



Collaborative Process Modelling with Multi-Touch Tables

Diploma Thesis at Ulm University

Submitted by:

Judith Burkhardt
judith.burkhardt@uni-ulm.de

Reviewer:

Prof. Dr. Manfred Reichert
Prof. Dr. Peter Dadam

Supervisor:

Jens Kolb

2013

Version August 1, 2013

© 2013 Judith Burkhardt

This work is licensed under the Creative Commons Attribution-NonCommercial-ShareAlike 3.0 License. To view a copy of this license, visit <http://creativecommons.org/licenses/by-nc-sa/3.0/de/> or send a letter to Creative Commons, 543 Howard Street, 5th Floor, San Francisco, California, 94105, USA.

Satz: PDF-L^AT_EX 2_ε

Abstract

In business process modelling, as in any application domain, collaborative work gets more and more important as the complexity of the problem increases. Conventional computing workspaces, (e.g. desktop PC, laptop) are not suitable for active collaboration of several contributors on the same device at the same time. In this thesis we develop collabView, a proof-of-concept implementation, to model business processes on a multi-touch table such that multiple users can interact at the same time. Our application connects as a client to the proView business process modelling server application. collabView encourages all collaborators to work actively together and interact with the process models in a direct and intuitive way.

Kurzfassung

Wie in jedem anderen Anwendungsbereich wird die persönliche Zusammenarbeit beim modellieren von Geschäftsprozessen umso wichtiger, je komplexer das Problem wird. Herkömmliche Computerarbeitsplätze (z.B. Desktop PC, Laptop) sind nicht für die aktive und zeitgleiche Zusammenarbeit mehrerer Mitarbeiter am gleichen Gerät ausgelegt. In dieser Arbeit entwickeln wir collabView, eine prototypische Implementierung, die nachweist, dass Benutzer auf einem Multi-Touch-Tisch gleichzeitig an einem Geschäftsprozessmodell interagieren können. Unser Programm verbindet sich als Client mit dem proView-Server für Geschäftsprozessmodellierung. collabView fördert die aktive Zusammenarbeit aller Beteiligten und ermöglicht die direkte und intuitive Interaktion mit dem Prozessmodell.

Contents

1. Introduction	1
1.1. Motivation	1
1.2. Contribution	2
1.3. Overview	2
2. Fundamentals	3
2.1. Process Model	3
2.2. The proView Project	7
2.3. Samsung SUR 40	11
2.4. Collaborative Work	12
3. Requirements	13
3.1. Requirements for Multi User Support	13
3.2. Requirements for Collaborative Work	15
3.3. Requirements for Main Menus	16
3.4. Requirements from proView	17
3.5. Manipulation Operations of Process Views	18
4. User Interaction Design Concept	19
4.1. Main Menu Design Concept	19
4.1.1. Pie Menu	20
4.1.2. Sticky Note Menu	21
4.1.3. Element Menu	21
4.1.4. Rectangle Menu	23

Contents

4.1.5. Choice of Main Menu Concept	24
4.2. Presentation of Process Model	25
4.3. Manipulation Operations of Process Views	27
4.3.1. Manipulating Process Models through Touch Gestures	27
4.3.1.1. Manipulating Edges in Process Models with Touch Gestures	28
4.3.1.2. Manipulating Nodes with Touch Gestures	29
4.3.2. Manipulating Process Models with Context Menus	30
4.3.2.1. Manipulating Nodes	30
4.3.2.2. Title as Button	33
4.4. Insert Menu	34
4.5. Modal Dialogue	35
5. Process Model Layout	37
5.1. Abstract Schema of a Process Model Layout	37
5.2. Calculating Width and Height of a Box and a Path	38
5.3. Vertical and Horizontal Position of Process Nodes	40
6. Proof-Of-Concept Implementation	45
6.1. Server communication	46
6.2. Multi User Support	46
6.3. Visual Components	47
6.3.1. Main Menu	47
6.3.2. Process Area	50
6.4. Manipulation of Process Area	53
6.4.1. Touch Gestures to Manipulate the Process Area	54
6.4.2. Buttons to Manipulate the Process Area	58
6.4.3. Modal Dialogues for Entering Text	60
6.5. Live Updates	61
6.6. Discussion	61
7. Conclusion	65

A. Source Code

69

1

Introduction

1.1. Motivation

Process models are a graphical representation of business processes. Business processes may get very large the more complex the real world business processes are and the more users are involved in. In order to face this problem the *proView* framework support users through *Personalized and Updatable Process Visualizations*¹. Therefore, personalised *process views* for complex process models are provided, which abstract the underlying process model for the specific need of a user. Generally, modelling process models involves different domain experts to capture the whole business process. In particular, domain experts need to discuss about fragments of a process model in which, e.g., they are involved in. For this, they usually need to come together to discuss and remodel process fragments.

¹www.dbis.info/proView

1. Introduction

During this process modelling session domain experts utilise, e.g., paper, whiteboard, or flip charts to document process changes. Afterwards the modifications can be assigned to the actual process model. To avoid this intermediate documentation a multi-touch table to present and change business processes models can be a helpful tool. Hence we implement a proof-of-concept for modelling business processes with multi-touch tables called *collabView*.

1.2. Contribution

- We show that multi-touch tables offer a possibility to provide powerful tooling for active, real-time collaboration of multiple contributors on one project.
- We show that multi-touch table applications are suitable for integration into distributed server-client architectures.
- We show how an intelligent colour concept can emphasises the ownership of screen items and therefore support smooth collaboration of multiple users.
- We show how sophisticated graphical user interface (GUI) concepts can significantly contribute to optimal usage of the limited space when multiple users work at the same time.

1.3. Overview

The remaining of this thesis is structured as follows: Section 2 introduces fundamentals of process models, the *proView* and the hardware we use for *collabView*. Section 3 describes the requirements of *collabView* needs to meet. Section 4 discusses the user interaction design. Section 5 depicts how a graph for a process model is drafted to visualise for the user. Section 6 elaborates how *collabView* on the multi-touch table is implemented. Section 7 concludes the thesis.

2

Fundamentals

Section 2.1 defines the structure of a process model. Section 2.2 introduces the framework which is used to get process models. The multi-touch table used as well as development software is shown in Section 2.3. Section 2.4 explains collaborative work and why multi-touch tables are beneficial for it.

2.1. Process Model

A *process model* is a directed graph with different types of nodes. Nodes represent *activities*, *data elements*, or a *gateways* of the business process. In the following different types of elements of the process model graph are presented.

ControlEdges connect two nodes in a process model. Each *ControlEdge* has a specific start node and a specific end node. It is graphically represented as an arrow, which

2. Fundamentals

points from the preceding to the succeeding node. *StarEvent* and *EndEvent* mark the beginning and the end of a process model. These event are mandatory for a process model. A *StarEvent* has no incoming *ControlEdges* and *EndEvents* no outgoing ones. An activity is the most common part of each process model. Each activity is depicted as a square including the name of the action of the real-world process. Activities have exactly one incoming and one outgoing *ControlEdge* (cf. Figure 2.1).



Figure 2.1.: Graphical Representation of an Activity with *StartEvent*, *EndEvent* and *ControlEdge*

ANDSplit and *ANDJoin* gateways represent an *AND branching block* (cf. Figure 2.2 a), i.e., activities on the outgoing branches of the *ANDSplit* may be executed in parallel. They can only exist as a pair. *ANDSplits* and *ANDJoins* are also nodes. *ANDSplits* can have annotations. These annotations are called name and are at the right side of the *ANDSplit*. *ANDSplits* have one to n outgoing *ControlEdges*. The corresponding *ANDJoin* has the same amount of incoming *ControlEdges*.

XORSplit and *XORJoin* gateways represent an *XXOR branching block* (cf. Figure 2.2 b), i.e., activities on the outgoing branches of the *XORSplit* may be executed conditionally. They can only exist as a pair. *XORSplits* and *XORJoins* are also nodes. *XORSplits* can have annotations. These annotations are called name and are at the right side of the *XORSplit*. *XORSplits* have one to n outgoing *ControlEdges*. The corresponding *XORJoin* has the same amount of incoming *ControlEdges*.

LoopEdges can be added between *StartLoop* and *EndLoop* gateways. They look like gateways of an *XOR branching block*. *StartLoops* can have annotations which are conditions for the *LoopEdges* (cf. Figure 2.3).

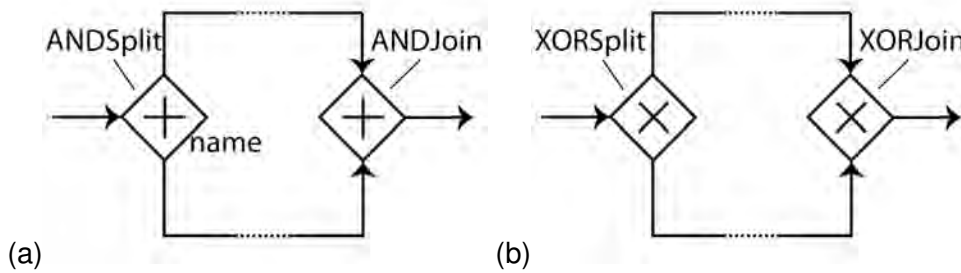


Figure 2.2.: Graphical Representation of an AND Branching Block (a) and an XOR Branching Block (b)



Figure 2.3.: Graphical Representation of a LoopEdge

SynchronisationEdges may be added between two activities on parallel branches. *SynchronisationEdges* are represented through dashed lines (cf. Figure 2.4).

Generally, process models need to be well-structured (cf. Figure 2.5) [Rei00], e.g., no overlapping branching blocks are allowed between different split and join blocks, only nested ones (cf. Figure 2.6).

All our notations are based on *Business Process Model and Notation 2.0 (BPMN)* [OMG11].

2. Fundamentals

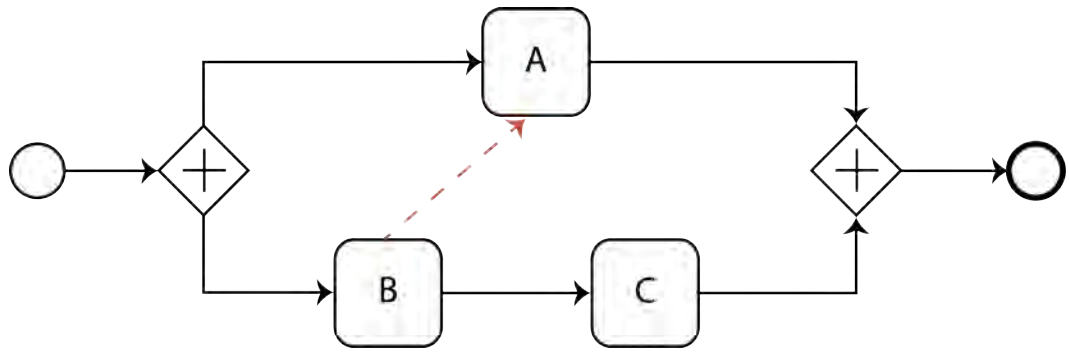


Figure 2.4.: Graphical Representation of a SynchronisationEdge

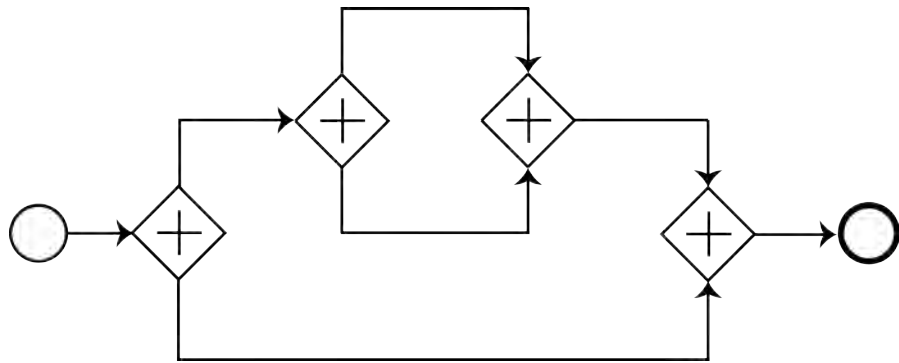


Figure 2.5.: Possible Process Model

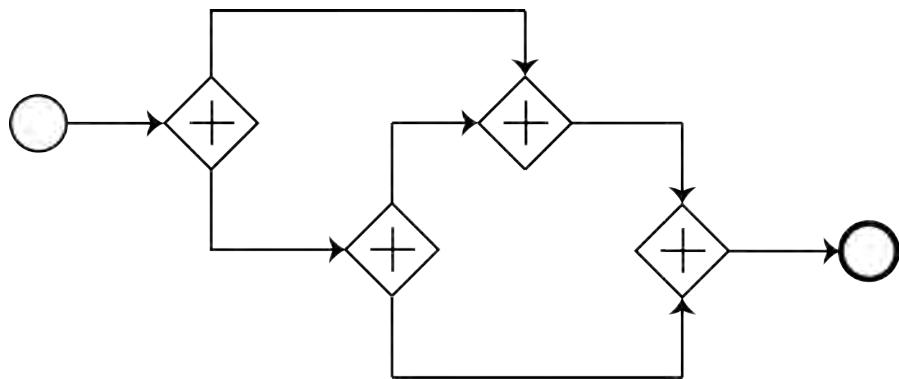


Figure 2.6.: Impossible Process Model

2.2. The proView Project

proView is a research project aiming to support human-centric business process management. Therefore, a framework with server-client architecture was developed to support end user in understanding and evolving complex process models [KR13a, KR13b].

Generally, proView provides personalised perspectives for different user groups to model large business processes [KKR12], since process models may get large, e.g., when many users of different domains are involved in, or if the process model maps a complex real-world business process. The larger a process model is, the more complex it gets. It is assumed that not all users need to see all process steps or are not able to understand them. Manager, for example, require a more abstract view than users with a technical background.

Therefore, the user can create a personalised *process view* based on a *Central Process Model (CPM)* for their specific needs (cf. Figure 2.7). A CPM captures and represents a complex business process [KKR12]. Thus, process views reduce the complexity of a CPM.

To personalise a process view two types of operations may be applied: *reduction* and *aggregation* operations. Reducing or aggregating in a process view does not affect the CPM.

Reduction of activities means the respective activities are hidden in the process view. For example, the Reduce(A) operation in Figure 2.7 reduces activity A and its control edges. New control edges are inserted from the predecessor to the successor of activity A.

Aggregation means that two or more neighbouring activities are assembled to one abstracted activity. Users may enter a name for the resulting abstract activity. For example, the Aggregate(B, C) operation in Figure 2.7 combines the activities B and C and their control edges into a new abstract activity. The resulting activity is inserted between the predecessor node of activity B and the successor node of activity C. New control edges are inserted to the respective nodes.

2. Fundamentals

Furthermore the operation $\text{Reduce}(G, H, I, J, K)$ in Figure 2.7 results in an empty branch. In such cases proView automatically refactors the process view to remove the respective gateways in order to make the process view more comprehensible [KKR12].

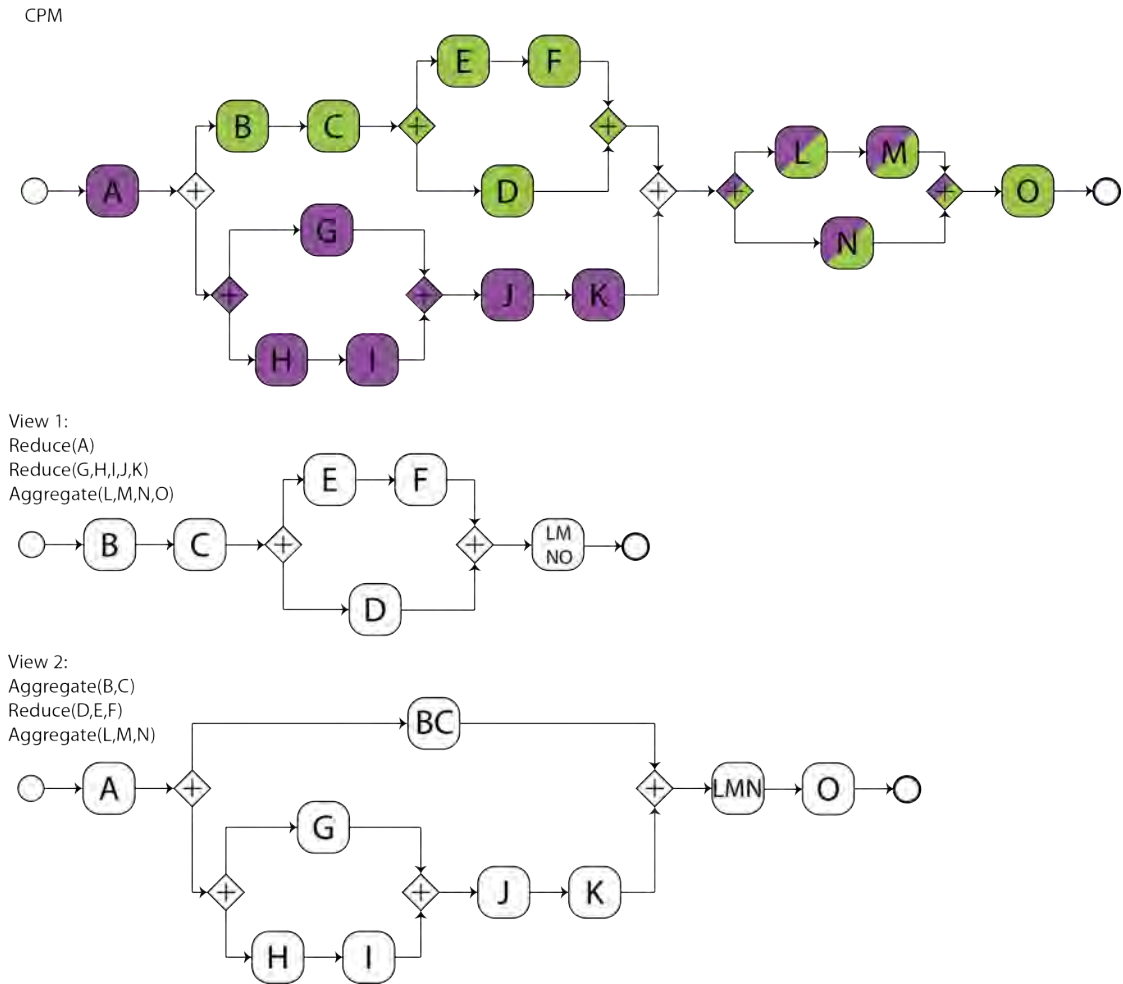


Figure 2.7.: CPM with two Associated Views and Operations

Generally, no changes can be performed directly in the CPMs to evolve the process. Changes in the CPM may be an error-prone task for users. In order to evolve a process users need to change their process view. A change may be *inserting*, *deleting*, or *renaming* process fragments.

In Figure 2.8 a CPM with its associated views can be seen. If change operations are performed in a process view (cf. Figure 2.8 b), the corresponding CPM is updated. Afterwards all other associated process views are updated to the new version.

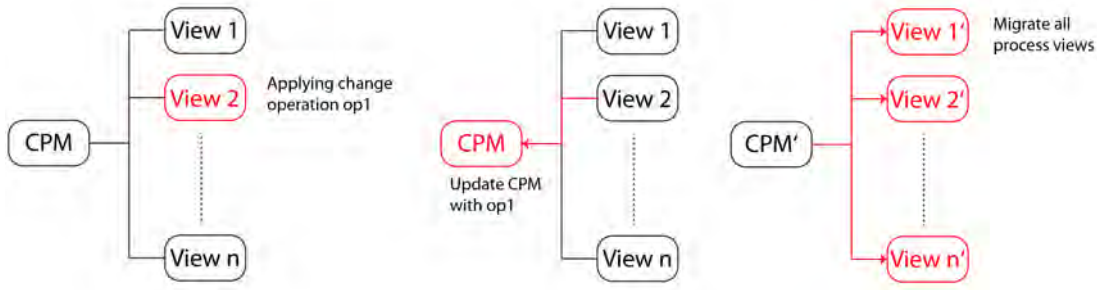


Figure 2.8.: CPM with its Dependent Process Views (a), Change Operation on Process View 2, Update of the Associated CPM, and Update of the Dependent Process Views

If a process view is personalised to the users needs, it might not be possible to insert an activity on a unique position of the corresponding CPM. [KKR12] use a policy which supports parametric propagations. These operations with possible parameters are shown in Table 2.1. The bold types are the default parameters.

2. Fundamentals

Operation	Parameter & Value	Description
$InsertSerial(V, n_1, n_2, n_{new})$	$InsertSerialMode = \{\mathbf{EARLY}, LATE, PARALLEL\}$	Insert activity n_{new} between n_1 and n_2 in view V . The parameter describes the propagation behaviour of this insertion.
$InsertParallel(V, n_1, n_2, n_{new})$ $InsertCond(V, n_1, n_2, n_{new}, c)$ $InsertLoop(V, n_1, n_2, n_{new}, c)$	$InsertBlockMode = \{\mathbf{EARLY_EARLY}, LATE_LATE, LATE_EARLY, EARLY_LATE\}$	Insert activity n_{new} as well as an AND/XOR/Loop block surrounding the <i>Single Entry Single Exit</i> block defined by n_1 and n_2 in view V . the first (last) part of the parameter value before (after) the underline specifies the propagation behaviour of the split (join) gateway.
$InsertBranch(V, g_1, g_2, c)$	$InsertBranchMode = \{\mathbf{EARLY}, LATE\}$	Insert an empty branch between split gateway g_1 and join gateway g_2 in view V . In case of conditional branchings or loops, a condition c is required.
$InsertSyncEdge(V, n_1, n_2)$	-	Insert a sync edge from n_1 to n_2 in View V , where n_1 and n_2 belonging to a different branch of a parallel branching.
$DeleteActivity(V, n_1)$	$DeleteActivityMode = \{\mathbf{LOCAL}, GLOBAL\}$	Deletes activity n_1 in view V . The parameter decides whether the activity is deleted <i>locally</i> (i.e., reduced in the view) or removed from the CPM (i.e, global).
$DeleteBranch(V, g_1, g_2)$	-	Deletes an empty branch between gateways g_1 and join gateway g_2 in view V .
$DeleteSyncEdge(V, n_1, n_2)$	-	Deletes a sync edge between activities n_1 to n_2 in View V .
$DeleteBlock(V, g_1, g_2)$	$DeleteBranchMode = \{\mathbf{INLINE}, DELETE\}$	Deletes and AND/XOR/Loop block enclosed by gateways g_1 and g_2 in view V . The parameter describes whether elements remaining in the block shall be <i>inlined</i> or <i>deleted</i> .

Table 2.1.: Update Operations for Process Views with Parameters [KKR12]

2.3. Samsung SUR 40

The *Samsung SUR 40 with Microsoft PixelSense* technology is a multi-touch table. Its measurements are 27.8" in width, 43.1" in height and 4" in depth and can recognise up to 50 simultaneous inputs. In our setup it is used as a table, but it may also be mounted on a wall. The screen has 40" in diagonal screen size and a 1920x1080 px, i.e., FullHD, resolution [Sam13].

Microsoft PixelSense is a technology where infrared light is used in pixels to recognise fingers, hands, and objects. PixelSense works with four layers. The first is *protection glass*. Below the protection glass a *LCD glass panel with integrated sensors* comes. Then an *optical sheet* comes where the PixelSense processing is done. The lowest layer is a *backlight with visible and infra-red LEDs* (cf. Figure 2.9 [Mic13c]).

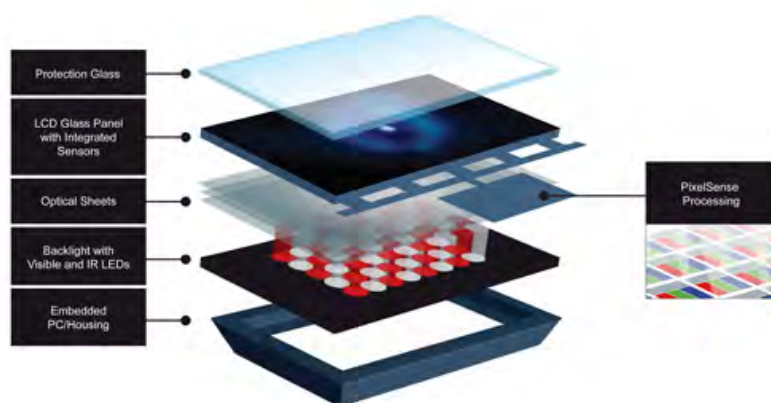


Figure 2.9.: Architecture of the PixelSense Technology

The backlight with visible and infra-red LEDs reflect light throughout the overlaying layers. If a finger is placed on the protection glass it reflects the infra-red light from the backlight. The integrated sensors in the second layer see the reflected light and convert it into electrical signals. The signals from all sensors are reported to the optical sheet. There

2. Fundamentals

the whole picture is analysed with image processing technology. The output is send to the embedded PC [Mic13c].

To develop for PixelSense Microsoft Surface 2.0 SDK is used. This SDK has two development layers. We use the *Presentation Layer*. The Presentation Layer uses *Microsoft Windows Presentation Foundation (WPF)*. WPF supports custom controls as well as custom graphics and touch enabled standard controls. It combines *XAML* with *C#* respectively *Expression Blend 4* with *Visual Studio 2010* [Mic13a].

2.4. Collaborative Work

The term “collaborative” is derived from the latin word “collaborare”, i.e., to work together. We use collaborative in the meaning of working together by two or more people solving a problem at one place.

Collaborative work is a consequent step from the other applications of proView which are developed for single user workspaces. Different setups are possible to work collaborative.

A study identifies four different setups to model business processes in a collaborative way and evaluated the advantages and disadvantages in user studies [Wit12]:

S1: Vertical touch-screen with an optional keyboard and mouse

S2: Multi-touch table

S3: Laptop and a video projector

S4: Whiteboard without technical support

Further, in sessions utilising a multi-touch table users are more aware of what they are doing and are more communicative. In comparison so setup S2, in setup S1, S3, and S4 not all users actively modified the process model. This lack of activeness may lead to reduced concentration and a wrong or incomplete process model. Avoiding this and involving all users a multi-touch table can be a useful tool to work actively together.

3

Requirements

In the following the requirements are laid out. In particular requirements to support multiple users especially for collabView are presented in Section 3.1. Section 3.2 introduces requirements for successful collaborative work on multi-touch tables. Requirements for the main entry point in collabView, the main menu, are presented in Section 3.3. In Section 3.4 the requirements from proView are presented. Further, Section 3.5 describes requirements for the manipulation operations of process views.

3.1. Requirements for Multi User Support

Complex process models require the collaboration of several domain experts to cover the whole real-world process, since a single domain expert, usually, does not have the whole knowledge of an application domain. Moreover, various domain experts may have

3. Requirements

a different perspective on the respective business process. Therefore, it is necessary to meet and discuss the details of the business process.

[MKG02] shows that the success and efficiency of collaborative work on complex problems strongly depends on the availability of a clear and understandable representation.

This leads to the following requirements:

- MUR1 (Avoid Distracting User Interface (UI) Elements): Distracting UI elements would lead the focus away from the actual task that needs to be performed.
- MUR2 (Limit Desktop-style Menus): Desktop-style menus should only be used where strictly necessary, since they need a lot of space. This space cannot be used for the graphical representation of the actual process model. Space is especially limited on a multi-touch table when multiple users are working together. Therefore, desktop menus need to be avoided or limited.
- MUR3 (Minimalistic Process Model Representation): The representation of a process model should be reduced to as few as necessary graphical or geometrical elements. The representation of these elements should be consistent within each other and clearly communicate their function.
- MUR4 (Fixed Process Model Layout): It might be argued that freely movable graph elements give the user more freedom in adapting their process views. However, a fixed layout leads to a clear and ordered representation of the process model. Furthermore, fixed layouts of a process model require less UI interaction. Therefore, the user needs to remember less gestures, which reduces the cognitive load.
- MUR5 (Hide Unnecessary UI Widgets): UI widgets should be hidden as long as they are not specifically called through a gesture. To not disturb the minimalistic representation of a process model, UI widgets required for manipulations of the process model should be hidden as long as they are not needed.

3.2. Requirements for Collaborative Work

Several requirements have to be taken into account when users need to work collaboratively on multi-touch tables. In [SGM03] eight guidelines are introduced for multi-touch tables which should be guaranteed for co-located and collaborative work:

- CWR1 (Interpersonal Interaction): Communication among users should be encouraged. Gesturing, deictic referencing, and meeting coordination activities support collaborations. Pointing to objects helps understanding the point of interest.
- CWR2 (Transitions between Activities): Different manipulations should be possible to perform with one input device for example a finger.
- CWR3 (Personal and Collaboration Work space): Distinguish between personal and collaboration work. Personal activities should not change the whole table and disturb collaboration work. Best transitions between personal and collaboration work space has to be determined.
- CWR4 (Multi-Touch Table Collaboration and External Work): The work done on the multi-touch table needs to be accessible on different workspaces. Also work done on other workspaces need to be accessible on the multi-touch table.
- CWR5 (Physical and Digital Objects): Task-related and non-task-related objects should be recognised and supported by the multi-touch table. So, users can apply their experience from collaborating on conventional workspaces like tables and whiteboards.
- CWR6 (Shared Access to Physical and Digital Objects): To facilitate pointing on objects and facilitate awareness within collaborative tasks should take place on a single object. Orientation of text can be an issue as users are located around the multi-touch table and may have different view angles.

3. Requirements

CWR7 (Appropriate Arrangements of Users): User interactions should be possible from all positions around the multi-touch table, as a table stands free in the room and users can walk around and change their position. Therefore all interaction elements should be reachable from every position of the table. For this reason interaction elements should be free movable and not attached to the margins. Automatic presented information should appear in an appropriate orientation and should be rotatable. Especially menus should not be attached to the margins.

CWR8 (Simultaneous User Actions): Concurrent interactions on the multi-touch table should be possible. Interacting with multiple software components should be possible at the same time.

3.3. Requirements for Main Menu

Due to the novelty of multi-touch tables, no generally accepted conventions, how a main menu should look like to be good and efficient exist so far. Main menus in a conventional UI are commonly located at the top borders. For multi-touch tables this is not appropriate as we argue in the following [Mic11]:

MMR1 (Ownership of Main Menu): In order to give all users equal interaction possibilities and to enable concurrent work on process models each user gets their own personalised main menu.

MMR2 (Fast Capturing of Content): The main menu is not the main interaction point in the multi-touch application. Therefore, the main menu should be small (cf. MUR1 (Avoid Distracting UI Elements)). Further, it should not bind the users attention. The user should be able to capture its functionality and content within one glance.

MMR3 (Variable Amount of Entries in Submenus): The content of all submenu entries is not known on design time. Further, while working with collabView the amount of entries may vary. Therefore the main menu needs to be flexible in size.

MMR4 (Personalised Registration): Personalised registration for users should be supported. This registration could be done with a business mobil phone or a company card. The registration possibilities are called *ID token*.

3.4. Requirements from proView

The proView framework itself has requirements that collabView has to meet [KR13a, KR13b]:

PVR1 (Presentation of Process Model): collabView needs to present process models to the user as a graphical or textual representation, the user understands easily (cf. Section 2.1).

PVR2 (Generation of Process Views): The speciality of proView is the generation of process views out of CPMs. Therefore, collabView needs a possibility to generate a process view out of a CPM.

PVR3 (Personalised Process Views): Process views are personalised in proView, that is why collabView needs to support these personalisation creation functionalities, too.

PVR4 (Deleting of Process Views): A process views should be able to be deleted again by the user, so collabView needs to support such a deletion function.

PVR5 (CPM and Associated Process Views): The CPMs can only be changed through their associated process view, therefore, CPMs need to have an association to their process views and vice versa.

PVR6 (Update CPM and Associated Process Views): If a change in process views is made (cf. PVR5 (CPM and Associated Process Views)) the corresponding CPM and the associated process views need to be updated.

3.5. Manipulation Operations of Process Views

To manipulate process views and thus CPMs with all its supported functionalities (cf. PVR5 (CPM and Associated Process Views)) the following requirements have to be met [KR13b, WPTR13]:

- MVR1 (Insert Process Elements): The proView framework knows an activity, an AND branching block, and an XOR branching block. It should be possible to insert them into process views.
- MVR2 (Delete Process Elements): The process elements in a process view, an activity, an AND branching block, and an XOR branching block, should be possible to be deleted.
- MVR3 (Aggregate Process Elements): It should be possible to aggregate all activities to an abstracted activity.
- MVR4 (Reduce Process Elements): Activities should be able to be reduced in a process view.
- MVR5 (Rename Process Elements): It should be possible to rename activities.
- MVR6 (Change Process Elements): The branching condition of Loop and XOR branching blocks should be possible to change.
- MVR7 (Insert Control Flow Edges): LoopEdges, additional branches in AND branching blocks as well as in XOR branching blocks, and SynchronisationEdges should be possible to be inserted.

MVR3 and MVR4 only affect the process view in which the manipulation operation is performed. All other manipulation operations affect the corresponding CPM and its associated process views.

4

User Interaction Design Concept

In the following the user interaction design concept is introduced based on the defined requirements. Section 4.1 presents the main menu, four different versions, and discusses them. In Section 4.2 the presentation of the process model is introduced. Section 4.3 discusses two options of touch manipulations. For each manipulation option of a process model different possibilities are presented and discussed. Three menus for inserting new nodes are visualised and discussed in Section 4.4. To insert text, a modal dialogue is shown in Section 4.5.

4.1. Main Menu Design Concept

Accessing CPMs and process views, a central starting point is needed. This should be an individual main menu for each user (cf. MMR1 (Ownership of Main Menu)). This

4. User Interaction Design Concept

might be done with a unique colour, or the ID token. The content of the main menu items should be readable (cf. MMR2 (Fast Capturing of Content)). Main menus need to cope with variable amount of menu items (cf. MMR3(Variable Amount of Entries in Submenus)). The ID token for personalised registration (cf. MMR4 (Personalised Registration)) is taken into account during the development of the concepts for the main menus. The main menu can have a boarder to move it on the multi-touch table or the ID token may be a tangible part of the main menu for moving (cf. CWR7 (Appropriate Arrangements of Users)).

Four concepts are considered and discussed in the following. These are a *Pie Menu*, an *Element Menu*, a *Sticky Note Menu*, and a *Rectangle Menu*.

4.1.1. Pie Menu

A type of main menu commonly used on multi-touch tables is the pie menu [BWB06, BLS⁺09, EBBDL09, HFS09]. In this type of main menu all menu items are arranged in a circular way around a centre point.

For personalised registration with an ID token (cf. MMR4 (Personalised Registration)) the central point has to be large enough to contain the ID token. The ID token can be used as tangible part of the main menu. The main menu can be moved on the table by moving the ID token (cf. CWR7 (Appropriate Arrangements of Users)). It further can illustrate the ownership of the main menu (cf. MMR1 (Ownership of Main Menu)).

There are various ways how the menu items may be arranged around the central point and may look like. Two item designs are considered. First, bubbles around the central point (cf. Figure 4.1 (a)) and second, beagle cutouts around the central point (cf. Figure 4.1 (b)) represent the individual menu items.

The bubble styled menu items are floating round the central point and have no connection between each other and central point. If an item has sub-items, they appear outside the bubbles as floating bubbles as well. A connection to the initial item needs to be made.

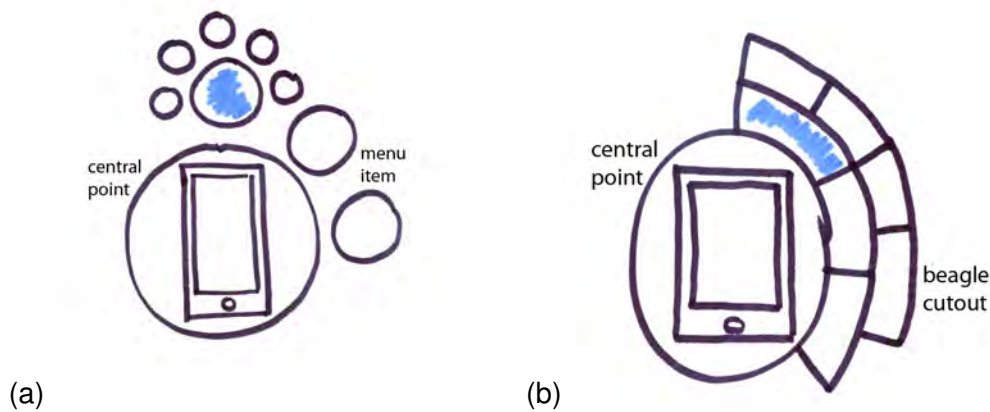


Figure 4.1.: Pie Menu with (a) Bubble-styled and (b) Beagle Cutout-styled Menu Items

The beagle cutout-styled menu items are directly connected to the central point and are connected with each other. If a menu item has sub-entries, another round of beagle cutouts appears at the border of the beagle cutout with the initial item.

4.1.2. Sticky Note Menu

Similar to the pie menu concept, the ID token serves for personal registration (cf. MMR4 (Personalised Registration)) and as a tangible basis (cf. CWR7 (Appropriate Arrangements of Users)). At one side of the ID token sticky notes like rectangular shaped buttons strike out (cf. Figure 4.2 (a)). The titles of the individual menu items are written on the respective sticky notes.

If the user clicks on one sticky note a new column is opened representing submenu items (cf. Figure 4.2 (b)). If there are too many items, the submenu gets scrollable (cf. MMR3 (Variable Amount of Entries in Submenus)).

4.1.3. Element Menu

An *element menu* is the default solution of the Microsoft Surface 2.0 SDK and is a cascading menu [Mic13d]. The element menu has a start button. If the user taps this

4. User Interaction Design Concept

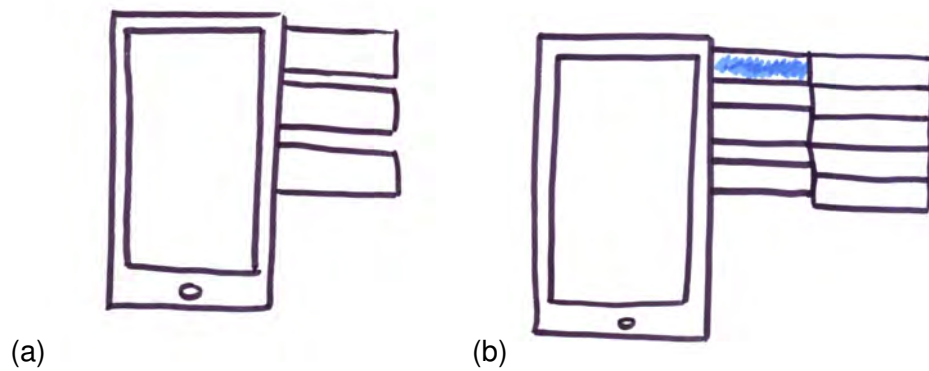


Figure 4.2.: Sticky Note Menu with (a) Menu Items and (b) Submenu Items

button the user gets further element menu items. If there is another hierarchy of element menu items, it indicates this with a triangle in the upper right corner (cf. Figure 4.3).

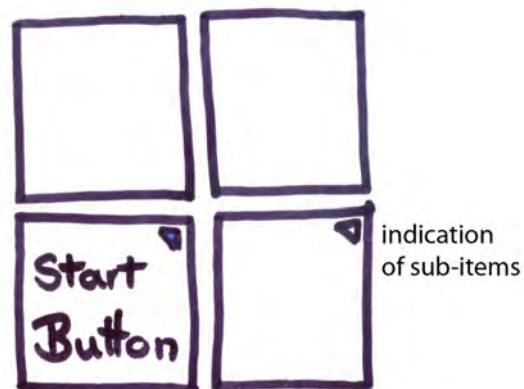


Figure 4.3.: Element Menu

The element menu collapses to its initial start button after some seconds, if the user does not hold their finger down. To select another cascading menu item they move with the finger to the next item or tap the next item.

4.1.4. Rectangle Menu

To register, the user puts an ID token on the multi-touch table (cf. MMR4 (Personalised Registration)) and the respective menu appears behind the ID token. This menu needs to be slightly bigger than the ID token otherwise the user does not see the appearance of a *rectangle menu* (cf. Figure 4.4). After registration, the user needs to lift the ID token to use the rectangle menu.

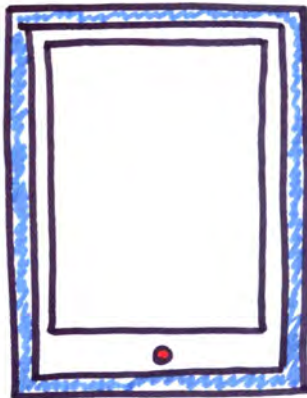


Figure 4.4.: Rectangle Menu with ID token

The rectangle menu represents menu items as squares. The top of the rectangle menu is personalised with the user's name (cf. Figure 4.5 (a) and MMR1 (Ownership of Main Menu)). When the user clicks one of the menu items the main menu vanishes and the respective submenu appears. To go back to the main menu the user clicks the user name. If there are too many entries in the submenu, a scrolling area is used (cf. Figure 4.5 (b) and MMR3 (Variable Amount of Entries in Submenus)).

The rectangle menu has to have a broad border so the user can move it on the table and reaches it from everywhere (cf. Section 3.2 CWR7 (Appropriate Arrangements of Users)).

In addition to the personalisation with the name, the menu gets a specific colour. This colour is reused on windows the user opens from this main menu. So no other efforts have to be made to identify the ownership of CPMs and process views.

4. User Interaction Design Concept

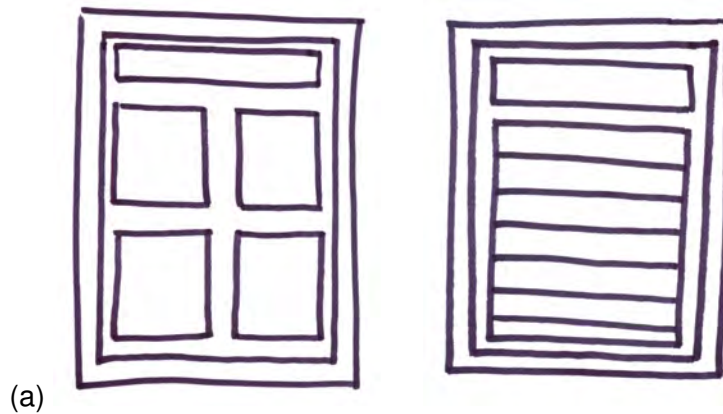


Figure 4.5.: (a) Rectangle Menu and (b) with Sumenu

4.1.5. Choice of Main Menu Concept

Main menus with an ID token as central point take a lot of space of the multi-touch table. This space is taken up permanently and may not be available for other windows in collabView.

If the ID token is taken off the table, empty space without functionality emerges. Therefore a separate representation of the main menu is needed. To visualise that these two representation of the main menus belong together we need a graphical transition between them, e.g. a smooth animation. The ownership of the main menu is clear through a personal ID token, after such a transition the ownership has to be communicated with, e.g., a name or a colour like on rectangle menu.

Pie menus using bubbles or beagle cutouts have limited amount of bubbles or beagle cutouts depending on the size of the central point. Additionally, the size of the bubbles or beagle cutouts limit the amount, too. The amount of element menu items is limited by its natural appearance unless it loses its clarity in which hierarchy level the user is in. This contradicts requirement MMR3 (Variable Amount of Entries in Submenus) in Section 3.3.

The title of the menu items on the beagle cutouts are rounded to fit into the shape. However, rounded typography is hard to read and not for a fast capturing of the content which contradicts MMR2 (Fast Capturing of Content).

4.2. Presentation of Process Model

As these two columns at the sticky note menu, i.e., main menu and submenu items of sticky notes are already consuming much space, a button, gesture, or timer has to be introduced to collapse the submenu column again.

The element menu does not support personal registration as required in MMR4 (Personalised Registration). Further, a boarder for moving on the multi-touch table and colours for personalisation have to be introduced for CWR7 (Appropriate Arrangements of Users) and MMR1 (Ownership of Main Menu).

Hence we choose the rectangular menu. It has a fixed size. This is because the menu is not a main item users should concentrate on. So the little possibilities users have the less time they spend with it. All information of the initial state of the menu is given on one glance. The information entropy will not increase in a bigger menu. Only if the user is in a submenu the information entropy could be increased with a bigger menu. In the submenu information can be reached within seconds of vertical scrolling. One could argue that the menu takes a lot of space during the time users concentrate on the process models and therefore it should be possible to minimise it. But the other visual items can be placed over the main menus. So no main menu is taking no extra space which cannot be used anymore. The user also has fast access to the main menu if it is not possible to minimise.

4.2. Presentation of Process Model

After the user chose a CPM or a process view from the main menu we need to present the process model. This happens in a window-like area, called *process area*, which contains the process model. To meet CWR7 (Appropriate Arrangement of Users) we use a border around the process area.

As each main menu is personalised the process models opened from a main menu are assigned the same colour as the main menu. Further, we add the title of the process model to the process area for fast identification.

4. User Interaction Design Concept

Additionally, the process model has to be minimalist designed and furthermore arranged in a fixed grid (cf. MUR3 (Minimalistic Process Model Representation), MUR4 (Fixed Process Model Layout)).

The UI design of process elements Activities, ANDSplits, ANDJoins, XORSplits, XORJoins, StartEvent, and EndEvent indicates that they are similar to buttons. Therefore, the elements have the same border and the same colours as, e.g., close buttons. The second visual indication is the text at some activities. If the text is too long for the activity, the text ends with three dots. This indicates the user, that they somehow can access the rest of the text. Most users who use collabView might have worked with one or more prototypes of proView before. So a third indication for them is, that they know they can click nodes. The only widgets that are not clickable, but look like buttons are the StartEvent and the EndEvent. They are mandatory for all process models, so they cannot be changed. But to keep the UI design consistent they look like all other process elements.

Some process models are complex and thus large. To not inhibit the whole multi-touch table with one process area, the initial size is limited. The user may resize the process area as they want. A process area with a process model may look like Figure 4.6

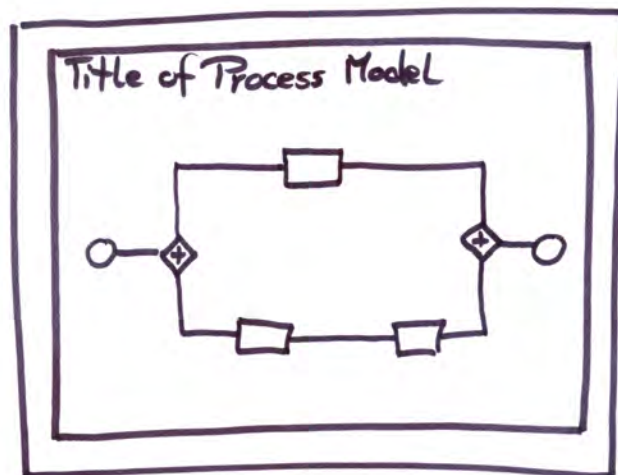


Figure 4.6.: Process Area with Process Model

4.3. Manipulation Operations of Process Views

To change graphs of process models on multi-touch tables there are two main possibilities: The first possibility is *tap*, the second possibility are *touch gestures*. By tap the user is clicking with his finger instead of a mouse. It is called *tapping*. This technique is mostly used for buttons. By touch gestures the user draws with their finger on the surface of the multi-touch table similar to using a pencil.

As computers are widely spread over the years simple buttons are commonly known. However, some conventions for buttons exist which have to be taken into account. For example, a cross on a button implies the closing of a window. Buttons on multi-touch surfaces are not working any different from conventional UIs.

Touch gestures are getting more and more common as smart phones and tablets are spreading, i.e., users can manipulate content with movements, e.g. *single-touch* or *multi-touch*. As multi-touch technology is only a few years commercially spread, conventions are still developing. Each application adjusts touch gestures to its own use.

We try to avoid buttons if possible as required in MUR2 (Limit Desktop-style Menus) in Section 3.1. However, in some cases it is hard to find adequate touch gestures, then we fall back to buttons. In other cases a solution using buttons requires a lot of additional redundant menus, then intuitive touch gesture are applied.

As touch gestures highly depend on the purpose of the applications, no common style guide over all operating systems exists. We designed our interactions mostly based on two user studies who concern the manipulation of graphs on multi-touch devices [FHD09, KRR12]. A third study was used which concerns manipulation gestures on multi-touch tables without a specific context [WMW09].

4.3.1. Manipulating Process Models through Touch Gestures

In the following, possible touch gestures for the given tasks (cf. Section 3.4 and cf. Section 3.5) are introduced and their suitability for collabView discussed.

4. User Interaction Design Concept

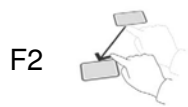
4.3.1.1. Manipulating Edges in Process Models with Touch Gestures

To add edges between nodes [FHD09, FHD10] suggest three different ways to create edges as required in MVR7 (Insert Control Flow Edges).

Also[KRR12] have a core gesture to connect two nodes with a line.



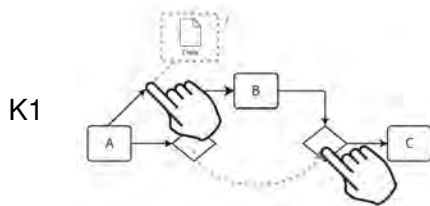
Sequential tapping from one node to another node.



Drawing a line between nodes with an arrow at the end.



Drawing a line and holding the source with another finger.



Drawing a line with a finger from one node to another. The direction of drawing indicates the direction.

F1 creates a collision with other gestures, therefore it is rejected. F2 needs additional time to draw the arrow, which might frustrate the user. F3 needs two hands. Users prefer gestures with one hand [WMW09]. This leads to the conclusion to use gesture K1 in collabView.

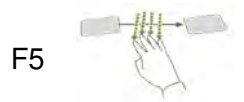
Further a visual feedback is added to complement the touch gesture and aid the user. As the user draws with their finger, a line will appear in order to guide to which nodes they connect.

The SynchronisationEdge has another visual representation unlike other edges. It is dashed compared to the continuous edges of ControlEdges and LoopEdges. [FHD09] suggest three possible ways to change the type of edges:

4.3. Manipulation Operations of Process Views



Short orthogonal lines are drawn over the line that the user wants to change.



The user scratches with a few fingers over the line. They call it the “rake” gesture.



The user taps on an edge for a menu and selects the containing possible edge types.

As SynchronisationEdges are not the most frequent edges we decided, that nothing is different during the drawing. After drawing the line it would appear as a dashed line.

To delete edges two gestures are suggested [FHD09] :



Wipe over the edges.



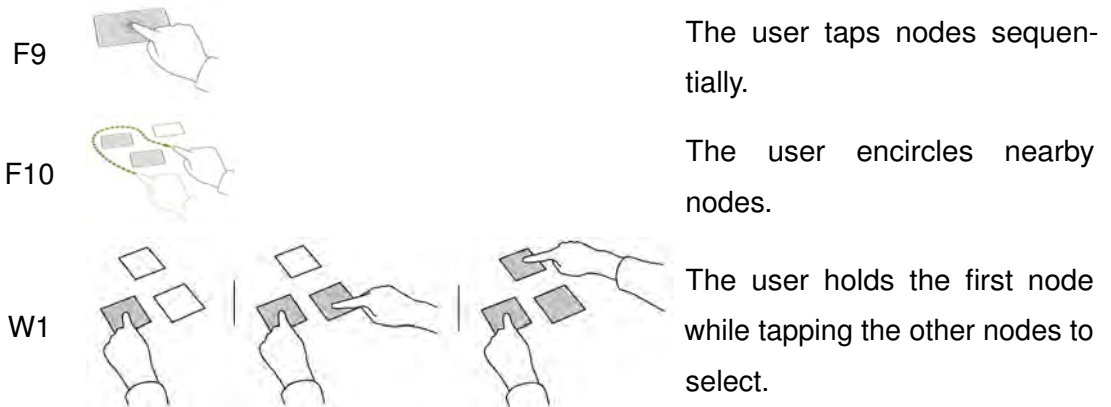
Drag the edges of the screen.

F7 creates a collision with other gestures. F8 is not possible in collabView, because any part of the process model can be moved or dragged. The user does not necessarily know which edges they can delete. Therefore, no edges can be deleted explicitly on collabView. Edges are connected with their nodes. They are deleted if one of their nodes is deleted.

4.3.1.2. Manipulating Nodes with Touch Gestures

One core functionality in proView is the aggregation of nodes to combine nodes in process views [KKR12]. To achieve this, the user needs to select nodes to aggregate. For selecting a set of nodes three gestures are suggested [FHD09, WMW09].

4. User Interaction Design Concept



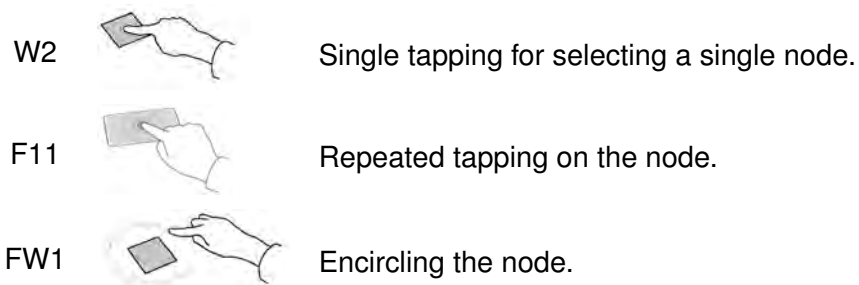
F9 and F10 creates a collision with other gestures, therefore both are rejected. W1 is very intuitive. To support the user in collabView the tapped nodes get a darker colour so they see which nodes they have already tapped.

4.3.2. Manipulating Process Models with Context Menus

Not all manipulations can be done by touch gestures since for some manipulations buttons (e.g., closing a window) are still the faster and more intuitive option and are commonly known from conventional UIs. However other buttons are less obvious because they look like normal text or UI widgets. To recognise them, user needs to know the functionality. Tough these buttons are not that obvious they will be functional. They do not disturb the organisation of the process model. So the whole attention stays there.

4.3.2.1. Manipulating Nodes




For selecting a single node the following possibilities exist [FHD09, WMW09]:



4.3. Manipulation Operations of Process Views

F11 is rejected because it is too similar to two single tapings in a row. FW1 interferes with a touch gesture. Therefore we use W2. It also is very simple.

For deleting nodes following interactions exist [FHD09, KRR12, WMW09].

- FKW1  Wiping over the node like one would do with a rubber.
- FW2  Drag a node of the screen.
- KW1  Draw two crossing lines like an X.

As mentioned before, proView has the concept of reducing nodes. Therefore, we need to find an option for deleting and an option for reducing elements. We rejected all suggestions (FKW1, FW2, KW1) and use buttons, because the communication which touch gesture is for deleting and which for reducing increases the cognitive load.

If users want to see the whole text of an activity with shortened text they tap on the activity. To go back to the initial state of the activity a touch gesture for accepting or going back is needed. [WMW09] suggest a touch gesture for accepting something.

- W3  Drawing a tick.

This action falls into the same category like the deletion touch gesture. To be consistent, we reject W3 as well and fall back to a button.

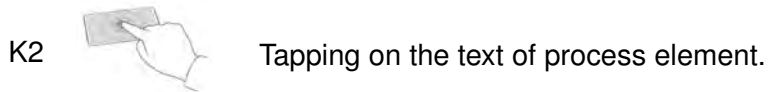
Therefore, if the user taps an element, three buttons appear on top of the element. From left to right: a tick, a bin and a cross. These three buttons are the same for all elements except for the StartEvent and the EndEvent as already explained why. If the user taps the tick, the node goes back to its initial state. The next two buttons tend to be used as synonyms, but in collabView they will have different meanings. If the user taps the bin, the node is deleted from the CPM and its associated process views. Whereas if the user taps the cross the node is reduced from this process view. To complete all functionalities of a node we further need renaming of a node (cf. MVR5 (Rename Process Elements)). To indicate this the name of the node is highlighted (cf. Figure 4.7).

4. User Interaction Design Concept



Figure 4.7.: Node after Tapping

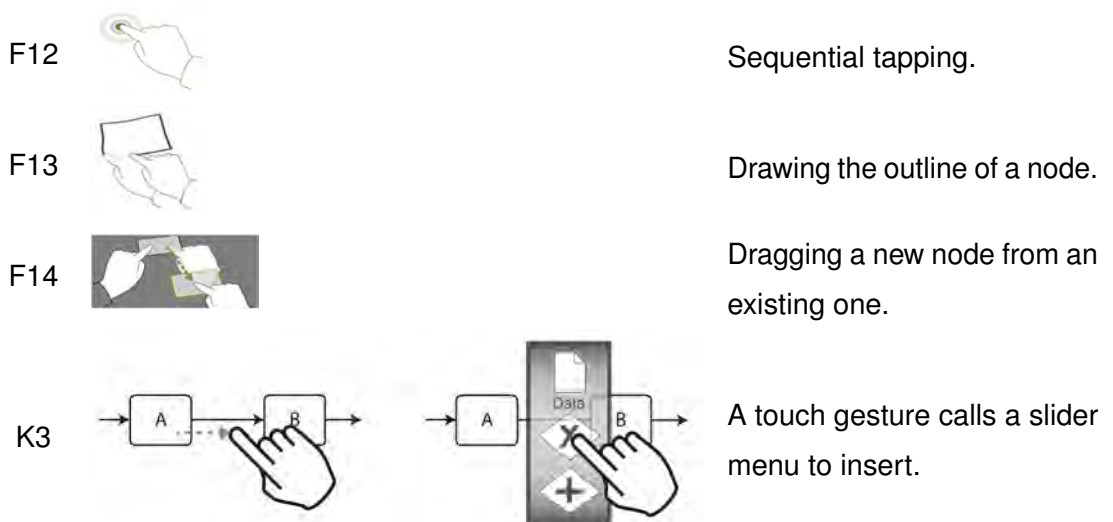
For renaming elements [KRR12] have a core gesture.



The name of a node can then be edited.

Because we have a different result for tapping on an element, we slightly alter K2. If the user taps the node not only the three buttons appear, but also the text of the element changes its appearance. So the user sees they can change the text. They just need to tap it again.

As required in MVR1 (Insert Process Elements), new nodes have to be inserted into the process model. Therefore, the following possibilities exist [FHD09, KRR12]:



Using F12 only one type of node could be inserted. We have to support three types of nodes that can be inserted. F13 creates a collision with the scrolling gesture. F14 is

4.3. Manipulation Operations of Process Views

rejected because we have already decided that no elements can be dragged or moved in collabView. Furthermore, the user could not create a node of a type which is not already present in the process model. Instead, we decided to use a variant of K3. [FHD09] suggested to call a menu by clicking on an edge in F6. We adapt this idea and combine it with K3 such that the menu to insert new nodes is called by tapping the edge (cf. MVR1 (Insert Process Elements)).

4.3.2.2. Title as Button

Users may know from other proView prototypes that it is possible to create process views. There is no obvious menu for the creation of the process view. If the user taps on the title of a process model, three buttons with different functionalities, depending if the process model is a CPM or process view, appear.

If the user taps the title of a CPM, they get three buttons from, left to right, a tick, a cross, and an eye symbol (cf. Figure 4.8 (a)). If the user taps on the tick, the initial state is re-established. If the user taps on the cross, this process model is closed. If the user taps on the eye symbol a new view is created.

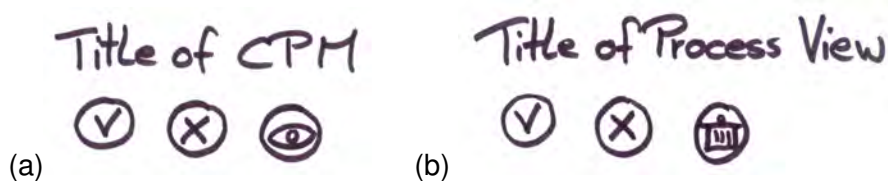


Figure 4.8.: Title of a (a)CPM and (b) Process View with Buttons

If the user taps the title of a process view they get three buttons: a tick, a cross, and a bin symbol (cf. Figure 4.8 (b)). The first two have the same functionalities like the buttons of the CPM. If the user taps the trash bin the process view is deleted.

4.4. Insert Menu

To insert new nodes into a process model as required in MVR1 (Insert Process Elements) a menu is applied. This menu needs to support three different actions: insert an activity, AND branching block, and XOR branching block. Therefore three concepts are presented:

The first menu style is similar to [KRR12]. A vertical menu is used, which appears after a finger tapping on the line (cf. Figure 4.9). All menu items are displayed without scrolling and contain the outlines of the nodes that may be inserted.

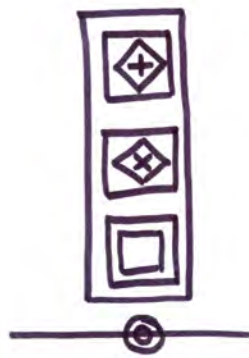


Figure 4.9.: Insert Menu Column

The problem with this occurs with plane process models where, e.g., no AND branching block is in the process model. The menu could stick out of the process area.

The second menu style is structure horizontally, i.e., items appear in a horizontally order. The items are the outlines of the node that can be inserted. To connect all the nodes with the tap point on the edge, they are situated in an area that looks like the top of an hourglass shape (cf. Figure 4.10).

Note that in this menu style the additional outline for buttons is missing. Therefore, they are not that obvious than in the first menu style. The hourglass shape contradicts our clear geometrical design, anticipated for the process model design.

The third menu style combines the buttons with the extra outlines of the first menu style and the horizontal form of the second menu style. We use a rectangular shape instead

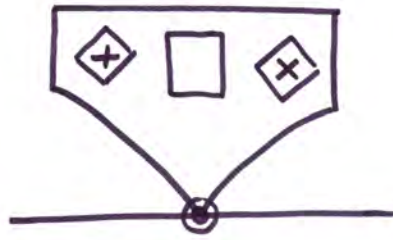


Figure 4.10.: Insert Menu Shaped like an Hourglass

of the hourglass shape (cf. Figure 4.11). The connection to the tap point on the edge is the middle of the menu.



Figure 4.11.: Insert Menu Horizontal

The ordering in the menu of the buttons is that way, that the button of the activity is in the middle. The activity is the most common node to be inserted.

4.5. Modal Dialogue

Triggering modifications, to change names for headlines or nodes needs a *modal dialogue* to input text values.

All modal dialogues consist of three parts. The topmost is a heading describing the adaption to make. Below a text input area is located to insert text using the on-screen keyboard. Two buttons are below the text input area - a "Cancel" and an "OK" button. A modal dialogue looks like in Figure 4.12.

4. User Interaction Design Concept

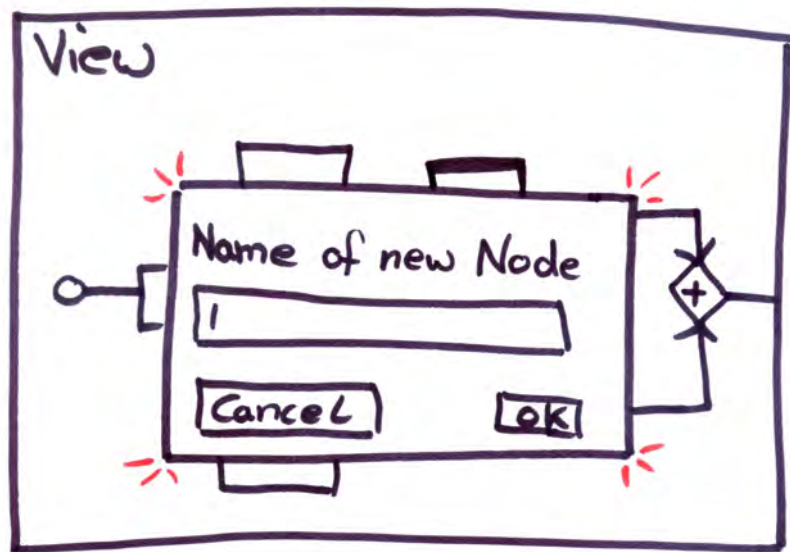


Figure 4.12.: A Modal Dialogue

5

Process Model Layout

This section introduces the formulas we used to calculate the node positions in the process model layout. Section 5.1 describes the abstract schema of a process model layout. In Section 5.2 we present the formulas for calculating the width and height of components. The formulas used for calculating the actual positions of the nodes are defined in Section 5.3.

5.1. Abstract Schema of a Process Model Layout

To simplify the definition of the geometric positions of the nodes of a process model, we use an abstraction of the process model introduced in Section 2.1.

We assume that process models are build of two types of nested regions which we call *boxes* and *paths* (cf. Figure 5.1). A box comprises either one split node, its corresponding

5. Process Model Layout

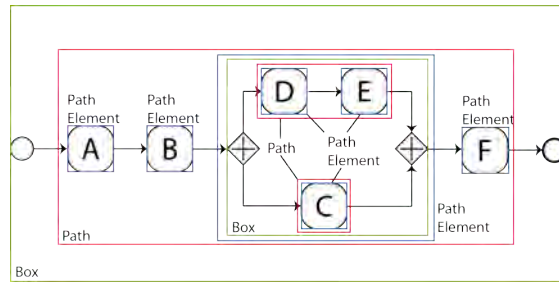


Figure 5.1.: Schematic Representation of Boxes, Paths, PathElements and Activities in a Process Model

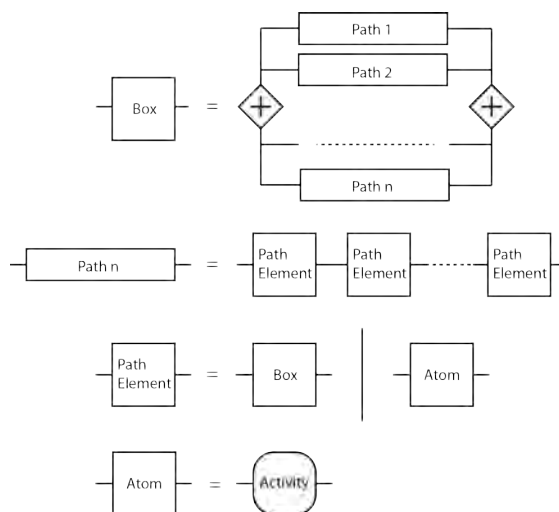


Figure 5.2.: Schematic Definitions of Box, Path, and Path Element

join node, and all nodes and edges in between. A path consists of zero to n consecutive path elements and their connecting edges. The sequence of path elements may be empty in order to represent empty edges. A path element is either a box or an atom which is an activity (cf. Figure 5.2) .

5.2. Calculating Width and Height of a Box and a Path

The following global parameters are used in the formulas: w_{Atom} , h_{Atom} represent the width, height of an atom. w_{Split} , h_{Split} represent the width, height of a split node. w_{Join} ,

5.2. Calculating Width and Height of a Box and a Path

h_{Join} represent the width, height of a join node. Furthermore we use the parameter d , called *default distance*.

The following variable names are used in the formulas: a stands for an activity. e stands for a path element. p stands for a path. b is a box. We use the function name $w()$ for width and $h()$ for height for the respective parameter.

In the following, we assume that there are no empty paths and $h_{Atom} \geq h_{Split}$. Should empty paths occur, our definitions can be extended in the obvious manner.

The width of any activity w_a is defined as the value of the global parameter w_{Atom} .

$$w(a) = w_{Atom}$$

The height of any activity h_a is defined as the value of the global parameter h_{Atom} .

$$h(a) = h_{Atom}$$

We assume that path p consists of the path elements e_1, \dots, e_n . We add the sum of the widths of its paths $w(e_i)$ and one default distance d between each neighbouring pair of path elements to get the width of a path $w(p)$.

$$w(p) = \sum_{i=1}^n w(e_i) + (n - 1)d$$

We assume that path p consists of the path elements e_1, \dots, e_n . The height of a path $h(p)$ is defined as the maximum height of its path elements $h(e_i)$.

$$h(p) = \max_{i=1}^n (h(e_i))$$

5. Process Model Layout

We assume that box b consists of the paths p_1, \dots, p_n . The width of a box $w(b)$ is defined as the sum of the width of a split node w_{Split} and the maximum width of its paths $w(p_i)$ and the width of a join node w_{Join} and one default distance d between each neighbouring pair of paths.

$$w(b) = w_{Split} + d + \max_{i=1}^n(w(p_i)) + d + w_{Join}$$

We assume that box b consists of the paths p_1, \dots, p_n . We add the sum of the heights of all paths $h(p_i)$ and one default distance d between each neighbouring pair of paths to get the height of a box $h(b)$.

$$h(b) = \sum_{i=1}^n h(p_i) + (n - 1)d$$

In abuse of the definition of StartEvent and EndEvent in Section 2.1, we assume that EndEvents are instances of split nodes and join nodes, respectively. Therefore, a process can be viewed as an instance of a box. Now we can calculate the width and height of each box and path in the process model and then the width and height of the whole process model.

5.3. Vertical and Horizontal Position of Process Nodes

To each element and region of a process model, we assign a so-called *anchor point*. By anchor point we mean a fixed point on an element or region which we use to specify the position of that element or region with respect to a coordinate system. For example, we might say that “node N is positioned at point (140,50)”. That is: node N is positioned such that its anchor point coincides with point (140,50) on the coordinate system of reference (cf. Figure 5.3). In the following, we assume that the anchor point of every node is positioned vertically centred on the left border of its respective area.

To position process nodes, we give each box and each path their own coordinate system. Each element of a box and of a path is placed regarding this coordinate system. The

5.3. Vertical and Horizontal Position of Process Nodes

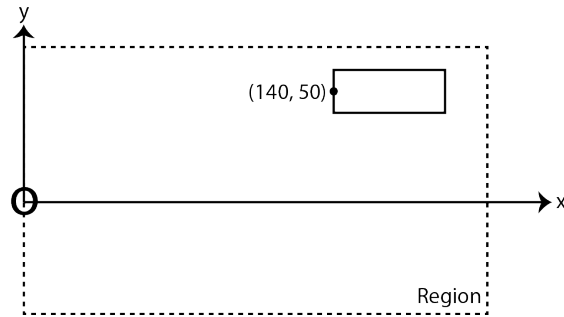


Figure 5.3.: Relative Origin of the Internal Coordinate System

coordinate system of a box is chosen such that the anchor point of the split node coincides with the origin of the coordinate system. This implies that the split node is bisected by the x-axis and touches the y-axis. The coordinate system of a non-empty path is chosen such that the anchor point of the leftmost path element coincides with the origin of the coordinate system. This implies that the leftmost path element is bisected by the x-axis and touches the y-axis.

The anchor point of every box and path are chosen to coincide with the origin of the internal coordinate system of the region.

In the following the global parameters and variable names of Section 5.2 are assumed. Further, we use the function $x()$ for x-coordinate values and $y()$ for y-coordinate values.

Anchor points in a path: The anchor point of the first path element e_1 is positioned at the origin of the coordinate system of the path.

$$x(e_1) = 0, \quad y(e_1) = 0$$

The anchor point of the second path element e_2 , the x-coordinate is defined as the sum of the width of the first path element $w(e_1)$ and one default distance d . We obtain the y-coordinate by zero.

$$x(e_2) = w(e_1) + d, \quad y(e_2) = 0$$

5. Process Model Layout

The anchor point of the n^{th} path element e_n , we obtain the x-coordinate by adding the sum of the widths of the preceding elements $w(e_i)$ and one default distance d between each neighbouring pair of path elements. We obtain the y-coordinate by zero.

$$x(e_n) = \sum_{i=1}^{n-1} (w(e_i)) + (n - 1)d, \quad y(e_n) = 0$$

Anchor points in a box: The anchor point of the split node in a box is positioned at the origin of the coordinate system of the box.

$$x(s) = 0, \quad y(s) = 0$$

For the anchor point of the first path p_1 , the x-coordinate is defined as the sum of the width of the split node w_{Split} and one default distance d . We obtain the y-coordinate by the half height of the first path $h(p_1)$.

$$x(p_1) = w_{Split} + d, \quad y(p_1) = h(p_1)/2$$

For the anchor point of the second path p_2 , the x-coordinate is defined as the sum of the width of the split node w_{Split} and one default distance d . We obtain the y-coordinate by the height of the first path $h(p_1)$ and one default distance d and half the height of the second path $h(p_2)$.

$$x(p_2) = w_{Split} + d, \quad y(p_2) = h(p_1) + d + h(p_2)/2$$

For the anchor point of the n^{th} path p_n , the x-coordinate is defined as the sum of the width of the split node w_{Split} and one default distance d . We obtain the y-coordinate by the sum of n minus one of the heights of each path $h(p_i)$ and half the height of the n^{th} path $h(p_n)$ and one default distance d between each neighbouring pair of paths.

$$x(p_n) = w_{Split} + d, \quad y(p_n) = \sum_{i=1}^{n-1} (h(p_i)) + (n - 1)d + h(p_n)/2$$

5.3. Vertical and Horizontal Position of Process Nodes

For the anchor point of the join node in a box the x-coordinate is defined as the sum of the width of the split node w_{Split} and the maximum width of the paths $w(p_i)$ in the box and two default distances d . We obtain the y-coordinate by zero.

$$x(j) = w_{Split} + \max_{i=1}^n(w(p_i)) + 2d, \quad y(j) = 0$$

Our formulas give us the relative position of every node and region within its surrounding region. In order to render the nodes, however, we need their absolute position, i.e., their position within the coordinate system of the overall process model. The absolute position of a node is calculated simply as the sum of its own relative position and the relative positions of all regions in which the node is contained (cf. Figure 5.4).

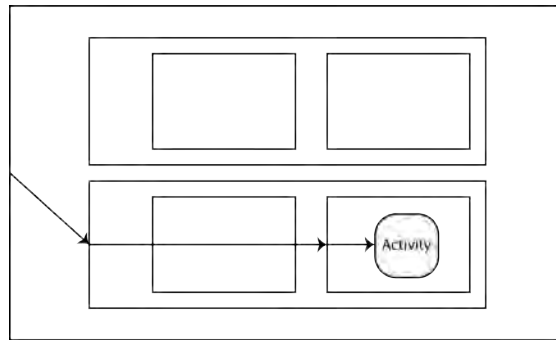


Figure 5.4.: Example how to Calculate Absolute Position

6

Proof-Of-Concept Implementation

This chapter presents the proof-of-concept implementation of collabView. In Section 6.1 the communication with the proViewServer is clarified. Section 6.2 shows how multiple users are supported working with collabView. Section 6.3 address with the implementation of the individual visual components, as well as its appearance and functionalities. The implemented manipulations which may be performed in the process area are shown in Section 6.4. A discussion about live updates of process models after modifications is presented in Section 6.5. Section 6.5 gives a discussion which requirements collabView meets.

6. *Proof-Of-Concept Implementation*

6.1. Server communication

To get the information of CPMs and process views we need to communicate with the proViewServer. This is done with an existing RESTLibrary [Hes13]. This library handles all HTTP requests and responses between the proViewServer and collabView.

The methods for modifying a process model which are not fully implemented on the server yet are also not implemented in the RestLibrary. We added these methods to the RESTLibrary to get the error messages from the server. For example, to insert an XOR branching block (cf. MVR1 (Insert Process Elements)) or to insert a LoopEdge (cf. MVR7 (Insert Control Flow Edges)).

6.2. Multi User Support

In the proof-of-concept implementation collabView, the PixelSense SUR40 is used (cf. Section 2.3). Through its limited screen size of 40" it is limited to about four to six users. It would have to be shown if too many users reverse the benefits of collaborative work.

In collabView, each user requires their own main menu to meet MMR2 (cf. Figure 6.1). Based on this the user may access all CPMs and process views they want to. Each user is assigned a unique colour. This colour is reflected in the user's personal menu. Every CPM, or process view that a user opens is also shown in their assigned colour. This is way the owner of each menu, CPM or process view may identified easily.

As soon as the proViewServer registers a modification of a process model, all CPMs and their associated process views are informed and updated. Hence, all users can work on an up-to-date version.

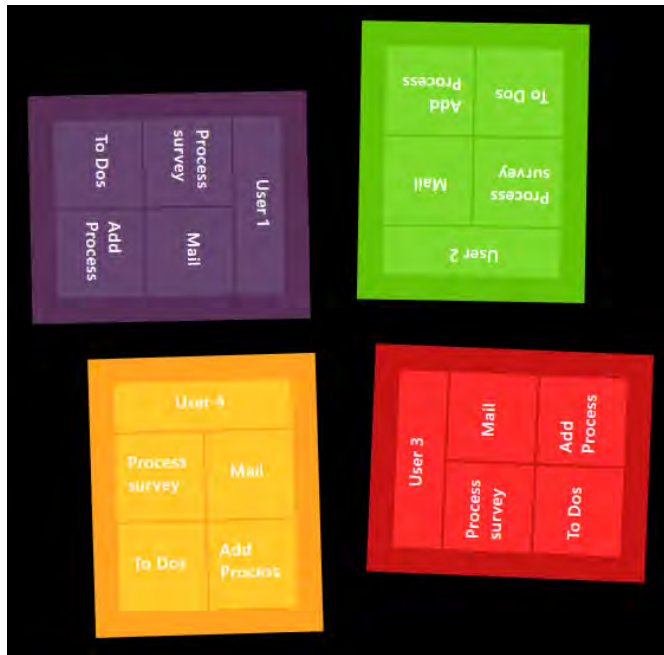


Figure 6.1.: Start Screen

6.3. Visual Components

collabView exists of two major visual components. These are main menus and process areas. All visual parts can be moved and rotated on the multi-touch table (cf. CWR7 (Appropriate Arrangements of Users))

All texts in these components are typeset in the *Segoe360* font. Segoe360 is specially designed for better reading from various angles [Mic11]. The implementation and the appearance of the visual components is explained in the following.

6.3.1. Main Menu

The main menu has a specific colour. The menu consists of an outer darker area called *play field* and an inner brighter one called *content field* (cf. Figure 6.2). The play field is needed because of CWR7 (Appropriate Arrangements of Users) to move and rotate the main menu on the multi-touch table. The content field shows the name of the user

6. Proof-Of-Concept Implementation

in the title. Thus, it is personalised for the particular user after the registration to meet MMR1 (Ownership of Main Menu).

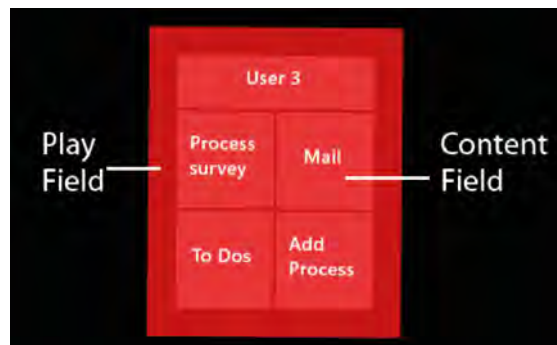


Figure 6.2.: Main Menu with Play Field and Content Field

Each main menu has four entries:

1. Process Survey: to access CPMs and process views
2. Mail: to view mails
3. ToDo: to view a list of things that need to be done
4. Add Process: to create a new CPM

Only the process survey functionality is implemented yet. The others are place holders with suggestions for further functionalities that could be added to collabView.

The transition between the different levels of the menu is done with a storyboard animation which is exemplarily shown in Figure 6.3.

Tapping on process survey opens a list of CPMs and process views (cf. Figure 6.4). Further, a scrollable list is visualised through the first or last entry which is not fully visible, because the list of processes may be too long (cf. MMR3 (Variable Amount of Entries in Submenus)). The user can choose the process model they want to expand. The last tapped item is marked.

Tapping on the user name reverts the main menu to its initial state again.

The *play field* of the main menu supports two functionalities. The user may move and rotate the main menu on the multi-touch table.



Figure 6.3.: Main Menu Animation After Tapping on Process Survey

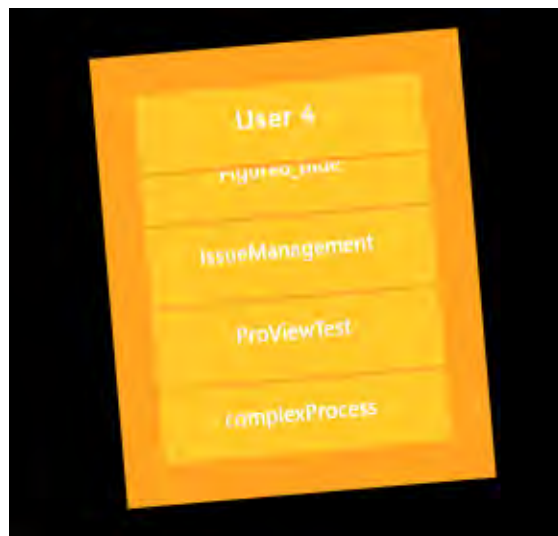


Figure 6.4.: Main Menu showing CPMs an Process Views

6. Proof-Of-Concept Implementation

The implementation of the main menu consists of a `Grid` component which resides in a `ScatterViewItem` component. The `ScatterViewItem` component has functionalities, like moving and rotating, natively implemented. The outer `Grid` components represent the before mentioned play field.

Inside the middle `Grid` component resides the main menu content based on five so called `SurfaceButtons`. If the user taps on process survey a storyboard animation is started where the four lowest rectangular buttons move outward and an area with a `SurfaceScrollViewer` component appears. In this `SurfaceScrollViewer` component reside rectangular `Buttons`. These buttons contain the names of the CPMs and process views. These names are called via the `REStLibrary`. If the user taps on one of the CPM or process view buttons the corresponding process area is opened near the main menu. The rotating angle of the new process area is the same as the main menu angle to meets CWR7 (Appropriate Arrangements of Users).

6.3.2. Process Area

In the following the process area is introduced.

After choosing a CPM or a process view, a new process area (cf. Figure 6.5) appears. The process area has a similar composition as the main menu. Furthermore, the colours are the same as in the main menu to assign it to a particular user.

The initial size of the process area depends on the process model contained. But as mentioned before, space is a big issue on multi-touch tables. Therefore, the initial maximum size of a process area is *840x540 Device Independent Pixels (DIP)*. Then more process models may be open and do not overlap. DIP is defined as $DIP := 1/96inch$, e.g., all objects are scaled properly on all devices without further effort [Mic13b].

Like the main menu, the process area consists of two parts. The *play field* and *content field*. Different colours are hints for different manipulation possibilities (cf. Section 6.4).

The content field distinguishes three different parts: *heading*, *close buttons*, and *process model*. Heading and close buttons are explained in Section 6.4.2. The structure of the process model is explained in the following: Each process node in a process model has

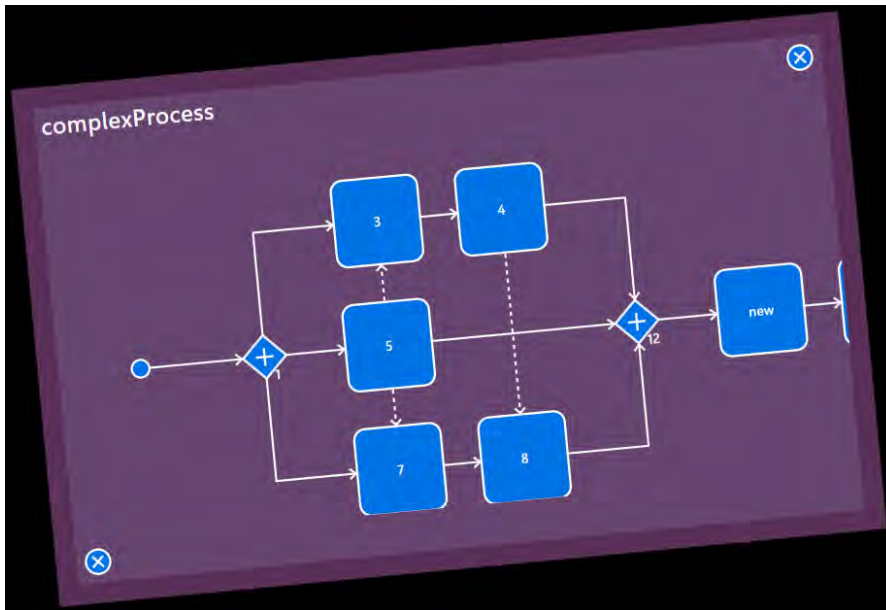


Figure 6.5.: Process Area

an evolutionary found size of $h_{Box} = 80DIP$ and $w_{Box} = 80DIP$. The space between nodes is $d = 40DIP$. For most activity names is the therefore resulting space enough, but some activity names require more than $40DIP$, they are shortened (cf. Figure 6.6).



Figure 6.6.: Node with a Shortened Name

The process area has many nested components implemented. To support the touch functionalities, e. g., moving, rotating, and scaling of Surface 2.0 SDK, process areas needs to reside in a `ScatterViewItem` component. However, the component is not visible to the user. In the `ScatterViewItem` component resides a `Grid` component which consists of three columns and three rows. The outer columns and rows are used

6. Proof-Of-Concept Implementation

to move, rotate, or scale the process area. In the middle column and row resides another `Grid` component, which has three rows: One for the heading and the right close button, one for the left close button, and one for the next residing `SurfaceScrollView` component. This `SurfaceScrollView` component supports horizontal and vertical scrolling, if the process area is not large enough to show the process model.

Inside this `SurfaceScrollView` component is a `Grid` component and inside this a `Canvas` component. This `Canvas` component is required to draw the edges between the nodes. Also manipulation events of the user are captured through this component. These events will be explained later in this section.

Inside the `Canvas` component resides a `ScatterView` component. This `ScatterView` component is needed for process nodes. Nodes are residing in `ScatterViewItems` component so the `ScatterView` component is needed to insert them at all. How the positions of the nodes of the process model are calculated is explained in Section 5.

After `PreviewTouchDown` is called, it depends on whether the process model is a CPM or a process view. If it is a CPM only the scrolling on the `SurfaceScrollView` is activated. If it is a process view it depends on where the user has touched the `Canvas`. Four options are possible:

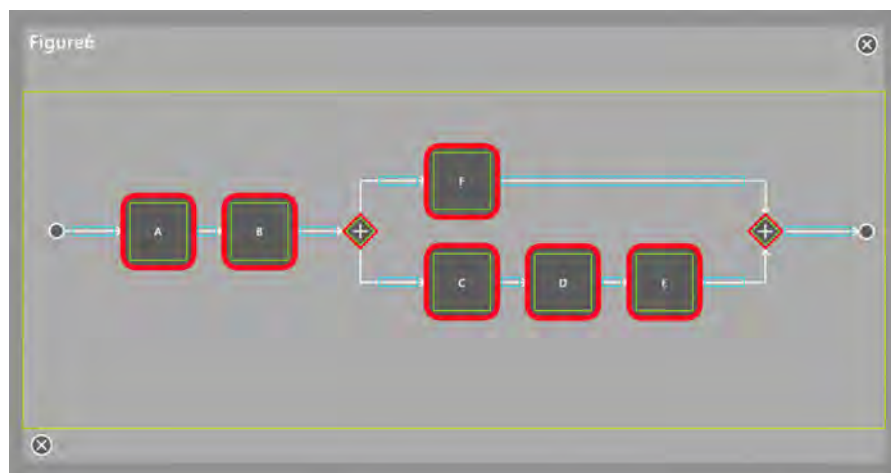


Figure 6.7.: Touch Regions for Insertion Menu (blue region), Manipulation of Nodes (green regions), Drawing Edges (red regions), and Scrolling (yellow region without containing regions) in a Process Area

6.4. Manipulation of Process Area

1. If the user taps a ControlEdge (cf. Figure 6.7 blue regions) an insertion menu opens (cf. Figure 6.8).
2. If the user taps a process nodes (cf. Figure 6.7 green regions), the event is forwarded to the handling of the process node.
3. If the user taps on free space (cf. Figure 6.7 yellow region where no other regions are) they may scroll.
4. If the user taps on one of the borders of a process node (cf. Figure 6.7 red regions) and moves their finger `PreviewTouchMove` is called.

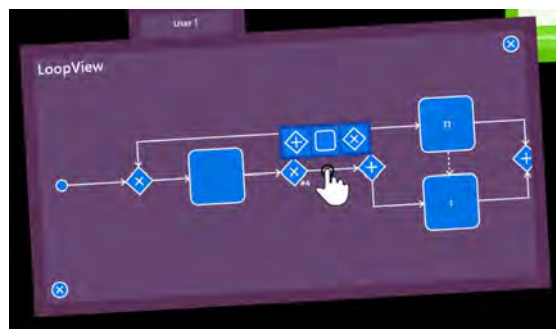


Figure 6.8.: Insert Node with Insertion Menu

The `PreviewTouchMove` event is called as soon as one of the three values of `x`, `y`, or orientation changes. This event handler (cf. Listing A.1) calls a method to draw a line from the previous `x` and `y` values to the new `x` and `y` values (cf. Listing A.2).

The `TouchLeave` event handler (cf. Listing A.3) vanishes the drawn line and sends the origin and the destination of the line to a method, which checks if it is possible to insert a new `ControlEdge`, `LoopEdge`, or `SynchronisationEdge` (cf. Listing A.4) .

6.4. Manipulation of Process Area

In `collabView` all process nodes are on a fixed position with a grid (MUR4 (Fixed Process Model Layout) in Section 3.1). Further, `ControlEdges` are represented in a grid and

6. Proof-Of-Concept Implementation

after a SplitNode with more than one outgoing edges, the ControlEdges are straight and angles are only 90° (cf. Figure 6.9)

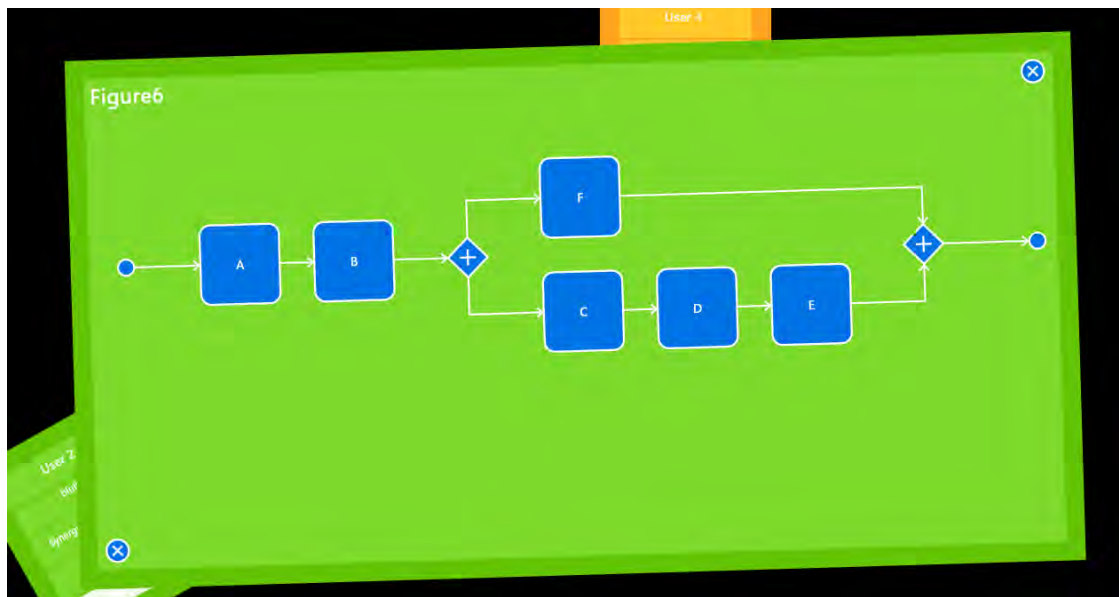


Figure 6.9.: Example of a Process Model

6.4.1. Touch Gestures to Manipulate the Process Area

As aforementioned process areas consists of a play field and a content field and colour differences suggest different functionalities possible on the areas.

Generally, the play field has three functionalities:

First, it is for *moving* the process area around the table. The user taps down and holds the play field and moves it around (cf. Figure 6.10). If the user stops and takes their finger up, the process area stays at this position. If the user takes their finger up during the movement, the process area slides further in the same direction.

The second functionality of the play field is *rotating*. This is also performed with one finger. To rotate the process area the user taps down the play field and rotates their finger as if tuning a screw (cf. Figure 6.11). As with movements, if the user takes their finger up during rotation, the process area spins further.

6.4. Manipulation of Process Area



Figure 6.10.: Moving Process Area

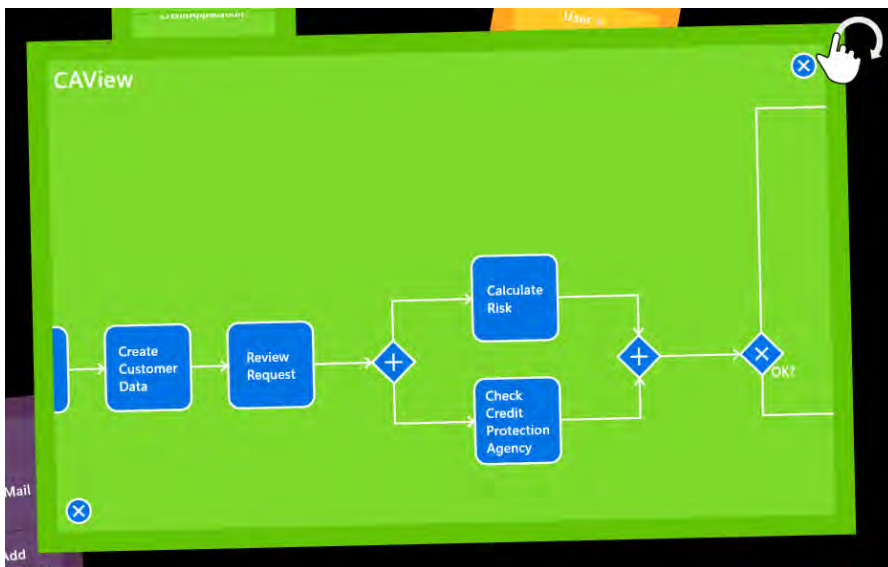


Figure 6.11.: Rotating Process Area

6. Proof-Of-Concept Implementation

The third functionality is *resizing* the process area. This is performed with two fingers. The user taps down play field with two fingers on two opposite corners (cf. Figure 6.12). Moving the fingers apart makes the process area larger. Moving them together makes the process area smaller.

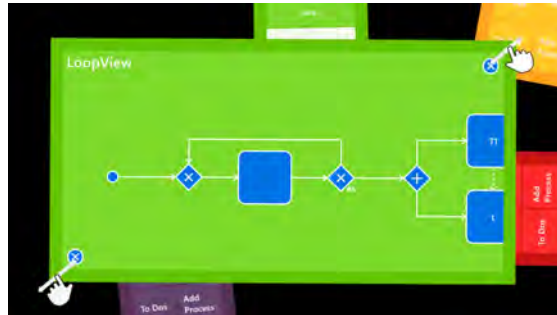


Figure 6.12.: Scaling Process Area

The content field of the process areas is divided in three obvious parts. The top, middle and bottom region. The top and the bottom will be elaborated in Section 6.4.2. In the middle region of the process area the actual process model can be seen. Here the process model can be changed with tapping on buttons and touch gestures. The tapping on buttons will be explained in Section 6.4.2 and the touch gestures in the following.

Per default the process area is 840x540 DIU large. But most process models require more space than this. If a user does not want to enlarge the process area, but wants to see other parts of the process model they may scroll inside the middle part using one or two fingers (cf. Figure 6.13).

In the following, the different lines that can be inserted are explained, as well as the possible mistakes that might occur:

Adding a new ControlEdge between an ANDSplit and ANDJoin or an XORSplit and XORJoin the user needs to tap down at the border of the split nodes and draw a line with their finger to the corresponding join node (cf. Figure 6.14). Giving a visual feedback a white line appears behind their finger. If the finger is lifted on an area where there is no node, the line vanishes and nothing else happens. If the finger is lifted at a corresponding join node, a modification operation is sent to the proViewServer that the user wants to

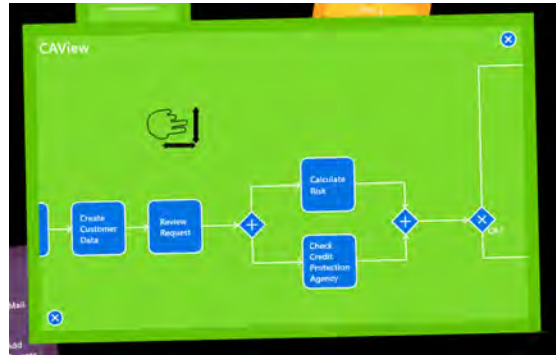


Figure 6.13.: Scrolling Process Area

add a ControlEdge between these two nodes. Since the proViewServer has this action not yet implemented, it sends an error back. As soon as the proViewServer implements the function the corresponding process model is redrawn. If it is not possible to add a line between the two nodes for logical reasons the line vanishes and nothing further happens.



Figure 6.14.: Insert new Edge

To add a SynchronisationEdge, the user needs to connect two respective activities.

Aggregating multiple nodes, the user needs to tap on a node with one finger and holds it down (cf. Listing A.5). Then, the user selects the other nodes, they want to aggregate as well (cf. Figure 6.15). Lifting up the first finger (cf. Listing A.6) aggregates the set of nodes and a modal dialogue appears to name the aggregation (cf. Section 6.4.3).

6. Proof-Of-Concept Implementation

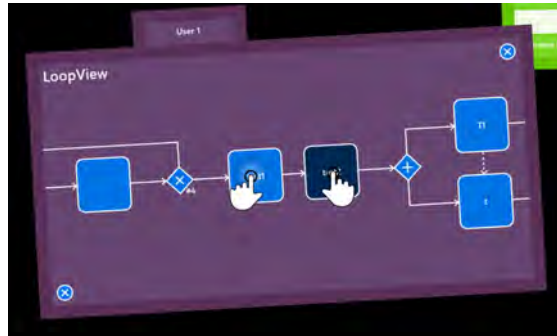


Figure 6.15.: Aggregate Nodes

6.4.2. Buttons to Manipulate the Process Area

Each process area has two close buttons. To avoid closing accidentally, a user needs to tap both buttons. This can be done with one or two fingers. For the first method the user taps one close button and within five seconds the second one. Alternatively the user taps on both close buttons (cf. Figure 6.16).

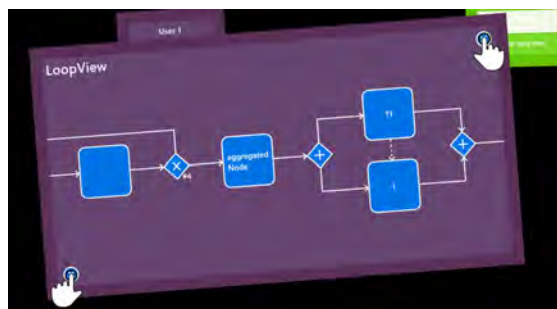


Figure 6.16.: Closing of Process Area

The manipulation possibility that appears if a node is tapped, looks slightly different for activities (cf. Figure 6.17 (a)) than for ANDSplit, ANDJoin (cf. Figure 6.17 (b)), XORSplit, and XORJoin. For activities the name moves to the upper left corner and is coloured differently to visualise the interaction possibility. For all other nodes, the text only gets another colour.

6.4. Manipulation of Process Area

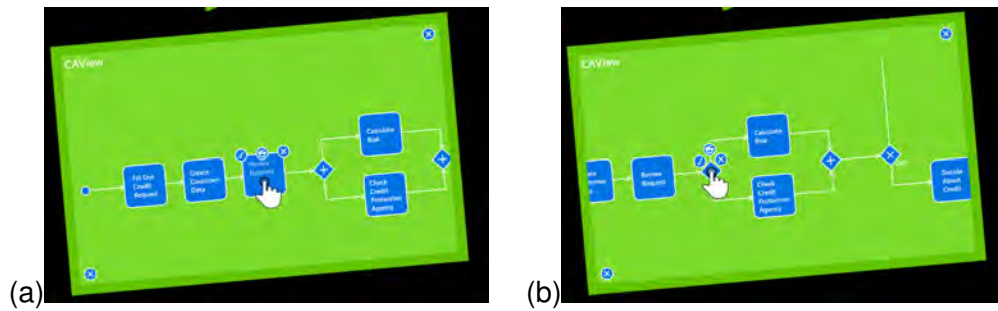


Figure 6.17.: (a)Activity and (b) Split with Buttons

For modifications of the text the user just needs to tap on the text and a keyboard appears to change the text (cf. Figure 6.18). The keyboard limits the collaboration of users since only one keyboard can be shown at a time [Mic13e]. After changing the text the user taps the “Tick” on top of the node to confirm.



Figure 6.18.: Renaming an Activity with Keyboard

To insert nodes the user needs to tap on an edge between two existing nodes and an insertion menu appears (cf. Figure 6.8). This menu shows an AND branching block, an activity and an XOR branching block. After three seconds, the menu vanishes if the user does not tap it.

If the user taps the activity button, the modal dialogue appears to enter the name of the new activity.

6. Proof-Of-Concept Implementation

If the user taps the AND or the XOR button, a new AND branching block or XOR branching block is created. The neighbouring existing activity node is moved into one branch of the block and a new activity node is created in the parallel branch. Again, the name of the new activity node is entered, using the modal menu.

6.4.3. Modal Dialogues for Entering Text

On inserting a new node or creating a view a modal dialogue to enter a name of title appears (cf. Figure 6.19). The dialogue inhibits all functionalities of the process area, except movement) until some text has been entered and confirmed.

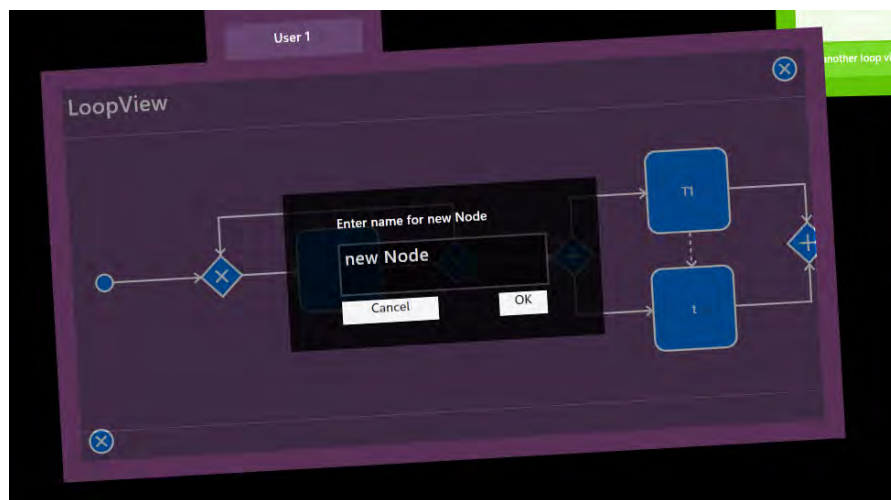


Figure 6.19.: Modal Dialogue for Inserting a New Node

The dialogue consists of a heading, a text field, and two buttons. If the user taps on the text field a keyboard appears (cf. Figure 6.20). Tapping on the “Ok” button confirms the entered text. Tapping “Cancel” restores the state before opening the modal dialogue.



Figure 6.20.: Modal Dialogue with Keyboard

6.5. Live Updates

The RESTLibrary informs all registered listeners about changes in their process model. Therefore, all opened process models are registered (cf. Listing A.7). If the library registers a change in collabView all associated and opened views and the superior CPM are informed, therefore two methods have to be implemented (cf. Listing A.8)). If parallel changes in another client of proView are made, the listeners are not informed by the RESTLibrary.

6.6. Discussion

We established some requirements in Section 3. In this section we discuss if collabView meets those requirements.

- MUR1 ✓ (Avoid Distract UI Elements): Closing buttons are the only UI widgets that are permanently visible. They are at the edges of the process field and do not distract from the actual task.
- MUR2 ✓ (Limit Desktop-style Menus): The title menu of a process model can be seen as a desktop menu. However, the title is needed to distinguish the different process areas. We therefore gave the title a functionality.

6. Proof-Of-Concept Implementation

- MUR3 ✓ (Minimalistic Process Model Representation): Only basic geometrical forms are used to create the process model.
- MUR4 ✓ (Fixed Process Model Layout): No parts of the process model are movable. They are all arranged in a fixed grid. The only lines that not fit in a grid are the SynchronisationEdges.
- MUR5 ✓ (Hide unnecessary UI Widgets): Buttons in the title menu and on the nodes are not visible as long as the user does not tap the items.
- CWR1 ✓ (Interpersonal Interaction): Communication and gesturing are not affected. It is possible for each user to work for himself because they can have their own process areas. However, space on the multi-touch table is limited. Therefore they will probably work on one process area.
- CWR2 ✓ (Transitions between Activities): Only fingers are possible to use as input devices.
- CWR3 (✓) (Personal and Collaboration space): We only support personal workspace. The group is forced to work on one of the personal workspaces. Therefore, one person leads the group and is responsible that the task is fulfilled.
- CWR4 ✓ (Multi-Touch Table Collaboration and External Work): The created views and changed process models are also changed on the proViewServer and therefore on all other implementations of the proView framework.
- CWR5 (✓) (Physical and Digital Objects): We do not support the use of physical objects. The process modelling is done only in a digital form.
- CWR6 X (Shard Access to Physical and Digital Objects): As orientation and thus to increase readability we decided to give each user their own main menu and process area. They can arrange them in the best way so they can read the texts easily.
- CWR7 ✓ (Appropriate Arrangement of Users): Users can arrange themselves round the table as they want. Newly opened process areas open in the angle of and near the corresponding main menu. Furthermore, each UI element is rotatable.

- CWR8 ✓ (Simultaneous User Action): Users can simultaneously interact with their process areas and main menus. Only trying to insert nodes at the same time on the same positions could lead to errors on the proViewServer. Further, only one keyboard can be used at a time.
- MMR1 ✓ (Ownership of Main Menu): Each user has an assigned colour. This colour is reflected in their own main menu.
- MMR2 ✓ (Fast Capturing of Content): The main menu is designed in a reduced way. Information in the submenu can be reached within seconds of scrolling.
- MMR3 ✓ (Variable Amount of Entries in Submenus): The submenu can contain a variable amounts of entries.
- MMR4 X (Personalised Registration): We do not support personal registration.
- PVR1 ✓ (Presentation of Process Model): Process Models are presented in the process areas.
- PVR2 ✓ (Generation of Process Views): Views can be derived from CPMs by tapping first on the title and then on the button with the eye.
- PVR3 ✓ (Personalised Process Views): Views can be personalised by reducing nodes and aggregating neighbouring nodes together.
- PVR4 ✓ (Deleting of Process Views): Views can be deleted by tapping first on the title and then on the bin.
- PVR5 ✓ (CPM and Associated Process Views): No changes can be made directly in a CPM, only in their associated process views.
- PVR6 (✓) (Update CPM and Associated Process Views): If a change is made the proViewServer informs the CPM and associated process views. Changes from outside collabView are not updated in real-time.
- MVR1 ✓ (Insert Process Elements): Nodes can be insert through an insertion menu.
- MVR2 ✓ (Delete Process Elements): Nodes can be deleted by the button with the X which is called by tapping on the node.
- MVR3 ✓ (Aggregate Process Elements): Nodes can be aggregated by tapping one node with one finger and selecting other nodes to it with another finger.

6. Proof-Of-Concept Implementation

- MVR4 ✓ (Reduce Process Elements): Nodes can be reduced by the button with the bin which is called by tapping on the node.
- MVR5 ✓ (Rename Process Elements): Nodes can be renamed by tapping on the text which is made editable by first tapping on the node.
- MVR6 ✓ (Change Process Elements): Conditions for loops can be changed by tapping on the text of the condition which is shifted by first tapping on the node.
- MVR7 ✓ (Insert Control Flow Edges): New edges can be inserted by drawing a line with a finger between the nodes where a new edge is wanted.

MVR1 (Insert Process Elements), MVR5 (Rename Process Elements), MVR6 (Change Process Elements), and MVR7 (Insert Control Flow Edges) are currently not fully supported by the proViewServer.

7

Conclusion

We have created a concept and a proof-of-concept implementation of a collaborative process modelling tool on the multi-touch table Samsung SUR40 with PixelSense. This tool, called collabView, is part of the proView business process framework. collabView shows that it is possible to create an effective tool on multi-touch tables to work collaboratively on process models. The collabView tool offers a more productive and more communicative way to work together on one process model than on a shared screen on a conventional computer.

collabView already supports a large segment of the functionalities of the proView framework. The user interface, e.g., scrolling and creating a new node, is strictly designed with no double meanings of touch gestures. collabView gives real-time visual feedback for inserting new edges and on aggregating neighbouring nodes. A clear colour-concept and smart auto-positioning of process areas and menus simplify collaboration of multiple

7. Conclusion

users. collabView has an intelligent use of context menus which therefore optimally uses space with a minimalistic interface design.

In the future some things need to be done on the side of collabView as well as on the proViewServer side.

On the collabView side a next step would be the implementation of a personalised login. So the user only sees the CPMs and their personal views in the menu. Further, "Staff Assignments", where identifications of the user who is responsible for this node could be added to the nodes the user inserts.

Another functionality needs to be added, the data elements describing the data flow. They are not implemented yet on our prototype. If they will be added, new considerations about using and segmenting space need to be made. Further the design needs to be considered. How can the data elements fit into the design and suggest the possible or impossible functionalities they have compared to nodes and in respect of touch gestures.

To distinguish between CPMs and proces views there needs to be thought of a sign or hint, as well as on the main menu as on the process area. At the moment, the user needs to know the names of the CPM and the process views. This would reduce the cognitive load of the users.

Another function that needs to be considered is a scaling or zooming function for the process model itself. Now the user can scale the process area as a whole and scroll to the clipping in the process model they want. Maybe it is useful to scale the process model down to get an overview.

Some functionalities which are essential for modelling all the functionalities in a process model work on collabView, but are not supported from the proViewServer yet. For instance, insertion of an XOR branching block, the insertion of a LoopEdge, the insertion of another ControlEdge between split nodes and the renaming of nodes.

Renaming of CPMs and process views is neither supported in collabView nor on the proView server yet.

The limitation of a single keyboard provided by the Surface 2.0 SDK could be replaced with an own design of a keyboard in a productive system. With multiple keyboards more

space is needed but that might improve the workflow of the modelling of the process model.

For the future the prototype needs to be tested by expert users with special focus on usability and suitability of the gestures and buttons.

A

Source Code

Some important code fragments are listed here:

Listing A.1: OnPreviewTouchMove Event Handler

```
1 OnPreviewTouchMove() {  
2     TouchPoint tp = GetTouchPoint(Canvas);  
3     AddLineFromTo(canvas, lastPosition, tp);  
4 }
```

Listing A.2: Method to Draw a Line

```
1 AddLineFromTo(Canvas canvas, Point lastPosition, Point tp){  
2     Line l = new Line  
3         {  
4             X1 = LastPosition.X;
```

A. Source Code

```
5         Y1 = LastPosition.Y;
6         X2 = tp.X;
7         Y2 = tp.Y;
8     };
9     canvas.Add(1);
10 }
```

Listing A.3: OnTouchLeave Event Handler

```
1 OnTouchLeave() {
2     touchUp = GetTouchPoint(Canvas);
3     canvas.Remove();
4     CreateNewEdge(touchUp, canvas)
5 }
```

Listing A.4: Create a new Edge

```
1 CreateNewEdge(Point touchUp, Canvas canvas) {
2     if(touchUp on node1 & _global.touchDown on node2) {
3         Rest.InsertSynchEdge(node1, node2);
4     }
5     else if(touchUp on split & _global.touchDown on join) {
6         Rest.InsertControlEdge(split, join);
7     }
8     else if(touchUp on startLoop &
9         _global.touchDown on endLoop) {
10        Rest.InsertLoopEdge(startLoop, endLoop);
11    }
12    else
13        canvas.Remove();
14 }
```

Listing A.5: PreviewTouchDown Event Handler of a Node

```
1 Node_PreiewTouchDown() {
2     if(FirstTouchId == null) {
3         FirstTouchId = this.TouchDevice.Id;
4     }
5     touchedObjects.Add(this.TouchDevice.Id, this)
6 }
```

Listing A.6: PreviewTouchUp Event Handler of a Node

```
1 Node_PreiewTouchUp() {
2     if(FirstTouchId == this.TouchDevice.Id) {
3         String name = NameOfAbstractNode();
4         Rest.AggregateNodes(touchedObjects, name);
5         FirstTouchId = null;
6         touchedObjects.Clear();
7     }
8     else if(touchedObjects.count == 1) {
9         FirstTouchId = null;
10        touchedObjects.Remove(this.TouchDevice.Id);
11        ShowEditWidgets();
12    }
13    else if(touchedObjects.count < 1) {
14        touchedObjects.Add(this.TouchDevice.Id, this)
15        DarkenColor();
16    }
17 }
```

A. Source Code

Listing A.7: Subscribing as Observer for Events on Initialising Process Area

```
1 InitProcessArea() {
2     Rest r = Rest.Instance;
3     //subscribe as observer for events
4     r.Subscribe(this);
5     ...
6 }
```

Listing A.8: Observer Methods Notified from RESTLibrary

```
1 OnNext(Event value) {
2     RebuildProcessModel();
3 }
4
5 OnError(Exception error) {
6     _serverNotification.Text = error.ToString();
7     RebuildProcessModel();
8 }
```


List of Figures

2.1. Graphical Representation of an Activity with StartEvent, EndEvent and ControlEdge	4
2.2. Graphical Representation of an AND Branching Block (a) and an XOR Branching Block (b)	5
2.3. Graphical Representation of a LoopEdge	5
2.4. Graphical Representation of a SynchronisationEdge	6
2.5. Possible Process Model	6
2.6. Impossible Process Model	6
2.7. CPM with two Associated Views and Operations	8
2.8. CPM with its Dependent Process Views (a), Change Operation on Process View 2, Update of the Associated CPM, and Update of the Dependent Process Views	9
2.9. Architecture of the PixelSense Technology	11
4.1. Pie Menu with (a) Bubble-styled and (b) Beagle Cutout-styled Menu Items	21
4.2. Sticky Note Menu with (a) Menu Items and (b) Submenu Items	22
4.3. Element Menu	22
4.4. Rectangle Menu with ID token	23
4.5. (a) Rectangle Menu and (b) with Sumenu	24
4.6. Process Area with Process Model	26
4.7. Node after Tapping	32
4.8. Title of a (a)CPM and (b) Process View with Buttons	33
4.9. Insert Menu Column	34

List of Figures

4.10. Insert Menu Shaped like an Hourglass	35
4.11. Insert Menu Horizontal	35
4.12. A Modal Dialogue	36
5.1. Schematic Representation of Boxes, Paths, PathElements and Activities in a Process Model	38
5.2. Schematic Definitions of Box, Path, and Path Element	38
5.3. Relative Origin of the Internal Coordinate System	41
5.4. Example how to Calculate Absolute Position	43
6.1. Start Screen	47
6.2. Main Menu with Play Field and Content Field	48
6.3. Main Menu Animation After Tapping on Process Survey	49
6.4. Main Menu showing CPMs an Process Views	49
6.5. Process Area	51
6.6. Node with a Shortened Name	51
6.7. Touch Regions for Insertion Menu (blue region), Manipulation of Nodes (green regions), Drawing Edges (red regions), and Scrolling (yellow region without containing regions) in a Process Area	52
6.8. Insert Node with Insertion Menu	53
6.9. Example of a Process Model	54
6.10. Moving Process Area	55
6.11. Rotating Process Area	55
6.12. Scalling Process Area	56
6.13. Scrolling Process Area	57
6.14. Insert new Edge	57
6.15. Aggregate Nodes	58
6.16. Closing of Process Area	58
6.17. (a) Activity and (b) Split with Buttons	59
6.18. Renaming an Activity with Keyboard	59
6.19. Modal Dialogue for Inserting a New Node	60
6.20. Modal Dialogue with Keyboard	61

List of Tables

2.1. Update Operations for Process Views with Parameters [KKR12] 10

Listings

- A.1. OnPreviewTouchMove Event Handler 69
- A.2. Method to Draw a Line 69
- A.3. OnTouchLeave Event Handler 70
- A.4. Create a new Edge 70
- A.5. PreviewTouchDown Event Handler of a Node 71
- A.6. PreviewTouchUp Event Handler of a Node 71
- A.7. Subscribing as Observer for Events on Initialising Process Area 72
- A.8. Observer Methods Notified from RESTLibrary 72

Bibliography

- [BLS⁺09] BRANDL, Peter ; LEITNER, Jakob ; SEIFRIED, Thomas ; HALLER, Michael ; DORAY, Bernard ; TO, Paul: Occlusion-Aware Menu Design for Digital Tabletops. In: *CHI '09 Extended Abstracts on Human Factors in Computing Systems*. New York, NY, USA : ACM, 2009 (CHI EA '09), pp. 3223–3228
- [BWB06] BENKO, Hrvoje ; WILSON, Andrew D. ; BAUDISCH, Patrick: Precise Selection Techniques for Multi-Touch Screens. In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. New York, NY, USA : ACM, 2006 (CHI '06), pp. 1263–1272
- [EBBDL09] ECKER, Ronald ; BROY, Verena ; BUTZ, Andreas ; DE LUCA, Alexander: pieTouch: a Direct Touch Gesture Interface for Interacting with In-Vehicle Information Systems. In: *Proceedings of the 11th International Conference on Human-Computer Interaction with Mobile Devices and Services*. New York, NY, USA : ACM, 2009 (MobileHCI '09), pp. 22:1–22:10
- [FHD09] FRISCH, Mathias ; HEYDEKORN, Jens ; DACHSELT, Raimund: Investigating Multi-Touch and Pen Gestures for Diagram Editing on Interactive Surfaces. In: *Proceedings of the ACM International Conference on Interactive Tabletops and Surfaces*. New York, NY, USA : ACM, 2009 (ITS '09), pp. 149–156
- [FHD10] FRISCH, Mathias ; HEYDEKORN, Jens ; DACHSELT, Raimund: Diagram Editing on Interactive Displays using Multi-Touch and Pen Gestures. In: *Proceedings of the 6th International Conference on Diagrammatic Representation and Inference*. Berlin, Heidelberg : Springer-Verlag, 2010 (Diagrams'10), pp. 182–196

Bibliography

- [Hes13] HESS, Hayato: *Hand Gesture-based Process Modeling for Updatable Processes*. 2013. – Bachelor Thesis, Ulm University
- [HFS09] HESSELMANN, Tobias ; FLÖRING, Stefan ; SCHMITT, Marwin: Stacked Half-Pie Menus: Navigating Nested Menus on Interactive Tabletops. In: *Proceedings of the ACM International Conference on Interactive Tabletops and Surfaces*. New York, NY, USA : ACM, 2009 (ITS '09), pp. 173–180
- [KKR12] KOLB, Jens ; KAMMERER, Klaus ; REICHERT, Manfred: Updatable Process Views for User-Centered Adaption of Large Process Models. In: *Proceedings of the 10th International Conference on Service-Oriented Computing*. Berlin, Heidelberg : Springer-Verlag, 2012 (ICSOC'12), pp. 484–498
- [KR13a] KOLB, Jens ; REICHERT, Manfred: Data Flow Abstractions and Adaptations through Updatable Process Views. In: *28th Symposium on Applied Computing (SAC'13), 10th Enterprise Engineering Track (EE'13)*, ACM Press, March 2013, pp. 1447–1453
- [KR13b] KOLB, Jens ; REICHERT, Manfred: A Flexible Approach for Abstracting and Personalizing Large Business Process Models. In: *Applied Computing Review* 13 (2013), March, No. 1, pp. 6–17
- [KRR12] KOLB, Jens ; RUDNER, Benjamin ; REICHERT, Manfred: Towards Gesture-Based Process Modeling on Multi-Touch Devices. In: *1st Int'l Workshop on Human-Centric Process-Aware Information Systems (HC-PAIS'12)*, Springer, June 2012 (LNBIP 112), pp. 280–293
- [Mic11] *Microsoft Surface 2.0 Design and Interaction Guide Principles and Guidelines for Designing and Developing Surface Applications*. 2011
- [Mic13a] *Developing Presentation Layer Surface Applications*. <http://msdn.microsoft.com/en-us/library/ff727733.aspx>. Version: 17th July 2013
- [Mic13b] *WPF Graphics Rendering Overview*. <http://msdn.microsoft.com/en-us/library/ms748373.aspx>. Version: 19th July 2013

- [Mic13c] *The Power of PixelSense*. <http://www.microsoft.com/en-us/pixelsense/pixelsense.aspx>. Version: 25th July 2013
- [Mic13d] *ElementMenu Control*. <http://msdn.microsoft.com/en-us/library/ff727728.aspx>. Version: 26th July 2013
- [Mic13e] *On-Screen Keyboard and Numeric Keypad*. <http://msdn.microsoft.com/en-us/library/ff727766.aspx>. Version: 29th July 2013
- [MKG02] MARK, G ; KOBASA, A ; GONZALEZ, V: Do Four Eyes See Better than Two? Collaborative Versus Individual Discovery in Data Visualization Systems. In: *Proceedings. 6th International Conference on Information Visualisation, 2002.* , IEEE Comput. Soc, 2002, S. 249–255
- [OMG11] *Business Process Model and Notation (BPMN) Version 2.0*. <http://www.omg.org/spec/BPMN/2.0/>. Version: January 2011
- [Rei00] REICHERT, Manfred ; DADAM, Peter (Ed.): *Dynamische Ablaufänderungen in Workflow-Management-Systemen*. July 2000. – PhD Thesis, Ulm University
- [Sam13] *Samsung SUR40 with Microsoft PixelSense*. <http://www.samsung.com/us/business/displays/digital-signage/LH40SFWTGC/ZA>. Version: 17th July 2013
- [SGM03] SCOTT, Stacey D. ; GRANT, Karen D. ; MANDRYK, Regan L.: System Guidelines for Co-Located, Collaborative Work on a Tabletop Display. In: *Proceedings of the 8th Conference on European Conference on Computer Supported Cooperative Work*. Norwell, MA, USA : Kluwer Academic Publishers, 2003 (ECSCW'03), pp. 159–178
- [Wit12] WITTERN, Hauke: Empirical Study Evaluating Business Process Modeling on Multi-Touch Devices. In: *Proceedings of the 2012 IEEE International Conference on Software Science, Technology and Engineering*. Washington, DC, USA : IEEE Computer Society, 2012 (SWSTE '12), pp. 20–29
- [WMW09] WOBROCK, Jacob O. ; MORRIS, Meredith R. ; WILSON, Andrew D.: User-Defined Gestures for Surface Computing, In: *Proceedings of the 27th*

Bibliography

International Conference o Human Factor in Computer Systems CHI 09.
ACM Press, 2009 (CHI '09), pp. 1083–1092

- [WPTR13] WEBER, Barbara ; PINGGERA, Jakob ; TORRES, Victoria ; REICHERT, Manfred: Change Patterns in Use: A Critical Evaluation. In: *Proceedings of the 14th International Conference of BPMDS 2013*, Springer, June 2013 (LNBIP 147), pp. 261–276

Name: Judith Burkhardt

Matrikelnummer: 623583

Erklärung

Ich erkläre, dass ich die Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Ulm, den

Judith Burkhardt