<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Importing external packages/referencing other packages</td>
<td>18</td>
</tr>
<tr>
<td>Working with Sequence Diagrams</td>
<td>18</td>
</tr>
<tr>
<td>Using SMSMediators</td>
<td>19</td>
</tr>
<tr>
<td>Mediator constructor arguments (SMSInput/Output)</td>
<td>19</td>
</tr>
<tr>
<td>SMS Mediators</td>
<td>19</td>
</tr>
<tr>
<td>Transforming with JFT</td>
<td>20</td>
</tr>
<tr>
<td>Executing a JavaFrame Application with JFDebug</td>
<td>21</td>
</tr>
<tr>
<td>Executing a JavaFrame Application with JFTrace</td>
<td>21</td>
</tr>
<tr>
<td>Running the application</td>
<td>22</td>
</tr>
<tr>
<td>Running Google Earth</td>
<td>22</td>
</tr>
<tr>
<td>Notes on SMS syntax/semantics</td>
<td>23</td>
</tr>
<tr>
<td>Known Bugs</td>
<td>24</td>
</tr>
<tr>
<td>Papyrus/All diagram editors</td>
<td>24</td>
</tr>
<tr>
<td>Sequence Diagram Editor</td>
<td>24</td>
</tr>
<tr>
<td>State Machine Diagram Editor</td>
<td>25</td>
</tr>
<tr>
<td>Tips</td>
<td>26</td>
</tr>
<tr>
<td>Working on a Project (Team-work)</td>
<td>27</td>
</tr>
<tr>
<td>Sharing a Project using Subclipse</td>
<td>28</td>
</tr>
<tr>
<td>Importing a project from your repository</td>
<td>29</td>
</tr>
</tbody>
</table>
Changes in 1102 from previous version 1101
Changes in this document from the previous version (1101) are:

- Added sections to Working with JavaFrame
- Added new bugs to bug list: Known Bugs
- Added information about executing with JFDebug instead of JFTrace: Executing a JavaFrame Application with JFDebug
Installing Papyrus IF UML and Setting up a JavaFrame Project

Installation
Download and unpack PapyrusIFUML.zip

Creating a project
1. In your new blank workspace create a new JavaFrame Project: File -> New -> Other -> JavaFrame -> JavaFrame Project

![Create a JavaFrame Project](image)

2. Define a project name and click Finish. You may be asked if you would like to switch to the Java-perspective, you may choose to or not.

3. The first time you create a JavaFrame Project a companion project is generated that contains the essential resources that are needed in the course (JavaJars project). Some of these jars are referenced to be your newly created project and are needed to build the project.

A JavaFrame Project is similar to a regular Java project with the following modifications:

- Sourcepath is set to generated.
- Output classpath is set to bin.
- Essential Java-libraries needed to use JavaFrame and positioning are referenced in the classpath.
• 1 profile and 2 profiles are added to the project itself. *JavaFrame.profile.uml*, *JavaFrame.library.uml*, *SMSPorts.library.uml*. The first is needed for various stereotypes that are needed to create JavaFrame transformable UML-models. The two latter are libraries that your project may reference.

4. A new, empty JavaFrame-project looks like this in the Java-perspective:
**The Resource Jars**

**FakePats**
FakePats is an emulator for PATS such that we are not dependent upon the real PATS laboratory. FakePats was made by Frank Davidsen and includes also features that were used when we collaborated with Trafikanten in 2005. This part of FakePats is not significant in the ICU context, but the world seen by FakePats is limited to the area covered by Bus 37.

**PATSClient**
The real PATS client that connects to the PATS laboratory. Does not function in its current form, per 01.09.2008.

**JFTrace**
A stand-alone tracing program that can trace JavaFrame executions (a new improved debugging tool is under development and should be available shortly).

**JFDebug**
An advanced JavaFrame debugger developed by Jonas Winje.

**SMSMediators**
Defines the SMSPorts and connects to the PATs service.

**JavaFrame**
Runtime library needed to compile JavaFrame projects.
Creating a Papyrus-model

1. Open the Papyrus-perspective (Window -> Open perspective -> Other -> Papyrus) or find the button and click Other in its menu.

2. Click File -> New -> Create an empty model or right-click your project folder New -> Create an empty model

3. Name your project and click Finish
4. 2 files are generated:
   a. A **DI2** file: This file contains the graphical aspect of your diagrams.
   b. A **UML** file. This file contains the model aspect of your UML-model.

5. The **Outline** view is where a lot of your work will be done. It displays a Tree-structure of your Papyrus-model.
Configuring your project for JavaFrame

A Papyrus-model needs to be configured for us to use JavaFrame, PATS and Java specific terms, which ultimately allows us to transform the UML-model into a runnable JavaFrame-based system and send/receive SMS-messages.

This is done in 4 steps.

Import the JavaFrame-profile into the model

This allows your model to access JavaFrame-specific stereotypes.

1. Choose your model in the Outline-view and click Profile in the Properties-view and click the +-sign next to Applied profiles.

2. Choose JavaFrame.profile.uml that resides in your project folder and click OK.

3. A dialog-box will open. Mark the check-box JavaFrameProfile and click OK. Under Applied Profiles JavaFrameProfile should now be visible.
Import the SMSPorts-library into the model

1. Right-click your Papyrus-model in the Outline-view and choose Import Package -> Import Package from workspace models

2. Under Select a resource to open write SMS and select the file SMSPorts.library.uml and click OK.
3. A dialog-box will open. Mark the check-box **smsmediators** and click **OK**.
Import the JavaFrame-library into your model

1. Right-click your Papyrus-model in the Outline-view and choose Import Package -> Import Package from workspace models

2. Under Select a resource to open write JavaFrame and select the file JavaFrame.library.uml and click OK.

3. A dialog-box will open. Mark the check-box se.ericsson.eto.norarc.javaframe and click OK.
Import the JavaPrimitiveTypes-library into your model
This library gives you a wider range of primitives than UML-supports.

1. Right-click your Papyrus-model in the Outline-view and choose Import Package -> Import library from repository

2. In the dialog window that opens select JavaPrimitiveTypes and click OK.

3. Mark the check-box JavaPrimitiveTypes and click OK.
Confirming that everything has been imported successfully
If all the above has been imported successfully an expanded view of both Imports and Applied Profiles should look like the below in the Outline view.
Working with JavaFrame

Section is not complete. More advanced JavaFrame specific tips will be added later.

Composites
Creating composites is accomplished by add a Class to any package and applying to Composite stereotype. Composites consist of ports, parts and connectors.

Ports are explained in detail here: Ports / Mediators.

A Part can be either another Composite or a State Machine. If you want to add a State Machine as a part of a composite, first create it in the Outline view under the Class that represents your Composite. Then add a new Property to your Composite diagram, and type that Property to your State Machine by selecting it in your diagram and clicking the plus symbol in Properties View -> General -> Value(s) Definition -> Type.

When adding Ports to a State Machine part in the diagram, the port element will be added to the State Machine in the Outline view and to all other Parts of that type.

If a State Machine that is used as a Part has parameters you need to add arguments to the Part. Select the part and go to its Property view and add the parameter to its Default Value.

Bug: Adding an OpaqueExpression as a Default Value does not work. Work-around: Add a LiteralString.

To create a Composite that JavaFrame can interpret you need to add a stereotype to the Class.

Add the Composite (JavaFrameProfile::Composite) to the class in question.
Main composites
The Composite stereotype has a property called main. If this is set to true there will be generated a Main class which creates this composite.

If the composite is a main composite you need to change the Property of main:Boolean to true (default is false). Expand the Composite stereotype, select main:Boolean[1..1] = false, double-click the Property and replace false with true.

Attributes
Attributes on Composites are treated like State Machine parameters in that they will be added to the constructor of the composite as a parameter. Anytime a composite is used as a part the attributes will need to be set in the part’s default value field just like State Machine parameters.

Multiplicity
Parts can have multiplicity values 1 (default) and *. Any other multiplicity values will be treated as *. If * is chosen there is initially not added any instances of that part but a State Machine can create instances with a <<Create>>Activity/Action. Parts with * multiplicity should not have any default value.

Signals
Signals are sent between ports and cause State Machines to trigger transitions. Signals can have attributes, which will be added as a variable to the generated java class for the signal. Attributes will also be added as parameters to the constructor. Signals can be abstract and they can extend other signals.

A Signal element is transformed to a class extending the JavaFrame Message class, unless it extends another signal.

Signals should be added under the root package of your system by either drawing them in the Class Diagram defining your system or by adding them through the Outline view. You may also want to group them in separate Packages by adding child packages to your root package.

Ports / Mediators
Ports are transformed to JavaFrame mediators. A standard JavaFrame mediator can only have one address and will forward all signals it receives to that address. If you want to define your own mediator behavior, create a class and apply the SimpleRouterMediator, the InputEdgeMediator or the Mediator stereotype. This class can then be used as a type for ports.

Forwarding
If you want to redefine what happens when the Mediator receives a Signal you must add an Activity to the class any apply the ForwardActivity stereotype to it. This is mandatory if you use SimpleRouterMediator.

All Activities in the model will be translated to code. If the Activity has an Action the code will be the name of the Action. If there is no Action the code will be the name of the Activity. The ForwardActivity code will be run every time the mediator receives a signal. The code has access to a sig:Message pointer which points to the signal received.

If the SimpleRouterMediator stereotype is applied you will have access to a mediatorList:List variable. This list contains Addressable objects, which can be either Mediators or State Machines. Otherwise you will have access to an address:Addressable variable. The Addressable interface has one method: forward(Message sig)
Attributes
Mediators can have attributes just like Signals. However if a Mediator has attributes, any port that uses it as a type must set them in Default Value. There you will have access to all Attributes/Parameters defined in the Composite/State Machine which owns the Port.

State Machines
State Machines consist of states, pseudostates and transitions. Supported state types are: State, State defined by a Submachine, and Final State. Supported pseudostates are Initial, Entry, Exit, Choice and Junction.

Attributes / Parameters
State Machines can have both attributes and parameters. Both are added as a variable to the generated java class for the State Machine. Parameters must be set by the Default Value field of a part as explained under Composites. In order to add Parameters right click the State Machine in the Outline view and add the element Parameter. Attributes owned by State Machines can be added the same way be adding a Property instead.

Add information about Timers and the TimerMsg stereotype here.

States
All states can have entry/exit activities. In the activity code for entry/exit you have access to a method: output(Message, Mediator, StateMachine) and a csm pointer which points to the enclosing State Machine.

States can be either regular states or states set as a Submachine. A SubmachineState has another State Machine as its submachine. A State Machine used as a Submachine in a SubmachineState, must be owned by the State Machine which owns the Submachinestate.

State Machines used as submachines can have entry/exit points, but are not automatically reflected in the SubmachineState. This can be achieved by drawing a ConnectionPointReference on the state, and setting it to its respective Entry/Exit point after you first have defined the points in the State Machine.

Transition-triggers
Transitions from States should have trigger(s). Triggers can be added to a transition by selecting the Transition and a SignalEvent to the list of triggers. For SignalEvents that use signals not previously used, you need to define a new Signal Reception that points to the Signal in question.

Add information about using signals that extend TimerMsg here.

When a State Machine receives a signal it will check all the triggers of the outgoing transitions in it current state, no assumptions about the order the transitions are check should be made. If none of the current state’s transitions fire, transitions on the enclosing state are checked. If no transitions fire all the way to the top-level state the signal will be ignored.

A state can have deferrable triggers. To add a deferrable trigger, create a trigger element in the list of Deferrable Triggers. Deferred signals do not trigger any transitions, but are saved until the State Machine enters another state where they can trigger transitions just like newly arrived Signals.

Transition-effects
Transitions can have Effects. An Effect is either an Activity or a OpaqueExpression. They are both transformed to java code and are run whenever the transition fires. If the activity has an Action element the code will be the name of the Action. If there is no Action the code will be the name of the Activity.
Transition effect code has access to these pointers/methods:

- output(Message, Mediator, StateMachine)
- sig: pointer to the signal that triggered the transition
- csm: to top-level statemachine this region is part of

If the transition can be triggered by more than one signal the signal is of type Message, which is the supertype of all signals. If the transition is triggered by only one signal, the sig point has the type of that signal.

Only the top-level statemachines are transformed to a StateMachine class and the csm pointer points to an object of that type. E.g. a Submachines csm pointer points to the parent Statemachine and not to the Submachine in question.

**Transition create effects**

If you want to add an instance to a part with multiplicity * in the composite that owns this statemachine, apply the Create stereotype to the Activity element. The name of the Activity should be a java call statement with the name of the method equal to the name of the part. If the part is a statemachine and has parameters you need to add arguments to the call. Set the `compositeOwner` property of the Create stereotype to the name of the composite that owns this statemachine. This requires that the statemachine is only used as a part in one composite.

For example, you have a composite TestComposite containing a part create:CreatorStateMachine with multiplicity 1 and another part sm:TestStateMachine with multiplicity *. You want to make a transition inside CreatorStateMachine add a TestStateMachine instance to sm. The name of that transitions <<Create>> activity should be `sm();` The `compositeOwner` property of the Create stereotype should be TestComposite. Using the Create stereotype requires that the CreatorStateMachine is only used in TestComposite.

**Transition guards**

Outgoing transitions from Choice Points should have a guard. A guard must either be a Boolean expression or [else]. If the Choice Point doesn’t have an out-transition with a [else] guard you must make sure that one out-transition always fires when the system reaches the Choice Point. Remember that when a transition has fired it must always end up in another state, otherwise the model is ill-formed (and the generated program will crash).

Junction and Choice points are treated equally in that they both can have multiple in and out transitions. The difference is that a Choice point should have more than one out transition.

**Importing external packages/referencing other packages**

Todo

**Working with Sequence Diagrams**

Todo
Using SMSMediators

**Mediator constructor arguments (SMSInput/Output)**
JavaFrameTransformation looks for Comments associated with Ports to determine if they require constructor arguments. If it finds a comment that starts with “args=“ it will interpret the value of the comments as constructor arguments. If no such comment is found a default constructor is used.

**SMS Mediators**

![Diagram showing SMS IN and SMS OUT mediators]

Select the Port typed with SmsInputMediator.

![Port properties showing comment args=ARGS[0],ARGS[0]]

In the Ports properties page, select Comments and click Add. Edit the comment and write args=ARGS[0],ARGS[0]. Repeat this process for the port typed with SmsOutputMediator.
• Transform JavaFrame projects by right-clicking and selecting **Run JavaFrame Transformation** from the context-menu.

• You want to transform the package that contains your System, not the packages that contain usecases and context definitions. (In the case above, we want to transform ICUo that contains the statemachines, composite structures etc. that actually define the system).

• After compilation change to the **Java**-perspective.

• The result of the transformation lies in the src-folder **generated**. The transformation operation replaces any files that resided there previously.
• A JavaFrame project is very similar to a regular Java project in Eclipse:
  o If you have errors in your code or if something went wrong during the transformation you will most probably see error warnings in the code and in Eclipse’s Problems view.

Executing a JavaFrame Application with JFDebug
The application JFDebug made by Jonas Winje, is a new addition to Papyrus IFI UML.
It runs automatically and does not require any special modifications. We do not however need to run JFTrace anymore.
Follow the guide below, but leave out the arguments for JFTrace and don’t start JFTrace.
The arguments for the Run Configuration should therefore look like this:
username
Instead of:
username –remote localhost:54321

Executing a JavaFrame Application with JFTrace
Before you can execute the java program generated, you need to make a Run Configuration.
• Make sure that you are in the Java-perspective.
• Right-click your project and select Run-as -> Open Run Dialog..
• Select Java Application and press the New button
• Name your Run Configuration
• Click the Search button and select the appropriate Main class for your configuration
• Click tab Arguments and in the pane named Program Arguments, put your user name and then the following which is used for communication with JFTrace -remote localhost:54321
• A properly configured configuration should look like below, given that your project is named ICU and that when running JFTrace you use the standard port number
• Click **Apply** to save your configuration.

If you are going to use **FakePATS** and **JFTrace** these standalone applications may be run by double-clicking their respective **Jars** in Eclipse.

• These should both be started **before** you start your JavaFrame application.

• After starting JFTrace you should also open its socket before starting your application.
  - **File -> Open Input Socket -> OK**
  - After a successful connection has been made between your application and JFTrace, click **Apply Filter** to see the traces.

**Running the application**

To run your application when you have created the run configuration and when you have optionally started (fakepats and) JFTrace, you may use **Run / Run ...** and select the correct run configuration, or you may find it on the menu from the run-icon (white triangle pointing to the right in green circle).

• It may be wise to show the Console tab to see if run-time errors occur.

**Running Google Earth**

If your system generates KML files you may want to display the information through GoogleEarth. Download and read up on GoogleEarth at [http://earth.google.com/](http://earth.google.com/)

Here is how you may display the information in GoogleEarth:

1. Start GoogleEarth
2. Add/Network Link ...

Then a dialog appears

3. Give it an appropriate name – such as “ICU”
4. Click Browse and specify the produced .kml file. Usually in the project folder of the target java project.

5. You will now see where your mobile is on the picture.

If you doubleclick on any of the mobiles, GoogleEarth will fly you to where they are.

Enjoy!
Notes on SMS syntax/semantics

As of per the 03.09.2008 the real PATS interface does not function. But the guidelines given below are compatible with FakePATS and should still be followed.

ICU and other programs that uses the SMSPorts library will require the following conventions for the syntax of the SMS-es sent to the system.

- The SMS-messages are sent to 2034. The PATS system does not work unless you have a Telenor subscription. Only using the Telenor network is not sufficient.
- The syntax of the SMS-text must be like this:
  Stud1 konto username sometext
  where username is your normal user name at Ifi and sometext may be anything
- Alternative syntax is
  Ifi konto username sometext
  but there is a slight difference in semantics which is very important.

Here are some very important points that must be understood:

1. The username defines the queue of the messages at Ifi. This means that if more than one program uses the same name at the same time, there will be confusion with messages.
2. When the user applies the Stud1-syntax the program will only see a “static identifier“ (static id) and not the user’s real telephone number. On the other hand, the user can be positioned through the message PosRequest. SMS-messages can be sent to static ids. Static ids are dependent upon properties of the SIM-card and the phone. Due to the possibility to position a mobile from its static id, you may want to keep your static id non-public.
3. When the user applies the Ifi-syntax, the program will see the real telephone number, but positioning is not possible.

As for the cost of sending and receiving SMS-es, the following applies.

The SMS-messages that you send are charged in the normal way. You do not pay for the SMS-messages that you receive. You do not pay for positioning either, but positioning should be considered a scarce recourse and with a price of 1-3 NOK. Abundant positioning will not be tolerated because it may cause the experimental PATS laboratory to degrade its performance and apply too many resources from the regular Telenor services. Thus it is not allowed to make programs that perform positioning at regular short intervals over an extended period of time.
Known Bugs

Papyrus/All diagram editors

- Copy/Paste between models of UML/diagram elements does not function properly and can corrupt the model.
- Copy/Paste of text is possible via right-clicking, but not by Ctrl-C/Ctrl-V.
- In certain situations elements that you want to see in your diagram are not visible as default, but you can see that you have defined them in the Properties view (like Guards, Body-content etc.)
  - Look in the Properties view -> Appearance tab for a check-box that will make the desired element visible.
- Sometimes Papyrus hangs during certain operations and may hang for up to 30 seconds, but recovers and continues to work afterwards.
- Comments elements are only selectable if you click the frame around the comment.
- When adding elements to another element (Like adding a Port to a Part in a Composite diagram), diagrams other than the one you are editing will not necessarily reflect your changes. To combat this redraw the element in their respective diagrams.

Sequence Diagram Editor

- **IMPORTANT**: Scaling an Interaction or the diagrams containing window to the point where it exists outside of the bounds of “frame/window”/making the interaction larger than visible, introduces some mysterious errors, which might not be obvious immediately but might ultimately corrupt the di2 and UML files.
  - A solution is to maximize the view of your diagram, so that you can scale it at least to the largest size possible on your screen. Double-click the tab of the diagram you want to maximize. To minimize double-click it again, but do not edit it in this view if the Interaction is larger.

- Undo/Redo is not stable and can introduce some serious errors.
- If a Lifeline is vertically larger than the Interaction causes it to become stuck. Scale the Lifeline so it fits within the Interaction to be able to move it again.
**State Machine Diagram Editor**

- After adding an Effect to a Transition of type OpaqueBehavior, you have to Double-click the effect in the field to be brought to its properties. (This also holds for several other elements in Statemachine diagrams).

![Effect Diagram](image)

- A similar bug to the above is that once an element (like a Submachine) has been created, a diagram for the element has not. You therefore have to navigate to the element in the **Outline** view and add a diagram to the element manually.
  - The new diagram will initially be empty. Drag and drop the element that the diagram is going to represent from the **Outline** view to the diagram canvas.

- Moving labels like transition Trigger/Effect may not work properly. The label may jump to a position that you have not selected.
Tips

- Use the outline view to navigate your way through the structure and to find elements.
  - Many elements end up nested within other elements. So immediately after creating an element make sure you give it a relevant name so it will be easier to locate in the Outline-view.

- Always use Delete From Model (in the outline or in the diagram) if you are going to delete something that you want removed from the Outline.

- Referring to above: **Do not use the Delete button on your keyboard** (this equals the “Delete from Diagram” command) if you want to remove elements from your model for good.
  - Using the Delete key without deleting the element from using for instance the Outline will cause your model to become cluttered and may ultimately make your model non-transformable.
  - However, if you have multiple diagrams sharing the same element and you don’t want to remove the element from all of them, do NOT use Delete From Model as it will vanish in all diagrams.

- Tip
  - If you find it inconvenient to write code in the model, try using the comment //TODO instead of the real code. After you transform the model, the location of all the //TODO comments will be shown in the Tasks view. Replace the comment with the real code using the regular java editor and finally paste it back into the model.
Working on a Project (Team-work)

These steps are recommended for the project work, and are not necessary for individual assignments, although you may still wish to use an SVN to **backup** your individual assignments as well.

Using a shared repository to collaborate and – importantly – to **backup** your project files is recommended. Papyrus (and the IFI UML tool package) unfortunately suffers from some instability issues, which in some cases can be quite serious resulting in corrupted Di2 and UML files. Backups are therefore highly recommended, and using a SVN can accomplish this.

**PapyrusIFIUML** comes with the **Subclipse**-plugin integrated and is the recommended method of connecting to a **SVN repository**.

The initial process described below of committing your group’s project to the SVN should only be done once by one member.

Uploading your project consists of two steps.

1. The first step involves uploading your **JavaFrame project**.
2. The second involves uploading the **JavaJars resource project**. Although not absolutely necessary, if team members share the same **JavaJars** resource folder it will be easier to distribute possible new versions of the Jars.
Sharing a Project using Subclipse

1. First, go to Windows -> Preferences -> Team -> Ignored Resources
2. Click Add Pattern and add the pattern bin (we do not want to have compiled code on the SVN).
3. Right-click the root-folder of your project and click Team -> Share Project…
4. Select Create a new repository location and click Next
5. Write the URL to your groups SVN

Select: Use project name as folder name

This operation and the creation of the JavaFrame project should only be performed by one person on each group.

The same process should be repeated with the JavaJars.
6. A successful upload of the 2 projects should look like the below (depending on the icon set chosen.
   for Subclipse in Preferences).

**Importing a project from your repository**
Create a new repository location and type the URL.
Ctrl-select both projects (the JavaFrame project and the JavaJars resource project).
Select “Check out into the workspace as projects” and choose the “Head Revision” to check out.

A successful import should look like the above.