DICES technical report

Requirements on the system design phase for PROGRESS-IDE

Project name: DICES
Contract number: No. 03/07
Author(s): Juraj Feljan, Luka Lednicki, Ana Petričić, Séverine Sentilles

Executive summary

This technical report gives an overview of requirements to be used as guidelines for the development of PROGRESS-IDE. PROGRESS-IDE is a tool that will support development of embedded systems using ProCom, covering the systems' lifecycle from the design- to the deployment phase. The document is focused on the requirements for the architectural editors that will enable graphical definition of ProCom component types, and graphical composition of systems consisting of ProCom components. The requirements for the architectural editors are defined in detail by use case diagrams. Prior to detailing requirements, a general overview of PROGRESS-IDE is given.

Table of Contents

1 Introduction

PROGRESS-IDE is an integrated development environment used for software development of distributed embedded systems, primarily in the automotive, automation and telecom domains. It is being developed at Mälardalen University (MdH), Västerås, Sweden, as part of the PROGRESS project [1]. The purpose of this technical report is to give a detailed description of the requirements on PROGRESS-IDE concerning the system design phase. This includes the specification of the *architectural editors* used for modeling a system with the PROGRESS Component Model (ProCom) [2] (Section 3) and a detailed specification of use cases (Section 4). Prior to the requirements specification, we give a brief overview of the vision behind PROGRESS-IDE and the current state of the environment in Section 2. In the remainder of Section 1 we give definitions of used terms, a list of used acronyms and abbreviations and a comparison of PROGRESS and DICES [3] projects.

1.1 *Definitions*

Table 1: Definition of terms used

Keyword	Definitions
PROGRESS	Strategic research centre funded by the Swedish Foundation for Strategic Research
PROGRESS-IDE	integrated development environment for developing distributed embedded component based systems
MRTC	Mälardalen Real-Time Research Centre
ProCom	component model developed at Mälardalen University, successor to SaveComp Component Model (SaveCCM)
ProSys	upper layer of ProCom component model
ProSave	lower layer of ProCom component model
ProCom component	ProSys subsystem or ProSave component
ProSys subsystem	ProSys composite subsystem or ProSys primitive subsystem
ProSys composite subsystem	ProSys subsystem whose internal structure is built using ProSys subsystems
ProSys primitive subsystem	ProSys subsystem whose internal structure is built using ProSave components or realized by code
ProSave component	ProSave composite component or ProSave primitive component
ProSave composite component	ProSave component whose internal structure is built using ProSave components
ProSave primitive component	ProSave component realized by code
black-box	component whose internal structure is at this point undecided (either on the ProSys or ProSave level)

Figure 1 shows the hierarchical relation between terms defined in Table 1.

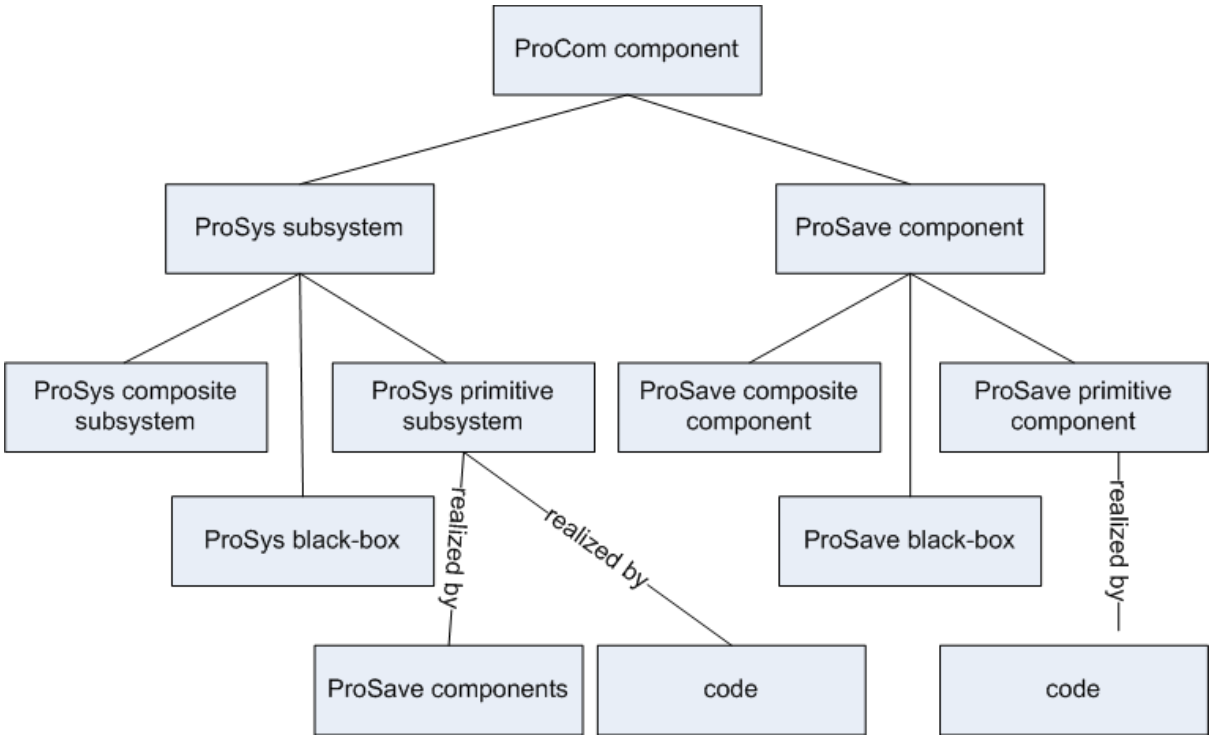

Figure 1: ProCom type and realization definitions

1.2 *Acronyms and abbreviations*

Table 2: Acronyms and abbreviations used

Acronym or abbreviation	Definitions
EMF	Eclipse Modeling Framework
GEF	Graphical Editing Framework
ES	Embedded Software
IDE	Integrated Development Environment
OCL	Object Constraint Language
REMES	Resource Model for Embedded Systems

1.3 *Relationship between PROGRESS and DICES*

PROGRESS aims to be a world-leading centre known for applied research in development of predictable component-based Embedded Software (ES), with emphasis on ES in the vehicular, automation and telecom domains, specifically focusing on:

- ES component technology,
- reuse of legacy ES, and

- verification and analysis for predictability [4].

DICES has a goal to advance development of distributed embedded software systems with emphasis on software reusability and predictability of software quality [3].

It is apparent that PROGRESS and DICES overlap. Thus, the DICES researchers (Juraj Feljan, Luka Lednicki and Ana Petričić) are involved also in PROGRESS, as this cooperation is believed to be beneficial for both projects, contributing to dissemination of knowledge.

Research results from PROGRESS are to be built in PROGRESS-IDE, which would provide engineering support for efficient development of predictable embedded software. The DICES researchers are involved in development of ProCom. Apart from contributing in the theoretical research, they are also taking part in implementing the design phase part of PROGRESS-IDE. The IDE will be used in later stages of their PhD studies. Also, their future research results will be integrated into the IDE. In conclusion, DICES has a direct interest in and benefit from PROGRESS-IDE.

2 PROGRESS-IDE

In this section first we give overall considerations of PROGRESS-IDE. Then we discuss the technical aspects of how the environment is being implemented. The section is finished with an overview of the current state of the environment.

2.1 *Overview of PROGRESS-IDE*

The grand vision of PROGRESS-IDE is to cover the whole software development cycle of distributed embedded systems for automotive, automation and telecom domains, from early system design to deployment and synthesis, all this according to the PROGRESS approach [4]. The development process of such a system requires a strong emphasis on analysis, verification and validation in order to ensure the necessary quality of the delivered product, therefore PROGRESS-IDE will also integrate various analysis tools. Compared to the majority of existing IDEs that focus mainly on the programming aspect, PROGRESS-IDE uses the notion of component as a first-class concept allowing the manipulation and modeling of components, and supporting:

- component- and system design,
- system analysis,
- model transformations, and
- system verification.

In the following list we give some overall reflections that apply to PROGRESS-IDE.

- PROGRESS-IDE will cover the whole system development process.
- PROGRESS-IDE will be developed as a stand-alone application on top of Eclipse RCP.
- PROGRESS-IDE will consist of a number of views:
  - system design,
  - component design,
  - analysis,
  - verification,
  - target platform specification,
  - deployment,
  - synthesis (code generation).
- A view will consist of a number of editors:
  - architectural editors,
  - attribute editors,
  - node descriptions (virtual/physical),

  - allocation editors,
  - code generation,
  - various analysis editors,
  - configuration editors.
- PROGRESS-IDE will be user-friendly by providing:
  - user interface respecting the usability features,
  - partial auto-completion features,
  - cheat sheets (integrated manuals that users can follow to learn how to use the environment),
  - default values.
- PROGRESS-IDE will be a real integrated framework where changes are propagated between various elements of editors, views, etc.
- PROGRESS-IDE will integrate already well-tried features of classical development environments (coding, debugging, documentation...).

2.2 *Implementation of PROGRESS-IDE*

The Eclipse platform [5] is chosen as the supporting architecture for PROGRESS-IDE. However, in order to reduce the overhead which exists when using Eclipse directly as the main integration environment, it has been decided that the PROGRESS-IDE will be developed as a stand-alone application built on top of the Eclipse Rich Client Platform (Eclipse RCP) [6]. This decision is driven by the will to keep the control over the features present in the environment such as, for example, avoiding the presence of menus not directly related to the particular use of the PROGRESS-IDE. When using Eclipse RCP, only the features explicitly added to the environment are present. In other words, the layout and function of the environment are fully controlled by the plugin developers.

The system design part of the environment is to be implemented using EMF [7] and GMF [8]. EMF is a modeling framework and code generation facility for building tools based on a structured data model. It offers a graphical editor for describing metamodels. With the aid of this, the ProCom metamodel (model of the ProCom model) was defined. From this graphical description of the metamodel, EMF automatically generates Java classes representing the model, classes used for modifying the model and textual tree editors for the model. GEF then takes this existing application model (i.e. the model generated using EMF) and quickly creates a rich graphical editor. However, this automatically generated editor has rudimentary functionality and needs to be further manually modified and tweaked in order to achieve the desired functionality. Here OCL [9] will be used to define constraints on graphical modeling of systems, to ensure valid models. For instance, OCL will be employed to prevent connecting ports of different types or adding ports onto other ports and similar actions that are defined illegal by ProCom syntax and semantics.

2.3 *Current state of PROGRESS-IDE*

Progress-IDE is being developed by the three DICES researchers and by a number of people with relatively low commitment, who can devote to the development only a fixed period of time. These are mostly master students at MdH who spend typically from 2 months up to 6 months in the project. Thus the project is divided into relatively small iterations with small but concrete goals and a special effort must be put on documenting the work (both in source code and report) and providing appropriate guidelines on how to perform the work or modify the code.

Tomáš Bureš, Jan Carlson, Ivica Crnković and Séverine Sentilles from MdH are responsible for ProCom, overall architectural design of the environment and for supervising master students and the DICES researchers doing the implementation.

Currently, the ProCom metamodel is defined, however it is open for modifications and improvements. Four parts of the environment have been implemented by master students. The first three portions of work are directly related to the design phase of the system life-cycle, while the last part handles the deployment phase.

The first part is a PROGRESS-IDE skeleton done by Aleksandar M. Petkov[1]. This skeleton represents the basis on which all other parts of the environment will be integrated upon. It handles project manipulation, i.e. management of files which make up the project. This skeleton covers the Project explorer (see Section 3.2).

The second portion of work is done by Rumen Kyusakov [10]. He investigated and implemented a way to organize components in the repository. The repository is a key prerequisite for reuse in component systems. It allows storing a component and retrieving it into a different project. It also manages component evolution by providing component versioning. This part of the work matches the Repository explorer (see Section 3.1).

In the third segment of work Petr Štěpán deals with attribute handling [11]. Throughout the process of modeling, ProCom entities (components, ports, etc.) can be assigned attributes. These pieces of information are to be used by various computations over the model in different phases of the development process. Štěpán investigated and implemented an attribute framework. His work covers the Attributes view (see Section 3.4.1).

The fourth part handles component allocation and deployment modeling [12]. Component allocation means determining the device where the component will be executed. Allocation is then used to deploy the system, i.e. to create standalone software units, optimize them for the appropriate hardware devices and prepare them to run. To manage deployment and allocation, an IDE must be able to model system architecture and its hardware requirements. Deployment modeling for PROGRESSIDE has been investigated and implemented by David Senkerik. It is done by defining a virtual nodes architecture as an abstraction of target platform devices where components are allocated.

The DICES researchers are to implement the design phase part of PROGRESS-IDE, consisting of architectural editors supporting system modeling using ProCom. This design phase part is in the focus of this report, and is described in detail in the following section. The implementation is currently under way.

1 Developed as part of an ongoing master thesis an MdH, titled “Managing Components in PROGRESS-IDE”.

3 System design in PROGRESS-IDE

The part of PROGRESS-IDE used for system design (i.e. system modeling using ProCom) will consist of the following elements:

- Repository explorer,
- Project explorer,
- System view,
- Properties view
  - Attributes view,
- Component editor
  - Externals editor
  - Structure editor
    - Palette
  - Code editor
    - Binding table.

The listed elements are described in the following sections.

3.1 *Repository explorer*

As component-based development promotes building software systems out of preexisting components, component repositories storing pre-developed ProCom components will be available. In order to browse through available components and to import them in the current project, the Repository explorer will be used. The first version of the Repository explorer is implemented by Rumen Kyusakov [10].

3.2 *Project explorer*

The Project explorer is intended to display all relevant information about the current project and it is a reflection of the file system structure of the project. Besides project meta-data, documentation, testing and analysis information, the Project explorer will contain a listing of all ProCom components available in the project, in a flat and non-hierarchical manner.

The proposed Project explorer structure is as follows:

- ProjectInfo file – contains meta data about the project, including the definition of a "top node".
- SystemModel folder – contains information related to the model of a system. The internal structure and contents are not decided yet.
- ProSys folder – contains all ProSys subsystems existing in the project. It has one subfolder

for each ProSys subsystem.

- ProSave folder – contains all ProSave components existing in the project. It has one subfolder for each ProSave component.
- Doc folder – contains documentation and requirements definition.

Other possible elements of the Project explorer are:

- Analysis folder – consists of subfolders for the different analysis tools, for input, output, temporary storage, etc.
- Test folder
- Deployment folder – contains models of the virtual platform, allocation of ProCom components to virtual nodes, model of the physical platform, allocation of virtual nodes to physical, etc.
- Synthesis folder – contains artifacts from the synthesis process for example task models, relations between tasks and components, generated glue code, makefiles, and the final output.

Each ProCom component will be stored in a separate folder with the structure as shown below. The folder structure is the same in the project and the repository.

- ComponentInfo file – contains component meta-data.
- Models folder
  - Architectural model
    - Conceptual model
    - Graphical model
  - REMES [13]
    - Conceptual model
    - Graphical model
- Analysis folder – contains subfolders for the different types of analysis that can be performed on the component in isolation (not on a concrete component instance in the system). Most of analysis will probably be filed under the system instead.
- Doc folder
- Src folder – holds source code, in the case of a primitive ProSys subsystem/ProSave component.
- Bin folder – holds binaries (executable code), in the case of a primitive ProSys subsystem/ProSave component.

The first version of the Project explorer is implemented by Aleksandar M. Petkov.

3.3 *System view*

The System view will display the hierarchical view of the system in the form of a tree view. This view will be in the same tab group as the Project explorer.

In the first version of PROGRESSIDE it will be possible to have only one system model in the project, therefore, the root node presented in this view is the "top node" of this system model. The user will be able to see all levels of hierarchy and it will be possible to collapse or expand some parts of the view. If the root node is not defined, then the System view will be empty.

3.4 *Properties view*

The Properties view will be used to view and modify various properties of ProCom entities, such as names of components, types of ports and so on. The Properties view will much resemble the standard properties view in Eclipse.

3.4.1 Attributes view

The Attributes view is implemented as a tab in the Properties view. It shows all attributes for the currently selected/edited ProCom entity and clicking one of the values will open the associated editor. The first version of the Attributes view is implemented by Petr Štěpán [11].

3.5 *Component editor*

The Component editor will allow development of ProCom components, both their internals and externals. It will consists of multiple editors available by tabs. Some of these editors are only available for specific types of ProCom components. Table 3 shows available editors depending on the ProCom component type and realization.

Table 3: Editors available for specific types of ProCom components

	Realization by	Externals editor	Structure editor	Code editor
ProSys	ProSys subsystems	☑	☑	☒
	ProSave components	☑	☑	☒
	Code	☑	☒	☑
	Black-box	☑	☒	☒
ProSave	ProSave components	☑	☑	☒
	Code	☑	☒	☑
	Black-box	☑	☒	☒

3.5.1 Externals editor

Externals of ProCom components will be editable in a form-like editor, called the Externals editor.

For ProSys subsystems the External editor will consist of:

- name of the component,
- type of the component,
- type of realization (ProSys, ProSave, code, black-box),
- state of the component – locked/unlocked, singleton/multi-instance,
- list of input message ports (name, type),
- list of output message ports (name, type).

For ProSave components the External editor consists of:

- name of the component,
- type of the component,
- type of realization (ProSave, code, black-box),
- state of the component – locked/unlocked, singleton/multi-instance,
- list of input port groups (name (it is automatically generated from the service name))
- list of input ports (name (in case of trigger port it is automatically generated from the group name), type),
- list of output port groups (name)
- list of output ports (name (in case of trigger port it is automatically generated from the group name), type),
- list of services (name).

3.5.2 Structure editor

The Structure editor will be used for modeling the internal structure of ProCom components that are realized with other components. It will show the graphical representation of the edited ProCom component and the elements it consists of. Just the first level of the hierarchy will be shown. Additionally the user will be able to add or remove external ports of the edited component from this editor.

The user will be able to add a new ProCom component to the currently edited component in different ways:

- by using *drag&drop* from the Project explorer,
- by choosing from the Palette (a set of ProCom components and other modeling elements that are available in the Palette depends on the type and realization of the component which is edited). In this case a "wizard" will appear, prompting the user to either choose one of the

existing components in the project, or helping him to create a new component.

The Palette will also provide the means to add new connectors, connections and clocks. Editing of the subcomponents will be discussed in a separate section.

Table 4 provides the listing of all ProCom components' parts that will be visible in the Structure editor and modeling elements that will be available in the Palette, depending on the ProCom component type and realization.

Table 4: Structure editor for specific types of ProCom components

	Realization	Structure editor	Elements available in the Palette
ProSys	ProSys subsystems	- borders of the subsystem - input and output message ports - internal ProSys subsystems (with their ports) - internal message channels - connections between internal ProSys subsystem ports and internal message channels, and message ports of the outer component	- message ports (input, output) - ProSys subsystem - message channel - connection
	ProSave components	- borders of the subsystem - input and output message ports - internal ProSave components (with their services, groups and ports) - internal connectors - connections between internal ProSave component ports and internal connectors, and message ports of the outer component	- message port (input, output) - ProSave component (composite, black box, primitive) - connection - connector - clock - output group - ProSave data port (input, output) - service

ProSave	ProSave components	- borders of the composite component - input and output groups - input and output ports - internal ProSave components (with their services, groups and ports) - internal connectors - connections between internal ProSave components' ports and internal connectors and message ports of the outer component	- ProSave component (composite, black box, primitive) - connection - connector - output group - ProSave data port (input, output) - service

3.5.3 Code editor

The Code editor will be used for defining internals of the components that are realized by code. It will be a textual editor. Additionally it will have a binding table which is intended for binding component ports with function parameters. It will provide the user with a list of all component's ports so he/she is able to associate each port with a function parameter.

3.6 *Editing components*

ProCom components that have been imported to the project from a repository or exported to the repository are by default in read-only mode. We call these *locked components*. In order to modify a locked component, the user needs to explicitly unlock it. At this point it becomes an *unlocked component.* It is always the component that is unlocked, not a particular instance. Thus, by editing the component, all instances are affected. When exporting a modified unlocked component back to the repository, the user needs to decide whether this is a new version of the original component or a new component.
When editing the internal structure of a component in the Structure editor, its subcomponents are uneditable, it is not possible to add/remove/edit the subcomponents' ports from this view[2]. However, as syntactic sugar, we allow editing of subcomponents if they are *singletons*. A singleton is a component with only one instance currently present in the project. In this manner we call a component with several instances a *multi-instance*. For a locked component it is not important whether it is a singleton or a multi-instance, since it is uneditable.
All this is depicted in Figure 2.

2 In order to do so, the user needs to open the particular subcomponent he/she wishes to modify, in its own editor.

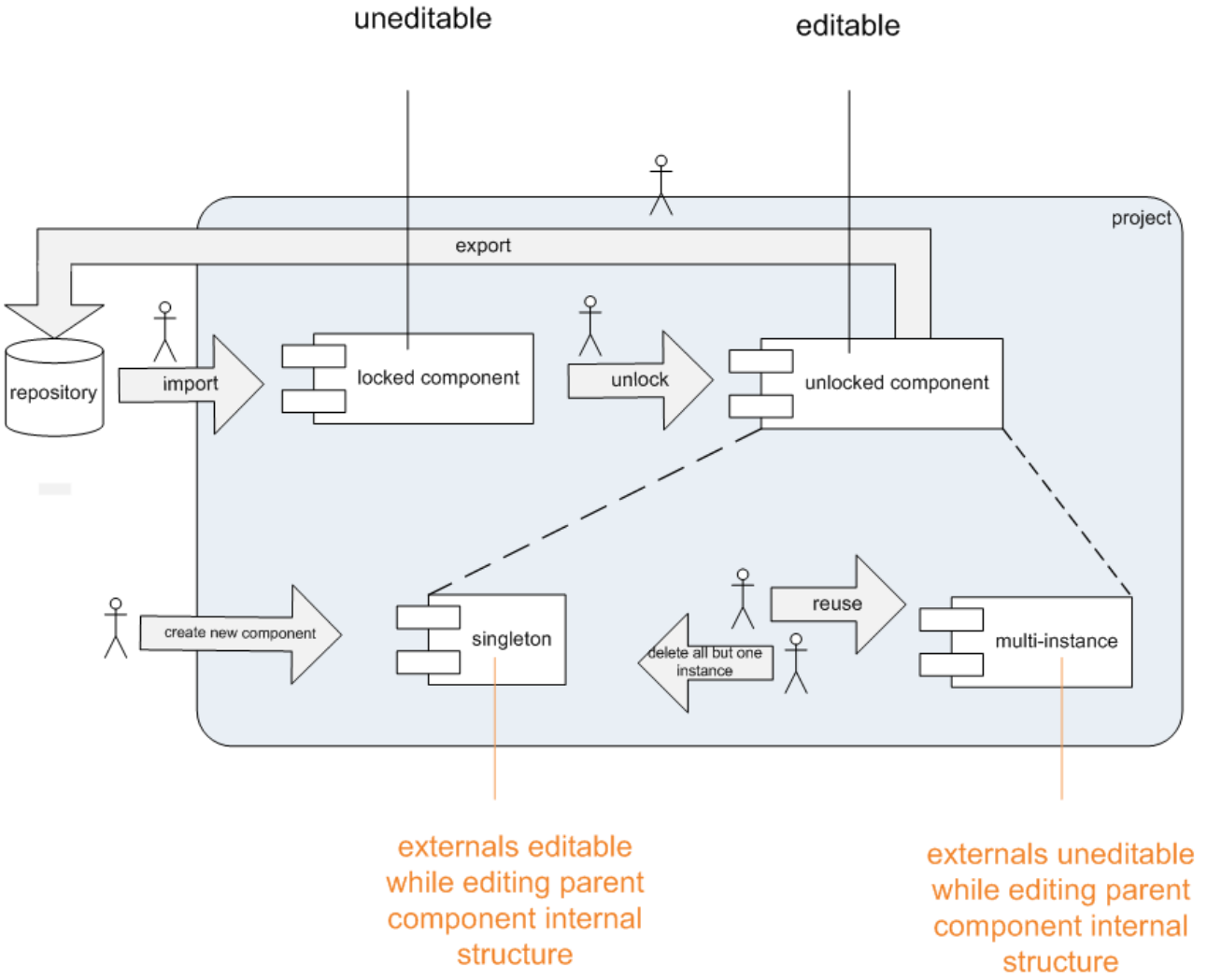

Figure 2: Locking and instance concepts

In order to distinguish between locked and unlocked components, all components will have a small icon which shows if they are locked. This icon will be visible in the Project explorer and in the Structure editor. Each component will be marked with an icon which shows if it is a singleton. The icon will be visible in the Project explorer and in the Structure editor. This way the user can immediately see what he can and cannot edit from the current view.

3.6.1 Handling services, port groups and ports in ProSave

The relation between input groups and services is one-to-one, so there is no need to add or remove the input group of a service. Similarly, there is a one-to-one mapping between groups and trigger ports. Thus, the user can specify a name of a service, and then for each input group and its trigger port, the names are generated automatically from the service name. The same applies to output groups and their trigger ports – the trigger port name is automatically generated from the group name. For example for the "Control loop" service, we get the "Control loop input group", "Control loop input trigger", and for the "Speed" output group we get the "Speed output trigger" etc.

Further more, the things that are one-to-one mapped are not handled as two separate concepts in the editor. So, it is not possible to delete an input group (instead, the user has to delete the service) or a trigger port (instead, the user has to delete the group). This also means that they are created together:

- When a new component is created, is has one service (with a default name which can be edited).
- When a new service is created (including the default service of a new component), the input group is created.
- When a new group is created (including the default group of the default service), the trigger port is created.

This also means that, the user is able to:

- add/delete a service to/from the component,
- add/delete an output group to/from a service,
- add/delete a data port to/from a group,
- edit all ports, port groups or services.

3.7 *Future development and open questions*

3.7.1 Project explorer

Working with non-hierarchical listing of components might become impractical if there is a large number of components in the project. For example, after importing a composite component that has a very complicated hierarchy could lead to a large number of components shown in the Project explorer (as all the subcomponents will be shown in the same level as the top-level component). In order to avoid this problem, the Project explorer should be extended with a way to hide some components or group them (similar to Java packages, but just for visual purposes). Another solution would be filtering of listed components according to some criteria or implementing a search functionality.
Another improvement considering the Project explorer, that might be considered in future development of ProgressIDE, is the support for having more than one workspace (currently there can be only one workspace).

3.7.2 System view

At the moment, the System view is intended to show the tree structure of the system model, with the root component of the system set as the top node of the view. If there is a component in the project which is not part of the system, it will not be possible to see it in the system view. For example if a user imports some component from the repository the IDE will not provide him with the tree view of this component, and he will not be able to see the structure of the imported component..

In future versions of the tool, the System view should be extended to be able to dynamically change the top node which is presented, so the user will be able to import some component from the

repository and to see its hierarchical structure before using the component. Also, this way it will be possible to support having more than one system model in the project.

3.7.3 Editing components

For a future version of ProgressIDE we consider the feature of making a new component out of a particular instance. By that we would effectively allow editing a particular instance (or several instances, depending on the user's need).
Also, future developers could consider adding a possibility to drag&drop the component from the component repository directly into the Component editor (instead of importing the component into the Project explorer).

4 Requirements on system design in PROGRESS-IDE

In this section we give a detailed specification of requirements on the system design part of PROGRESS-IDE, by specifying use cases.

4.1 *Use case "Using PROGRESS-IDE for system design"*

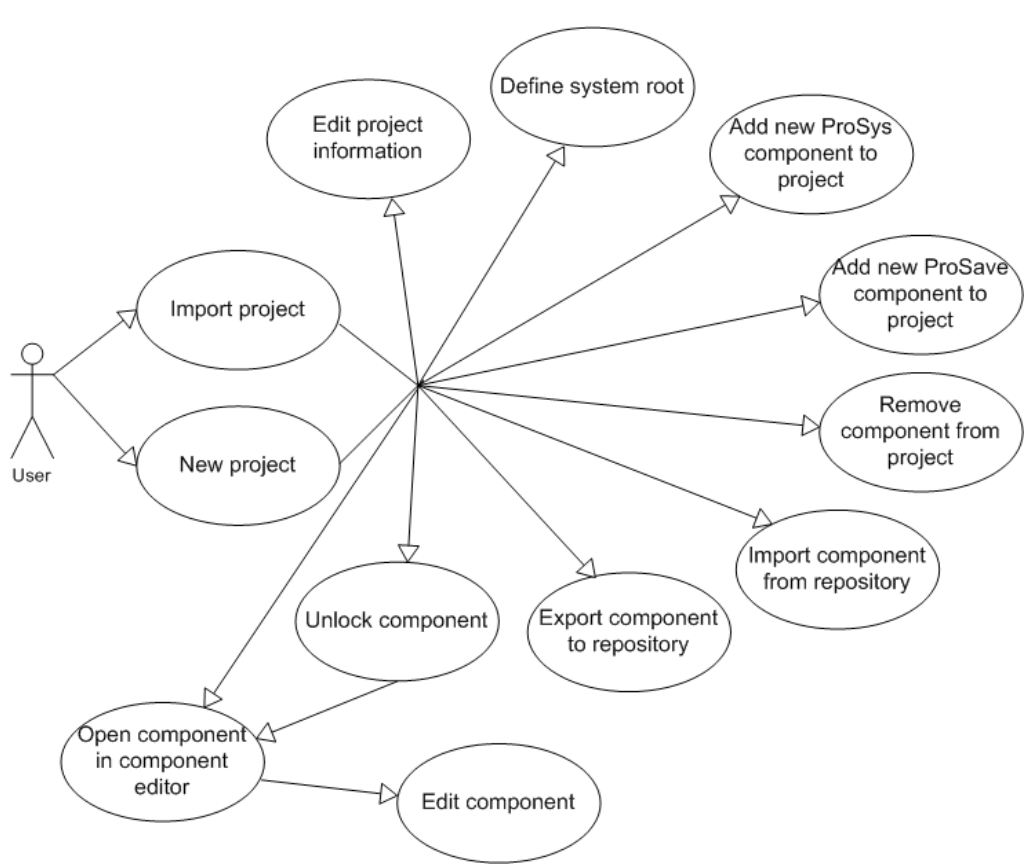

Figure 3: Using ProgressIDE for system design

4.1.1 Use case "New project"

Initiator: *PROGRESS-IDE user.*

Goal: *Add a new ProCom project to the project explorer.*

Main Scenario:

1. *From the "File" menu select "New" and then "ProCom project"*
   OR
   Right click on the "Project explorer" and select "New", then "ProCom

project".
The same functionality can be accessed if selecting "New" and "Project...". In that case the user needs to select "ProCom project" from the Eclipse's wizard.
2. *New project wizard opens.*
3. *The user defines the project name and chooses to either use a default location for the project or defines a custom location.*
4. *The user clicks on the "Finish" button.*
5. *A new project, with the default project structure is available in the project explorer.*

Extensions: The u*ser can abort the creation of the project at any time by clicking on the "Cancel" button.*

4.1.2 Use case “Import project”

Initiator: *PROGRESS-IDE user.*

Goal: *Import an existing project into the project explorer.*

Main Scenario:

1. *From the "File" menu select "Import..."*
   OR
   Right click on the "Project explorer" and select "Import...".
2. *Import project wizard opens.*
3. *Select "Existing project into workspace".*
4. *Select the root directory of the existing project or an archive file containing the project.*
5. *Projects found in the detected root directory or archive file appear in the project list of the wizard.*
6. *The user selects the project(s) to import and clicks the "Finish" button. A check-box makes it available for the user to copy all the project files into workspace.*
7. *Projects selected for importing appear in the "Project explorer"*

Extensions: *User can abort the creation of the project at any time by clicking on the "Cancel" button.*

4.1.3 Use case “Edit project information”

Initiator: *PROGRESS-IDE user.*

Goal: *Editing general project information (e.g. selecting the root ProSys system for the project).*

Main Scenario:

1. *Find the "ProjectInfo" file under the project's node in the "Project*

explorer".
2. *Double-click the "ProjectInfo" file*
   OR
   right-click on it and select "Open".
3. *Form editor for editing project information opens.*
4. *Make changes.*
5. *Save the changes by clicking on the "Save" icon in the tool-bar or by selecting "Save" form the "File" menu.*

Extensions: *User can use the "Undo" feature while the changes are not saved.*

4.1.4 Use case “Define system root”

Initiator: *PROGRESS-IDE user.*

Goal: *Selecting ProSys component that will be the root system for the project. The same goal can be achieved by editing project information.*

Main Scenario:

1. *Right-click on the ProSys component that you want to use as root system.*
2. *Select the "Use as top node".*
3. *A window asking for confirmation appears. The user has an option to skip this step if he checks the "don't ask me again" check-box.*
4. *The selected component is now the top node of the system model.*

4.1.5 Use case “Add new ProSys component to the project”

Initiator: *PROGRESS-IDE user.*

Goal: *Adding a new ProSys component to the project.*

Main Scenario:

1. *Right-click on the project node in the "Project explorer" and select "ProSys component" from the "New" menu*
   OR
   Select "ProSys component" from the "New" menu in the "File" menu.
2. *Select a ProCom project to which to add the component to. If the wizard was started by right-clicking in the "Project explorer" the project node that the action was performed is selected by default.*
3. *Select the name for the component. The wizard suggests a default name.*
4. *Select the realization type (ProSys subsystem, ProSave subsystem, code, black-box).*
5. *Fill in optional information about the component (e.g. provider).*
6. *Click on the "Finish" button..*
7. *The newly defined component is available under the "ProSys" node of the project.*

4.1.6 Use case “Add new ProSave component to the project”

Initiator: *PROGRESS-IDE user.*

Goal: *Adding a new ProSave component to the project.*

Main Scenario:

1. *Right-click on the project node in the "Project explorer" and select "ProSave component" from the "New" menu*
   OR
   Select "ProSave component" from the "New" menu in the "File" menu.
2. *Select a ProCom project to which to add the component to. If the wizard was started by right-clicking in the "Project explorer" the project node that the action was performed is selected by default.*
3. *Select the name for the component. The wizard suggests a default name.*
4. *Select the realization type (primitive, composite, black-box).*
5. *Fill in optional information about the component (e.g. provider).*
6. *Click on the "Finish" button..*
7. *The newly defined component is available under the "ProSave" node of the project.*

4.1.7 Use case “Remove component from the project”

Initiator: *PROGRESS-IDE user.*

Goal: *Removing an existing component from the project.*

Main Scenario:

1. *Right-click on the component in the "Project explorer"*
2. *Select "Delete" from the context menu.*
3. *A window asking for user confirmation appears.*
4. *If the component is used in a subsystem or a composite ProSave in the project, a warning window appears listing all the instances of the component and asking user for the confirmation of the delete action again.*
5. *If user confirmed the deletion of the component, the component is removed from the project. If it was part of a subsystem or composite ProSave in the project, all its instances are removed from that components.*

4.1.8 Use case “Import component from the repository”

Initiator: *PROGRESS-IDE user.*

Goal: *Importing an existing component from the component repository to the project.*

Main Scenario:

1. *Right-click on the component in the "Repository explorer".*
2. *Select "Import to project".*
3. *A dialogue with the list of available projects, and "Finish" and "Cancel" buttons appears.*
4. *Select the desired project to which the component is to be imported and click "Finish".*

4.1.9 Use case "Export component to the repository"

Initiator: *PROGRESS-IDE user.*

Goal: *Exporting a component in from the project to the component repository.*

Main Scenario:

1. *Right-click on the component in the "Project explorer".*
2. *Select "Export to repository".*
3. *A dialogue with the list of available repositories, and "Finish" and "Cancel" buttons appears.*
4. *Select the desired repository to which the component is to be exported and click "Finish".*

4.1.10 Use case "Unlock component"

Initiator: *PROGRESS-IDE user.*

Goal: *Unlock the component imported from the repository or exported to the repository and make it available for editing.*

Main Scenario:

1. *Right-click on the component in the "Project explorer".*
2. *Select "Unlock" from the context menu.*
3. *The component losses its version information.*
4. *The component is available for editing.*

4.1.11 Use case "Open component in component editor"

Initiator: *PROGRESS-IDE user.*

Goal: *Editing an existing component.*

Main Scenario:

1. *Right-click on the component in the "Project explorer" and select "Edit"*
   OR
   Double-click the component.
2. *A multi-page Component editor appears. Different pages of the editor are available depending on the realisation type of the component (e.g. primitive, composite...) and the component type itself (ProSys or*

ProSave).

4.2 *Use case "Edit component"*

The use case "Edit component" is further refined in this section, as it consists of a number of other use cases. These can be performed either in the Structure editor, Externals editor or both, as shown in the figure below.
Some use cases have several options how to proceed from a particular step and/or depend on the type of element we are dealing with. However, the idea here was to unify as many use cases as possible into one, rather showing the general scenario than particular details.

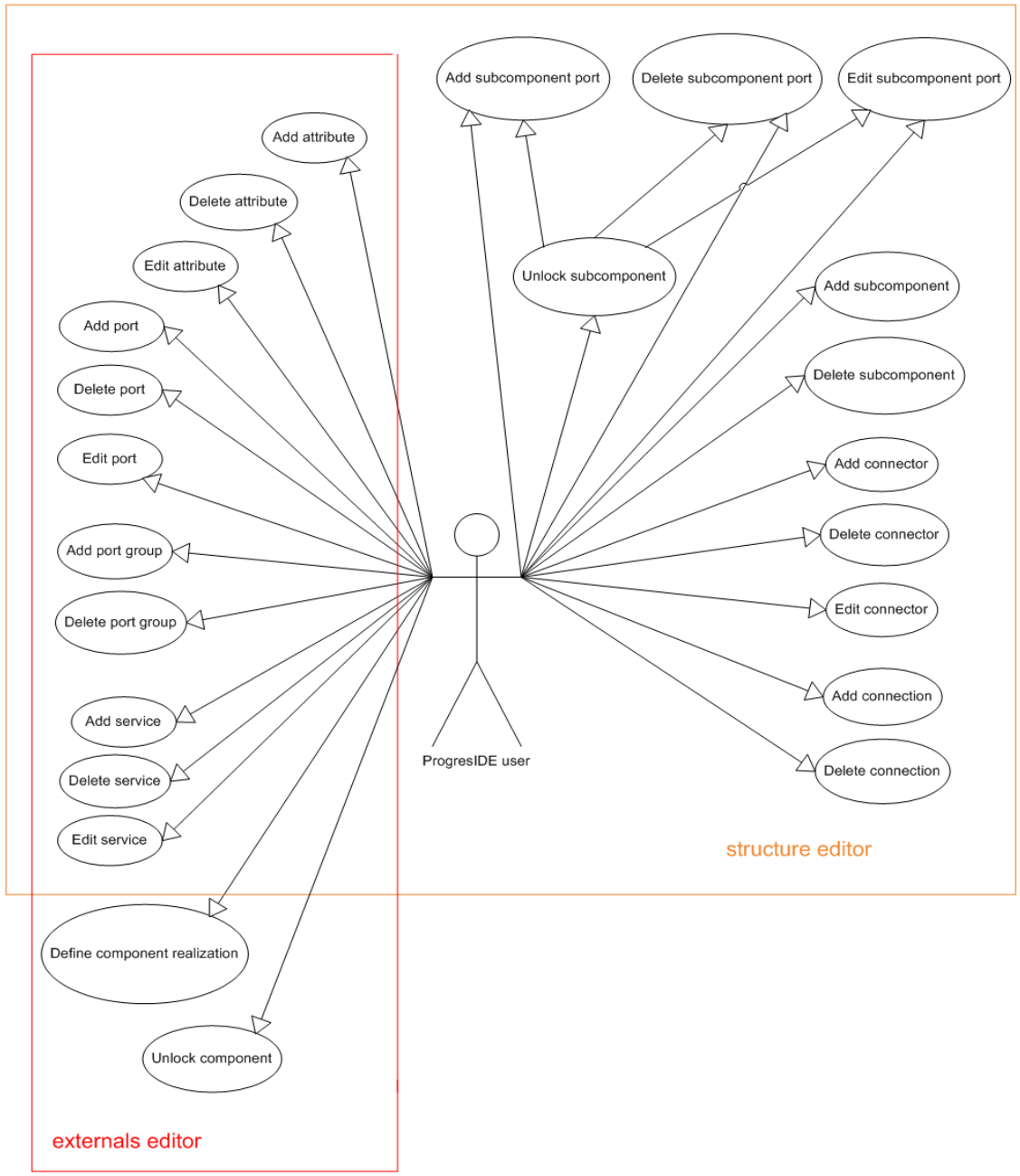

Figure 4: Edit component

4.2.1 Use case "Unlock subcomponent"

Initiator: *PROGRESS-IDE user*

Goal: *Unlocking a subcomponent of the enclosing component currently being*

edited.

Precondition: *The subcomponent is imported from the repository or exported to the repository, i.e. is not a newly defined component.*

Main Scenario:

1. *Right-click on the desired subcomponent in the Structure editor of the enclosing component and select "Unlock".*
2. *The component losses its version information.*
3. *The component is available for editing.*

4.2.2 Use case "Add subcomponent port"

Initiator: *PROGRESS-IDE user.*

Goal: *Adding a port to a subcomponent of the enclosing component currently being edited.*

Precondition: *If the subcomponent to which the port will be added is imported from the repository or or exported to the repository, then this subcomponent has to be unlocked.*

Main Scenario:

1. *If the subcomponent is not a singleton, then it has to be opened in its own component editor. See use case 3.1.11. Otherwise, follow the next step.*
2. *Right-click on the desired subcomponent (or on a port group in case of a ProSave component) in the Structure editor of the enclosing component and select "Add" and then select the desired port.*
   OR
   Select the desired port from the Palette and click on the subcomponent (or port group) to which the port needs to be added.
3. *The port is then added.*

4.2.3 Use case "Edit subcomponent port"

Initiator: *PROGRESS-IDE user.*

Goal: *Editing a port information of a subcomponent of the enclosing component currently being edited.*

Precondition: *If the subcomponent whose port will be edited is imported from the repository or exported to the repository, then this subcomponent has to be unlocked.*

Main Scenario:

1. *If the subcomponent is not a singleton, then it has to be opened in its own component editor. See use case 3.1.11. Otherwise, follow the next step.*

2. *Click on the desired port in the Structure editor.*
3. *Port properties appear in the Properties view and port attributes appear in the Attribute view.*
4. *Edit port information in the Properties view and in the Attribute view.*

4.2.4 Use case "Delete subcomponent port"

Initiator: *PROGRESS-IDE user.*

Goal: *Deleting a port of a subcomponent of the enclosing component currently being edited.*

Precondition: *If the subcomponent whose port will be deleted is imported from the repository or exported to the repository, then this subcomponent has to be unlocked.*

Main Scenario:

1. *If the subcomponent is not a singleton, then it has to be opened in its own component editor. See use case 3.1.11. Otherwise, follow the next step.*
2. *Right-click on the desired port in the Structure editor and select "Delete".*
   OR
   Click on the desired port in Structure editor and press the DEL keyboard button.
3. *A confirm dialogue appears, with the options to confirm or cancel the deleting.*
4. *If the user confirmed the delete action, the port is deleted. Otherwise no modification is carried out.*

Extensions: *The confirm dialogue from step 2 of the Main scenario has a check-box "Do not ask me again", which when checked stops showing the dialogue and immediately deletes the port.*

4.2.5 Use case "Add subcomponent"

Initiator: *PROGRESS-IDE user.*

Goal: *Adding a subcomponent to the enclosing component currently being edited.*

Main Scenario:

1. *Right-click on empty space in the Structure editor of the enclosing component and select "Add" and then select the desired subcomponent (a set of offered subcomponents depends on the type and realisation of the enclosing component).*
   OR
   Select the desired subcomponent from the Palette and click on empty space in the Structure editor of the enclosing component.
2. *The subcomponent is then added.*

4.2.6 Use case “Delete subcomponent”

Initiator: *PROGRESS-IDE user.*

Goal: *Deleting a subcomponent of the enclosing component currently being edited.*

Main Scenario:

1. *Right-click on the desired subcomponent in the Structure editor of the enclosing component and select "Delete".*
   OR
   Click on the desired subcomponent in the Structure editor and press the DEL keyboard button.
2. *A confirm dialogue pops up, asking the user to confirm or cancel the deleting. If the subcomponent to be deleted has several instances in the project, the confirm dialogue is expanded with a warning message and a list of all current instances.*
3. *If the user confirmed the delete action, the subcomponent with all its instances is deleted. Otherwise no modification is carried out.*

4.2.7 Use case “Add connector”

Initiator: *PROGRESS-IDE user.*

Goal: *Adding a connector to the enclosing component currently being edited.*

Precondition: *The edited component must be a ProSave component or a ProSys primitive subsystem realized by ProSave*

Main Scenario:

1. *Right-click on empty space in the Structure editor of the enclosing component and select "Add” and then select the desired connector.*
   OR
   Select the desired connector from the Palette and click on empty space is the Structure editor of the enclosing component.
2. *The connector is then added.*

4.2.8 Use case “Edit connector”

Initiator: *PROGRESS-IDE user.*

Goal: *Editing a connector of the enclosing component currently being edited.*

Precondition: *The edited component must be a ProSave component or a ProSys primitive subsystem realized by ProSave*

Main scenario:

1. *Click on the desired connector in the Structure editor.*

2. *Connector properties appear in the Properties view (a list of properties depends on the connector type).*
3. *Edit connector information in Properties view.*

4.2.9 Use case “Delete connector”

Initiator: *PROGRESS-IDE user.*

Goal: *Deleting a connector from the enclosing component currently being edited.*

Precondition: *The edited component must be a ProSave component or a ProSys primitive subsystem realized by ProSave*

Main scenario:

1. *Right-click on the desired connector in the Structure editor of the enclosing component and select "Delete".*
   OR
   Click on the desired connector in Structure editor and press the DEL keyboard button.
2. *A confirm dialogue appears, with the options to confirm or cancel the deleting.*
3. *If the user confirmed the delete action, the connector is deleted. Otherwise no modification is carried out.*

Extensions: *The confirm dialogue from step 2 of the Main scenario has a check-box “Do not ask me again”, which when checked stops showing the dialogue and immediately deletes the connector.*

4.2.10 Use case “Add connection”

Initiator: *PROGRESS-IDE user.*

Goal: *Adding a connection between two sub-elements of the enclosing component currently being edited.*

Main scenario:

1. *Right-click on the desired source element in the Structure editor of the enclosing component and select "Add” and then “Connection".*
   OR
   Select Connection from the Palette and click on the source element of the connection.
2. *Click on the destination element of the connection.*
3. *If the elements are compatible (for instance an input and output data port of the same type), the connection is established. Otherwise no connection is made.*

4.2.11 Use case “Delete connection”

Initiator: *PROGRESS-IDE user.*

Goal: *Deleting a connection between two sub-elements of the enclosing component currently being edited.*

Main Scenario:

1. *Right-click on the desired connection in the Structure editor of the enclosing component and select "Delete".*
   OR
   Click on the desired connection in the Structure editor or and press the DEL keyboard button.
2. *A confirm dialogue appears, with the options to confirm or cancel the deleting.*
3. *If the user confirmed the delete action, the connection is deleted. Otherwise no modification is carried out.*

Extensions: *The confirm dialogue from step 2 of the Main scenario has a check-box “Do not ask me again”, which when checked stops showing the dialogue and immediately deletes the connection.*

4.2.12 Use case “Add attribute”

Initiator: *PROGRESS-IDE user.*

Goal: *Adding an attribute to the component currently being edited.*

Main Scenario:

1. *If the Externals editor of the edited component is opened and in focus, component's attributes are listed in Attributes view.*
   OR
   If the Structure editor is opened and in focus click on empty space in the Structure editor of the enclosing component – the attributes appear in Attributes view.
2. *Add new component attribute in Attribute view (since this is already implemented by a master student, description of the process of adding attributes is out of the scope of this document).*

4.2.13 Use case “Edit attribute”

Initiator: *PROGRESS-IDE user.*

Goal: *Editing the attribute of the component currently being edited.*

Main Scenario:

1. *If the Externals editor of the edited component is opened and in focus,*

component's attributes are listed in Attributes view.
OR
If the Structure editor is opened and in focus click on empty space in the Structure editor of the enclosing component – the attributes appear in Attributes view

2. *Edit the desired component attribute in Attribute view (since this is already implemented by a master student, description of the process of editing attributes is out of the scope of this document)*

4.2.14 Use case "Delete attribute"

Initiator: *PROGRESS-IDE user.*

Goal: *Deleting the attribute of the component currently being edited.*

Main Scenario:

1. *If the Externals editor of the edited component is opened and in focus, component's attributes are listed in Attributes view.*
   OR
   If the Structure editor is opened and in focus click on empty space in the Structure editor of the enclosing component – the attributes appear in Attributes view
2. *Delete the desired component attribute in Attribute view (since this is already implemented by a master student, description of the process of deleting attributes is out of the scope of this document)*

4.2.15 Use case "Add port"

Initiator: *PROGRESS-IDE user.*

Goal: *Adding a port to the component currently being edited.*

Main Scenario:

A. From the Externals editor

1. *In the Externals editor click on the "Add port" button next to the list of all existing ports and then select the desired port from the offered list of ports.*
2. *The port is added to the list of component ports.*
3. *The user can edit the port information in the port list.*

B. From the Structure editor

1. *Right-click on empty space in the Structure editor of the enclosing component (or on a port group in case of a ProSave component) and select "Add" and then select the desired port.*
   OR
   Select the desired port from the Palette and click on empty space of the enclosing component in the Structure editor (or on a port group in case

of a ProSave component)
2. *The port is then added.*

4.2.16 Use case "Edit port"

Initiator: *PROGRESS-IDE user.*

Goal: *Editing a port of the component currently being edited.*

Main Scenario:

A. From the Externals editor
1. *Select the desired port by clicking it in the port list. Its properties are visible and editable in the Properties view.*
2. *Edit the desired property.*

B. From the Structure editor
1. *Select the desired port by clicking its image. Its properties are visible and editable in the Properties view.*
2. *Edit the desired property.*

4.2.17 Use case "Delete port"

Initiator: *PROGRESS-IDE user.*

Goal: *Deleting a port of the enclosing component currently being edited.*

Main Scenario:

A. From the Externals editor
1. *Select the desired port by clicking it in the port list.*
2. *Click on the "Delete" button next to the port list.*
   OR
   Press the DEL keyboard button.
4. *A confirm dialogue appears, with the options to confirm or cancel the deleting.*
5. *If the user confirmed the delete action, the port is deleted. Otherwise no modification is carried out.*

B. From the Structure editor
1. *Select the desired port by clicking its image.*
2. *Press the DEL keyboard button.*
   OR
3. *Right click on the port image.*
4. *Select "Delete" from the context menu.*

3. *A confirm dialogue appears, with the options to confirm or cancel the deleting.*
4. *If the user confirmed the delete action, the port is deleted. Otherwise no*

modification is carried out.

Extensions: *The confirm dialogue from steps A.3 and B.3 of the Main scenario has a check-box "Do not ask me again", which when checked stops showing the dialogue and immediately deletes the port.*

4.2.18 Use case "Add port group"

Initiator: *PROGRESS-IDE user.*

Goal: *Adding a port group to the enclosing ProSave component currently being edited.*

Precondition: *The edited component must be a ProSave component or a ProSys primitive subsystem realized by ProSave.*

Main Scenario:

A. From the Externals editor

1. *Click on the "Add port group" button next to the list of all existing port groups.*
2. *The port group is added to the component.*

B. From the Structure editor

1. *Click on the empty space in the Structure editor of the enclosing component.*
2. *Select the Port group from the Palette.*

4.2.19 Use case "Delete port group"

Initiator: *PROGRESS-IDE user.*

Goal: *Deleting a port group of the enclosing component currently being edited.*

Precondition: *The edited component must be a ProSave component or a ProSys primitive subsystem realized by ProSave.*

Main Scenario:

A. From the Externals editor

1. *Select the desired port group by clicking it in the port group list.*
2. *Click on the "Delete" button next to the port group list.*
   OR
   Press the DEL keyboard button.
3. *A confirm dialogue appears, with the options to confirm or cancel the deleting.*
4. *If the user confirmed the delete action, the port group is deleted. Otherwise no modification is carried out.*

B. From the Structure editor

1. *Select the desired port group by clicking its image.*
2. *Press the DEL keyboard button.*
   OR
3. *Right click on the port group image.*
4. *Select "Delete" from the context menu.*
5. *A confirm dialogue appears, with the options to confirm or cancel the deleting.*
6. *If the user confirmed the delete action, the port group is deleted. Otherwise no modification is carried out.*

Extensions: The confirm dialogue from steps A.3 and B.3 of the Main scenario has a check-box "Do not ask me again", which when checked stops showing the dialogue and immediately deletes the port group.

4.2.20 Use case "Add service"

Initiator: *PROGRESS-IDE user.*

Goal: *Adding a new service to the component currently being edited*

Precondition: *The edited component must be a ProSave component or a ProSys primitive subsystem realized by ProSave.*

Main Scenario:

A. Using Externals editor

1. *In Externals editor click on the "Add service" button next to the list of all services*
2. *The new service is added to the list.*

B. Using Structure editor

1. *Right-click on empty space in the Structure editor of the enclosing component and select "Add service".*
   OR
   Select Service from the Palette and click on empty space in the Structure editor of the enclosing component.
2. *The service is then added.*

4.2.21 Use case "Edit service"

Initiator: *PROGRESS-IDE user.*

Goal: *Editing a service of the component currently being edited.*

Precondition: *The edited component must be a ProSave component or a ProSys primitive subsystem realized by ProSave.*

4.2.22 Use case “Delete service”

Initiator: *PROGRESS-IDE user.*

Goal: *Deleting a service from the component currently being edited.*

Precondition: *The edited component must be a ProSave component or a ProSys primitive subsystem realized by ProSave.*

Main Scenario:

A. Using Externals editor

1. *In Externals editor click on the desired service*
2. *Click on the “Delete service” button next to the list of all services or press the DEL keyboard button.*
3. *The service is deleted and it is removed from the list*

B. Using Structure editor

1. *Right-click on empty space within the desired service in the Structure editor of the enclosing component and select "Delete service”.*
2. *The service with associated port groups and all internal sub-components are then deleted.*

4.2.23 Use case “Define component realization”

Initiator: *PROGRESS-IDE user.*

Goal: *Defining how the component functionality will be realized (ProSys, ProSave, code, black-box)*

Main Scenario:

1. *In Externals editor select the component realization type from the drop-down list (the set of offered realization types depends on the type of the edited component)*
2. *Click on the “Define realization” button next to the drop down list*
3. *If the component already has a realization specified, a confirm dialogue appears, with the options to confirm or cancel the deleting of existing component realization.*
4. *If the user confirmed the delete action, existing realization is deleted and appropriate editors for new realization type appear. Otherwise no modification is carried out.*

4.2.24 Use case “Unlock component”

Initiator: *PROGRESS-IDE user.*

Goal: *Unlocking the edited component*

Precondition: *The component is imported from the repository or exported to the repository, i.e. is not a newly defined component.*

Main Scenario:

1. *In externals editor find the information about lock status of the component*
2. *Click on the "Unlock" button next to the status*
3. *The component losses its version information*
4. *The component is available for editing*

References

[1] Mälardalen University, The Progress project, http://www.mrtc.mdh.se/progress/

[2] Tomáš Bureš, Jan Carlson, Ivica Crnković, Séverine Sentilles, Aneta Vulgarakis, ProCom - the Progress Component Model Reference Manual, version 1.0, MRTC report, Mälardalen University, 2008

[3] University of Zagreb/Mälardalen University, DICES, http://www.fer.hr/dices/

[4] Hans Hansson, Ivica Crnkovic, Thomas Nolte, The World according to Progress

[5] Eclipse, http://www.eclipse.org/

[6] Eclipse RCP, http://wiki.eclipse.org/index.php/Rich_Client_Platform/

[7] Eclipse Modeling Framework, http://www.eclipse.org/modeling/emf/

[8] Eclipse Graphical Editing Framework, http://www.eclipse.org/gef/

[9] Object Management Group, Object Constraint Language, http://www.omg.org/technology/documents/modeling_spec_catalog.htm#OCL

[10] Rumen Kyusakov, Model-based and component-based development of embedded systems, Master's thesis, Mälardalen University, 2008

[11] Petr Štěpán, An extensible attribute framework for ProCom, Master's thesis, Mälardalen University, 2009

[12] David Senkerik, Modeling deployment and allocation in the Progress IDE, Master's thesis, Mälardalen University, 2009

[13] Cristina Seceleanu, Aneta Vulgarakis, Paul Pettersson, REMES: A Resource Model for Embedded Systems, MRTC report, Mälardalen University, 2008