or may address a full

project lifecycle, in which case it can be described as a delivery process.

This definition is compatible with the definition that we proposed therefore we used EPF to develop KAOS-β systematically. Using EPF helps us to evaluate KAOS-β by illustrating the missing or miss-placed elements. Next section introduces EPF in more detail.

1.2 Eclipse Process Framework

What is EPF

Eclipse Process framework (EPF) is "a tool platform for process engineers, project leaders, project and program managers who are responsible for maintaining and implementing processes for development organizations or individual projects" [7, P. 4].

This tool could be used for developing software processes in particular. It proved this ability by presenting OpenUp as an example. OpenUp will be introduced in more detail in section 1.4. The main goals of EPF are [4, Slide. 5]:

- An extensible framework and tooling for authoring, configuring and publishing processes
- Exemplary processes - first delivered is OpenUP

We used this tool author, configure and publish KAOS-β process model. We also used this tool to analyse and evaluate KAOS-β. The aim is to apply a systematic approach to define KAOS-β and search for part that could be improved. EPF is a systematic approach to define a process model like KAOS-β because it introduces process's elements and their relation automatically. Section 1.3 introduces the elements that are used for KAOS-β in more detail.

1.3 EPF Elements

The main elements of a process are its tasks, roles and work products. This section defines these elements and some other ones that are used in KAOS-β development from [3].

Task A task is an assignable unit of work. Every task is assigned to a specific role. The duration of a task is generally a few hours to a few days. Tasks usually generate one or more work products.

Role A role is a well-defined set of related skills, competencies, and responsibilities. Roles can be filled by one person or multiple people. One person may fill several roles. Roles perform tasks.

Role set A role set is used to group roles with certain commonalities together. For example, in a software development environment, an Analyst role set could be used to group together roles such as Business Process Analyst, System Analyst and Requirements Specifier. Each of these roles work with similar techniques and have overlapping skills, but may be responsible for performing certain tasks and creating certain work products. Role sets can be organized using role set groupings.

Step A step is a part of the overall work described for a task. The collection of steps defined for a task represents all the work that should be considered to achieve the overall goal of the task. Not all steps are necessarily performed each time a task is invoked in a process. Steps are generally unordered and can be performed in any order.

Work product Work product is a general term for task inputs and outputs, descriptions of content elements that are used to define anything used, produced, or modified by a task. The three types of work product are: Artifact, Outcome, Deliverable.

Activity Activities are the main building blocks for processes. An activity is a collection of work breakdown elements such as task descriptors, role descriptors, work product descriptors, and milestone. Activities can include other activities. Activities can be presented in work breakdown structures and activity diagrams that graphically describe the flow of work by showing which activities precede other activities. phase and iteration are special types of activities that define specific properties.

Artifact An artifact is a tangible work product that is consumed, produced, or modified by one or more tasks. Artifacts may be composed of other artifacts. For example, a model artifact can be composed of model elements, which are also artifacts. Roles use artifacts to perform tasks and to produce other artifacts. Each artifact is the responsibility of a single role, making responsibility easy to identify and understand, and promoting the idea that every piece of information produced in a method requires the appropriate set of skills. Even though only one role is responsible for an artifact, other roles may use the artifacts.

Checklist A checklist is a specific type of guidance that identifies a series of items that need to be completed or verified. Checklists are often used

in reviews such as a walkthroughs or inspections.

Domain A domain is a hierarchy of related work products grouped together based on timing, resources, or relationship. While a domain categorizes many work products, a work product belongs to only one domain. Domains can be further divided into sub-domains.

Outcome An outcome is an intangible work product that may be a result or state. Outcomes may also be used to describe work products that are not formally defined. Phase

Roadmap A roadmap is a specific type of guidance that describes how a process is typically performed. Often processes can be much easier understood by providing a walkthrough of a typical instance of the process. In addition to making the process practitioner understand how work in the process is being performed, a roadmap provides additional information about how activities and tasks relate to each other over time.

After defining the main terms that are used in EPF and KAOS-β and before starting to explain how we implement KAOS-β in EPF it is useful to review a current software engineering process that is implemented in EPF. Reviewing OpenUp helps us to illustrates the power of EPF. Checking the current examples and tutorials of EPF usually use OpenUp completely or part of it. We also used OpenUp to learn how to present KAOS-β in EPF.

1.4 OpenUp

OpenUP is the example process that ships with EPF Composer. “An Agile Inspired process with its roots in the UP” [4, Slide. 8].

“An iterative software development process that is minimal, complete, and extensible

- Minimal - Contains vital roles, tasks and guidance
- Complete - Complete for small co-located teams
- Extensible - Serves as a foundation that can be extended and tailored“ [4, Slide. 8]

Example of its tasks are assess results, design solutions, and detail requirements [1]. Examples of its roles are analysts, architecture, and developer.

[1] org.eclipse.epf/libraries/OpenUP/openup/tasks

Some of the work products defined by OpenUp are build, design, and use-case. We reviewed these elements in OpenUp to check if some of the elements could be valid for KAOS-β too. However, we considered that KAOS-β is a process model for the early stages of software system development whereas OpenUp is for the complete software system development, therefore KAOS-β is more focus and OpenUp is in a bigger scale.

1.5 KAOS-Beta Results

In the last sections we mentioned KAOS-β frequently, this section introduces this process model in more detail. KAOS-β is a process model that is an adaptation of KAOS methodology in the enterprise information systems (EIS) domain. KAOS is a goal-driven methodology, designed to elicit and validate requirements and to prove their consistency [2].

To tailor KAOS we applied it to the Stroke care example which is an example of EIS. The result of applying KAOS to Stroke care demonstrates the required steps for defining KAOS-β. Thus an activity diagram was created that illustrates the basic steps or activities of KAOS-β process model. Considering this activity diagram and the rules applied by EPF we add more detail information that was required by EPF to create Kestrel which is a tool to support KAOS-β. kestrel is a web-base tool that illustrates the elements and steps of KAOS-β.

1.6 Summary

In summary this chapter presents a background to introduce various concepts about KAOS-β process model and EPF. These concepts will be used in the next chapter in using EPF to develop Kestrel.

Chapter 2

Implementation

The implementation section includes the methods (see Section 2.1) and results (see Section 2.2). In the documentation of EPF it can be understand that they are advising the users to edit the current libraries and Open-up process to create a new process. Indeed using the best practices helps the developers to follow a standard and make less mistakes, however we find editing these library plug-in and packages so complicated for a small process model like KAOS-β that increases the complexity without adding more value. In exchange, we developed the process from scratch with an eye on the best practices in Open-up and other EPF examples, to make it as simple as possible and as standard as possible. The first part of this chapter describes the general approach and guideline to develop a process model. In second part of this chapter we apply the results of KAOS-β to the process model tool.

2.1 Method: Eclipse Process Framework

> *The **goal** of this phase is to develop a tool to support a process using EPF.*

EPF provides the requirements for developing a process. Requirements such as roles and tasks. EPF is a framework that defined the process of developing a process, therefore, there are requirements and steps that should be followed to develop a process and define its elements in EPF. Next section illustrates these steps:

2.1.1 Steps

The EPF process is as follow:

1. Create a Method Library
2. Create a Method Configuration
3. Create a Plug-in
4. Create a Content Package (Method Content)
5. Publish the results

"All method contents are stored in a method library" [9, P. 12]."The Method Library contains Method Plug-ins and Method Configurations" [9, P. 35]. To create the new method library, method configuration, and plug-in, the developer simply goes to $File > New > ...$ then the required option should be chosen. The sequence of creating these elements are important and is forced by EPF. Method Configuration "allow you to specify working sets of content and processes for a specific context, such as a specific variant of the OpenUp framework that you want to publish and deploy for a given software project or as a foundation for a development organisation" [9, P. 32].

"All content and processes are organised in method plug-ins and method content packages." [9, P. 32] By creating a plug-in two sub categories of *Method Content* and *Process* are created under the newly created plug-in. To create a Content Package the developer should open the tree categories under *Method Content* and right click on the content package category. In this way there is an option to choose $New > ContentPackage$. The *method configuration* should be selected during the whole process.

Until this step the skeleton of a process is created. From now the information related to the given process should be inserted into the process elements. By creating a method content three sub categories will be created automatically. These three sub-categories are as follow:

- Content Packages
- Standard Categories
- Custom Categories

From top to bottom the information of each step should be created. However, it is just a suggestion if developers do not have a specific priority and plan, otherwise this is an iterative process that does not follow a restrict order.

Content Packages

Content packages "allow you to manage your content in configurable units" [9, P. 30].

To create a new content package for each process developers should right click on the Content Packages category and choose ($ContentPackages > New > ContentPackage$). By doing so a new_content_package is created that could be renamed to the relevant context.

By creating a content package, four main elements of a process which are roles, tasks, work products, and guidance are created. In the next step new instances out of each of these classes or categories should be created and filled with the process's informations. To create the instances of each of the elements, the similar process as creating a new object is required. Right click on each element, and choose new and then the particular element.

Standard Categories

"Standard categories provide a means to categorise core method content in line with best practices for creating structured methods. To encourage good method structure, there are standard categories for grouping tasks into disciplines, work products into domains, roles into role sets, and tool-mentors into tools. Unlike custom categories, standard categories, by definition, are linked to a specific type of method content" [9, P. 46].

After creating a set of roles, tasks, work products and Guidance the next step is to categorize them in relevant groups for a better presentation and following best practices. to created each category, developers should follow the same approach which is right click, new , and then the choose the required option.

Custom Categories

After creating the process elements and categorise them, it is the time to prepare them as a web-base tool. Custom category section will create the view for the web-base tool. Thus same as other element we can have different views by creating different custom categories. To create a custom category, right click on the $CustomCategory > New > Customcategory$. The new custom category requires the relevant information for that view in Description mode (see Figure 2.2).

Then in the Assign mode, the related content elements from standard categories could be assigned to the list. later this list could be chosen as the preview list in the web-base tool.

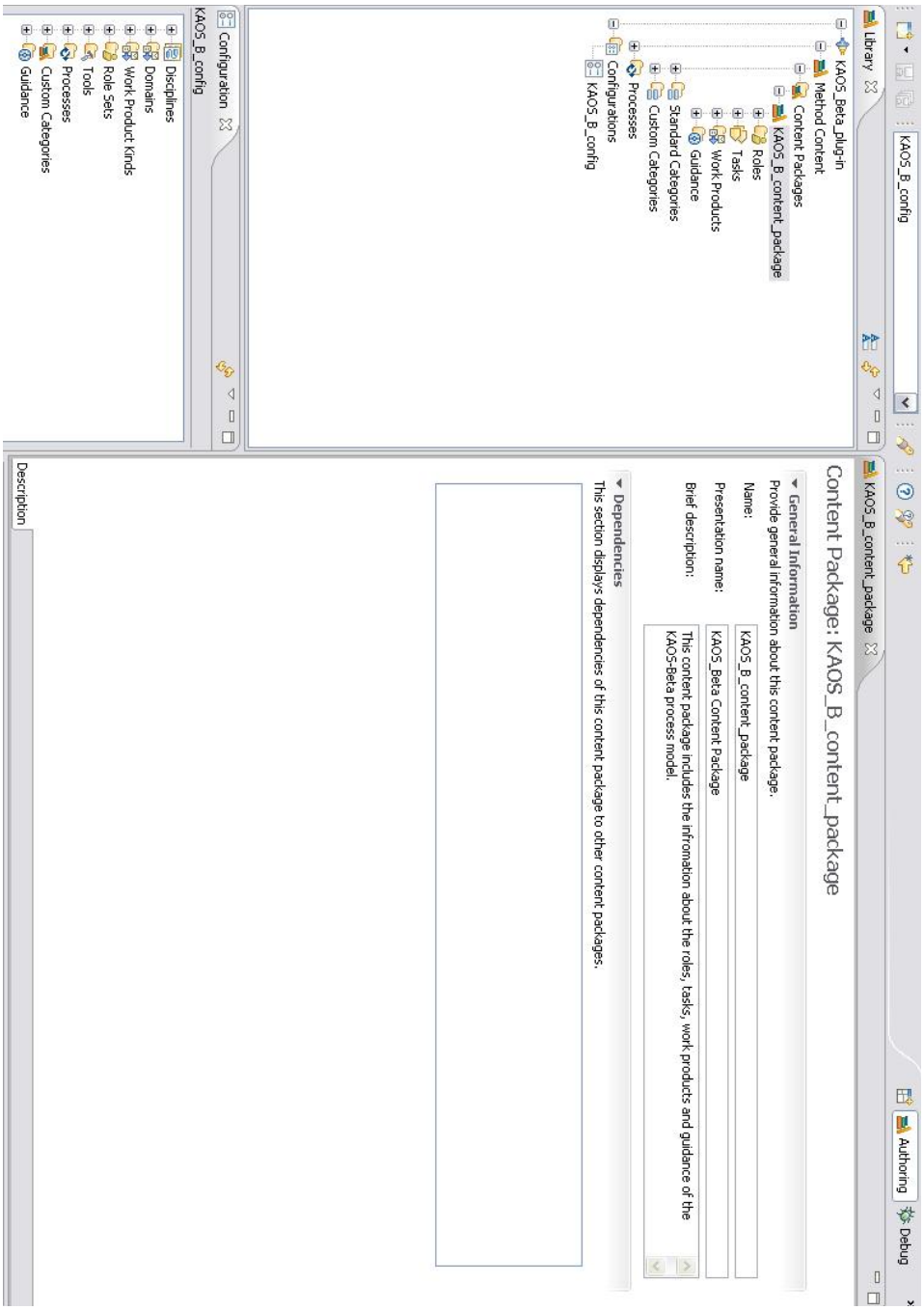

Figure 2.1: KAOS-B content package

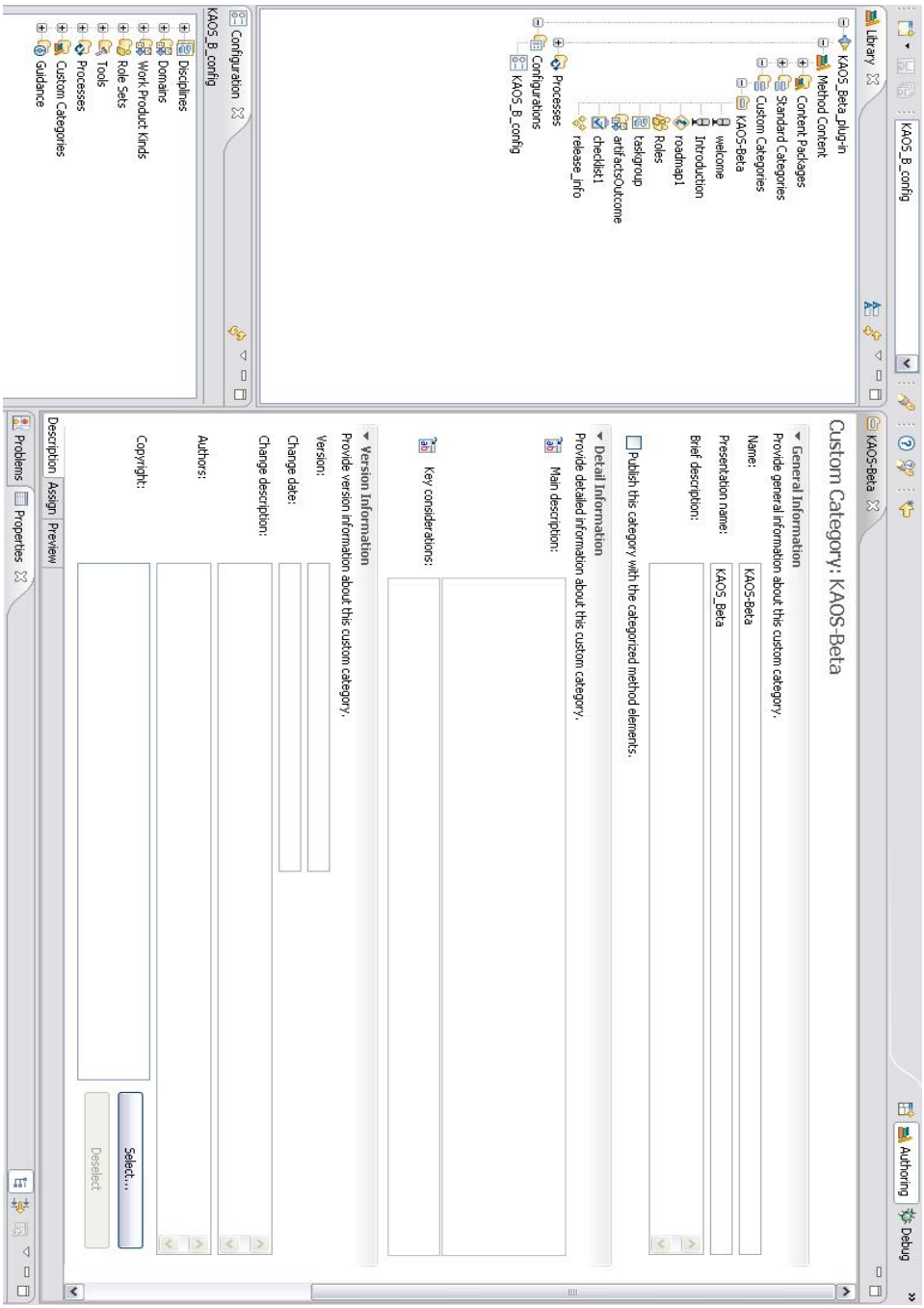

Figure 2.2: Custom Category Description

2.1.2 Activity Diagram

EPF gave the ability to create three types of diagrams in its *capability patterns* section: activity diagram, activity detail diagram, user defined dia-

grams. Designers also can create diagrams in delivery processes section. The difference between capability pattern and delivery process is defined as bellow:

> The work may have a relatively small scope, in which case it can be described as a capability pattern, or may address a full project lifecycle, in which case it can be described as a delivery process. Capability patterns are used as building blocks to compose delivery processes, they describe reusable clusters of activities in common process areas. A delivery process describes a complete and integrated approach for performing a specific type of project [9].

"Capability patterns are a special type of process that describe a reusable cluster of activities in common process areas. Capability patterns express and communicate process knowledge for a key area of interest, such as a discipline, and can be directly used by process practitioners to guide their work. Capability patterns are also used as building blocks to assemble delivery processes or larger capability patterns ensuring optimal reuse and application of the key practices they express" [3].

Base on this given definition, capability pattern describes a reusable cluster of activities to satisfy a process whereas delivery process "provides a complete lifecycle model with predefined phases, iterations, and activities" [3]. In the case of KAOS-β process model, the term *process model* emphasises the abstract characteristics of the KAOS-β. Based on our understanding, capability pattern is more suitable for the abstract description compare to delivery process which is a complete lifecycle model.

In the early version of the Kestrel, the KAOS-β includes a capability pattern. In this capability pattern we defined an activity diagram. To create an activity diagram create a new capability pattern (right click on capability pattern> new> capability pattern> in the work breakdown structure right click on the capability pattern> diagrams> open activity diagram)

In the activity diagram environment (see Figure 2.3) there is a set of tools to create the activity diagram and store the required information. This activity diagram will be added to the output of the published results.

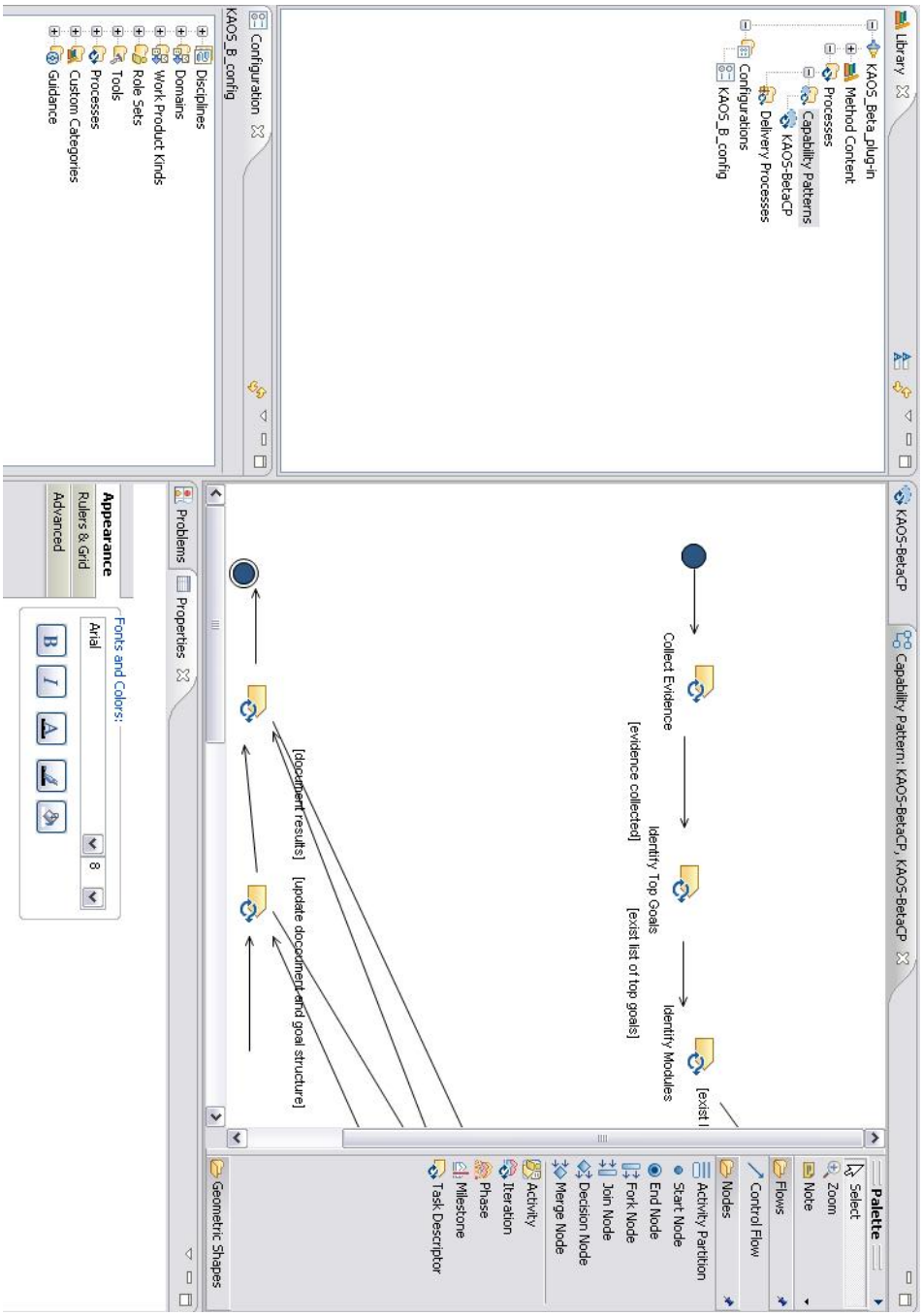

Figure 2.3: Activity Diagram

2.1.3 Publish the Tool

To publish the final results, developer should choose the configuration file from the drop-down list in the menu part and in the library, in the configuration section. For example, in our case the *KAOS-B_config* is the configuration file that is selected in the drop down menu and in the library under the Configuration category. By double clicking on the file the developer can have access to the settings of configuration file (see Figure 2.4). After ensuring that the related plug-in is selected in the *plug-in and Package Selection* mode, in the *Views* mode the developed view in *Custom Categories* section could be added by pressing *Add view* bottom and selecting one or more suitable custom categories. After Saving the results, the next step is to choose publish from Configuration menu. In the process of publishing developer can set some settings on how it should be presented. For example, part of the process plug-in or all of it could be static web site or Java application.

If you choose a web site, after publishing the results, a web site containing the given information will be published by EPF. Users can navigate through this web site to find the relevant information. How user friendly is the website is partly forced by a rigid structure from EPF but still it does not guarantee the most useful information for the final users because the content of information depends on the developers.

2.2 Results: Tool with the Data

> *The **goal** of this phase is to add the KAOS-β data to the skeleton of the tool that was developed in Section 2.1.*

After become familiar with the power and functionality of EPF, it is the time to put the information regarding to KAOS-β into the process's skeleton. This skeleton provides the structure for defining a process hence it gives the roadmap and requirements for a process. Using this structure, some of the hidden aspects of the KAOS-β become more clear.

Roles: Based on the results of applying KAOS in stroke care example to develop KAOS-β, we could extract following four basic roles:

Analyst is a team or individual who collect the suitable evidence from the environment of the enterprise information system. They create an understanding of the system for the designer team. Analyst collects various type of information that could help the designers.

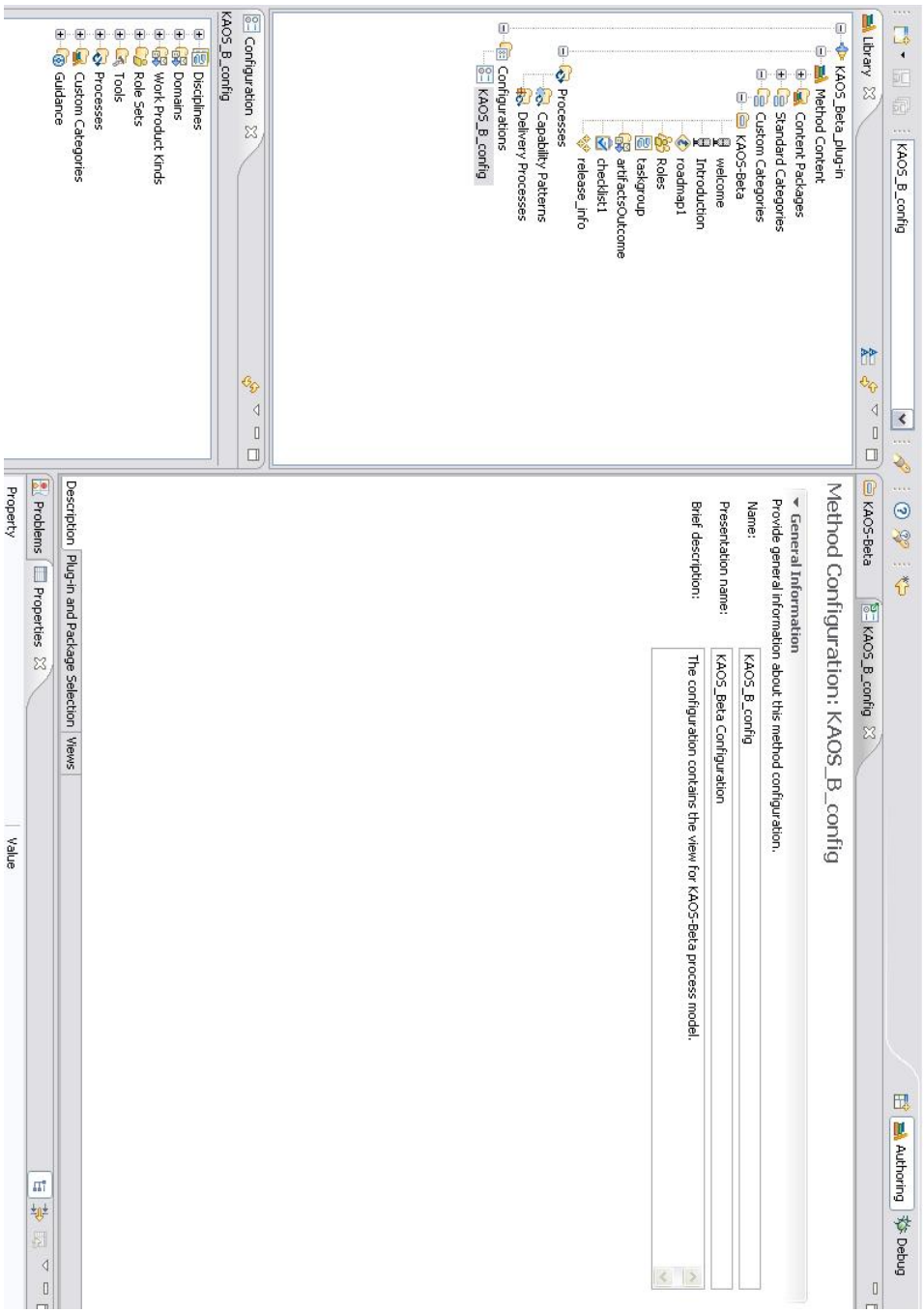

Figure 2.4: Configure Description

This team is better to be involved with the experts of the environ-

ment to collect a better understanding of the expectations from the system.

Designer team analyse the evidence collected by analyst and extract the required information regarding to the goals, refinement and agents of the system. They are familiar with the concepts of KAOS-Beta process model, hence they use the Kestrel tool (tool to support KAOS-Beta process model) to apply this method to the evidence collected by analysts.

Stakeholders are the agents in the systems. They are the experts, decision makers and users of the system. With the help of stakeholders, analyst collect evidence for designers. Testing team use stakeholders to evaluate the results.

Tester team is the team separated from designer and analyst team. They evaluate the goal structure with the help of stakeholders and pass the results to the designer team.

Tasks: Thirteen major tasks are detected for KAOS-β. Each of these tasks could be designed in more detail for each project.

- *Collect Evidence*: Collecting evidences such as document, interviews, report etc. that can help the designers to refine and define the goals from them.
- *Identify Top Goals*: In this task the designers derive some top goals from the collected evidence.
- *Identify Modules*: Considering the evidence and understanding from the top goals, designers group the top goals in different modules if possible to structure the system goals.
- *Prioritise Modules*: Based on the evidence and decision makers' opinion, designers will prioritise modules in compare to each other. This method helps to allocate suitable resources for each module.
- *Goals in Each Module*: Identify Goals in Each Module. this task focuses in each module individually. from this phase the tasks will be applied to each module.
- *Document Goal*: For each goal a form should be filled to help to document the goal in a consistent way.
- *Refine Links Between Goals*: When there is more than one goal in the goal structure there should be links between goal to justify their existence. in addition it helps to trace the goals to their

source goal and destination. In this task the designers derive some links between goals from the collected evidence and personal justification.

- *Document Links*: This optional tasks refers to documenting the links. It can be done as part of goal documentation or separately. It depends if it does not add extra cost without adding enough value. The aim is to structure the goal structure to manage the complexity by creating traceability.
- *Identify Agents*: Considering the definition of agent in KAOS process, identify the agents in the system and bare in mind that agents should satisfy one or more goals. Therefore, in later tasks the relationship between goals and agents should be clarified.
- *Link Goals to Agents*: The ultimate aim is to link each goal to suitable agents. This helps to allocate the responsibilities for satisfying each goal.
- *Identify Obstacles, threats, Conflicts*: This task can break to many smaller tasks that explain how to identify each of the defined groups using different techniques.
- *Link Modules*: Modules could be linked together using shared goals. Documenting these shared goals and making their relationship with different modules clear makes the links between modules more clear.
- *Evaluation*: Evaluate the goals and flow of the goal structure.

Work Products: Work products are the elements that clarifies the tasks inputs and outputs. After defining the tasks, work products help to create a smooth flow between tasks. We defined 12 work products that defines the input and output of each tasks. From the characteristics of the work product, it seems that there is a need to define more work products because we have defined thirteen tasks, considering each tasks has one input and one output. Thus there should be at least 14 work product for the whole process. After reviewing the defined work products we can define the missing ones.

- *Evidence*: A list of documents, reports, sample of other systems, interviews with experts and users and information that can help the designers to understand the goals or requirements of the systems.

- *Top Goals*: A list of top goals. This list in the following steps will be analysed in more detail to document each goal.
- *List of Modules*: List of possible modules, explicitly justified by point of view.
- *Prioritised Modules*: A list of prioritised modules.
- *Goal Documentation*: Forms to be filled for each goal.
- *Goal Links to Modules*: The goal is a link to other modules. The status could be yes and no. Based on this status suitable information should be added to the goal documentation.
- *Refinement*: Goal documentation with added refinement information.
- *Refinement Documentation*: This section could be the goal documentation including the refinement information.
- *Agents List*: A list of Agents and their relationships with the goals. Each agents should be linked at least to one goal.
- *Agent Goal Documentation*: Expanded version of the goal documentation that includes agents information.
- *Obstacle Documentation*: A list of possible obstacles, threats and conflicts between goals and the links between them.
- *Evaluation Results*: It could be a document that contains the results of evaluation.

Guidance: Guidance contains required information about KAOS-β that could help the reader to understand and apply this process model to their project. This guidance is organised in five different categories. It includes introduction, welcome message, roadmap, checklist and the release information.

- *Welcome*: This part of the guidance introduces the main elements of KAOS-β in Kestrel web-base tool. In this section it is emphasised that Kestrel is a tool to support the KAOS-β process model by guiding the users through this process rather than storing the results of applying the steps.
- *Introduction*: This section introduces the KAOS-β process model in a very high level. It provides some references to the main textbook of KAOS-β that could help the readers to find further information.

- *Roadmap*: This section presents the activity diagram that KAOS-β is following without extra information. This high-level diagram not only introduces the KAOS-β process model graphically but also presents the flow and links between tasks in the tasks section.
- *Checklist*: This section presents the criteria for evaluating the output of KAOS-β. These criteria are collected from literature and are qualitatively arguable. In cases such as completeness, the designers could argue over the lack of completeness, and completeness is not a measurable criteria. These criteria are suitable for evaluator team and designers to evaluate the results before discussing it with the stakeholders for the final evaluation process.
- *Release Information*: This section presents the required information about the version of the product. In the future attempt to improve this tool, this section will be updated.

Standard Categories: This category create standard categories for the elements created in content package section. If the process is in a large scale, it is useful to organise the elements in smaller related groups, such as roles, tasks, and artifacts.

Custom Categories: This section provides the view, hence we assign the elements to the publishable view. In Kestrel tool, we assigned all the created elements to the tool interface.

Capability Pattern: To present KAOS-β activity diagram in more detail, a capability pattern created in this section. The main information in this section is the activity diagram. EPF allows the designers of the process to choose the defined tasks and roles for the activity diagram. Therefore by clicking on the elements of the diagram final users can navigate through the KAOS-β process model elements easily.

Publish: To publish the results in the web-base or Java EE web application, EPF provides the publish functionality. This option can be chosen from configuration menu. To publish the results first double click on the KAOS-β configuration method and choose views sub-section. In this section designers could add view. If the custom category item has a defined item, it should be visible in list of possible views to choose. It is possible to add different views. In the case of kestrel we added one main view that contains all the defined elements so far. The result of publishing this view is a web-base tool that navigates the users through the defined information for KAOS-β process model. using this tool users can apply KAOS-β in their projects.

2.3 Summary

This chapter presented the EPF tool in practice. By applying KAOS-β process model to EPF and developing Kestrel, which is a tool to support KAOS-β we demonstrate the capabilities of EPF in practice. Discussion chapter (Chapter 3) illustrates part of the abilities and limitations of EPF and Kestrel that we faced during this empirical study.

Chapter 3

Discussion

In this chapter we review our experience of using EPF for developing Kesterel. EPF in the case of this project gave us the ability to analyse and evaluate KAOS-β. It defined a set of required elements for a process; hence, following these requirements made the development of KAOS-β more systematic. By using a systematic approach we can compare the elements of the process with the requirements of the tool and it helps us to improve the process and find out the missing parts. For example in the case of KAOS-β, we have the concept of agents which is very similar to the roles concept in the EPF, therefore we considered agents as roles and defined one or more role (s) for each task. However, even these roles are the agents but the agents in the KAOS-β are more than the roles defined in the EPF. Another helpful aspect of EPF which was implicit in early definition of KAOS-β was work product. EPF requested the developers to define work products for each tasks which basically defines the input and output for each task. These work products makes our activity diagram more complicated but it helps the users of the KAOS-β to have a clear expectations from each task or step of the process. In cases it can give a clear results for evaluation if it is required.

During using EPF and reading other users comments we observed some challenges that will be illustrated briefly.

3.1 Challenge

- One of the early challenges of using EPF that we faced with was the lack of documentation for EPF's elements and essential terms. Even though we could find definitions for the terms (see Section 1.3) in EPF but applying them in practice require some examples to clarifies the terms. After trying OpenUp and reading these definitions and other

documentations the terms become more familiar to use.

- Most of the tutorials focus on reusing the current practices in EPF such as editing OpenUp. Barely they explain how to develop a process from scratch. In our case, editing OpenUp creates more complexity without adding extra value. Therefore we needed to understand how to develop a process without reusing OpenUp elements.

- In practice, when a task was added, in some cases I wanted to move the created tasks up and down in the list to follow an structure that helps me and the reader to follow the sequence of the tasks. But I found no way to move the tasks and rules after creating them. By default EPF sort out the tasks and roles by name which in cases it is not convenient. For example in our case we wanted to order the tasks by their position in the activity diagram. Thus we gave numbers to the names to sort it in the way that we wish. It still is presented exactly in the order that we wish.

- How easy to use is the results is very dependent on the design and personal choice, but it is clear that this tool gives a clear understanding of the process requirements.

- We found publishing the results is very confusing for the beginner users. In general all these simple actions look and feel very complicated because of lack of user-friendly help and tutorials.

- Technical problems such as crashing and loosing the library in addition to the limitations of diagrams for presenting the process was repeatedly reported in EPF forum.

Bibliography

[1] Bill Curtis, Marc I. Kellner, and Jim Over. Process modeling. *Commununications of the ACM*, 35(9):75–90, 1992.

[2] Emmanuelle Delor and Robert Darimont and Andŕe Rifaut. Software Quality Starts with the Modelling of Goal-Oriented Requirements. [Accessed 20 Sep 2009] Available at:www.objectiver.com, 2009.

[3] EPF Team. Help- Eclipse Process Framework Composer [online]. [Accessed 30 March 2010] Available at: epf.eclipse.org, March 2010.

[4] EPF Team. Introduction to the Eclipse Process Framework [online]. [Accessed 30 March 2010] Available at: epf.eclipse.org, March 2010.

[5] Peter H. Feiler and Watts S. Humphrey. Software process development and enactment: Concepts and definitions. In *ICSP*, pages 28–40, 1993.

[6] Alfonso Fuggetta. Software process: a roadmap. In *ICSE - Future of SE Track*, pages 25–34, 2000.

[7] Peter Haumer. Eclipse Process Framework Composer (Part 1: Key Concepts). Technical report, IBM Rational Software, 2007.

[8] Ian Sommerville. *Software Engineering*. Addison Wesley, 8 edition, 2006.

[9] Bjorn Tuft. Eclipse Process Framework (EPF) Composer, Installation, Introduction, Tutorial and Manual [online]. [Accessed 20 Febrary 2010] Available at: epf.eclipse.org/uploads/14.pdf, Febrary 2010.