

Evaluating Process Modeling Capabilities of the XVSM Micro-Room Framework

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Business Informatics

eingereicht von

Johann Binder

Matrikelnummer 0727950

an der
Fakultät für Informatik der Technischen Universität Wien

Betreuung: Ao. Univ.-Prof. Dipl.-Ing. Dr. Eva Kühn
Mitwirkung: Projektass. Dipl.-Ing. Stefan Craß

Wien, 19. April 2017

(Unterschrift Verfasser)

(Unterschrift Betreuung)

Evaluating Process Modeling Capabilities of the XVSM Micro-Room Framework

MASTER'S THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Business Informatics

by

Johann Binder

Registration Number 0727950

to the Faculty of Informatics
at the Vienna University of Technology

Advisor: Ao. Univ.-Prof. Dipl.-Ing. Dr. Eva Kühn
Assistance: Projektass. Dipl.-Ing. Stefan Craß

Vienna, 19. April 2017

(Signature of Author)

(Signature of Advisor)

Erklärung zur Verfassung der Arbeit

Johann Binder
Großmotten 5, 3542 Gföhl

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit - einschließlich Tabellen, Karten und Abbildungen -, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

(Ort, Datum)

(Unterschrift Verfasser)

Acknowledgements

I would like to thank several persons for their support while creating this work. Special credit belongs to Eva Kühn and Stefan Craß for their useful and constructive recommendations during our regular meetings. I am also particularly grateful for Evelyn Binder proof-reading the thesis. Finally, I would like to thank my family for their support during the work.

Abstract

Nowadays, several modeling tools exist for creating business processes in enterprises. Most of them can be used by IT experts to semi-automatically create corresponding software artifacts of the given process. However, as they comprise a lot of complexity, none of these tools is usable by ordinary end users.

When searching for related work targeting process modeling tools for end users, only very few approaches can be found. An evaluation of the existing papers shows that they all have several drawbacks.

To overcome this gap, we have designed and implemented a modeling tool that is usable by end users to model their daily life workflows such as chatting or exchanging photos with a friend. Afterwards, a fully executable application can be created from the model automatically that can be downloaded and shared with friends.

The abstraction of the modeling tool is more comprehensible than those of other modelers as it uses the micro-room concept as a basis. It allows end users to think about their daily life workflows in terms of persons performing specific actions in several rooms they can visit in a predefined order. Additionally, the user interface of the modeling tool is kept as simple as possible to provide proper usability.

After presenting all details of the modeling tool's architecture, implementation and its deployment, the modeling tool itself as well as the underlying micro-room concept have been evaluated. Therefore, usability testing has been performed with the developed modeling tool and a competing modeling tool.

The evaluation shows that the micro-room concept indeed uses a more comprehensible abstraction if compared to other modeling languages. It is well-suited for modeling simple collaborative tasks, whereas other approaches are better if very complex processes have to be modeled or software should be specified formally. Also, the developed modeling tool is far more intuitive than the competing ones due to the simplifications allowed by using the micro-room concept. Finally, five key factors for creating an intuitive modeling tool in general have been derived, i.e. required functionality, undoability, system stability, easy learnability and understandability.

Kurzfassung

Heutzutage gibt es viele Modellierungs-Tools, mit denen Geschäftsprozesse in Unternehmen erstellt werden können. Die meisten davon werden von IT-Experten verwendet, um semi-automatisch die zugehörigen Software-Artefakte zu erstellen. Da diese Modellierungs-Tools allerdings viel Komplexität beinhalten, sind sie von einem gewöhnlichen Endanwender nicht benutzbar.

Leider existieren auch nur sehr wenige wissenschaftliche Beiträge zu anderen Modellierungs-Tools, die explizit für Endanwender gedacht sind. Bei einer näheren Evaluierung dieser Beiträge stellt sich außerdem heraus, dass sie alle einige Nachteile haben.

Um diese Lücke zu schließen, wurde in dieser Arbeit ein neues Modellierungs-Tool geplant und implementiert, das von Endanwendern verwendet werden kann, um ihre alltäglichen Arbeitsabläufe zu modellieren. Anschließend kann aus dem erstellten Modell mit nur einem einzigen Klick eine voll funktionsfähige Anwendung erstellt werden, die mit Freunden geteilt und heruntergeladen werden kann.

Damit die Abstraktion des neu entwickelten Modellierungs-Tools im Vergleich zu anderen Modellierungs-Tools besser verständlich wird, wurde das Micro-Room-Konzept als Grundlage verwendet. Dadurch können Endanwender ihre täglichen Arbeitsabläufe direkt in Räumen modellieren, in denen bestimmte Personen vordefinierte Tätigkeiten durchführen können und die in einer bestimmten Reihenfolge besuchbar sind. Zusätzlich wurde die Benutzeroberfläche des Modellierungs-Tools so einfach wie möglich gehalten, um eine möglichst gute Benutzbarkeit zu gewährleisten.

Nach der Beschreibung aller Details zur Architektur, Implementierung und der Verwendung des Modellierungs-Tools, wurde es selbst sowie das zugrundeliegende Micro-Room-Konzept evaluiert. Dazu wurden Usability-Tests mit dem entwickelten und einem vergleichbaren Modellierungs-Tool durchgeführt.

Die Auswertung der Usability-Tests zeigt, dass die Abstraktion des Micro-Room-Konzepts im Vergleich zu anderen Modellierungssprachen besser verständlich ist. Es ist sehr gut geeignet, um einfache kollaborative Arbeitsabläufe zu modellieren, wohingegen andere Modellierungssprachen besser geeignet sind um komplexe Prozesse oder formale Softwarespezifikationen zu modellieren. Das entwickelte Modellierungs-Tool ist aufgrund der Vereinfachungen, die durch die Verwendung des Micro-Room-Konzepts ermöglicht wurden, wesentlich intuitiver als das vergleichbare, ebenfalls getestete Modellierungs-Tool. Zur Erstellung eines intuitiven Modellierungs-Tools wurden folgende fünf Schlüsselfaktoren aus den Usability-Tests abgeleitet: Benötigte Funktionalität, Rückgängig-Machbarkeit, Systemstabilität, leichte Erlernbarkeit und Verständlichkeit.

Contents

1	Introduction	1
1.1	Problem Description & Motivation	1
1.2	Aim of the Work	2
1.3	Methodology	3
1.4	Structure of the Thesis	3
2	Related Work	4
2.1	Popular Process Modeling Languages	4
2.2	End-User-related Approaches	16
2.3	Comparison	27
3	Approach	32
3.1	Related Concepts	32
3.2	Initial Draft	42
4	Design	44
4.1	Evaluation of Technologies	44
4.2	Components	49
4.3	Modeling	51
4.4	Provisioning	57
4.5	Workflow Execution	58
5	Implementation	60
5.1	Structural View	60
5.2	XVSM Micro-Room Framework Adjustments	63
5.3	Modules	65
5.4	Provisioning	69
5.5	Further Problems & Solutions	77
6	User Guide	81
6.1	Deploying the XVSM Micro-Room Modeler	81
6.2	Extending the XVSM Micro-Room Modeler	81
6.3	Using the XVSM Micro-Room Modeler	83
6.4	Using applications created by the XVSM Micro-Room Modeler	86

7	Evaluation	89
7.1	Theoretical Evaluation of the XVSM Micro-Room Modeler	89
7.2	Usability Evaluation Approach	92
7.3	Usability Evaluation Results	97
8	Future Work	109
8.1	Realization with the Peer Model	109
8.2	UI Improvements to the XVSM Micro-Room Modeler	110
8.3	Port Forwarding	111
8.4	Download JRE if not Installed	111
8.5	Publish Platform for Custom Modules	111
8.6	UI Templates	111
8.7	Further Modules	112
9	Conclusion	113
	References	114
	Web References	120
	Appendix	I

List of Figures

2.1	Example of an UML Activity Diagram	7
2.2	Acceleo Eclipse Plugin	8
2.3	Sparx Systems Enterprise Architect	9
2.4	Precise Operational Style - Behavior View (UML Activity Diagram)	10
2.5	Precise Operational Style - Static View (UML Class Diagram)	11
2.6	Example of a BPD	12
2.7	Signavio Process Editor	12
2.8	Activiti Designer	13
2.9	Activiti-generated UI for a Human Task	14
2.10	Example of a WS-BPEL Model	15
2.11	NetBeans BPEL Designer	16
2.12	The ISEA Method	17
2.13	Organizational Perspective in ISEAsy	18
2.14	Story Creation in BPMerlin	21
2.15	UI of the Wizard in SOA4All-Composer	22
2.16	Form-based Web Service Composition - Example Process	23
2.17	Form-based Web Service Composition - Data Mapping	24
2.18	Web Service Composition Framework - Architecture	25
2.19	Collaborative Task Manager	26
2.20	Comparison of Modeling Techniques	27
3.1	XVSM P2P Communication	33
3.2	XVSM Container	33
3.3	XVSM Aspects	34
3.4	Peer Model Example	35
3.5	Micro-Room Example	36
3.6	XVSM Micro-Room Framework Component View	38
3.7	Archive Micro-Room Modeled with the Peer Model	39
3.8	Comparison of Modeling Techniques including PM and XMRF	42
3.9	XVSM Micro-Room Modeler Draft	43
4.1	GMF Overview	45
4.2	GMF Modeler Example	46
4.3	Visio with Custom Shape Library	47

4.4	Comparison of Technologies	48
4.5	Component Diagram of the XVSM Micro-Room Modeler	50
4.6	Graphical Notation of the Provided Modules	53
4.7	Sequence Diagram of Downloading and Sharing the Modeled Application	56
4.8	Provisioning in the XVSM Micro-Room Modeler	57
4.9	Draft of an Executable Application Created by the XVSM Micro-Room Modeler	58
5.1	Class Diagram of the XVSM Micro-Room Modeler	62
5.2	Content of the Distribution Folder in the XVSM Micro-Room Modeler	70
5.3	A Smart Link being dragged in the XVSM Micro-Room Modeler	79
6.1	Creating a new Project in the XVSM Micro-Room Modeler	83
6.2	Configuring Groups and Members in the XVSM Micro-Room Modeler	84
6.3	Creating a Model in the XVSM Micro-Room Modeler	84
6.4	Creating an Invitation Link in the XVSM Micro-Room Modeler	85
6.5	Logging into the Created Application	86
6.6	Using a Micro-Room in the Created Application	87
7.1	Comparison of Modeling Techniques including the XVSM Micro-Room Modeler	90
7.2	Approach of Usability Evaluation of Business Process Modeling Tools	93
7.3	Proportion of Usability Problems found depending on the Number of Users	93
7.4	Example Item in IsoMetrics ^S	96
7.5	Example Item in IsoMetrics ^L	97
7.6	Accumulated Average Score for S.12, L.1, L.4, L.5, L.6	98
7.7	Average Time for all Tasks in Minutes	99
7.8	Average Time / Task	99
7.9	Use Cases for the Micro-Room Concept and their Naming Count	101
7.10	Preferred Usage per Tool (EQ 3)	102
7.11	Total IsoMetrics Score per Tool	103
7.12	Average IsoMetrics Score per Principle	103
7.13	Total Number of Questions Asked	104
7.14	Reasons to use the XVSM Micro-Room Modeler and their Naming Count	105
7.15	Relations between Key Factors for the XVSM Micro-Room Modeler's Intuitiveness	106
7.16	Evaluated Tool Performance in each of the Key Factors	107

List of Tables

2.1	Humanistic Business Process Modeling vs. Mechanistic Business Process Modeling	19
5.1	XVSM Micro-Room Modeler Package Structure	61
5.2	XVSM Micro-Room Modeler Resource Folder Structure	61
5.3	XVSM Micro-Room Modeler Containers	63
6.1	Global JavaScript Variables Provided by the XVSM Micro-Room Modeler	82
6.2	JavaScript Helper Functions Provided by the XVSM Micro-Room Modeler	83
7.1	IsoMetrics ^S Items Related to Abstraction of the Underlying Concept	98
7.2	Most Important IsoMetrics ^S Items Across all Test Subjects	106
7.3	Key Factors for an Intuitive Modeling Tool Targeting End Users	107

List of Listings

2.1	Web Service Composition Framework - Control Rule	25
5.1	Example of new Micro-Room Annotations in the XVSM Micro-Room Framework	64
5.2	Example of new Plugin Annotations in the XVSM Micro-Room Framework	64
5.3	Example Manifest of a Module JAR File	65
5.4	JSON Structure of a parsed XVSM Micro-Room Framework Module	66
5.5	Example UI for a Module of the XVSM Micro-Room Framework	67
5.6	Content of start.bat	70
5.7	JSON Structure of an XVSM Micro-Room Modeler Project Object	71
5.8	JSON Structure of an XVSM Micro-Room Modeler Model Object	72
5.9	JSON Structure of an XVSM Micro-Room Modeler Micro-Room Object	73
5.10	JSON Structure of an XVSM Micro-Room Modeler Plugin Object	74
5.11	Example of a XVSM Micro-Room Framework Business Logic File	75
6.1	Deployment of the XVSM Micro-Room Modeler	81

List of Abbreviations

AJAX	Asynchronous JavaScript and XML
API	Application Programming Interface
B2B	Business-to-Business
BPD	Business Process Diagram
BPM	Business Process Management
BPMN	Business Process Model and Notation
CRUD	Create, Read, Update, Delete
CSS	Cascading Style Sheets
DEST	Destination
DOM	Document Object Model
DSL	Domain-specific Language
ebXML	Electronic Business using Extensible Markup Language
EMF	Eclipse Modeling Framework
EQ	Evaluation Question
EUPM	End-User-driven Process Modeling
FID	Flow Identifier
FWSC	Form-based Web Service Composition
GMF	Graphical Modeling Framework
HBPM	Humanistic Business Process Modeling
HTML	Hypertext Markup Language
HTTPS	Hypertext Transfer Protocol Secure
IDE	Integrated Development Environment
JAR	Java Archive
JAXB	Java Architecture for XML Binding
JRE	Java Runtime Environment
LDAP	Lightweight Directory Access Protocol
LPML	Lightweight Process Modeling Language
MBPM	Mechanistic Business Process Modeling
MVC	Model View Controller
OASIS	Organization for the Advancement of Structured Information Standards
OMG	Object Management Group
P2P	Peer-to-Peer
PC	Personal Computer
PIC	Peer-In-Container

PM	Peer Model
POC	Peer-Out-Container
POJO	Plain Old Java Object
REST	Representational State Transfer
RQ	Research Question
T	Task
TTL	Time-to-Live
TTS	Time-to-Start
UI	User Interface
UML	Unified Modeling Language
UPD	User Datagram Protocol
UPnP	Universal Plug and Play
URI	Uniform Resource Identifier
WPM	Wizard-based Process Modeling
WS-BPEL	Web Services - Business Process Execution Language
WS-CDL	Web Services - Choreography Description Language
WSCF	Web Service Composition Framework
WSDL	Web Services Description Language
WYSIWYG	What You See Is What You Get
XML	Extensible Markup Language
XMRF	XVSM Micro-Room Framework
XMRM	XVSM Micro-Room Modeler
XVSM	Extensible Virtual Shared Memory
XVSMP	XVSM Protocol

Introduction

In this chapter we explain our motivation and the problem. Additionally, we outline the expected results and describe the overall structure of the work.

1.1 Problem Description & Motivation

Today, processes in enterprises (e.g. purchasing, logistics or internal team coordination) are heavily supported by IT in order to increase automation and hence decrease labor costs. Therefore, they are modeled with special modeling techniques such as *Business Process Model and Notation* (BPMN) [DDD10] or *Web Services - Business Process Execution Language* (WS-BPEL) [AAA⁺07].

One problem of these modeling techniques is that they require a certain level of IT knowledge [LV10] and use an abstraction (i.e. modeling concepts and terminology for modeling elements) that is hard to understand. However, as the affected users are the only ones that really know their processes, they should be modeled by themselves in the first place and not by the IT department. Since many end users usually do not have these IT-focused skills, it is likely that they model their processes in a wrong way, hence creating unsatisfying solutions.

A second big problem of BPMN and BPEL is their insufficient tool support. Tools like the NetBeans BPEL Designer¹ or the BPMN platform Activiti² are highly complex to use as they offer a lot of functionality applying technical terms. That is why today often end users design the process with a simpler modeler at first, having no automation capabilities (e.g. Signavio Process Editor³ or Microsoft Visio⁴). Afterwards IT experts need to model the whole process again with the mentioned tools to enable automation support. [ODA⁺09]

Since the author of the thesis works with such modeling tools himself, it is a personal concern to overcome both of these problems.

¹<http://soa.netbeans.org/soa/>

²<http://activiti.org>

³<http://www.signavio.com>

⁴<http://office.microsoft.com/de-at/visio/>

1.2 Aim of the Work

As a consequence another way of modeling processes is developed in our contribution.

The first result is a self-developed graphical modeling tool. This tool uses a different kind of abstraction, i.e. processes are modeled via so-called “micro-rooms”. A micro-room can be seen as a room in which persons can perform predefined actions and that can be connected to other micro-rooms. Also, the tool provides full automation support, i.e. it derives the whole business logic and data layer from the model automatically. Additionally, a high focus is on usability. Summarizing, the tool tries to overcome the two problems stated in the problem description.

As this would be a rather complex task, the *Extensible Virtual Shared Memory (XVSM) Micro-Room Framework* [Bin13] is used as a basis for the modeling tool, which itself is related to the Peer Model [KCJ⁺13]. In the XVSM Micro-Room Framework, functionality is encapsulated within reusable self-definable rooms that can be connected with each other. Furthermore, it already supports the business logic and data layer code generation based on such micro-room models, including boilerplate code needed for security, privacy and replication. The framework uses XVSM to handle these tasks, which is a shared memory usable in a *Peer-to-Peer (P2P)*-based manner. Thus, by using the XVSM Micro-Room Framework, the actual workload for developing the modeling tool is significantly reduced.

The Peer Model on the other hand is a high-level programming model that allows developers to model the behavior, data and control flow of their components instead of implementing it. It targets highly concurrent and distributed environments across all domains. The XVSM Micro-Room Framework uses similar concepts like this programming model but with reduced features and complexity. E.g. it allows only one-to-one connections between rooms and rooms cannot be reassembled by other rooms, which would be possible with the Peer Model. The XVSM Micro-Room Framework is used in this work because its limited and more simple feature set is more suitable for a modeling tool targeting end users.

However, the XVSM Micro-Room Framework requires a model that is written in *Extensible Markup Language (XML)* as input. As a first result of our work, the developed modeling tool allows the graphical model to be transformed into the desired XML format directly. It is then packaged with the XVSM Micro-Room Framework to create the executable application.

Additionally, the micro-rooms usable by the XVSM Micro-Room Framework only specify logic that is executed on the backend. A *User Interface (UI)* has to be created by the user him-/herself. Thereby, the second result of this work is to allow micro-room creators to extend micro-rooms with UIs in the first place, so that the end users do not need to do this later on.

The third result is an evaluation of the stated modeling tool and the used micro-room concept itself. This is done via questionnaires and usability testing.

At the end, the following *Research Questions (RQ)* shall be answered:

- RQ 1: How can a suitable modeling tool for the XVSM Micro-Room Framework be designed and implemented?
- RQ 2: Is the abstraction used by the micro-room concept more comprehensible than those of common process modeling languages?

- RQ 3: In which domains or use cases is the micro-room concept better or worse than its competitors?
- RQ 4: Is the usability of the new modeling tool more intuitive in contrast to other modeling tools?
- RQ 5: If so, what are the key factors for the intuitiveness of our modeling tool, or a modeling tool for end users in general?

1.3 Methodology

At first, an evaluation of actual process modeling techniques and the corresponding tools is performed. Then, an overview about current problems of such platforms is given, leading to the justification of the new solution. Afterwards, an evaluation about how the XVSM Micro-Room Modeler shall be developed has to be carried out.

After that, the architecture of the modeler has to be designed, considering all of the evaluated problems of other modeling tools, which is followed by the implementation of the modeler. Finally, an evaluation of the modeler and the used micro-room concept is performed. Therefore, a usability testing of end users without special IT knowledge is done. Every user has to model several workflows with both our approach and another modeling tool. Afterwards, each user has to fill in a questionnaire. Both the questionnaires and our notes when observing the users are quantized and used for creating diagrams and answering the research questions.

1.4 Structure of the Thesis

The structure of the thesis is as follows: Chapter 2 covers related work on modeling techniques and tools, including a comparison of their problems. In Chapter 3 an initial draft and related concepts of our approach are shown. After that, the design of our modeling tool, i.e. the XVSM Micro-Room Modeler, is presented in Chapter 4, including technology evaluation and decisions. Chapter 5 contains details about the implementation of the XVSM Micro-Room Modeler, whereas Chapter 6 covers information about how end users and developers can actually use and extend the tool. The methodology and results of the usability testing, comparing our solution to others, can be found in Chapter 7. In Chapter 8 future work is outlined and Chapter 9 concludes the thesis.

Related Work

Regarding process modeling techniques several other contributions exist. In this chapter we will first outline well-known process languages and describe their drawbacks regarding end user modeling. Then, we will present the few end-user-specific approaches and compare all of the approaches in a matrix.

2.1 Popular Process Modeling Languages

Before describing popular process modeling languages we need to discuss their environment. According to [MTJ⁺10], there are four different types of process modeling languages:

- 1) **Traditional process modeling languages:** These languages focus on the understandability by people. They are not formal (i.e. they cannot be interpreted by the computer) and thus are not applicable for automated software generation. Nevertheless, they are of good use for understanding and analyzing processes. Examples for such languages are Petri Nets or Event Process Chains.
- 2) **Object-oriented languages:** They try to represent the world in a way both IT and domain experts can understand. Thereby, every object-oriented language can be used for one or more of the following four use cases according to [Iso01]: a) Understanding a problem in the real world, b) developing an application for simulating the real world, c) developing an application for dealing with entities of a cyber world or d) developing an application that automates business in the real world. The last usage scenario can be seen as what we call “automation support” in our work. The most famous language of this category is *Unified Modeling Language* (UML).
- 3) **Dynamic process modeling languages:** Such languages share at least the following three attributes: a) They cover the full spectrum of use cases (cf. object-oriented languages), i.e. they can be used for understanding a problem by humans and for automatic execution of the process in software. b) Every dynamic process modeling language explicitly

defines a serialization format for model interchange - usually in XML. c) They represent standardized languages by the industry. The two most popular dynamic process modeling languages are BPMN and WS-BPEL.

4) Process integration languages: Languages of this type are typically used in *Business-to-Business* (B2B) areas. They focus on integrating processes of various business partners. Therefore, they establish abstract, technology-independent interfaces and data exchange formats. Two examples for such languages are *Web Services - Choreography Description Language* (WS-CDL) and *Electronic Business using Extensible Markup Language* (ebXML).

As we focus on process languages that allow automation support only, traditional process modeling languages will not be further evaluated. Process integration languages will be skipped as well, since they focus more on the machine-to-machine communication between different ecosystems and not on process modeling by end users such as we plan to do.

Therefore, we will only discuss the most famous representatives of object-oriented languages and dynamic process modeling languages, i.e. UML, BPMN and WS-BPEL, in more detail. Additionally, we will describe the most popular of the few alternative modeling languages that focus on the end user.

2.1.1 UML

UML is developed by the *Object Management Group* (OMG) and can be seen as an industry standard in the software development process. The current version is UML 2.5.0, which was introduced in 2015. The language itself consists of various so-called language units. Each language unit can be used separately and focuses on a different aspect [OMG15]:

Classes: This language unit is intended for modeling classes (a “category” for objects) and their relationships.

Components: Can be used to define components (a modular unit of software), their interfaces and their relations to other components. Thereby, each component can be replaced by a new one or reused in another application.

Composite Structures: While with components only external connections are modeled when reaching a specific size (equal to a black box), composite structures are used to define the relations of inner parts of a greater whole.

Deployments: Allow to specify on which machine which software artifact has to be deployed. In contrast to the other language units, this one is very domain-specific.

Actions: With this language unit it is possible to declare actions which require input values on their input pins and produce output values on their output pins. By connecting various actions a specific behavior can be modeled.

Activities: Is a more detailed form of the actions language unit. Every activity contains a set of connected actions and can itself be used by other activities to form a bigger process. It is more like a control flow with a defined start and a defined end.

Common Behaviors: Can be used to describe the behavior of objects. Any behavior is a direct consequence of an action called on an object.

Interactions: Models of this language unit display the sequence of interactions or messages between systems, components or users.

State Machines: Allow to model discrete behavior via finite state-transition systems.

Use Cases: They are typically used to capture the requirements of a system by sketching which actors (users) interact with which systems in which way.

One reason why UML is so popular today is that it fits perfectly to modern object-oriented languages and allows to visualize every aspect of a modern application. This applies especially for characteristics which are hidden deep inside the code and thus rather difficult to see at all without a model, e.g. the relations of classes [SS07].

As can be seen, UML is designed for modeling many different aspects of software systems. However, for our purpose (i.e. to model processes) only the Activities language unit is qualified. Figure 2.1 shows what a typical activity diagram looks like.

Thereby the filled circle represents the starting point of the process and the framed circle the end point. Actions are represented by rounded rectangles and connected with each other by arrows. Decision nodes are modeled by diamonds whereas filled bars (fork and merge nodes) allow to split the process into two sub processes that run in parallel and later on merge it again.

As can be seen, the process can be modeled rather straight forward. Thus, one could assume that even end users should be able to model their own processes with only little knowledge about UML's structures. However, this assumption only holds if no automation support is needed.

According to Reggio et al. [RLRA12] activity diagrams can be used in five different styles for modeling business processes:

Ultra-Light Style: This style allows the modeler to produce the activity diagram he/she likes. There are no guidelines at all.

Light Style: In contrast to the *Ultra-Light Style* this style imposes some restrictions. These mainly ensure syntactical correctness, e.g. for each fork node there must be a matching merge node and for each decision node there must exist exactly one outgoing "else" arc. The model in Figure 2.1 corresponds to this style.

Disciplined Style: With this style the following additional elements are added to the model as comments: participants, objects and data. These have to be used in each action. The first action in Figure 2.1 thus could be called "CLIENT sends ORDER to EC", where "CLIENT" and "EC" are participants and "ORDER" is an object.

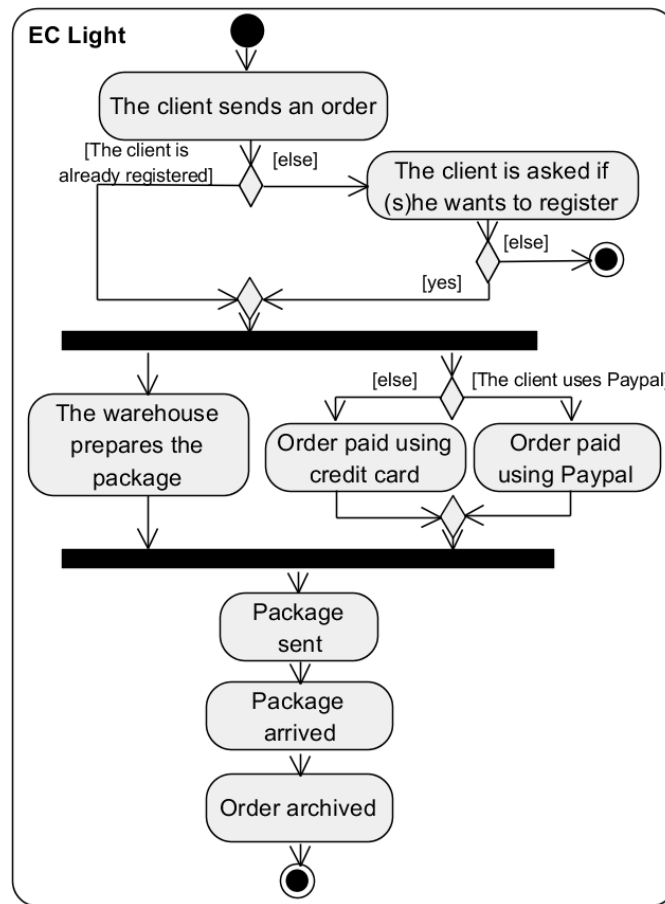


Figure 2.1: Example of an UML Activity Diagram [RLRA12]

Precise Conceptual Style: Instead of just defining participants, objects and data just as comments, in the *Precise Conceptual Style* they have to be modeled in a separate class diagram, the so-called static view. However, methods do not need to be modeled in the class diagram.

Precise Operational Style: This style is equal to the *Precise Conceptual Style*, except that it is more detailed. In the activity diagram, every action needs to call a method on an object that corresponds to the class diagram.

Reggio et al. come to the conclusion that for “a model to be used as the starting point of the (semi-)automatic generation of a BPEL implementation of a system supporting a business process” [RLRA12] the *Precise Operational Style* is required. So, if an activity diagram should provide any kind of automation support, this style is mandatory.

Tool Support

There are a lot of tools available for modeling UML diagrams. But when it comes to model activity diagrams with automation support, only very few remain, since most of them only support the automation of class diagrams.

A popular example for this restriction is Acceleo¹ (cf. Figure 2.2), which is a plugin for the popular *Integrated Development Environment* (IDE) Eclipse². It allows IT experts to model class diagrams for their object relations and generate code from these models. However, activity diagrams are not supported and the setup of Eclipse and the Acceleo plugin are far from being simple. The generation of code from the model is difficult as well, since it is likely that one has to edit the templates from which the code is generated. The full process of transforming a model to code is demonstrated in a 1.5-hour(!) video [1].

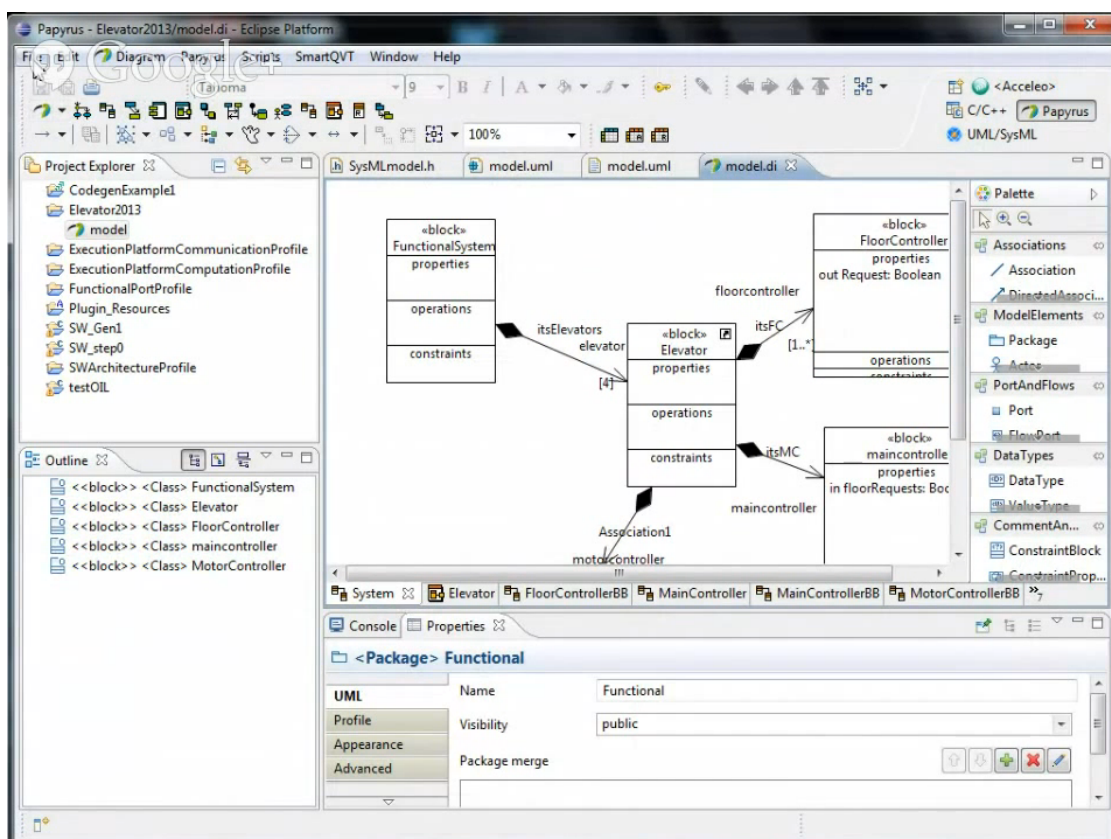


Figure 2.2: Acceleo Eclipse Plugin [1]

Another tool that also supports code generation from activity diagrams is Sparx Systems Enterprise Architect³ (cf. Figure 2.3). Again, this tool clearly focuses on IT experts and is

¹<https://www.eclipse.org/acceleo/>

²<https://www.eclipse.org>

³<https://www.sparxsystems.at>

complex as it supports all features of UML. Also, it is rather expensive: A single-user license for the version with business process modeling capabilities (i.e. “Business & Software Engineering Edition - Named User License”) can be bought for 569 euro, according to [2].

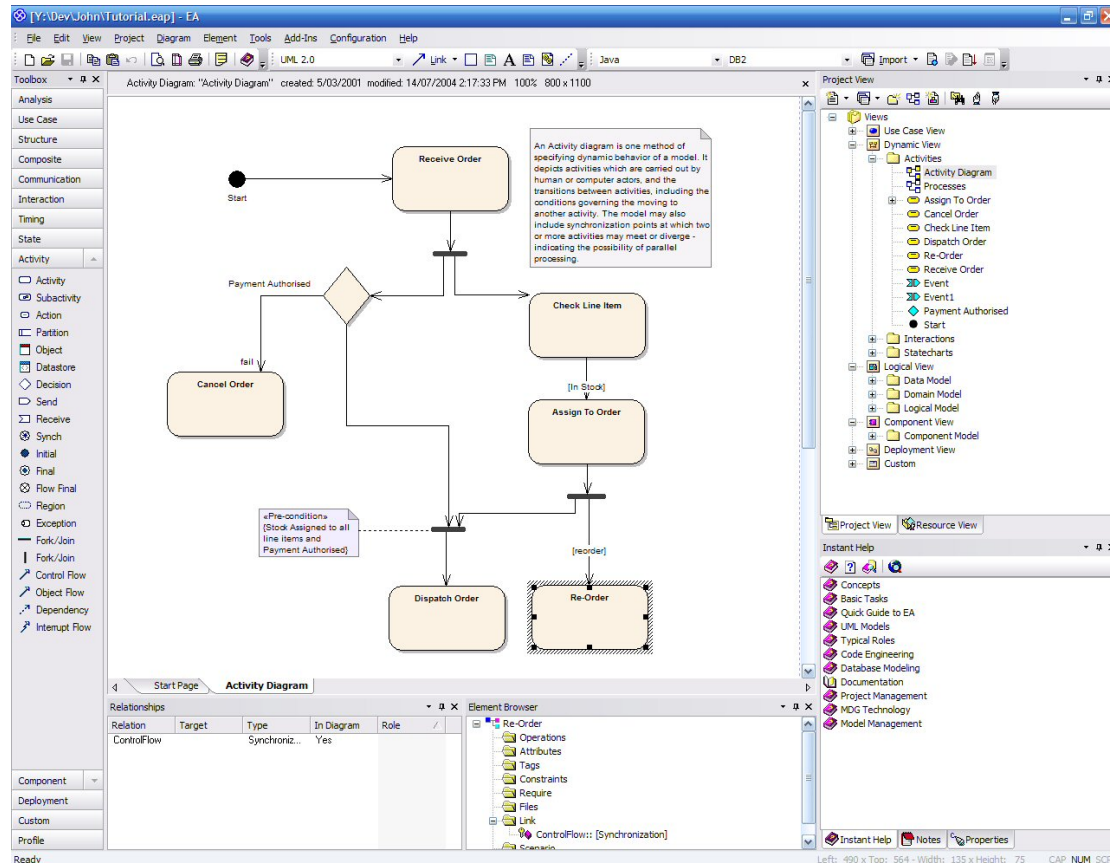


Figure 2.3: Sparx Systems Enterprise Architect [3]

Analysis

After presenting the general features of the language and some modeling tools, we will now discuss the suitability of UML regarding end user modeling.

The first problem is that for models with automation support the *Precise Operational Style* is mandatory. This style demands a very detailed model of not only the behavior view (cf. Figure 2.4) but also the static view (cf. Figure 2.5). With such models, code could be generated automatically with various approaches such as described in [HBR00] or [GS06].

By comparing the model in Figure 2.4 to the one in Figure 2.1 it becomes clear that the initial difficulties of activity diagrams rise significantly when automation support is needed. Both diagrams (Figure 2.4 and Figure 2.5) focus highly on IT experts with elements like objects, method calls and parameter lists and are thus not understandable by end users anymore. In a

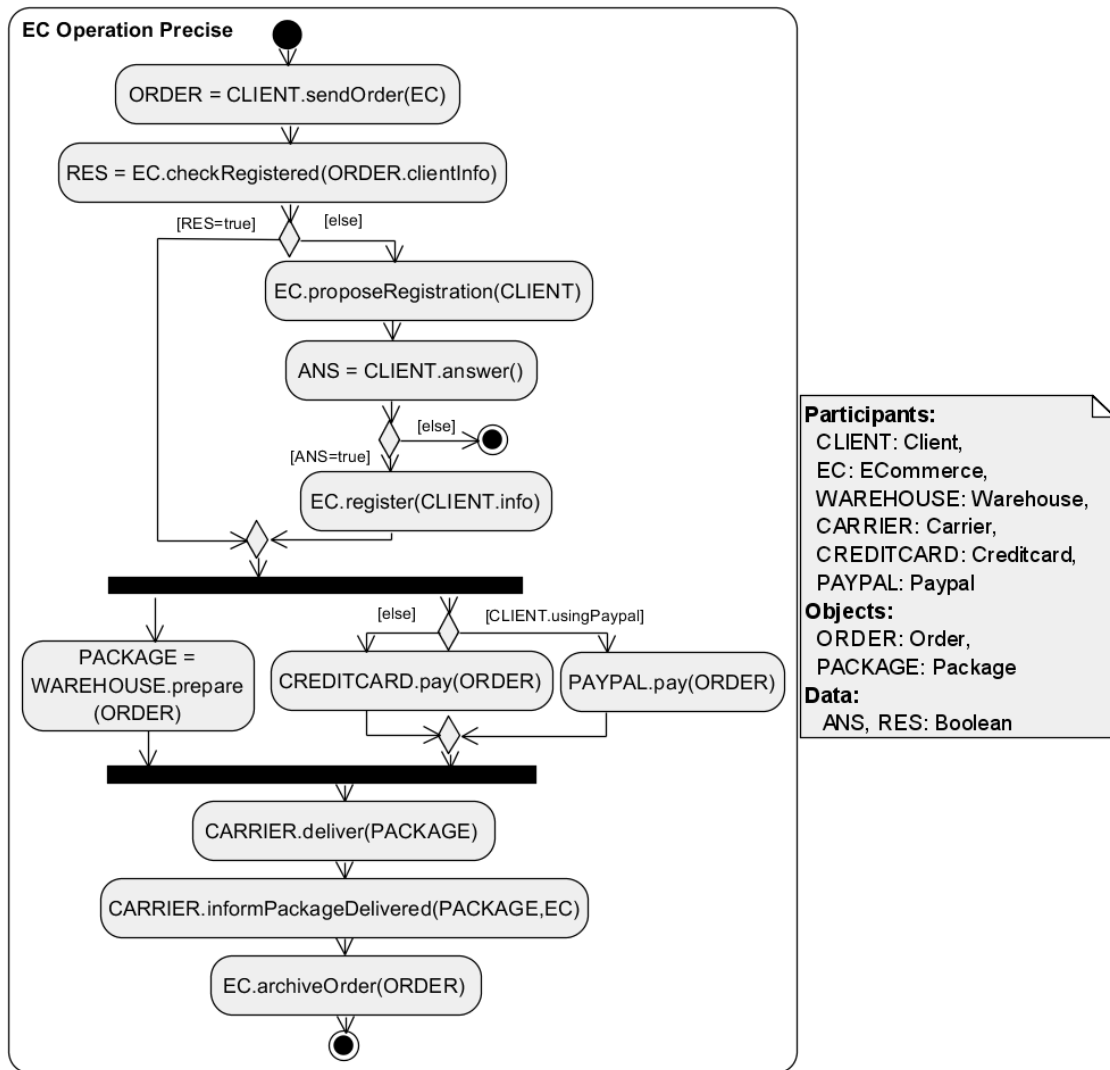


Figure 2.4: Precise Operational Style - Behavior View (UML Activity Diagram) [RLRA12]

survey of 171 UML experts, Dobing et al. found out that indeed “*the complexity of UML is a concern*” [DP06].

A second problem of activity diagrams has been stated by Wohed et al. [WAD⁺06]: “*There is no support for specifying any form of work distribution algorithm or employing varying styles of work distribution (e.g. push vs pull, offer vs allocation).*” So, the distribution capabilities are restricted, especially in dynamic P2P environments.

The third problem of UML is the complexity of its tools. As UML clearly targets IT experts, its tools are either based on or rather similar to IDEs and not easy to understand by end users (cf. Figures 2.2 and 2.3). Usually, these tools also support all of UML’s language units which makes them more complex than needed in our case.

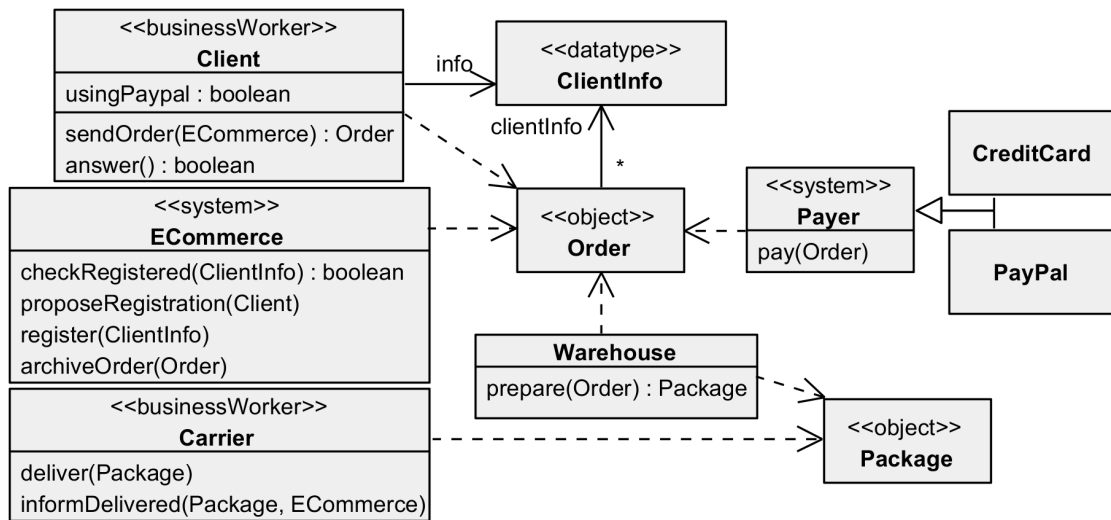


Figure 2.5: Precise Operational Style - Static View (UML Class Diagram) [RLRA12]

Finally, even if all the previous problems could be ignored, the generated code will not be helpful for the end user. It will just be a collection of empty classes which might be connected but does not provide any functionality at all. So, even if the end user could create a valid model and use it for code generation, he/she then would have to pass the generated code to IT experts again, since the code template still needs to be filled with business logic.

2.1.2 BPMN

BPMN is developed by the OMG as well and is the defacto standard when it comes to business process modeling. The current version is BPMN 2.0.2, which was released in 2013 [OMG13]. Its notation is extremely complex and thus leaves almost nothing to be desired. IT experts can model sequences, events, messages, sub tasks and much more.

In Figure 2.6 a typical *Business Process Diagram* (BPD), i.e. a BPMN model, can be seen. Again there is a starting point (circle to the left) and an end point (circle with bold line to the right). In between, the rounded rectangles represent activities, where the plus sign indicates that the activity is a sub-process, i.e. it is composed of a process itself. Diamonds represent gateways which can be used to split the process into two parallel processes and to join them again.

However, this is only a very small portion of BPMN's features. Concerning the process modeling capabilities of BPMN it is even more powerful than UML activity diagrams regarding to the findings of Wohed et al. [WAD⁺06] and Milanovic et al. [MGWD09].

Tool Support

Today, BPMN is already used by non-IT-experts for creating BPDs which are then handed over to developers. But these models are not appropriate for automation support, since they are mod-

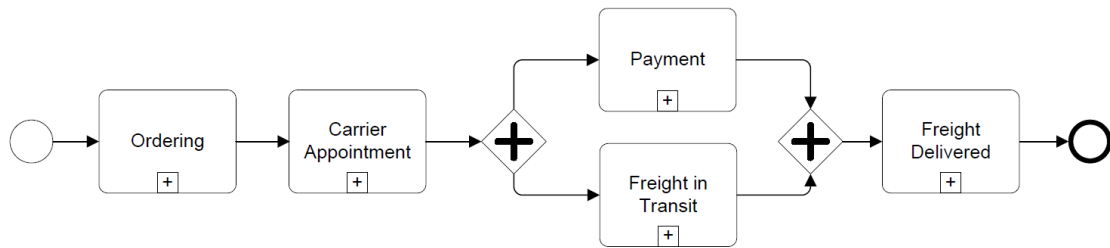


Figure 2.6: Example of a BPD [DDD10]

eled far too informally. Therefore, developers need to heavily adapt them or even worse, completely recreate them. [ODA⁺09]

One of the modelers which is often used by end users to create such informal BPDs is Signavio Process Editor⁴. As it can be seen in Figure 2.7, it is rather easy to use since it focuses on the core elements of BPMN.

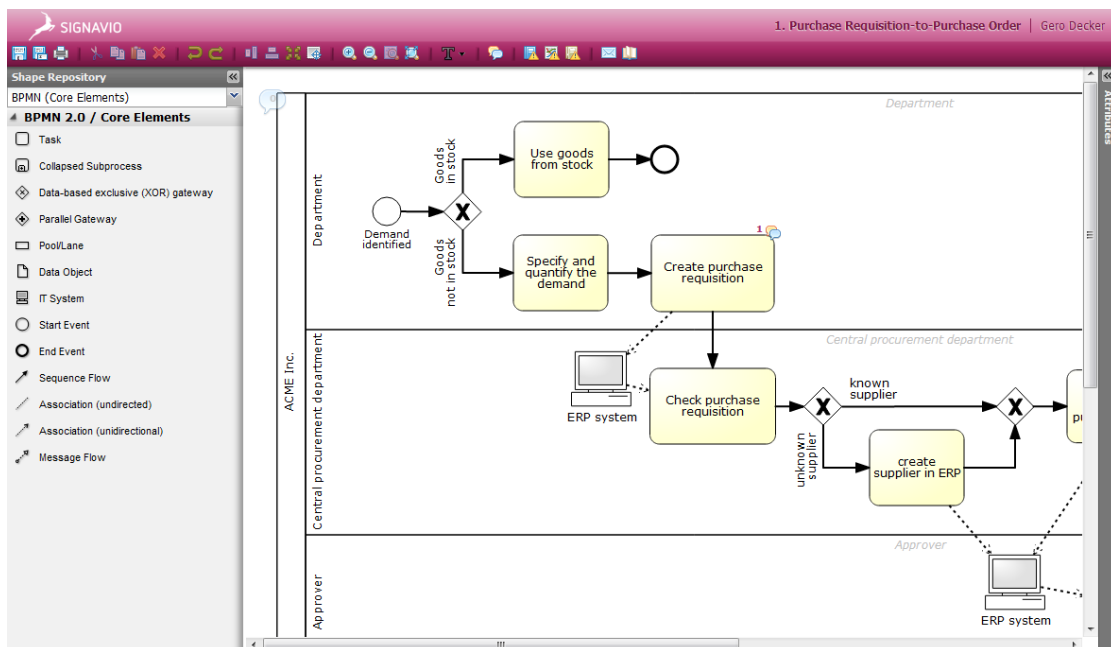


Figure 2.7: Signavio Process Editor [4]

However, one cannot model everything just with the core elements of BPMN. This is why BPDs created by end users with the Signavio Process Editor are often not sufficient for automation support.

If one needs to create such models from scratch, the Activiti⁵ plugin for Eclipse is an eligible

⁴<http://www.signavio.com>

⁵<http://activiti.org>

candidate (cf. Figure 2.8). It is a *Business Process Management* (BPM) platform that allows to both model and execute BPDs. Activiti supports more features of BPMN (e.g. Timers and Service Tasks) and allows its users to execute the generated models on the fly. However, the increased feature set comes at the cost of additional complexity. This becomes explicit when looking at the user guide at [5], having not less than 298 pages.

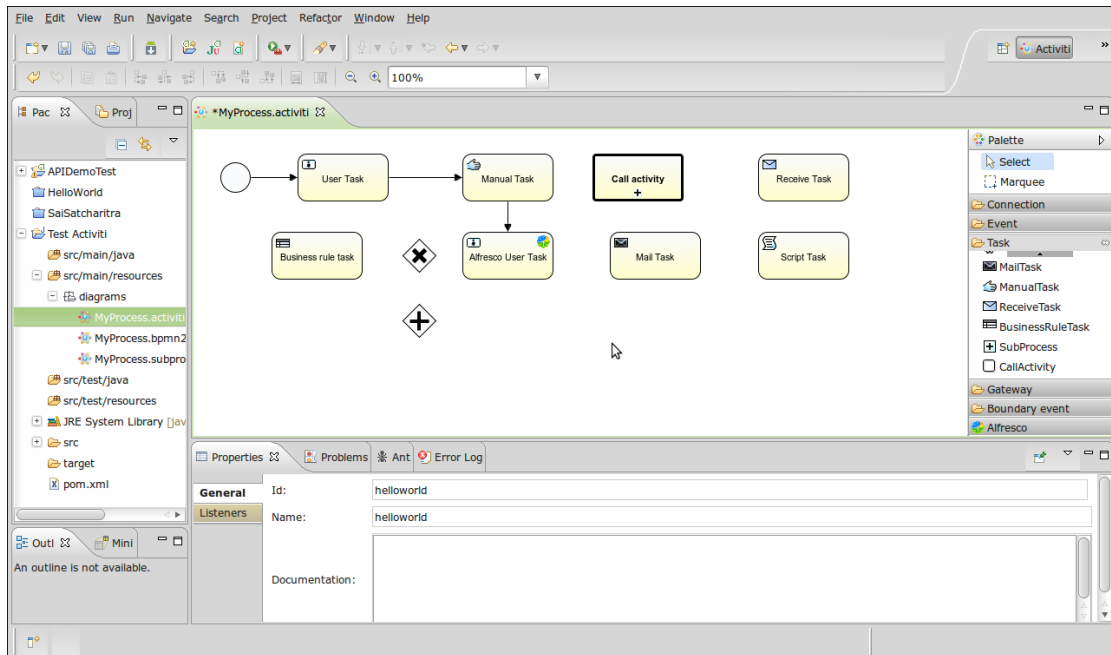


Figure 2.8: Activiti Designer [6]

Analysis

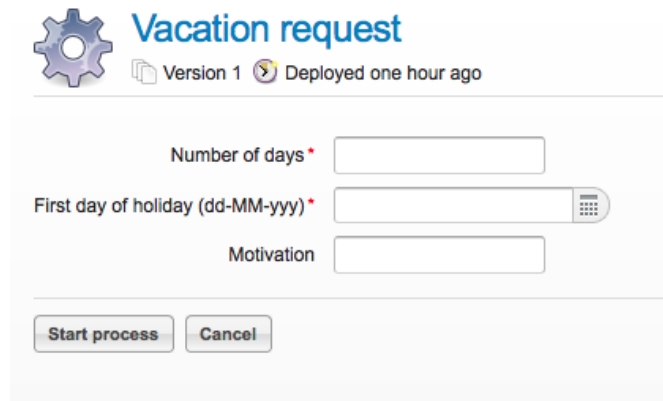
Basically, we are dealing with the same problems that have already been stated with UML activity diagrams. On the one hand, the modeling language can be understood and used by end users to create process models, but these models are too informal to enable automation support. On the other hand, if automation support should be enabled, the models and especially the modeling tools become far too complex to be understood by end users.

But BPMN also suffers from another problem: Due to its complexity the corresponding modeling tools usually support only a subset of the model notation. So, even if the language is very powerful in theory, the lack of proper modeling tools restrains this benefit again.

Also, in contrast to UML, BPMN does not provide that much data-related features. E.g. one cannot specify the relation between data objects but only the relation of data objects to a process activity [ZLC⁺12].

One advantage of Activiti is its human task support. If user input is required somewhere during the process, a simple UI is generated automatically which is capable of asking for specified data. An example for this UI can be seen in Figure 2.9. However, just as with UML, script tasks

need to be implemented by IT experts before the model can be fully used. Therefore, the end user on his/her own cannot create and execute a model.



The image shows a web-based form titled "Vacation request". At the top left is a gear icon. To its right, the text "Vacation request" is displayed in blue. Below this, there is a document icon followed by "Version 1" and a clock icon followed by "Deployed one hour ago". The form itself has three input fields: "Number of days *" with a text input box, "First day of holiday (dd-MM-yyyy) *" with a text input box and a calendar icon, and "Motivation" with a text input box. At the bottom of the form, there are two buttons: "Start process" and "Cancel".

Figure 2.9: Activiti-generated UI for a Human Task [5]

2.1.3 WS-BPEL

WS-BPEL is a part of the WS-* specification and developed by the *Organization for the Advancement of Structured Information Standards* (OASIS). The current version is WS-BPEL 2.0, which was released in 2007 [AAA⁺07]. In contrast to UML and BPMN, WS-BPEL focuses more on the implementation-based part than on the modeling part. Since the beginning, WS-BPEL models are based on XML and the specification itself does not specify any graphical notations. It is mainly used for IT experts to implement processes that use web services and provide well defined interfaces to other processes themselves. The big benefit of WS-BPEL is that later on, processes implemented with WS-BPEL can be orchestrated to other, new processes with low effort.

If we look at a WS-BPEL model in one of the proprietary graphical notations (cf. Figure 2.10), the focus on the technical aspects becomes apparent immediately. The model represents a typical seller process, where a seller iteratively adds new items to sell in his/her auction and can cancel auctions or bids. It can be seen that the representation of the underlying XML model is highly dependent on the modeling tool. In this case, calls of external web services are identified by rectangles with a small envelope and local method calls are represented by rectangles with a small gear.

Beside this small graphical model, the IT expert needs to specify each web service or method call with the exact name and parameters. So, there are a lot of additional specifications in the model that are not visible at first sight.

Due to its clear target group of IT experts, WS-BPEL is one of the best technologies for generating automated processes. Therefore, approaches like [ODA⁺09] exist to transform the more user-friendly BPMN models to WS-BPEL models. However, these transformations always encounter problems, due to the highly different model structures. BPMN supports any kind of

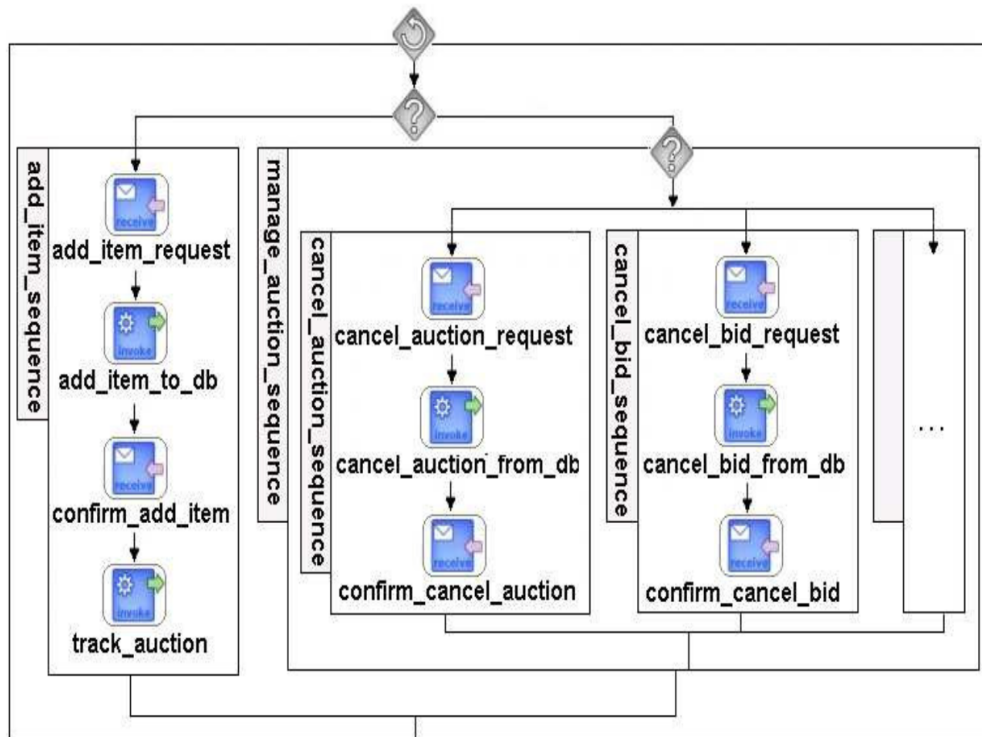


Figure 2.10: Example of a WS-BPEL Model [BEMP07]

control flow structures while BPEL supports just restricted control flow structures. Thus, one either has to restrict the allowed source BPMN models or the resulting BPEL models are very hard to read. [ODA⁺09]

Tool Support

WS-BPEL clearly aims for creating automated software systems which is why there does not even exist a modeling tool without automation support (such as with UML or BPMN). Thus, IDEs with BPEL support need to be used, e.g. NetBeans BPEL Designer⁶ which is a plugin for the NetBeans IDE. Again, as NetBeans is an IDE, it is too complicated to be used by end users. Especially the mapping of input to output parameters cannot be understood without any IT background (cf. Figure 2.11).

Analysis

Regarding end-user-suitability of WS-BPEL Brahe et al. [BS07] come to the following conclusion:

⁶<http://soa.netbeans.org/soa/>

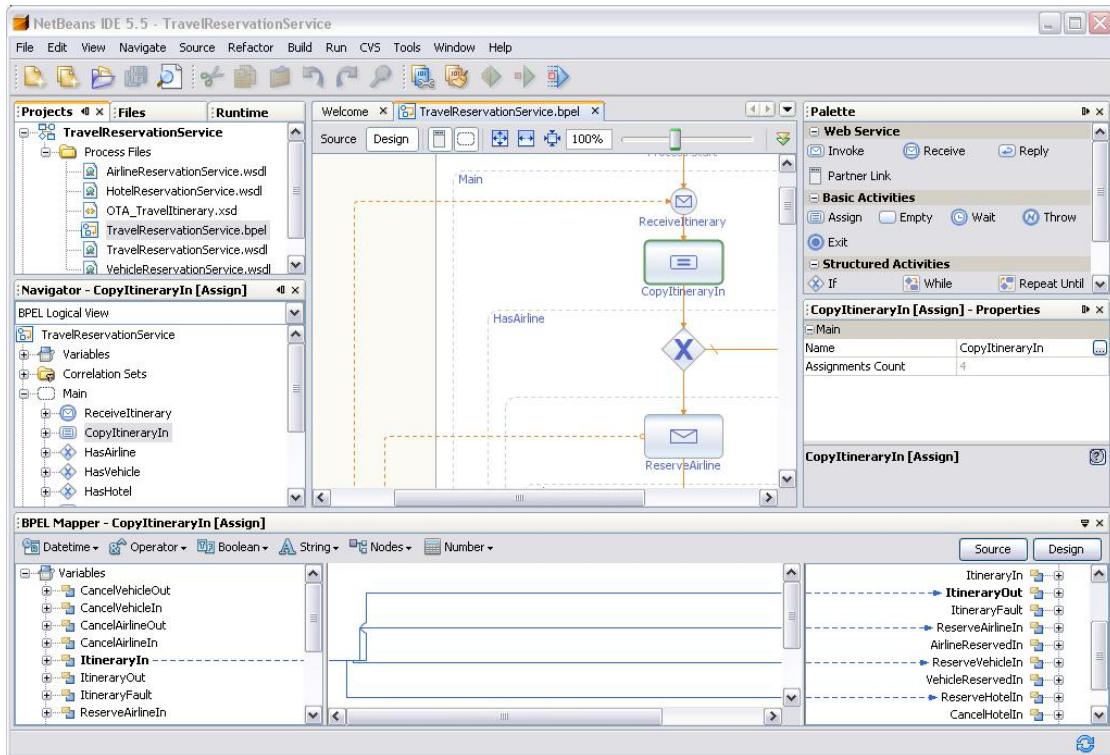


Figure 2.11: NetBeans BPEL Designer [7]

“Although BPEL in combination with SOA affords a remarkable increase in workflow design flexibility, we are far from a situation where ordinary workers are able to define and compose their own local computational workflows. The technology requires the ongoing intervention of highly trained technical specialists. And the development process requires great effort and takes time.”

This corresponds to our findings: WS-BPEL is of great use for IT experts, especially when it comes to reusing software components. But for the end user, it offers only very little to no opportunities.

2.2 End-User-related Approaches

As can be seen, the popular process modeling languages and their tools are not sufficient for end user process modeling. In [Ver04], Verner analyzed various business process modeling approaches and came to a similar result: *“Unfortunately, most BPMS products have been designed for developers, not for business analysts.”*

Since none of the modeling languages presented so far came close to satisfying our needs, we will now discuss alternative approaches in literature. However, it has to be stated that there are only very few and also very differing projects for end user process modeling.

2.2.1 ISEA

In [FRS14], Front et al. propose ISEA, a participative end user modeling approach. The acronym stands for *Identification*, *Simulation*, *Evaluation* and *Amelioration*, which describe the basic steps of the ISEA method, illustrated in Figure 2.12.

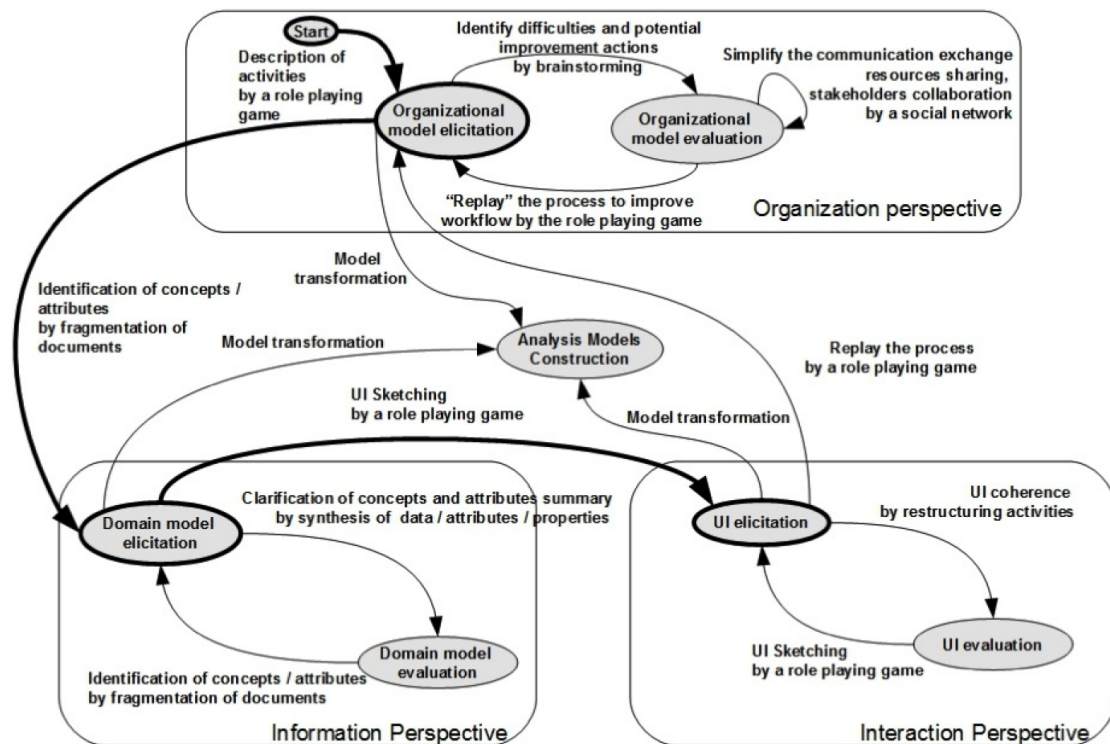


Figure 2.12: The ISEA Method [FRS14]

First, the processes and their activities need to be elicited from the end users (*Identification*). This is done via a computer-aided role-playing-game, where each end user plays him-/herself (*Simulation*). In this role-playing-game, end users basically simulate their whole work day by describing their actions and with whom they interact in a simplified graphical *Domain-specific Language* (DSL). Afterwards, possible problems of the process are analyzed together under guidance of IT experts (*Evaluation*) and the role-playing-game is replayed to improve the overall process (*Amelioration*).

This is done for three perspectives:

Organization Perspective: In this perspective, the method is as described above. Its aim is to elicit the activities, documents and actors of the business process from end users via an iterative role-playing-game.

Information Perspective: This perspective is similar, except that end users now collaborate to give details about the structure of all documents identified in the organizational perspec-

tive.

Interaction Perspective: Here, end users can make sketches of their UI for all activities identified in the organizational perspective.

After all three perspectives have been completed, they can be used for developing the corresponding application. Therefore, the *Organization Perspective* can be transformed to BPMN or Bonita BPM models automatically. Bonita BPM⁷ is an open-source BPM and workflow suite that allows to add automation support to the processes. The *Information Perspective* can be transformed to according UML class diagrams by an IT expert and the *Interaction Perspective* can be used as a basis for generating the UI by UI designers.

Tool Support

The ISEA method is aided by only one tool, i.e. ISEAsy, which allows end users to provide their valuable information according to all three perspectives in a playful way. Figure 2.13 shows the UI of the role-playing-game in the *Organizational Perspective*.

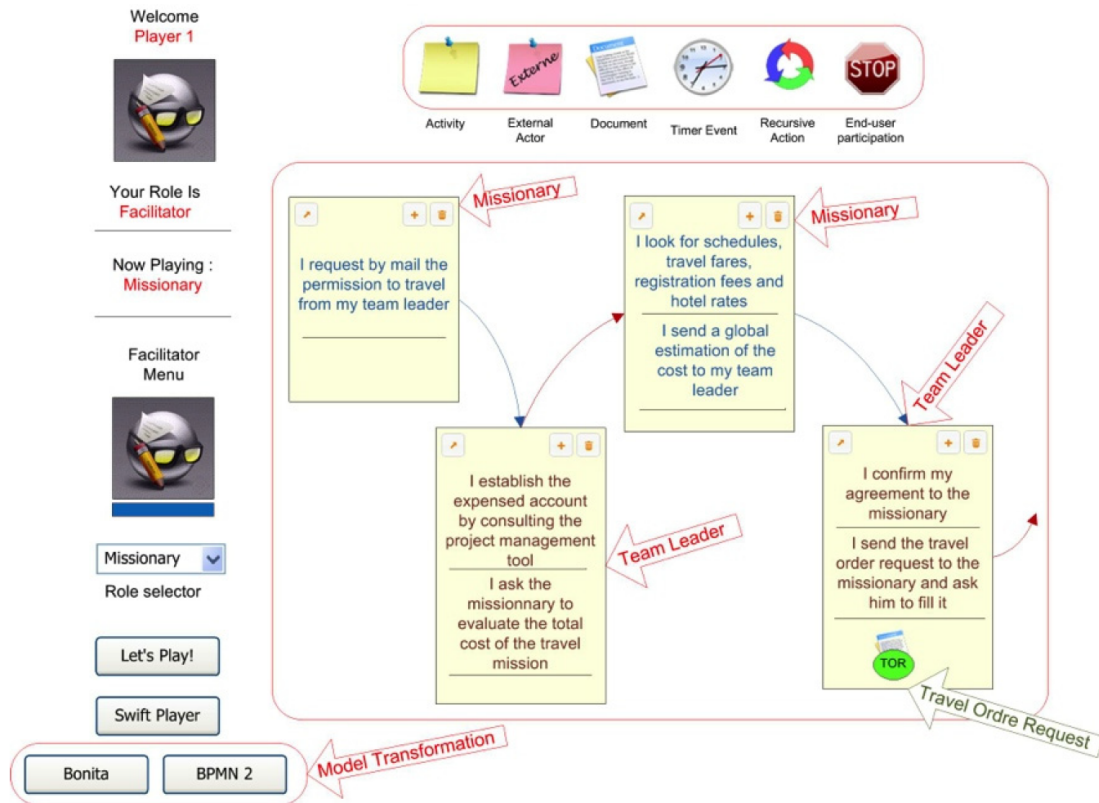


Figure 2.13: Organizational Perspective in ISEAsy [FRS14]

⁷<http://www.bonitasoft.com>

As can be seen, each user creates yellow post-its which represents his/her activities. The users then connect their activities with those of other users via arcs, thereby creating the business process. Additionally, they can add informal documents (green circle) to every activity, which are later described in more detail in the *Information Perspective*.

Analysis

In contrast to the traditional process modeling approaches, ISEA enables users to create processes in an easy way. The DSL and the modeling tool itself are very simple, thus enabling end users to model their processes on their own.

However, the resulting process, even if transformed to a BPMN or Bonita BPM model, does not provide automation support. Additionally, the *Information Perspective* and the *Interaction Perspective* do not provide any automated benefit at all, since they need to be transformed manually by IT experts. Also, the whole method is heavily guided by IT experts, which we want to avoid in our work.

2.2.2 Humanistic Business Process Modeling

The authors of [ASCP13] distinguish between *Humanistic Business Process Modeling* (HBPM) and *Mechanistic Business Process Modeling* (MBPM). Table 2.1 shows the basic differences between the both modeling styles:

Criteria	Definition	HBPM	MBPM
Formalization	The stating of formal rules in a business process specification	Low	High
Detail	The level of detail in a business process specification	Low	High
Agility	The capacity to adapt a business process specification to various external conditions	High	Low
Operationalization	The stating of operations involved in executing a business process	Low	High
Implicitness	What is implied when executing a business process	High	Low
Flexibility	The responsiveness to contextual changes when executing a business process	High	Low

Table 2.1: HBPM vs. MBPM [ASCP13]

The traditional process modeling languages presented in Section 2.1 all belong to MBPM. In contrast to MBPM, HBPM is a “softer”, less formal way of process modeling. Generally one could compare it to the *Light Style* of UML modeling as described in Section 2.1.1. Thus, at the one hand, modeling gets more intuitive for end users, but on the other hand, automation support can be added far more difficultly.

The authors of [ASCP13] created their own humanistic process modeling language based on story composition. Therefore, instead of modeling highly technical processes, end users simply tell their own stories. A story is composed of a sequence of scenes where each scene can be chosen from the scene library. The scene library contains several different archetypes of scenes, e.g. elaborating, discussing or signing a document.

Tool Support

To create the stories, the authors of [ASCP13] developed an easy-to-use tool, called BPMerlin. In Figure 2.14 the creation of a story with BPMerlin can be seen. On the left side, all available locations, situations, objects and document types are listed. On the right side, the actual story can be seen, composed of ten different scenes in the example. Special attention has to be paid to the graphical illustrations of each scene archetype, which is important for the acceptance of creating stories by end users, according to the authors.

Analysis

Just as ISEA, HBPM significantly eases the creation of process models for end users. In contrast to ISEA, no IT experts are needed for guidance during the creation of the stories, since they are very informal. However, this informality is the big drawback of the approach. Similar to the *Light Style* in UML modeling, HBPM cannot be used to enable automation support.

Additionally, the authors of [ASCP13] state nothing about BPMerlin being able to automatically transform the stories to a traditional process modeling language such as BPMN. Thus, adding functionality to the process gets even more difficult than in all other process modeling languages.

2.2.3 Wizard-based Process Modeling

Another approach for end user process modeling has been presented in [LV10]. Lombardi et al. state that “*enablement of a broader user-base incl. non-savvy BUs⁸ has been identified as a key requirement of the business process management of the future*” and that currently “*PM tools are too complex and error-prone for average users*”, both supporting our findings.

To overcome these problems Lombardi et al. proposed *Wizard-based Process Modeling* (WPM), which consists of two steps. First, IT experts create descriptions of lightweight process modeling activities, e.g. a conditional activity. All of these descriptions are then published as a wizard-model representation to a central repository.

In the second step, end users can browse through the repository and use all the wizard-model representations they need for creating their model. E.g. they can choose the conditional activity and then answer questions step by step. After finishing the wizard, the corresponding part of the model is created.

⁸Business Users

BPMERLIN - FRIENDLY BPM Welcome david ! [Log Out]

Home Stories Processes Organizations Scene Library Feedback About

LOCATIONS

Name
>> Service desk
>> Outside
>> Common space
>> Meeting room
>> Office room

Add Location

SITUATIONS

Name
>> Waiting
>> Approve/Validate
>> Communicate
>> Deliver/Receive
>> Formal Meeting
>> Casual Meeting
>> Working
>> Sign Document

Add Situation

OBJECTS

Name
>> Database/IS
>> Computer
>> Money
>> Document
>> Package
>> Fax
>> Material
>> Telephone

Add Object

DOCUMENT TYPES

Name
>> Curriculum Vitae
>> Email
>> Fax
>> Website
>> Score sheet
>> Assessment
>> Form
>> Notification
>> Software application
>> Budget

1 2 3 4 **Add Document Type**

Allow deletion

COMICS

#	Comic	Actors	Location	Situation	Objects
Edit > 1		1	Office room	Working	Document
Edit > 2		1	Office room	Working	Computer
Edit > 3		1	Office room	Working	Computer, Document
Edit > 4		1	Office room	Working	Money, Document
Edit > 5		1	Office room	Working	Computer, Money
Edit > 6		1	Office room	Working	Database/IS, Computer
Edit > 7		1	Office room	Working	Database/IS, Computer, Document
Edit > 8		1	Office room	Working	Material
Edit > 9		1	Office room	Working	Computer, Material
Edit > 10		1	Office room	Working	Document, Material

1 2 3 4 5 6 7 8 9 10 ... **Add Comic**

Figure 2.14: Story Creation in BPMerlin [ASCP13]

Tool Support

The authors wrote their wizards as a plugin for the SOA4All-Composer⁹, which is a process modeling tool for generating models in the proprietary *Lightweight Process Modeling Language*

⁹<http://soa4allcomposer.sourceforge.net>

(LPML) of SOA4All. With this process modeling language, one can compose executable processes from web services, similar to WS-BPEL.

The big difference is that in contrast to WS-BPEL, the LPML of SOA4All uses semantically annotated web services, thus reducing highly technical tasks like correlating input and output parameters and allowing end users to model processes with automation support on their own. An example of the wizard for the conditional activity in SOA4All-Composer can be seen in Figure 2.15.

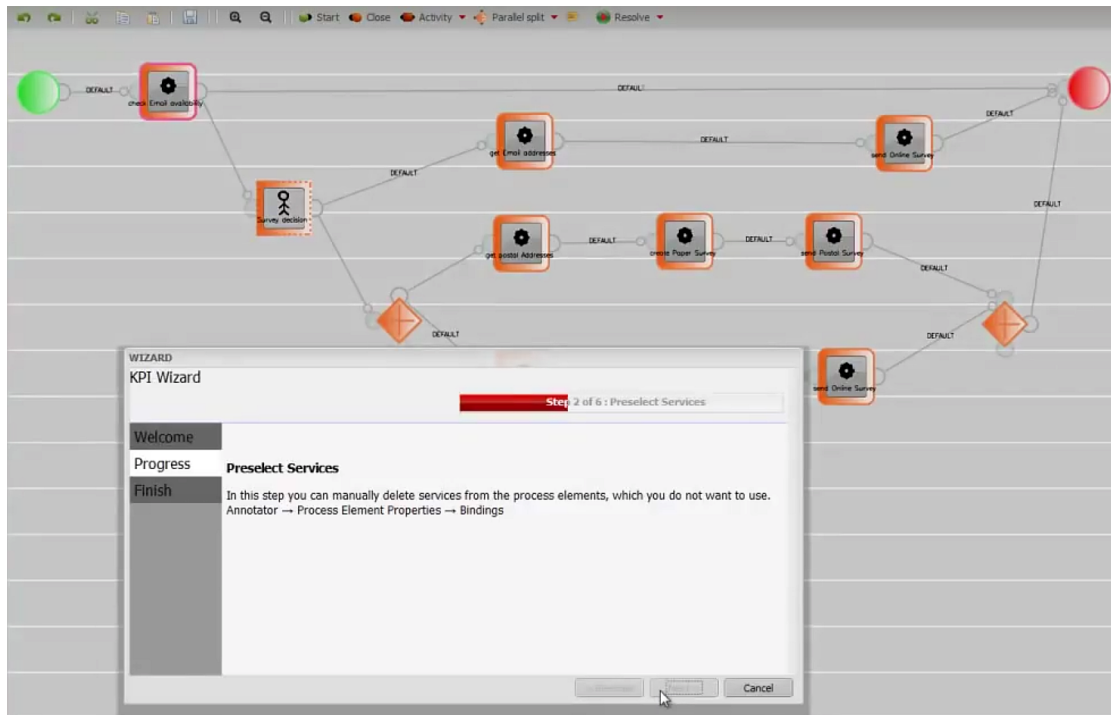


Figure 2.15: UI of the Wizard in SOA4All-Composer [8]

Analysis

Wizard-based process modeling is in some way similar to humanistic process modeling. IT experts create reusable units for end users in the first place (cf. scenes). Then, end users use these units to compose their process in a simpler way.

However, the big benefit of wizard-based process modeling is its integrated automation support. It uses semantically annotated web services in the background, thus the generated process can be executed without any further guidance of IT experts in the end which is very promising.

One drawback that remains is that the abstraction of the process modeling tool is still rather hard to understand and based on those known by traditional process modeling languages like WS-BPEL.

2.2.4 Form-based Web Service Composition

A quite similar approach is *Form-based Web Service Composition* (FWSC) as presented in [WPB13]. The approach relies on user-editable web services which are represented as forms.

As the previous two approaches, it is based on two steps. First, IT experts add web services to a central repository by providing their *Web Services Description Language* (WSDL) and setting a suitable icon for the service to increase user acceptance. They also need to provide human-friendly names for all fields and default values, if applicable.

In a second step, end users choose all services needed from the repository, connect and configure them. Thereby, the configuration is made via forms, containing one input field for each of the parameters needed for the web service. The values for these parameters can be either provided as static values directly or via mapping from in- or output of previous service calls.

After the end user completed the configuration a fully executable WS-BPEL model can be generated automatically.

Tool Support

Weber et al. developed a prototype that supports their approach. An example of a simple process generated by end users is shown in Figure 2.16.

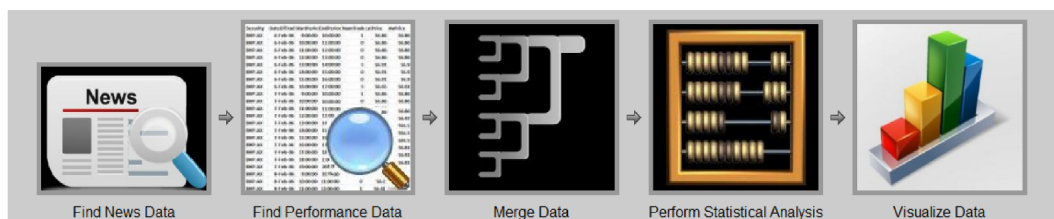


Figure 2.16: Form-based Web Service Composition - Example Process [WPB13]

Each of the five icons has been specified by IT experts in step 1 and represents a web service. The configuration of the services “Find News Data” and “Find Performance Data” is shown in Figure 2.17.

The colored boxes represent all data mappings, where blue fields are unmapped and other colors mean that there is either a data mapping or a static value assigned to the field. E.g. the value 100 is assigned to “Rows per page” as a static value and “RIC” on the left is mapped to “RICs to process” on the right via input to input mapping, indicated by the purple frame.

Analysis

Form-based web service composition is rather similar to wizard-based process modeling. However, form-based web service composition is closer to WS-BPEL which manifests both for IT experts when importing web services by their WSDL in step 1 and end users when defining data mappings between each service in step 2.

The approach enables the generation of WS-BPEL models and thus provides full automation support. However, due to its proximity to WS-BPEL it is still rather complex for end users. Data

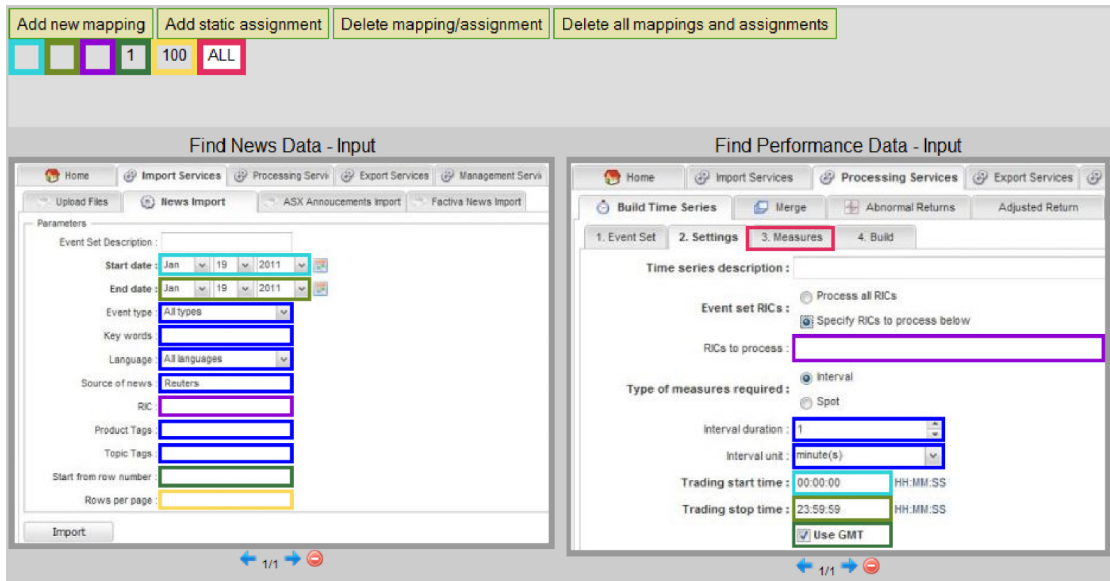


Figure 2.17: Form-based Web Service Composition - Data Mapping [WPB13]

mappings on the field level remain a rather abstract and error-prone task where end users are likely to fail. Also, the UI for the data mapping appears to be rather unintuitive with a lot of different colors and nested tabs.

2.2.5 Web Service Composition Framework

In [TYI05] Talib et al. present the *Web Service Composition Framework* (WSCF) which allows end users to create processes via web service composition. Web service composition has been used in the form-based web service composition approach as well, but the web service composition framework differs considerably from the previous approach. The architecture of the framework is shown in Figure 2.18.

The end user (i.e. composition modeler) provides information to the UI about the process he/she wants to model. This information is evaluated by an inference engine that extracts and deduces relevant parts and stores it in a special format in the internal relational repository. The WSDL analyzer reads the WSDL files of all web services required by the end user and stores the needed data structures in the internal relational repository.

Based on this repository the transformation engine can combine the data given by the user and the data needed for calling the web services and generate a BPEL model automatically. This model is then verified (e.g. so that no deadlocks occur) and executed.

Now, the interesting/unique aspect of this approach is how the user provides information. Talib et al. state: “*We advocate the use of a user friendly interactive graphical interface instead of a graphical modeling language to capture these concepts.*” [TYI05] So, the user does not need to know anything about the syntax of the underlying modeling language. Instead he/she provides the required information in form of control rules (cf. Listing 2.1).

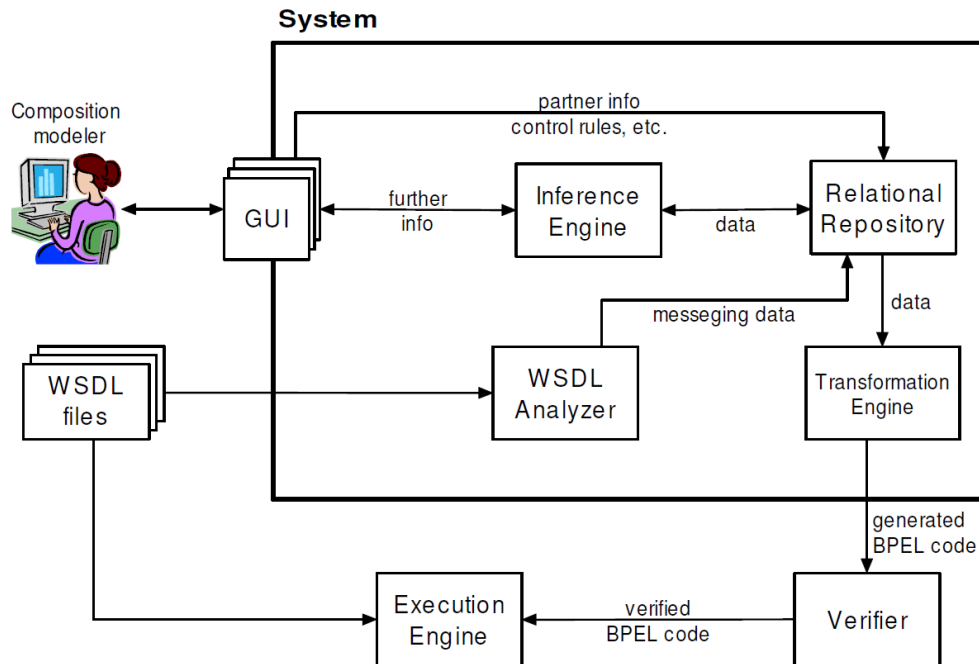


Figure 2.18: Web Service Composition Framework - Architecture [TYI05]

```

1 ControlRule {
2   from (activity)
3   [Condition (Boolean expression)]
4   to (activity)
5 }

```

Listing 2.1: Web Service Composition Framework - Control Rule

Thereby, *activity* needs to correspond to a name of a web service according to the provided WSDL files and the *condition* is optional. Unfortunately, nothing is stated about the input format of other important information, e.g. parameters for the web services.

Tool Support

The authors of [TYI05] state that “*tool development is still at its initial stage*”. So currently all the required information needs to be provided via text files.

Analysis

The basic idea of this approach, i.e. that the user does not need to model the process but simply provides informal information, is quite interesting. However, to develop such a framework is quite challenging and the whole complexity is translocated to the part of how the user specifies the information and how the information is interpreted and transformed.

Unfortunately, the presented approach lacks details concerning these questions. Neither does it explain how e.g. parameters are provided by users nor how the information is deduced by the inference engine so that it can be used for generating a BPEL process. Also, without any tool support at the moment, the approach provides no usability at all.

2.2.6 End-User-driven Process Modeling

The last approach we will analyze has been presented by Stoitsev et al. in [SSF08] and is called *End-User-driven Process Modeling (EUPM)*. It concentrates on only one domain, i.e. task management. Here, end users specify intuitive task delegation graphs which are then used by the tool to generate BPMN models.

Tool Support

The authors of [SSF08] developed the Collaborative Task Manager to support their approach. The tool can be seen in Figure 2.19.

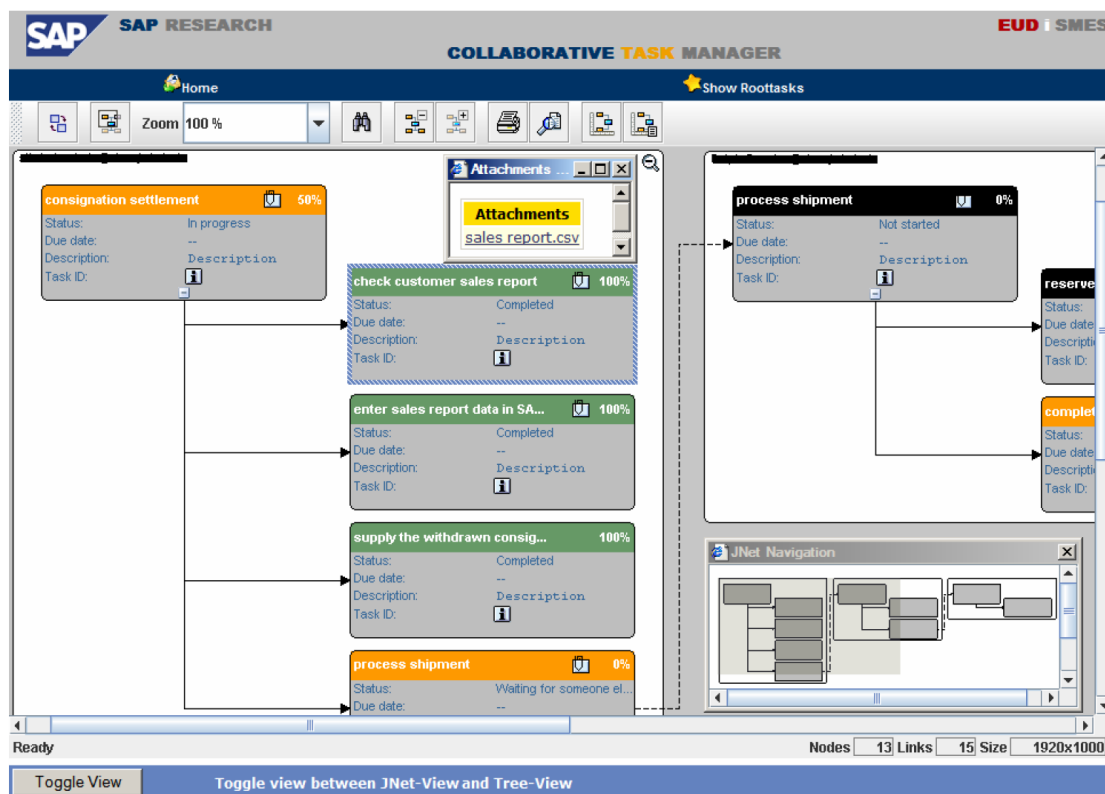


Figure 2.19: Collaborative Task Manager [SSF08]

The Figure shows the structure of a task delegation graph, i.e. one task, consisting of multiple sub tasks executed in a specified order. In the lower right corner one can also see that multiple task delegation graphs can be connected to each other.

Based on the structure of the graph one can easily think of a transformation to a BPMN model as the structure is quite similar.

Analysis

As can be seen in Figure 2.19, the structure of task delegation graphs is very simple, so the approach is suitable for end users. Also, even though the approach is fairly simple, it provides full automation support which is a rare combination.

However, the main reason for its simplicity is that it is highly tailored to the domain of task management. Therefore, all unnecessary generalisations can be omitted, leading to a very easy modeling notation and reduced complexity of the transformation algorithms to the corresponding BPMN models with automation support.

2.3 Comparison

After describing all of the promising approaches in literature, we will now compare them regarding their suitability for end users. Therefore we introduce a list of acceptance criteria for a decent modeling technique targeting end users. Most of the criteria are based on [MSMP11] (*Distributed Execution*), [MAKS12] (*Automation Support, Human Task Support, Multiple Domains*) and [Bin13] (*Easy Deployment*). Some have been deduced by us (*Tool Simplicity, Comprehensible Abstraction*), based on our findings when analyzing the approaches.

For the sake of completeness, also the popular modeling languages described in Section 2.1 have been added to the comparison, even though they do not target end users. The results of the comparison can be seen in Figure 2.20.

		UML	BPMN	WS-BPEL	ISEA	HBPM	WPM	FWSC	WSCF	EUPM
Features	Automation Support									
	Human Task Support									
	Distributed Execution									
	Multiple Domains									
End User Suitability	Tool Simplicity									
	Comprehensible Abstraction									
	Easy Deployment									

Figure 2.20: Comparison of Modeling Techniques

In the following, we will describe the meaning of the criteria and give details about the comparison results:

Automation Support: As this is one of our main premises, we will first deal with automation support of the modeling technique. Thereby we mean that a created model shall be executable and thus provide additional value in contrast to e.g. a much simpler flow diagram. Also Münch et al. define this as one of the main requirements for a process modeling language: “A process modeling notation should support the interpretation and execution of the process representation by a machine.” [MAKS12]

Strictly speaking, *UML* does not provide full automation support. It only generates empty classes and connects some method calls but the model itself is not executable without a lot of additional effort. *BPMN* and *WS-BPEL* on the other hand allow full execution of their models.

Coming to the end-user-related approaches, *ISEA* allows more or less automatic generation of *BPMN* models. However, these are not detailed enough for automation support and thus need additional manual adaption. *HBPM*'s stories are easy to create, however far too informal for allowing any automation support. *WPM* on the other hand enables full automation support due to its composition of semantically annotated web services. Since *FWSC* and *WSCF* internally generate *WS-BPEL* processes they provide full automation support. Finally, *EUPM* generates highly specialized *BPMN* models that can later be executed by an own workflow engine.

Human Task Support: If a modeled process can be executed, it also needs to provide possibilities for humans to interact with it. This means, at least the modeling end user him-/herself should be able to interact with the process at runtime via e.g. providing input or making decisions at some stages during the automated process. In literature this is often referred to as *human task support*. Münch et al. define this requirement as follows: “A notation for process representation should account for handling decisions made by humans during process performance” [MAKS12].

All modeling techniques that do not provide automation support will be neglected in the following since they fail the premise of an executable process. Both *BPMN* and *WS-BPEL* allow human tasks by *User Tasks* and *WS-BPEL4People* respectively. A *User Task* provides a simple UI to the user within the *BPMN* process engine and waits for him/her to fill out the specified form or click on the provided button [OMG13]. *WS-BPEL4People* is an extension for *WS-BPEL* that introduces so-called *people activities*. For such a task, the *WS-BPEL* execution engine requests a message from the corresponding endpoint which could be a UI where the user is allowed to enter forms or click buttons. After the user performed the corresponding action, a response message is sent back, thereby allowing the process to proceed [ICK⁺10].

Regarding *WPM*, the authors state nothing about human task support, but due to its *WS-BPEL*-related nature it might be addable without a lot of effort. *FWSC* allows the user to provide process-instance-specific data and thus lets the user influence the process behavior

at runtime. The authors of *WSCF* give no information whether their approach supports human tasks or not. Though, as it generates a WS-BPEL process at the end, WS-HumanTask integration might be possible with low effort. Since *EUPM* handles only the domain of task management, human task support is enabled in a way that end users can e.g. change the completion level of a task during runtime.

Distributed Execution: Beside human task support, processes modeled by end users especially require native support for distributed execution. This means that the process is executable not only on the end user's machine itself but can interact with machines of other end users as well. This is needed for many common use cases that include more than one user and thus is a very important requirement. Meyer also states that "*the execution of the process steps is usually distributed over the devices*" and that "*the orchestration of these distributed execution activities must be possible*" [MSMP11].

Again, all modeling techniques that do not provide automation support at all will be neglected for this criterion as they do not support the premises. In *BPMN* distributed process execution is highly dependent on the underlying workflow engine, but possible. Due to the orchestration of distributed web services, *WS-BPEL* is predestinated for distributed processes.

The authors of *WPM* do not state anything about the support of distributed process execution, but its relation to WS-BPEL might allow this feature. *FWSC* and *WSCF* provide support for all of WS-BPEL's features and thus also allow distributed process execution. Regarding *EUPM*, the process physically runs on one instance, however the modeling tool allows multiple users to change the task delegation graphs.

Multiple Domains: This criterion means that the modeling technique needs to be able to model various different domains. Its elements need to be general enough to be used independently of the domain. Concerning this requirement, Münch et al. state that "*a process modeling notation should enable a generic representation of information in order to allow for process models that can describe commonalities of processes from several different projects.*" [MAKS12]

The popular process modeling languages (*UML*, *BPMN*, *WS-BPEL*) are all general enough to model arbitrary scenarios of arbitrary domains. The same holds for *ISEA*, *HBPM*, *WPM*, *FWSC* and *WSCF*. However, *EUPM* completely targets the domain of task management, thus providing no possibility to model anything else.

Tool Simplicity: This criterion directly relies to end user suitability and means that the tool used for the modeling language has to provide a tidied up interface, needs to contain only few interface elements and requires a clear structure. If it contains complex features, at least user guidance via tooltips or help dialogues is mandatory. Finally, the tool has to be understandable by end users without IT background.

The popular process modeling languages (*UML*, *BPMN*, *WS-BPEL*) only provide tools designed for IT experts, often relying on IDEs and thus being far too complex for end users to understand.

ISEasy (*ISEA*) on the other hand is a rather simple tool, allowing end users to model their tasks in a playful way. This also holds for BPMerlin's (*HBPM*) story creation which is very intuitive. Regardless of the more abstract BPEL-based modeling language itself, SOA4All-Composer (*WPM*) provides a tidied up UI with a lot of user guidance thanks to its wizards. The tool supporting *FWSC* on the other hand is quite unintuitive, having a lot of nested tabs, configuration possibilities and no clear structure. Also the correlation between parameters is indicated via different colors which could have been done more intuitively, e.g. via connecting the parameters by lines. However, if compared to native WS-BPEL tools, it is still more intuitive. *WSCF* provides no tool at all, meaning that the user would have to provide his/her model by a text file which is unacceptable. Finally, the Collaborative Task Manager used in *EUPM* is fairly simple, having only few interface elements and a tidied up interface.

Comprehensible Abstraction: Beside the simplicity of the tool, the modeling language itself needs to use an abstraction that is easily understandable by end users. Therefore, its concepts and terminology have to rely on the real world as much as possible to reduce the amount of explanations required, thereby lowering the entry level.

In this comparison we are considering the abstraction of models that enable automation support as far as possible. Under this premise, *UML*, *BPMN* and *WS-BPEL* use an abstraction that is hard to understand (cf. to the precise operational style in *UML*, and the detailed parameter mappings in *BPMN* and *WS-BPEL*).

ISEA allows end users to define models in their known DSL but still requires guidance by IT experts due to the complexity of the overall process. The abstraction in *HBPM* can be understood easily, due to its story-based models. Since *WPM* relies on a WS-BPEL-based modeling language, its abstraction is similarly hard to understand, regardless of the good wizard-based UI. *FWSC* is also heavily related to WS-BPEL, thus requiring end user to specify parameter mappings of web services on their own, which is not possible without IT knowledge. The model required by *WSCF* basically constrains to control rules which could be done easily by end users if a tool existed. However, nothing has been stated about how the tedious parameter mappings are provided by end users. Modeling task delegation graphs in *EUPM* uses an abstraction that can be understood easily, since the notation of these graphs has been directly deduced from the domain of task management.

Easy Deployment: This criterion is deduced from [Bin13] and means that, given that the model has been created, the end user is able to execute the model him-/herself. So, after designing the model, no more tasks are required, such as implementing services that are used in the model or setting up a server for execution. Even if an approach allows easy modeling and has full automation support, it is worthless if the end user requires IT experts to deploy the model in order to see the impacts of different modeling decisions.

In *UML* the generated classes need to be filled with code and then compiled and executed, so IT experts are required before the model can be deployed. The workflow engines using *BPMN* are quite promising, as they can execute a model on the fly. However, script tasks need to be implemented by IT experts, so easy deployment is only possible if the end user

does not use such tasks. Given that the end user already has a fully functional model, *WS-BPEL* processes need to be compiled and deployed onto special WS-BPEL servers. This task is not as difficult as in UML, but still too tedious and error-prone for end users.

ISEA and *HBPM* do not provide any kind of automation support, thus there also is not any deployment. *WPM* allows deployment on the fly within SOA4All Composer whereas *FWSC* does not provide any integrated deployment support and relies on third-party WS-BPEL servers. The authors of *WSCF* state that their generated WS-BPEL model is forwarded to an external execution engine, i.e. a third-party WS-BPEL server equally to *FWSC*. Lastly, *EUPM* is a fully integrated workflow engine for task management and allows to deploy the generated models from within the tool.

As can be seen, none of the approaches presented fulfills all requirements for a decent end-user-related modeling technique. That is why we will present our approach in the following chapter.

Approach

In this chapter we will first discuss the related concepts used by our approach. Then, we will describe our approach based on the requirements specified in the last chapter and outline how each of them is fulfilled.

3.1 Related Concepts

We use several concepts that need to be described first before understanding their value for our approach.

3.1.1 XVSM

XVSM is an abbreviation for *eXtensible Virtual Shared Memory* and is a space-based middleware of the Space Based Computing Group of the Institute of Computer Languages at TU Wien. The Java implementation of XVSM is called *Mozartspaces*¹, which is used in this work.

As most space-based technologies, XVSM relies on the Linda model (cf. [Gel85]). The main goal of such tuple spaces is to decouple process communication in both time and space. This is achieved by providing a shared space where all processes can store and read data from.

XVSM extends the decoupling of inter-process communication on the same machine to distributed P2P environments. In Figure 3.1 exemplary P2P communication in XVSM can be seen.

Each instance of XVSM is represented by one core that is a part of the distributed space. Thereby, one to many cores can run on any peer or even on a centralized location. It is important to note that all parts of the space can be accessed by all peers.

The squares marked with C1 to C6 indicate containers that contain entries. Each entry can be an arbitrary type implementing the Java interface `Serializable`. Figure 3.2 shows the internal structure of a container.

¹<http://www.mozartspaces.org>

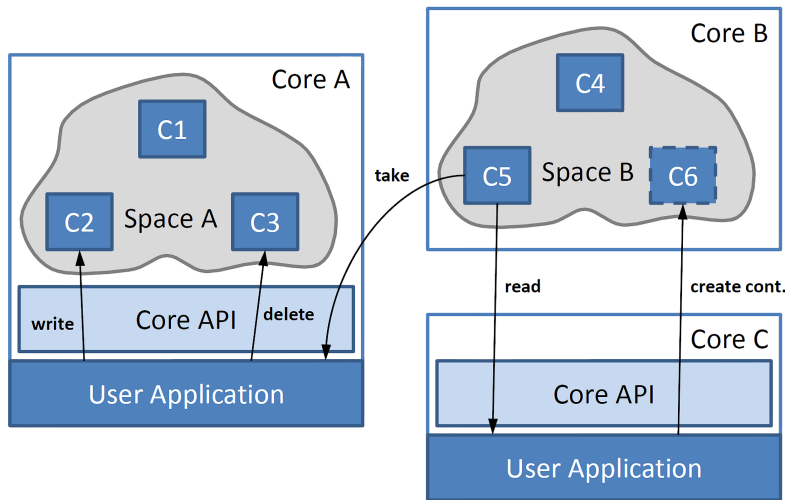


Figure 3.1: XVSM P2P Communication [Dön11]

Thereby, for each entry several coordinators can be placed which contain metadata concerning the accessibility of the entry. E.g. a `FifoCoordinator` returns entries in the same order as they were written into the container while the `RandomCoordinator` returns a random entry. There are many more coordinators such as the `TypeCoordinator`, `VectorCoordinator`, `LindaCoordinator`, `QueryCoordinator`, `LifoCoordinator`, `LabelCoordinator` and the `KeyCoordinator` which allow to use complex access patterns easily.

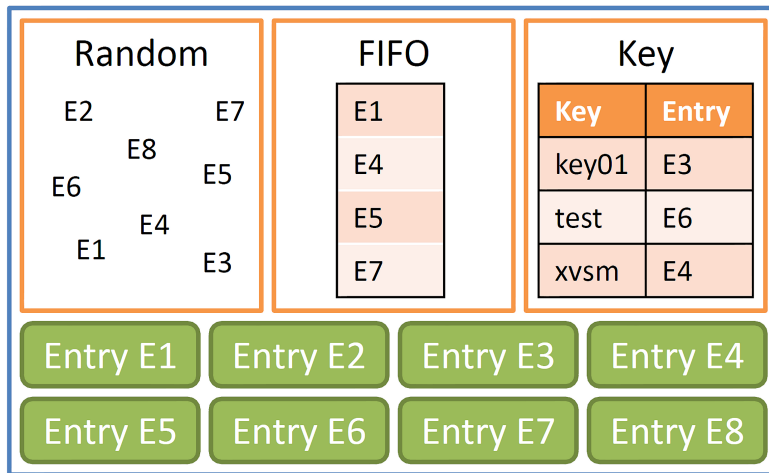


Figure 3.2: XVSM Container - Internal Structure [Dön11]

When accessing the entries of a container, the following operations are possible: `write`, `read`, `take`, and `delete` of one or more entries. If that is not sufficient, aspects can be added to each of the operations. An aspect is a user-defined piece of code that will be executed before

or after an operation, depending on where the aspect is registered. The processing order of aspects is shown in Figure 3.3.

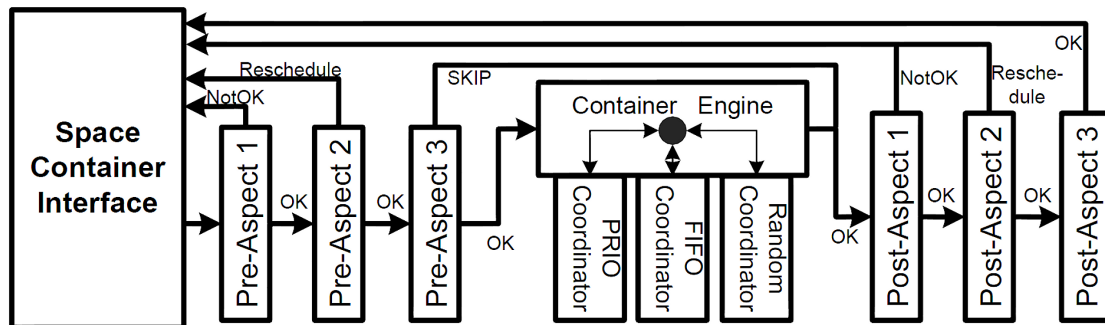


Figure 3.3: XVSM Aspects [MKS10]

A Pre-Aspect is perfectly useful to e.g. validate data, whereas a Post-Aspect could be used for logging successful operations. In case of errors, an aspect can either retry the current operation later (*Reschedule*), cancel the operation (*NotOK*) or skip the operation itself while still executing the other aspects (*SKIP*).

Further information about XVSM core features can be found in [Bar10], [Dön11] and [Cra10]. Beside these core features, also extended functionality is used by the XVSM Micro-Room Framework, i.e. XVSM Security [CK12, CDJK12, CDJ⁺13], XVSM REST *Application Programming Interface* (API) [Pro11], XVSM Persistence [Zar12] and XVSM Distributed Transactions [Brü13].

XVSM Security allows to secure the access on containers based on authenticated user roles or attributes provided by an external identity provider. Thereby, access restrictions can be set individually per container, container operation and user/role.

More about the other features is described in [Bin13].

3.1.2 Peer Model

The Peer Model [KCJ⁺13] is a high-level coordination model that is based on XVSM. It is intended for developers and targets highly concurrent and distributed environments across all domains.

The main goal of the Peer Model is to separate coordination logic from application code in distributed applications. Coordination logic usually relies on the same generic communication patterns that can be created or composed of simple sub-patterns with the Peer Model, instead of developing them over and over again in code [KCS15].

The key concepts of the Peer Model are described in the following [KCJ⁺13]:

Peer

A peer represents a re-usable component that can both contain internal coordination logic and execute service functionality. This internal logic is encapsulated via two external “interfaces”,

i.e. the *Peer-In-Container* (PIC) and the *Peer-Out-Container* (POC). With these containers, peers can collaborate with each other. For example, peer A gets some entries in its PIC, processes them in its internal stage and writes them into its POC from which they are forwarded to peer B's PIC and so on.

Internally, application-specific logic can be called in terms of pre-defined service methods. Also, so-called sub peers can be called which are peers themselves, thus allowing easy pattern composition as done in [KCS15].

Technically, each peer can be uniquely addressed via its *Uniform Resource Identifier* (URI), providing two separate XVSM containers (i.e. PIC and POC). Internally, a peer can provide no logic at all (*Space Peer*), only coordination logic (*Coordination Peer*) or application-specific logic (*Application Peer*).

Wiring

Figure 3.4 shows the typical structure of a peer (P_1), including PIC, POC, internal logic, a sub peer (P_2) and the communication with another peer (P_3). All connections between peers and within a peer are modeled with wirings. Each wiring consists of several mandatory parts: a name (e.g. W_1), one or multiple guard links specifying on which incoming entry types the wiring activates (e.g. R) and one or more action links defining where to put the resulting entries of the corresponding type (e.g. T).

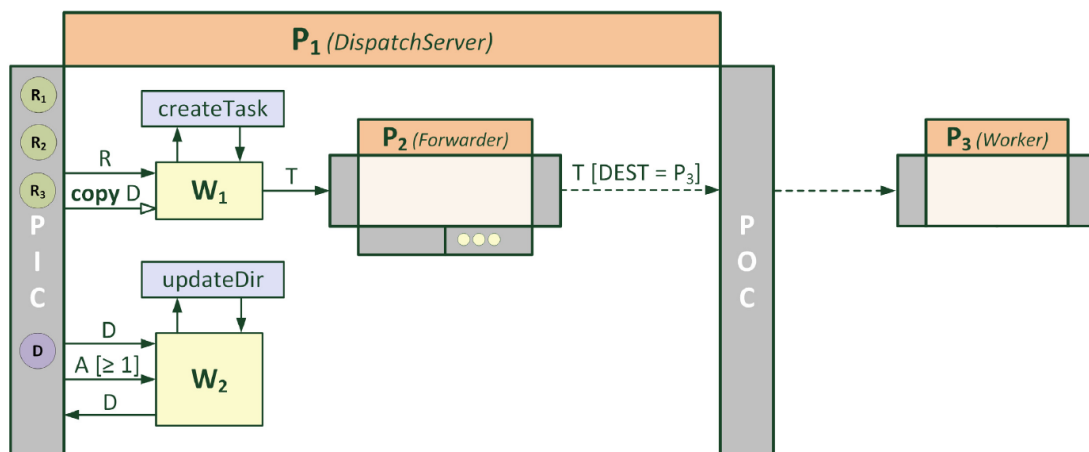


Figure 3.4: Peer Model Example [CJK15]

A wiring can also contain optional application-logic service calls (e.g. `updateDir`). Each guard link can be extended by an optional query that tries to select entries from the PIC, which also has to hold for the guard link to activate the wiring. Additionally, an optional `count` parameter defining the minimum and maximum amount of entries that should be selected can be provided. A guard link can only activate the wiring if at least the minimum amount of entries is available. Multiple guard links can be defined, all needing to be fulfilled for the wiring to trigger. If one guard link cannot be fulfilled the wiring will not activate.

Technically, guard and action links represent XVSM container operations (read/take/write) between PIC and internal wiring container (guard link) or internal wiring container and PIC/POC (action link) of the same peer or between a peer and a sub peer. Input links and guards are conceptually based on `TypeCoordinators` and `QueryCoordinators` of XVSM.

Entry

An entry contains its payload and several other key/value pairs that are described in this section.

The set of all wiring executions required to solve a task during runtime forms a flow. Hence, a flow can be seen as a runtime instance of an executed sequence of work. Each flow can be identified with a unique *Flow Identifier* (FID) which is created during runtime for the first entry starting the flow. All entries created afterwards as logical successors of this entry will contain the same FID. Each entry belongs to exactly one flow and a wiring only processes entries with matching FIDs.

Each entry can also contain *Time-to-Start* (TTS) and a *Time-to-Live* (TTL) property. The entry will only be scheduled after the TTS has expired. After the TTL has expired, the entry will be automatically removed from the container. The *Destination* (DEST) property of an entry can be used to model directed remote communication. If it is specified, the entry is injected into the PIC of the specified peer (e.g. [DEST = P3]).

3.1.3 Micro-Room Concept

The concept of micro-rooms has first been presented in [Bin13]. Micro-rooms allow to abstract fully-integrated software components in a way that is understandable also for end users without IT background. The concept will be described with the help of a practical example, i.e. the library shown in Figure 3.5.

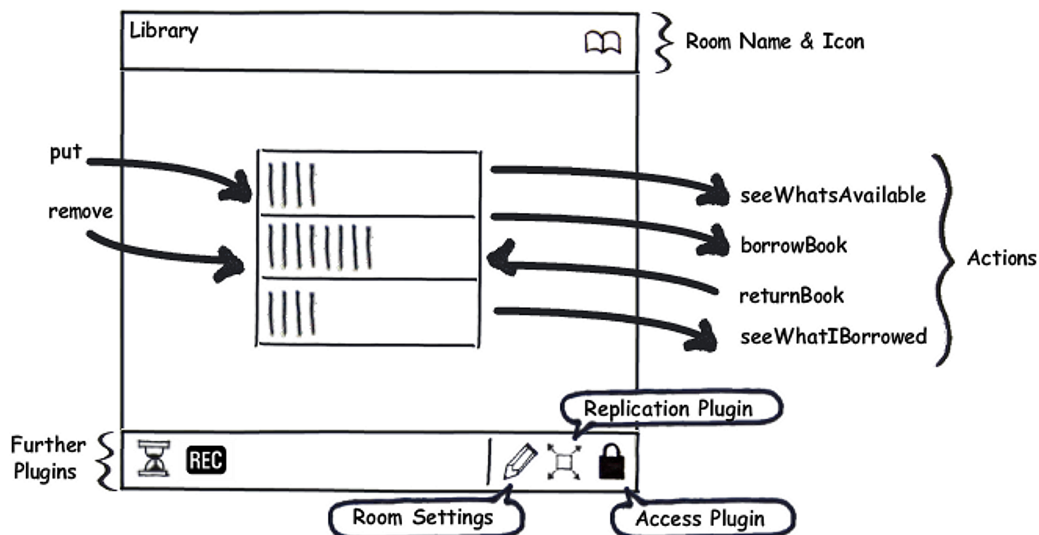


Figure 3.5: Micro-Room Example [Bin13]

The basic structure of a micro-room consists of a unique name, a clearly assignable graphical illustration, a set of room-specific properties (`Room Settings`), a set of actions (e.g. `put`) and a set of plugins (e.g. `Access Plugin`).

The graphical illustration helps end users to recognize each room type right away. Each room type can define arbitrary custom properties, in case of the library this could be e.g. the max time for which one is allowed to borrow a book. Actions represent all interaction possibilities within the room and can either be internal (not illustrated), incoming (e.g. `put`) or outgoing (e.g. `borrowBook`). Finally, the base functionality of the room can be extended by plugins, which can be placed on arbitrary room types. E.g. the `Access Plugin` allows to configure user- or role-based permissions for each action whereas the `Recorder Plugin` allows to protocol every call (user and timestamp) of a definable action. With the `Replication Plugin` it can be configured to which users or groups the room (i.e. the contained data) should be replicated.

Each room can be connected to other rooms by connecting outgoing to incoming connections with matching parameters. Thereby, only one-to-one connections are allowed to keep the model simple and understandable by end users.

What makes micro-rooms and plugins special is that they are fully integrated, i.e. they are implemented by IT experts and contain all the application logic. Each micro-room and plugin can be created by implementing a simple Java interface respectively (cf. [Bin13]). Also, they are very modular and extensible, as every IT expert can create their own set of micro-rooms and plugins with very low effort. Also, in the best case, the end user itself can create his/her functionality needed by extending a micro-room with a plugin.

End users can simply create a model of their needs by configuring and connecting several micro-rooms. The big advantage is that the thereby created model (i.e. “scenario”) can be executed by the corresponding runtime environment (i.e. the `XVSM Micro-Room Framework`) to start up a fully executable P2P application, tailored exactly to the needs of the modeling end user.

3.1.4 XVSM Micro-Room Framework

The `XVSM Micro-Room Framework` has been created in [Bin13]. It is a runtime environment that is based on `XVSM` and capable of running scenarios containing micro-rooms. Its functionality will be explained based on the `XVSM Micro-Room Framework` component view shown in Figure 3.6.

An application based on the `XVSM Micro-Room Framework` requires three types of input files:

1. A `Config File` specifying basic settings such as name of the user, location of module and business logic files and the URI of the central peer discovery server.
2. One or more `Module Files` (i.e. *Java Archive* (JAR) files) containing the implementation of all micro-rooms and plugins required.
3. One or more `Business Logic Files` (i.e. XML files) representing the micro-room workflow models (“scenarios”) defined by the user, containing all micro-room and plugin configurations as well as all action connections.

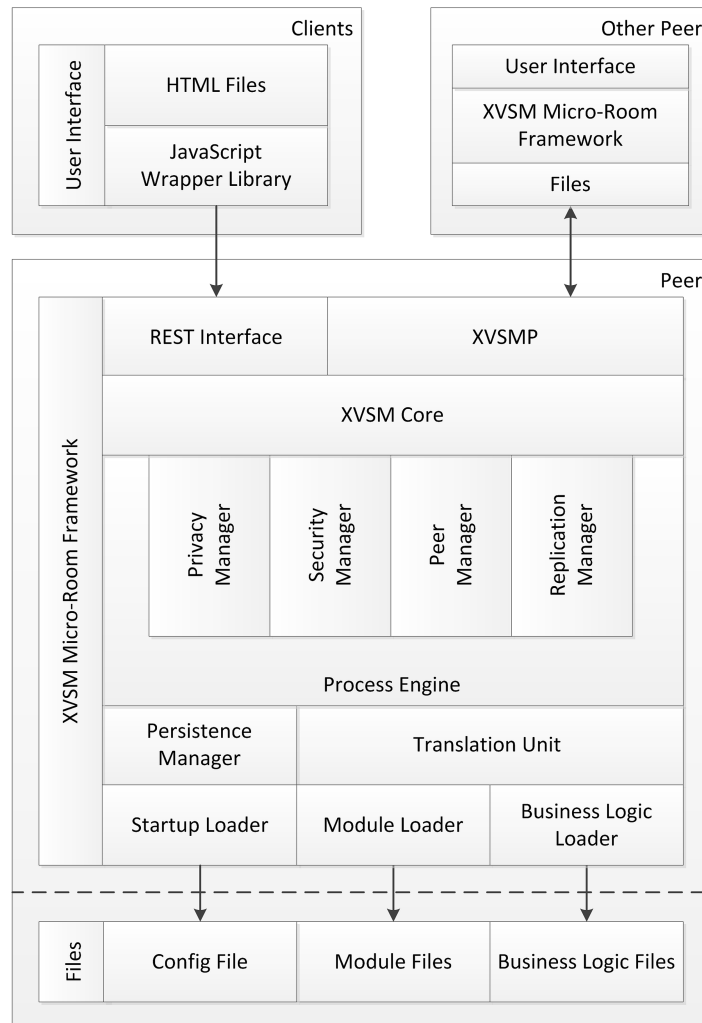


Figure 3.6: XVSM Micro-Room Framework Component View [Bin13]

These files are interpreted by the XVSM Micro-Room Framework on startup and thereby result in a fully functional decentralized P2P application (backend). As can be seen in Figure 3.6, the XVSM Micro-Room Framework handles a lot of a P2P application's boiler plate code, including replication, security and peer coordination.

Communication with other peers is based on replicated XVSM container operations directly, where each peer hosts the rooms that are replicated to him/her (cf. *Replication Plugin*). The communication with the UI is handled via *Representational State Transfer* (REST) interface which can be used to e.g. call actions of micro-rooms. To ease the access of these REST calls, a simple JavaScript wrapper library has been implemented in [Bin13].

Relation to the Peer Model

When looking at the other concepts presented, the XVSM Micro-Room Framework can be seen as a simplified version of the Peer Model that uses similar concepts. It has a reduced feature set but hence also reduced complexity. The Peer Model allows developers to model processes exactly, whereas the XVSM Micro-Room Framework hides most of the complexity by providing a set of fully defined micro-rooms.

When comparing terminologies with the Peer Model, a micro-room can be seen as a peer and its actions as wirings. The internals of a micro-room are not modeled with further wirings such as in the Peer Model, but with program code provided by IT experts directly. Wirings between micro-rooms have no guards but trigger immediately upon call.

Also there are no PIC and POC on a per-peer-basis but only one global PIC (i.e. the request container) and one global POC (i.e. the response container) of the XVSM Micro-Room Framework itself, handling all incoming action calls and outgoing action results. The micro-rooms select only their relevant entries by `QueryCoordinators`. Connections between actions are only allowed one-to-one, compared to arbitrary wiring possibilities in the Peer Model.

Room-in-a-room usage is not possible while in the Peer Model one can re-use peers as sub-peers in any other peer. However, in the XVSM Micro-Room Framework this drawback can be partly overcome with plugins. They represent some kind of sub-peer that can be used within arbitrary micro-rooms.

To outline the relation between the two concepts, Figure 3.7 shows how an `Archive` micro-room could look like in the Peer Model.

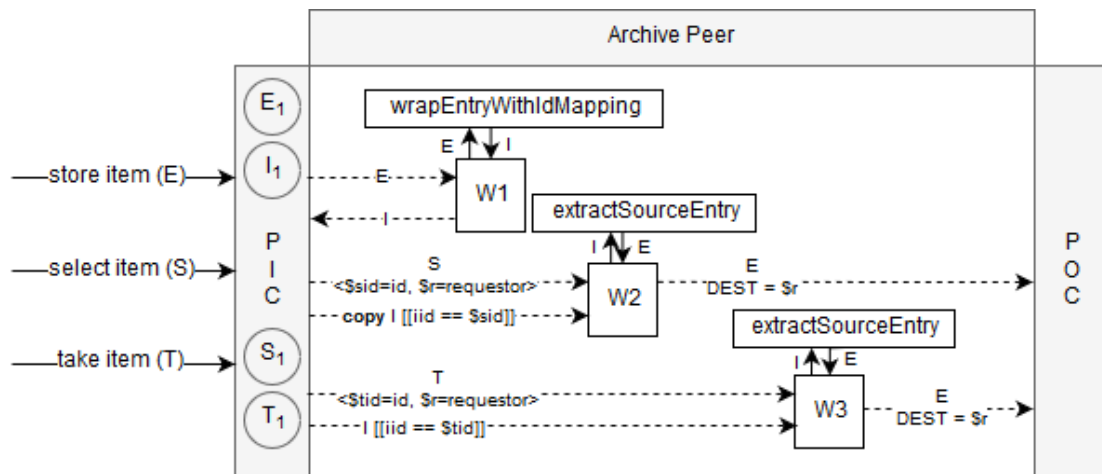


Figure 3.7: Archive Micro-Room Modeled with the Peer Model

When storing an item, it is encapsulated in an entry of type `E` and written to the archive peer's `PIC`. This triggers wiring `W1`, calling a service function that converts the entry of type `E` to an entry of type `I`, containing an additional mapping from its internal ID (application property provided by the XVSM Micro-Room Framework) to the source entry, e.g. via adding a `KeyCoordinator`. This entry is then written back to the `PIC`.

If one wants to select an item, an entry of type `S` is written into the `PIC`. Thereby, two local variables are set, i.e. `$sid` storing the ID of the item to be selected and `$r` containing the requestor's peer container. More about local variables in the Peer Model can be found in [Küh16]. This entry triggers wiring `W2` only if there is already an `I` entry in the `PIC` having the same ID (field `iid`) as requested in the `S` entry, i.e. there is already an item in the archive with the queried ID. Please note that the `iid` does not need to be known by the end user directly, since it is encapsulated in the items that the he/she is working with in the UI. If the user selects an item (e.g. by title or text), the internal id of the item is used as `iid`.

Please also note that the `I` entry is copied before it is taken out of the `PIC`, so that other select actions of the same item ID still can trigger wiring `W2`. Wiring `W2` then calls a service function that strips off the previously added ID mapping and writes the entry via `DEST` property to the requestor's peer container `$r`.

Taking an item works exactly as selecting an item, but this time, the `I` entry that has to be in the `PIC` already will not be copied. Thus, it will be taken out and no other select or take action for this ID can trigger wiring `W2` or `W3`. Also, for `S` and `T` entries there has to be defined a `TTL` to avoid that they stay in the `PIC` forever in case there will never exist a corresponding `I` entry.

As can be seen, the coordination logic of micro-rooms could be modeled via the Peer Model without problems. However, since we target end user modeling in our approach, internal modeling of micro-rooms as would be possible with the Peer Model is not required. Moreover, micro-rooms act as black boxes for end users to hide their internal complexity. Therefore, it is sufficient to implement them in Java directly (one class per micro-room) in our approach.

Current Drawbacks

Currently the *XVSM Micro-Room Framework* encounters two main drawbacks. First, the model required for the runtime environment needs to be specified in plain XML, which is not very intuitive for an end user.

The second drawback is that UIs are currently neither provided by the *XVSM Micro-Room Framework* itself (e.g. by automatically deriving them) nor by the module files (e.g. by including them). Hence, the created application represents only a fully working backend, allowing no direct interaction with the end user him-/herself.

3.1.5 Comparison to Related Work

Since the *Peer Model* (PM) and the *XVSM Micro-Room Framework* (XMRF) neither are classical modeling techniques nor target end users directly, they have not been incorporated into the comparison shown in Figure 2.20. Nevertheless, an evaluation regarding the requirements specified in Section 2.3 is of interest.

Automation Support: The Peer Model partly enables automation support since the coordination logic of the model could be translated to executable code automatically. However, the logic from the called service methods needs to be implemented by IT experts after modeling. This is similar to UML, where empty, connected classes can be generated automatically, which still need to be extended manually. In its current state, the Peer

Model cannot be compared to WS-BPEL, where service endpoints exist autonomously and can be queried and called during creation of the model. Moreover, service methods need to exist as methods in code, i.e. they are tightly coupled to the model itself.

The XVSM Micro-Room Framework provides full automation support by providing a proper business logic XML file using a set of available modules.

Human Task Support: Currently, no automatically generated form fields for user interaction are available in the Peer Model. To enable user interaction with a model during runtime, special service methods could be implemented that block the workflow until a user e.g. enters a form value, but this is insufficient according to the criterion definition.

Similarly, the XVSM Micro-Room Framework provides no UI automatically and hence users have no possibility to interact with the model during runtime without an IT expert creating a proper UI.

Distributed Execution: Since the Peer Model relies on XVSM containers for exchanging entries between PICs and POCs of different peers, it inherently supports distributed execution.

The XVSM Micro-Room Framework also relies on XVSM and uses containers to replicate executed actions between users. Hence, each user can execute his own part of the model and interact with the other parts without problems.

Multiple Domains: As the Peer Model is a general purpose framework for specifying arbitrary coordination patterns, it allows to model any domain.

The XVSM Micro-Room Framework can only model what is possible with the currently available modules (i.e. micro-rooms and plugins). However, since the framework can be extended by custom modules very easily, it can be stated that it also allows to model multiple domains.

Tool Simplicity: For the Peer Model, currently a modeling tool is under development but it is not yet finished. For the XVSM Micro-Room Framework, currently no modeling tool exists. Thereby, both fail the requirement at this moment.

Comprehensible Abstraction: The Peer Model targets IT experts and therefore its modeling concepts are very fine-grained. I.e. the modeling user needs to deal with several technical terms such as guards, variables or queries. Hence, the Peer Model cannot fulfill this criterion.

Currently, the XVSM Micro-Room Framework requires a business logic file in XML format as input without any tool supporting the creation of this file. Regarding this criterion, it is therefore exactly on the same level as WSCF, which is literally “*unacceptable*” (cf. Section 2.3).

Easy Deployment: Concerning the Peer Model, currently there is no easy way for end users to automatically run their models.

The XVSM Micro-Room Framework on the other hand allows to execute the created models easily by simply starting up the framework which then derives the logic to execute based on the provided model (i.e. business logic XML file).

The comparison of modeling techniques (cf. Figure 2.20), now including also both the Peer Model and the XVSM Micro-Room Framework, is visualized in Figure 3.8.

		UML	BPMN	WS-BPEL	ISEA	HBPM	WPM	FWSC	WSCF	EUPM	PM	XMRF
Features	Automation Support	⚠	✓	✓	✗	✗	✓	✓	✓	✓	⚠	✓
	Human Task Support	✗	✓	✓	✗	✗	⚠	✓	⚠	✓	✗	✗
	Distributed Execution	✗	✓	✓	✗	✗	⚠	✓	✓	⚠	✓	✓
	Multiple Domains	✓	✓	✓	✓	✓	✓	✓	✓	✗	✓	✓
End User Suitability	Tool Simplicity	✗	✗	✗	✓	✓	✓	⚠	✗	✓	✗	✗
	Comprehensible Abstraction	✗	✗	✗	⚠	✓	✗	✗	⚠	✓	✗	✗
	Easy Deployment	✗	⚠	✗	✗	✗	✓	✗	✗	✓	✗	✓

Figure 3.8: Comparison of Modeling Techniques including PM and XMRF

3.2 Initial Draft

In [Ver04] Verner states: “*What is missing is a seamless way to integrate design and development.*” In our approach, we try to achieve such a way by creating a new modeling tool meeting all of our defined acceptance criteria, in particular: the XVSM Micro-Room Modeler.

Our aim is to create a simple and intuitive modeler with which end users can model collaborative workflows based on the micro-room concept. This model can be translated by the modeler to a business logic XML file, readable by the XVSM Micro-Room Framework, thereby fixing the first drawback of the framework.

To fix the second drawback of the framework, UIs are added into the module files, hence generating a fully executable application on startup. To be flexible, the end user can also upload arbitrary module JAR files containing additional micro-room and plugin types as needed.

Finally, after completing all configuration and modeling tasks, the end user can create his/her modeled application by clicking a single button. In the background, the model translation and further tasks occur fully automatically, yielding an executable application based on the XVSM Micro-Room Framework.

In Figure 3.9 an initial draft of the XVSM Micro-Room Modeler is shown. As can be seen, the UI is very simple and intuitive. The user has the possibility to extend the modules available and to configure all micro-rooms as within the XML business logic file (cf. [Bin13]).

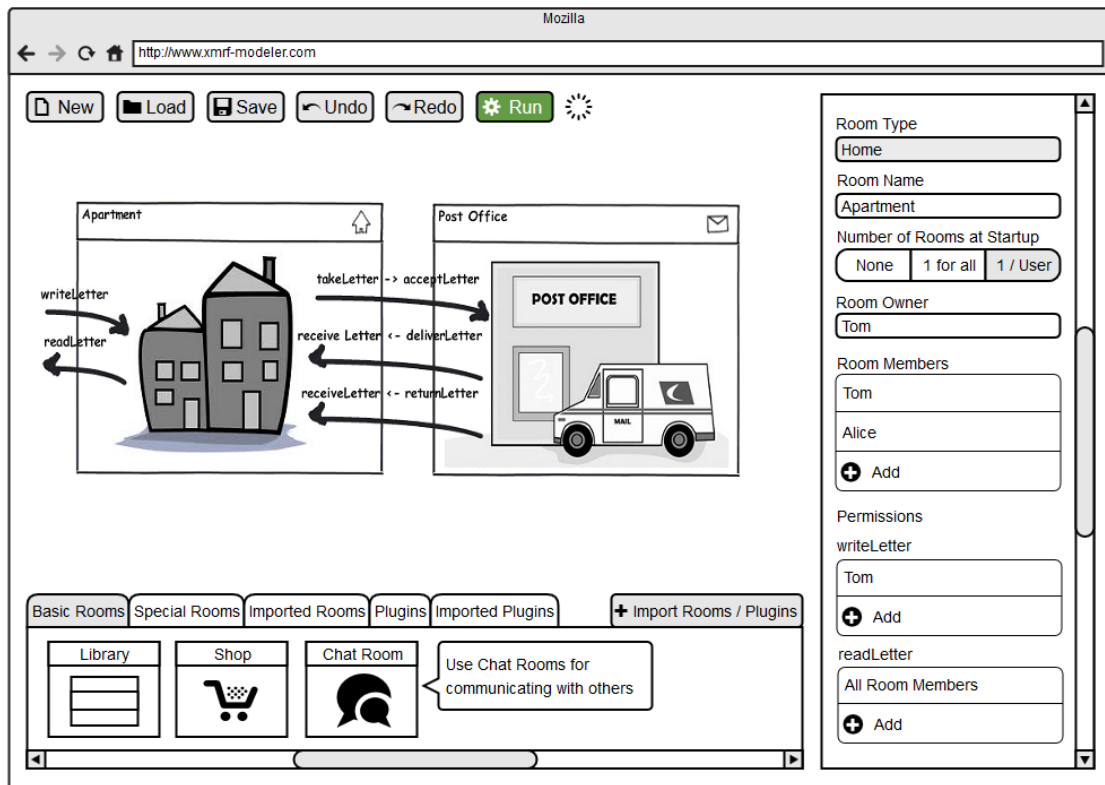


Figure 3.9: XVSM Micro-Room Modeler Draft

The stated modeler combines the best of all end-user-related approaches discussed in Chapter 2.2. The micro-rooms and plugins in the modeler use intuitive graphical illustrations to gain acceptance by the end user such as HBPM does. As in WPM, it has building-blocks with integrated automation support (i.e. micro-rooms and plugins), created by IT experts but usable by end users. As another contribution of this work, the custom configuration properties for micro-rooms and plugins come with reasonable default values, just like in FWSC. Finally, the XVSM Micro-Room Framework uses an “inference engine” to deduce the logic from user input (i.e. it interprets the generated business logic file to execute logic), similarly to WSCF.

After describing the related concepts and our approach, we will discuss its architecture in more detail in the next chapter.

Design

Based on the theoretical concepts presented in Chapter 3 we will now describe the technologies used to realize our modeling tool as well as its general architecture.

4.1 Evaluation of Technologies

First of all it has to be defined with which technology the modeler should be implemented. Therefore, we will define the requirements, discuss several approaches and evaluate which of them meets the requirements best.

4.1.1 Requirements

On the basis of the acceptance criteria for a decent modeling technique targeting end users presented in Chapter 2.3 we derived the following subset of requirements for a technology of a decent modeler targeting end users. All of these requirements directly influence the user experience of the end user.

Low Entry Level: The entry level of the modeler is defined by the amount of technical hurdles an end user needs to overcome for accessing the modeler. Thereby, this requirement is directly related to the technology used, e.g. because the end user needs to install an application or a library before the modeler itself can be started.

Tool Simplicity: In contrast to the entry level, which deals with the user experience before using the modeler, tool simplicity is related to the actual user experience when using the modeler. Thus, the technology used should allow the modeler to be as simple as possible, so e.g. no complex menus or technical terms should be necessary.

Easy Deployment: Converting the model to a XVSM Micro-Room Framework-compliant business logic file and packaging it into an executable application has to be possible with the technology used.

Extensibility: The XVSM Micro-Room Framework is highly extensible by simply providing new module files in a pre-defined folder before startup. Hence, also the modeler should allow this extensibility, i.e. it should be possible that the end user adds module files during runtime to the modeler, thereby extending its provided modeling elements.

4.1.2 Eclipse Modeling Framework

The first technology evaluated is the *Eclipse Modeling Framework* (EMF)¹. It is a modeling framework that allows to create your own modeling language, based on your `Domain Model`. Therefore you specify your model in terms of Ecore, which is the core meta-model of EMF. Additionally, EMF is capable of code generation, at least in terms of Java class stubs that need to be implemented by IT experts after creation.

In combination with the *Graphical Modeling Framework* (GMF)² modelers can be created based on these Ecore models. Figure 4.1 shows the components required to create the modeler.

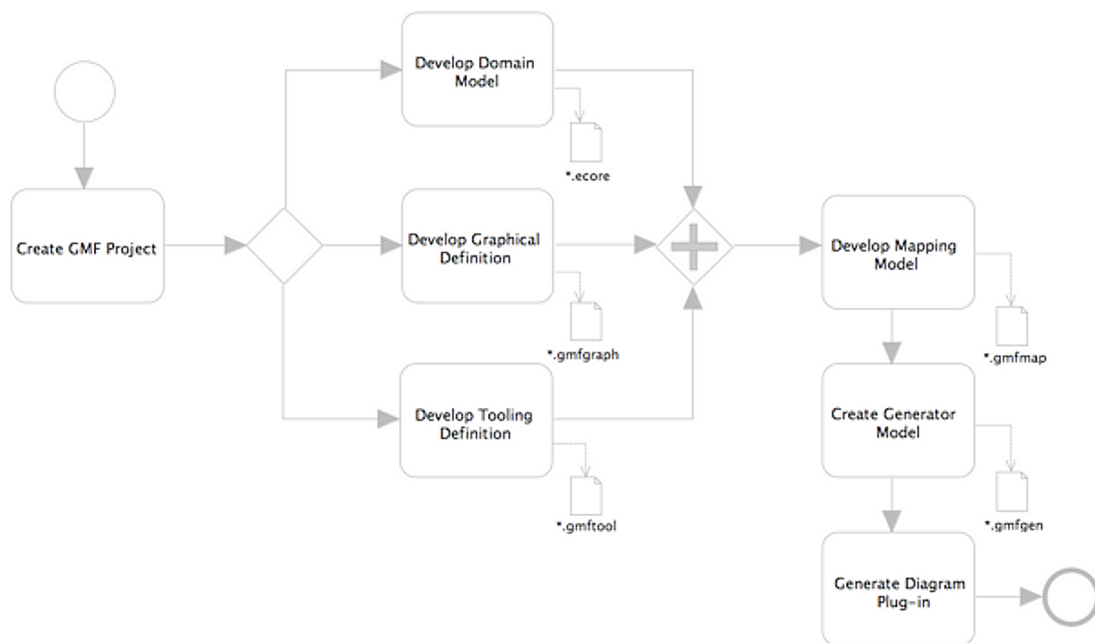


Figure 4.1: GMF Overview [9]

The `Graphical Definition` contains all information about the graphical elements available in the modeler, i.e. elements that can be used in the modeler pane. With the `Tooling Definition`, all other UI-related aspects of the modeler can be defined, e.g. menus and tool-bars.

In the `Mapping Model`, `Graphical` and `Tooling Definitions` are linked to the `Domain Model`. In the `Generator Model`, implementation details can be defined for the

¹<http://www.eclipse.org/modeling/emf/>

²<http://www.eclipse.org/modeling/gmp>

generation phase. Finally, the created Diagram Plug-In can be executed by the GMF runtime.

As can be seen, it is a rather complex task to get the modeler set up. Also, the modeler created will be based on the Eclipse IDE, hence containing a lot of additional menus, not required for the modeler in the first place (cf. Figure 4.2).

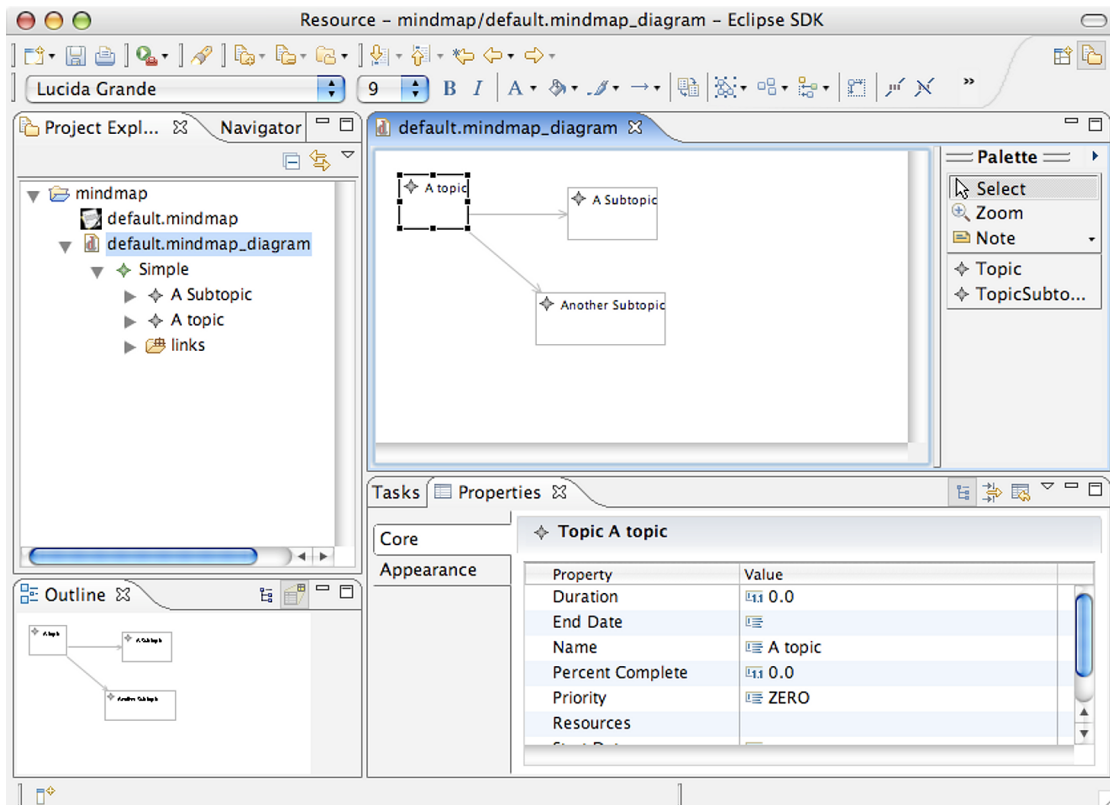


Figure 4.2: GMF Modeler Example [10]

To overcome this problem, a special GMF Lite Runtime [11] exists which does not contain any toolbars and reduces the Eclipse-IDE-specific features to a minimum. Nevertheless, the runtime has to be installed and a special launch profile has to be created and executed to launch the modeler.

Considering code generation, the code templates can be adjusted, so that a micro-room might be specified as a code template generating the underlying Java class. However, neither can the model be extended during runtime, nor can the generated modeler package a fully executable application.

4.1.3 Microsoft Visio

Microsoft Visio is another famous modeling tool that can be extended by custom elements. When looking at the SDK documentation [12] it becomes clear that you can create a custom set

of elements usable in Visio easily. Also, further elements (i.e. micro-rooms) could be imported during runtime without problems. Figure 4.3 shows Visio with a set of custom shapes available.

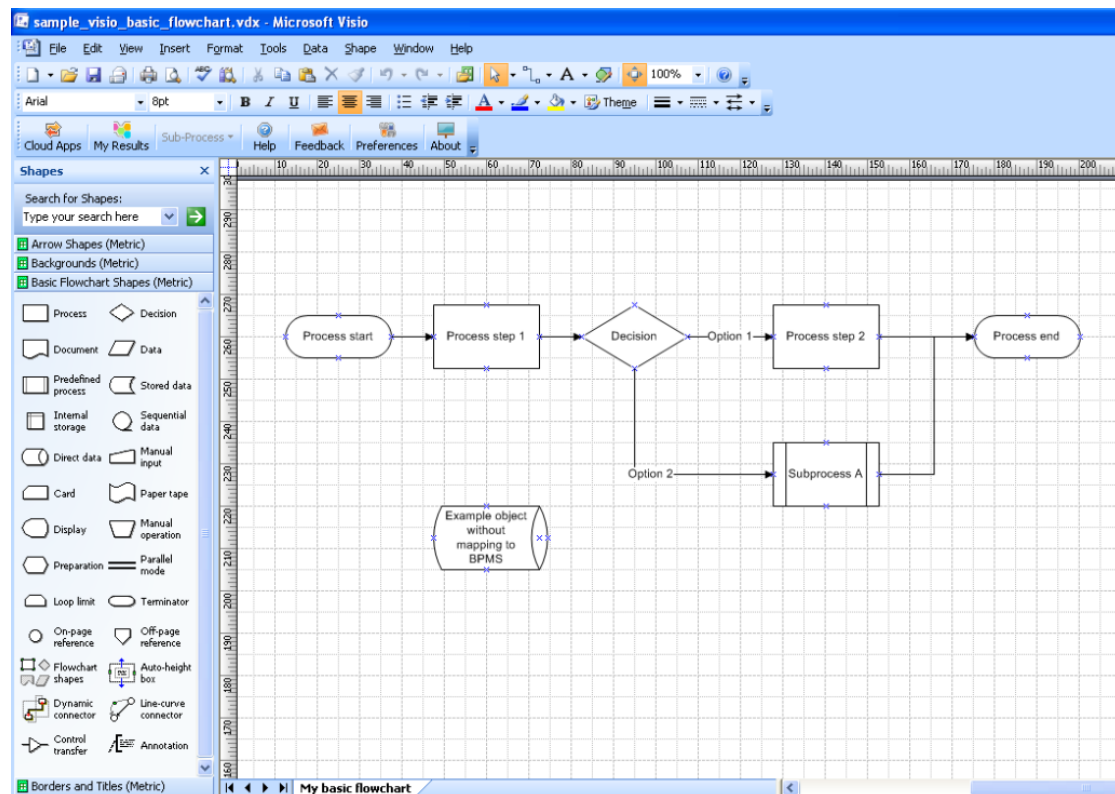


Figure 4.3: Visio with Custom Shape Library [13]

As can be seen, the UI is fairly simple and appropriate for end users. However, the big drawback is that with Visio you can neither create the business logic file in the desired format nor compose a fully executable application package.

4.1.4 Java Client Application

Another approach would be to implement the modeler completely from scratch as a client application in an arbitrary programming language. In this case, Java is chosen due to personal preferences. Thereby, the end user would need to download our application and the required runtime environment in order to access our modeler.

Concerning the usability of the modeler, the creation of the business logic file from the model, the extensibility during runtime and the packaging of the executable application, no limitations are present.

4.1.5 Java/AngularJS Web Application

The modeler could also be developed as a web application directly in any suitable programming language. As an example a Java backend could be extended with AngularJS³ on the frontend to create a client-application-like user experience. In contrast to the client application approach, the user does not need to download anything but can directly access the modeler with his/her browser on any platform.

Just as with the client application, there are no limitations concerning usability, business logic file creation, extensibility and generation of the executable application.

4.1.6 Comparison & Decision

Figure 4.4 shows the final comparison of all technologies discussed.

















	EMF	Microsoft Viso	Java Client Application	Java/AngularJS Web Application
Low Entry Level				
Tool Simplicity				
Easy Deployment				
Extensibility				

Figure 4.4: Comparison of Technologies

EMF looked promising at the first glance but is not suitable for our scenario. Its entry level is too high for end users, since they need to download the GMF runtime environment, create a specific launch profile for our plugin and execute it. Regarding tool simplicity the lite GMF runtime environment is mandatory for end users, because the regular runtime environment is far too complex. Nevertheless, the UI will always rely on the structure of the Eclipse IDE, thus limiting its usability. The EMF is intended for class generation, hence the creation of the business logic file from the model and the packaging to an executable application afterwards is not possible. Concerning extensibility, there is no way to add new graphical elements (i.e. micro-rooms or plugins) during runtime of the modeler.

Microsoft Viso makes it a bit easier for end users to access it. They need to install the software and import our shapes (i.e. micro-rooms and plugins). The tool itself is designed for end users and thus sufficient regarding this requirement. It is also extensible during runtime as the end user can simply import new shapes at any time. However, all of these benefits come with a main drawback: Converting the model to a business logic file and creating the executable application is very complex and error-prone due to the proprietary modeling format and limited documentation, hence rendering this approach useless.

³<https://angularjs.org/>

When creating a Java client application (e.g. in JavaFX), one is not limited in any concerns. The modeler can be designed completely from scratch, thus allowing to build it as simple as possible. Transformation of the created model to a business logic file and the creation of an executable application afterwards are no problem. However, Java client applications have one drawback: They are not accessible easy enough for end users, as they first need to download the application and then have to install the *Java Runtime Environment* (JRE) for the application to work. Also, they are only accessible on *Personal Computers* (PC) but not on other platforms, such as tablets or smartphones.

Finally, a web application using Java for the backend provides all the benefits of a Java client application if e.g. AngularJS is used for the frontend. With it a client application-like user experience can be created. Additionally, the web application can be accessed very easily by end users via their browsers. Thereby, they could even use a tablet or smartphone to access the modeler which is not possible with any of the other approaches.

As can be seen, a Java/AngularJS web application suits our needs best and hence these technologies are used to implement the modeler. Please note that regardless of the technology used for the modeler, the generated executable application will always require a JRE installed since it is based on the XVSM Micro-Room Framework, which itself uses Java.

4.2 Components

After defining the technologies for creating the XVSM Micro-Room Modeler, its structure is explained by the component diagram shown in Figure 4.5.

As can be seen the modeler is separated into two parts, i.e. the `Client Part` (frontend) and the `Server Part` (backend). The client part is executed in the browser of the modeling user and thus contains only *Hypertext Markup Language* (HTML), *Cascading Style Sheets* (CSS) and JavaScript. The required JavaScript is created with AngularJS as a basis which separates the code into controllers and services. `AngularJS Controllers` contain all browser-related logic, e.g. manipulating the *Document Object Model* (DOM), validating user input or listening to events such as `onclick`. `AngularJS Services` get called by controllers and execute *Asynchronous JavaScript and XML* (AJAX) calls to the backend, perform error handling and response parsing.

The backend called is implemented in Java as stated in the previous chapter. To avoid a lot of boilerplate code, we use the Spring Boot Framework⁴ as a basis. It launches an embedded Tomcat web server upon application startup which provides the REST endpoints (i.e. `REST Controllers`) of the backend. These endpoints get called by the `AngularJS Services`. When called, a `REST Controller` performs request parsing and validation and forwards data to the corresponding `Backend Service` containing the business logic.

Beside other services, there are three of higher importance: The `OpenAM Service` is used to communicate with the central `OpenAM`⁵ instance required for our approach. `OpenAM` is an access management system that provides authentication, authorization, single sign-on and a lot of other capabilities. In our work it is used in three places: First, when a user registers an account

⁴<https://projects.spring.io/spring-boot/>

⁵<https://forgerock.org/openam/>

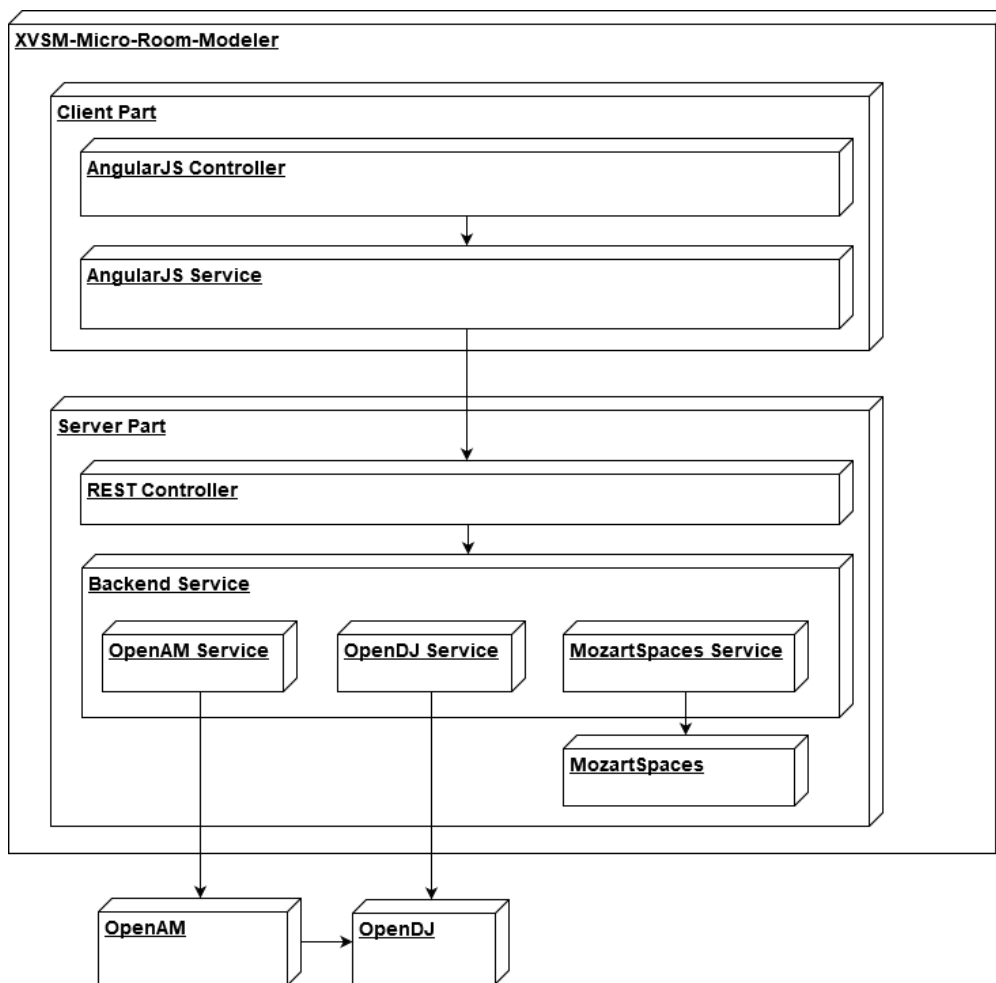


Figure 4.5: Component Diagram of the XVSM Micro-Room Modeler

for the XVSM Micro-Room Modeler, his/her credentials are added to OpenAM. Second, when logging into the modeler, the user credentials are validated against OpenAM. Third, when using the created application based on the XVSM Micro-Room Framework, the user needs to log in after startup. Thereby his/her account credentials are verified via OpenAM returning an access token, which can then be used by the XVSM Micro-Room Framework to enforce all XVSM-Security-related permissions (cf. Section 3.1.1 and [Bin13]).

Additionally, the `OpenDJ Service` allows to access the central `OpenDJ`⁶ instance directly. `OpenDJ` is an open source *Lightweight Directory Access Protocol* (LDAP) server that is used by `OpenAM` to store user and role information. Since for XVSM Security a special `memberof` attribute of the user is required for specifying his/her roles, direct access to `OpenDJ` is required, as with `OpenAM`'s REST API this attribute cannot be set. In the XVSM Micro-

⁶<https://forgerock.org/opensdj/>

Room Modeler a user has to be able to configure the other members and roles that should be able to access parts of his/her collaborative P2P application. When doing so, these roles are written into OpenDJ via the OpenDJ service so that they can be accessed via XVSM Security when using the generated application later on.

Finally, the `MozartSpaces Service` handles all modeler-related data and stores it in a persistent embedded `MozartSpaces` instance. Such data would be e.g. information about which user has created which project, project settings and the created models themselves.

4.3 Modeling

In this section the feature set of the XVSM Micro-Room Modeler regarding modeling is described. End users are able to register an account and log in. They can create new projects, save them within the modeler and load them later on. For each project, a set of persons and groups is definable upon project creation and modifiable later on. Those persons and groups are allowed to participate in the whole collaborative workflow or parts of it.

The workflow itself is modeled with a set of micro-rooms that can be connected with each other. To create a new micro-room, it can be dragged from the panel showing all available micro-rooms on the bottom to the central modeler pane (cf. Figure 3.9). After clicking on a micro-room, the properties panel appears to the right, which allows to configure common properties like its name, replication and security settings, as required by the XVSM Micro-Room Framework. These cover which persons/roles are allowed to enter a room and if so, who is allowed to perform which actions. Also, custom micro-room properties can be configured in this panel, i.e. properties that are unique per micro-room type.

Connections can be created by dragging an action port of one micro-room to a fitting action port of another micro-room. Thereby, fitting target actions need to have an input parameter with a type equal to the return type of the source action. To support the end user, fitting ports will be highlighted automatically.

Every micro-room can be extended with arbitrary plugins to enable additional functionality. Therefore, a plugin can be dragged from the panel showing all available plugins on the bottom to a micro-room in the central modeler pane. The plugin can be configured in the properties panel to the right after clicking on the micro-room it is assigned to.

Additionally, every micro-room is assigned to a UI template by default. This template specifies how to render the micro-room in the created application.

To show the possibilities of the XVSM Micro-Room Modeler, a base set of micro-rooms and plugins is provided with the modeler. It covers everything an end user needs to model the tasks required for the usability testing of this work. Those tasks address simple collaborative workflows ordinary end users might want to perform with their friends and are described in Section 7.2.1.

In more detail, the implemented micro-rooms are:

Archive: Represents a room where items can be stored, selected and taken. Thereby, an item can contain arbitrary information and/or represent a file.

Conference Room: A room in which all room members can chat with each other. Messages are transmitted in realtime between all members with a guaranteed ordering (cf. XVSM Micro-Room Framework replication based on distributed transactions [Bin13]).

Typing Room: Allows room members to write a document in a *What You See Is What You Get* (WYSIWYG) editor. Currently, only one member can edit the same document at a time but it could be extended to a fully-functional collaborative editor if required.

Mail Room: This room can be used for members to send mails across the world.

Decision Room: In the decision room, members can manually decide upon items and thereby specify what should happen with them next, e.g. whether they should be forwarded into another room or discarded.

Thereby items can contain arbitrary payload data and might be parameterized by the user to define which data they should be allowed to contain. However, for the tasks in the usability testing it is sufficient to fix the underlying payload to files, i.e. `byte` arrays. Also, with files, generality remains in some form and many use cases can still be modeled.

Additionally, two plugins are implemented which allow to extend the given micro-rooms with further functionality:

Recorder: Allows to record all calls of a specified action. Also provides an additional action to the micro-room with which the recordings (i.e. user name and time) can be viewed.

Notifier: Can be used to notify the user if a specified action has been called, thereby causing a page refresh of the active micro-room's UI.

Figure 4.6 shows the graphical notation of the seven modules in the XVSM Micro-Room Modeler.

Besides the two mentioned plugins, in the XVSM Micro-Room Framework the following two plugins are of major importance since they are responsible for core features. Therefore they are already included within the XVSM Micro-Room Framework itself instead of being provided by separate module files. A micro-room could be executed without them but it would not make much sense to do so. Therefore, in the XVSM Micro-Room Modeler both of these plugins are always added to each room automatically and thus will not be indicated with special icons on the corresponding micro-room. They can be configured by the end user via the micro-room's properties panel.

Security: Enables to configure the permissions of each action in the micro-room on a user or group level.

Replication: Allows to specify to which other peers the actions called on the micro-room should be replicated on a user or group level.

With these modules an end user can create e.g. the following collaborative workflows as outlined:

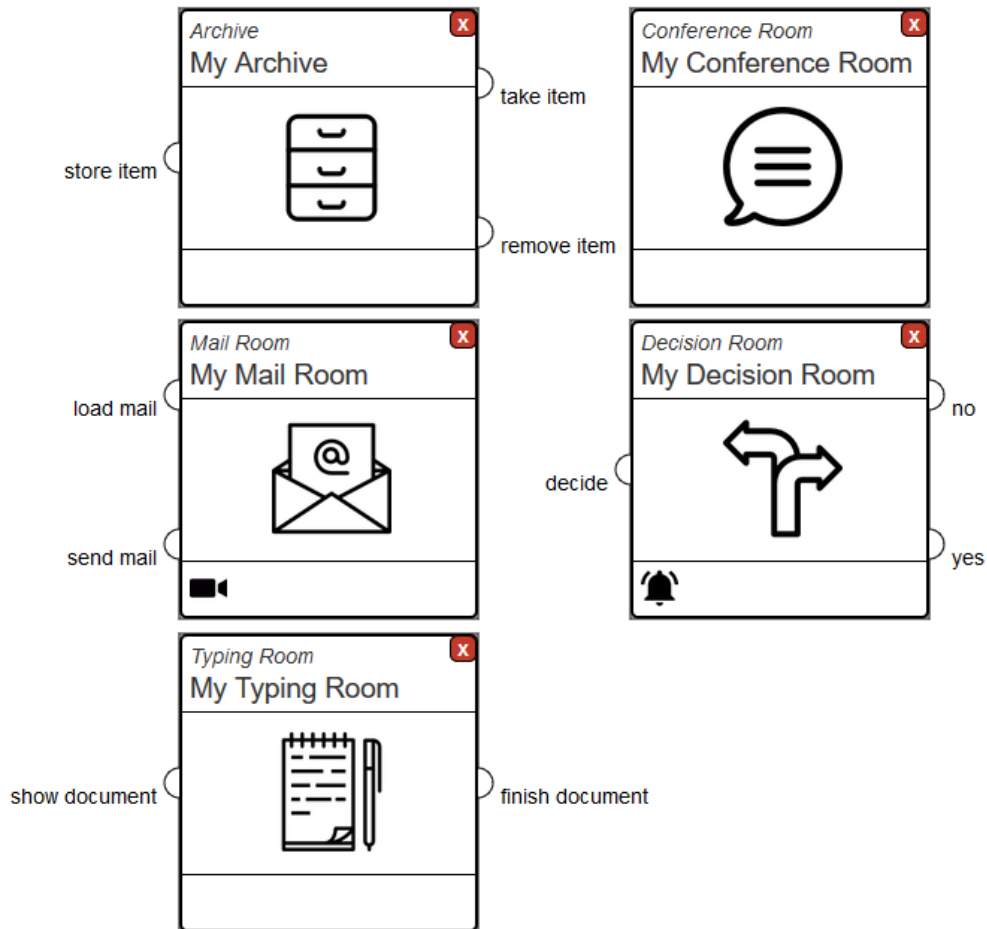


Figure 4.6: Graphical Notation of the Provided Modules

Chatting: Use a single `Conference Room` shared with your friends.

Message Boards: Use a `Conference Room` with an alternative UI template (“Message Board”) configurable via room properties.

File Sharing: Provide a single shared `Archive`.

Mailing Lists: Define a `Typing Room` for creating the mails, an `Archive` for storing them and a `Mail Room` for sending them. Items flow via connections from the `Typing Room` (`finish document`) over the `Archive` (`store item`) to the `Mail Room` (`send mail`).

Document Editing: Specify a `Typing Room` for creating documents and an `Archive` for storing them. Connect them with each other to store documents finished in the `Typing`

Room (finish document) into the Archive (store item) and to load documents stored in the Archive (take item) into the Typing Room (show document) for further editing.

Task Assignment: Use one Archive per user (permitted to the corresponding user only) and connect them to a set of Decision Rooms. By deciding upon an item of his/her own Archive, a user can forward it to the Archive of the desired user.

Reviews: Define a Typing Room for creating documents and connect it to an Archive for storing them. Connect the Archive (take item) to a Decision Room (decide) that is permitted to the reviewers only. Connect its yes action to another secured Archive containing all passed documents and its no action back to the first Archive's store item action.

Brainstorming: Specify either a Conference Room, a Typing Room or an Archive. With either of these rooms, users can add ideas dynamically so that other users can see them. When using an Archive, additionally a Notifier plugin can be added so that users immediately see new ideas of other users due to automatic page reload.

Profile Pages: Provide a Typing Room to add new text items to an Archive with an alternative UI template ("Show Full Item Text") configurable via room properties. This template renders the full item content instead of only the title. Allow access to the Typing Room and to the Archive's modifying actions only to your friends.

Blogs: Use a private Typing Room for creating HTML items and store them into an Archive with an alternative UI template ("Show Full Item Text") configurable via room properties.

Newsfeeds: Provide a private Typing Room to write news items and store them into an Archive with an alternative UI template ("Show Full Item Text") configurable via room properties.

Wikis: Define a Typing Room for creating wiki entries in HTML and store them into an Archive. Create a connection from the Archive to the Typing room to view and edit wiki entries. Add a Recorder plugin to the Archive's store action to record which user modified an wiki entry. Searching will not be possible but could be added with new Search plugin.

Shopping Lists: Use an Archive with an alternative UI template ("Simple List") configurable via room properties. Instead of up- and downloading files, simple text entries can be stored and viewed directly.

Wish Lists: Specify an Archive with an alternative UI template ("Simple List") configurable via room properties. Instead of up- and downloading files, simple text entries can be stored and viewed directly.

As can be seen, from a backend-based point-of-view, many use cases can be realized with the same room types but only differ in their frontend presentation. Please note that even though alternative UI templates could be specified without problems (cf. Section 5.4.5), they are not implemented in this work as they are not required for the tasks in the usability testing. They will be added in future work (cf. Section 8.6).

Beside the implemented modules, the following micro-rooms have been defined as well but are not implemented in this work:

Poll Room: A simple poll room, allowing members to answer questions that have been defined in advance.

Video Room: Allows members to stream videos between each other by e.g. using direct communication over *User Datagram Protocol* (UDP) instead of *XVSM Protocol* (XVSMP).

Please note that not using XVSM as the underlying technology does not influence the function of plugins that are applied to the room. E.g. the `Security` plugin will still work since the actions themselves are registered in a secured container by the XVSM Micro-Room Framework using XVSM Security. Also, the `Replication` plugin will still work as the XVSM Micro-Room Framework internally replicates called actions via XVSM Distributed Transactions, regardless of the technology used within the corresponding action.

So e.g. a `startVideo` action could be called with an URI of the video as parameter. This action can be secured and replicated without problems by using XVSM. Whether the action internally starts a video stream using UDP or any other protocol does not matter as its logic will be executed on each peer individually after receiving the replicated action call.

Playground: Provides various types of games that room members can play together. The type of game it shows could either be specified via room properties or the micro-room could be split into separate micro-rooms, i.e. one per type.

By extending the implemented modules with the additional micro-rooms, even more use cases could be modeled, such as:

1. Appointments
2. Online Exams
3. Video Conferences
4. Webinars
5. Gaming

As can be seen, the XVSM Micro-Room Modeler could be extended arbitrarily to support all use cases needed by end users. Such additional module files can be uploaded to the modeler,

which will then be available for the corresponding user in the tree view showing all available micro-rooms and plugins. In contrast to the previous XVSM Micro-Room Framework module implementation, they also contain the required UI in our approach, thus forming fully integrated building blocks.

While modeling, the end user always has the possibility to undo and redo his last actions. Also, he/she is supported by smart micro-room connectors, only allowing to connect source actions to compatible target actions, thereby preventing models that would lead to errors during runtime of the generated application.

After finishing the model, the end user can create an executable application (e.g. an executable JAR) with a single click. To invite other users to the designed collaborative workflow, the modeling user can create a special share link to the model. When another user follows the link, the modeler will create an executable application based on the model but customized for the specific user. Figure 4.7 shows this essential use case in form of a sequence diagram.

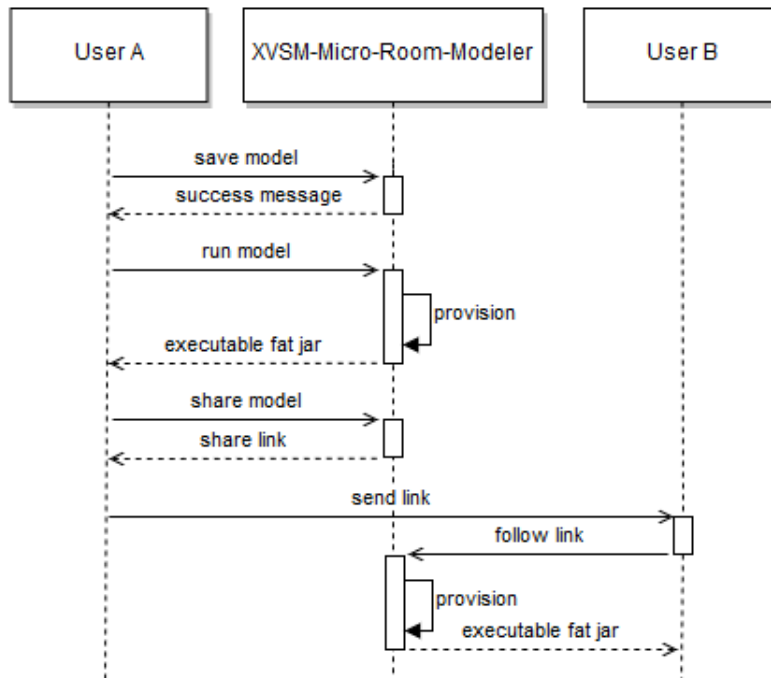


Figure 4.7: Sequence Diagram of Downloading and Sharing the Modeled Application

After both users downloaded their executable JAR, they execute them, thereby launching their local set of micro-rooms. The XVSM Micro-Room Framework replicates the state of these rooms between the applications of both users in the background via XVSM. Hence, interactions in a room become visible to all other users in this room.

When the created application has started up, it will show the customized UI depending on the model created by the user. So, e.g. only those micro-room UIs that are accessible will be shown and labels of UI elements will be adjusted to the micro-room or plugin properties specified by the modeling user.

4.4 Provisioning

Provisioning is the process of generating a fully executable application from the model created by the end user. Figure 4.8 shows which parts such an executable application (in this case in form of an executable JAR) has to contain.

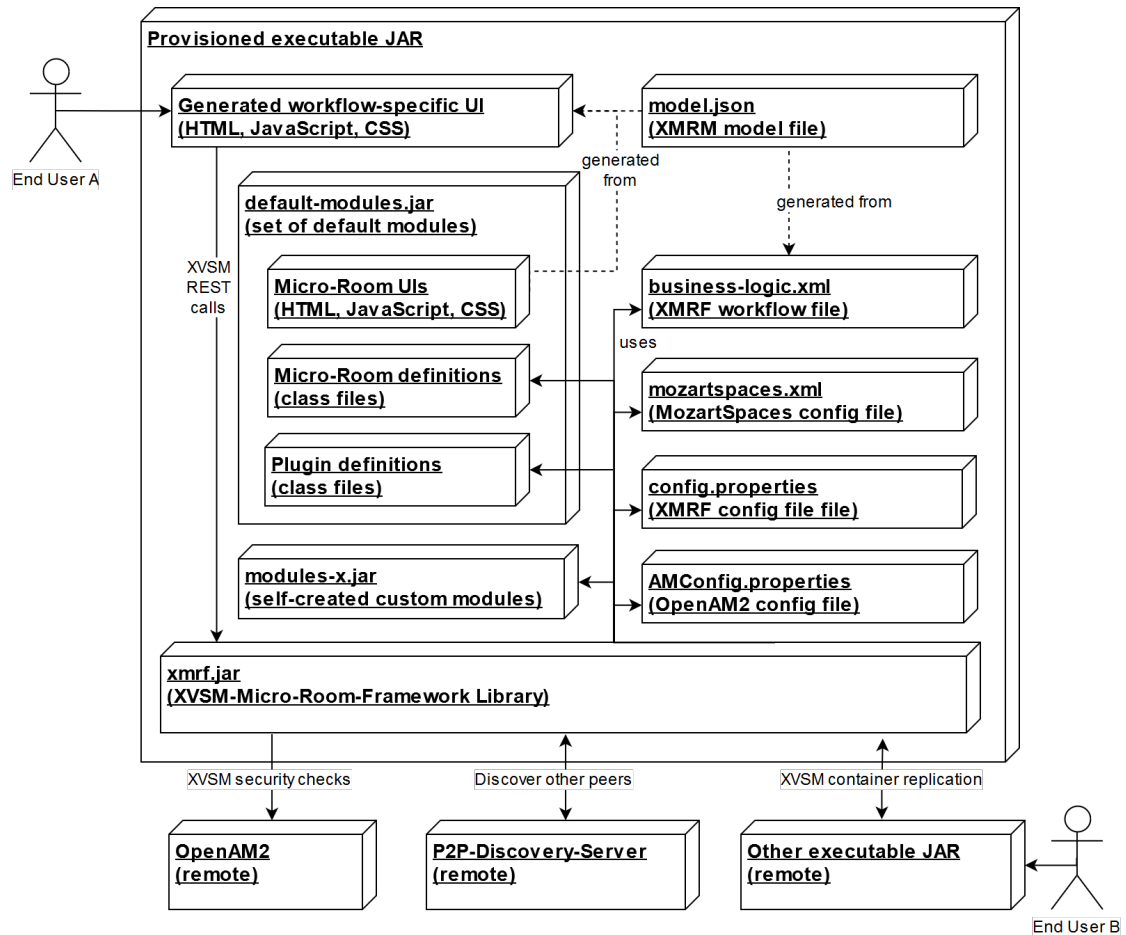


Figure 4.8: Provisioning in the XVSM Micro-Room Modeler

First of all, the created package needs to contain the XVSM Micro-Room Framework JAR file itself. This will be executed as a runtime environment, thereby loading its configuration files on startup. These contain `config.properties` defining XVSM Micro-Room Framework-specific settings, `mozartspaces.xml` configuring XVSM settings needed for P2P communication and `AMConfig.properties` defining security-related settings. These files will not differ much between the users and can be seen as static templates where several variables might be replaced according to the user creating the executable application.

Additionally, the XVSM Micro-Room Framework also needs one or more business logic files in XML format. The model created by the end user (which is stored in JSON format) is

transformed to exactly one such file and provided to the package. Also, all module files (i.e. the default modules as well as all additionally uploaded modules), containing all micro-room and plugin definitions, need to be added to the package.

Furthermore, the UI needs to be generated specifically for the end user. Thereby, the UI (HTML, CSS, JavaScript) files provided with each micro-room or plugin are extracted from the module file and “parameterized” with the JSON model created by the end user. Thereby, e.g. rooms or actions not accessible by a specific user can be hidden, buttons can be customized according to the micro-room properties defined in the model and the plugin UIs can be injected into the micro-room UI. Please note that these client-side adjustments only increase usability and do not enforce e.g. access restrictions to rooms or actions.

These are enforced by the XVSM Micro-Room Framework itself, which is why it communicates with the central OpenAM server. Besides that, also a central P2P discovery server is required for initial lookup of other peers (i.e. running executable JARs of other users). The interaction with the discovery server, as well as the replication with other executable JARs is handled by the XVSM Micro-Room Framework. It also has to be stated that neither the OpenAM server nor the discovery server need to be started by the end user him-/herself as they can be provided remotely. Only the provisioned application has to be launched by the user directly.

4.5 Workflow Execution

If the executable JAR is started, it shows the generated UI to the user in his/her default browser. A draft of such a created application is shown in Figure 4.9.

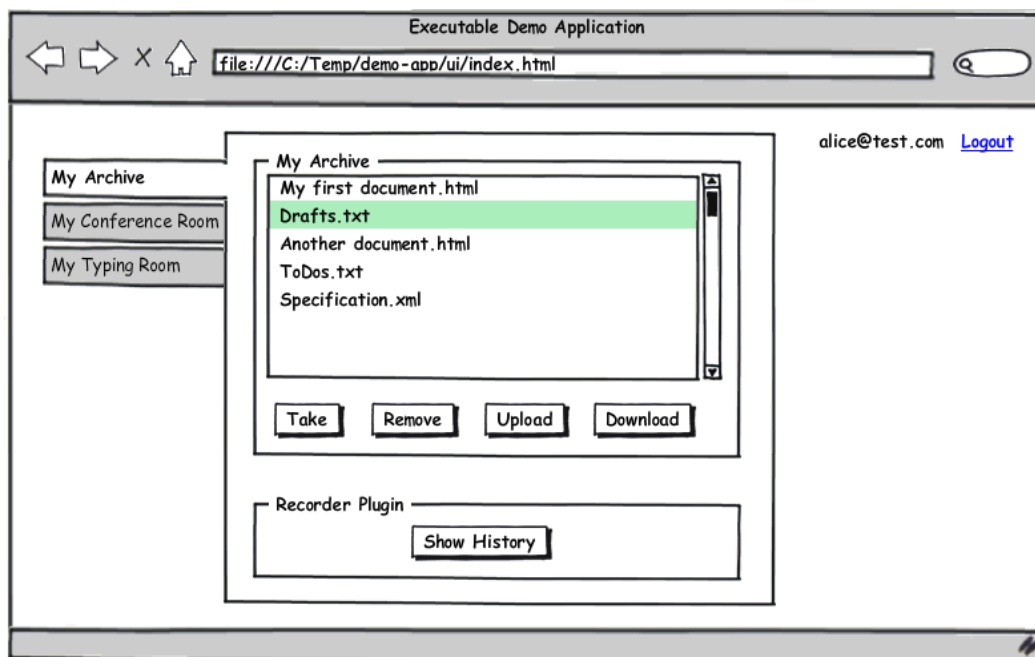


Figure 4.9: Draft of an Executable Application Created by the XVSM Micro-Room Modeler

As can be seen by the “Logout” link, the user has to be authenticated against the central OpenAM server to access the application (cf. Section 4.2). Upon login, a list of all micro-rooms available to the user is shown to the left. The first micro-room in the list is selected by default, thereby showing its UI in the main area. In the example an `Archive` can be seen that already contains several items that can be taken or removed via buttons. A click on one of these buttons triggers the corresponding action `take item` and `remove item` of the `Archive` (cf. Figure 4.6) respectively. Both of these actions could be connected to actions of other micro-rooms.

The “Upload” and “Download” buttons allow to interact with the room directly without triggering any connected actions. Thereby, the `Archive` is fully usable even if no other room is involved, since users can exchange their files with each other via one shared `Archive`.

Beneath the micro-room UI itself, the UIs of all plugins of the micro-room are shown. In the current example, the UI of a `Recorder Plugin` is shown, which consists only of a button to show all currently recorded events in a pop-up.

If the `take item` or the `remove item` action of the `Archive` is executed and it is connected to an action of another micro-room, this action will be called automatically as well. Such connected action executions are also replicated among all other peers, based on the functionality of the XVSM Micro-Room Framework (cf. [Bin13]).

Locally, a connected action also causes a UI transition if the user is allowed to enter the micro-room of the target action. If he/she is not allowed, the target action simply will not be executed and no UI transition occurs. E.g. if the `take item` action of the shown `Archive` is connected to the `show document` action of the `Typing Room`, a click on the “Take” button will cause a UI transition to the corresponding `Typing Room` automatically if the user is allowed to enter this room. Thereby, the item taken from the `Archive` is shown in the `Typing Room` as the names of the involved actions suggest. More details about the internal mechanics concerning UI transitions can be found in Section 5.5.1.

A uniqueness of this approach is that there exists no defined start and end point for the workflow. End users can start at any point that makes sense for them and follow the workflow path as long as they like. If they require a different micro-room to achieve their current task, they can always change to that room via the navigation panel, thus jumping around in the workflow. Thereby the current state of the old micro-room will remain the same. The user can continue with the work in this micro-room later by simply navigating to it again or another user could finish his/her work in the mean time.

So, strictly speaking, our approach does not provide a workflow that is executed from the start until the end in one go without any alterations. The users working with the created applications are more likely experiencing a collaborative working place that is guided by predefined flows in the form of automatic UI transitions. As these flows are only guided and not enforced, such as in other techniques, they can be iterated and adjusted on the go by jumping around between the elements as the current situation requires. Also, the work load can be shared dynamically by collaborating in the different micro-rooms according to the actual environment and work load. More about this can be found in Section 6.4.

Implementation

After describing the overall design we will now discuss implementation-related details, problems and solutions.

5.1 Structural View

First, we will outline the structure of the implementation of the XVSM Micro-Room Modeler and show the most important classes, interfaces and containers.

5.1.1 Package Structure

The package structure is very similar to the component diagram discussed in Chapter 4.2. Table 5.1 shows all packages and what they contain. All packages can be found in the `src/main/java` folder and are prefixed with `org.xvsm.microroom.modeler`.

Package	Description
<code>default</code>	Contains the main entry point of the application.
<code>config</code>	Spring-related classes to configure security and web settings.
<code>controller</code>	Non-REST controllers handling general tasks, e.g. registration and exception handling.
<code>controller.rest</code>	All REST controllers called by AngularJS.
<code>domain</code>	Domain objects required by the modeler.
<code>services</code>	Contains all service classes containing the core business logic.
<code>services.module</code>	Contains module-related services, e.g. for parsing or transforming.

<code>services.persistence</code>	Services that persist data to external systems such as OpenAM, OpenDJ and MozartSpaces.
<code>services.provisioning</code>	Classes containing logic for creating the provisioned executable package (cf. Section 4.4).
<code>services.provisioning.model</code>	<i>Plain Old Java Objects</i> (POJO) required for model-to-business-logic-file transformation.
<code>services.provisioning.model.json</code>	POJOs for the XVSM Micro-Room Modeler source model in JSON.
<code>services.provisioning.model.xml</code>	POJOs for the XVSM Micro-Room Framework target business logic file in XML.
<code>util</code>	Utility classes, e.g. for validation.

Table 5.1: XVSM Micro-Room Modeler Package Structure

5.1.2 Resource Folder Structure

Beside the Java packages, also the resource folders are of importance. Table 5.2 gives an overview of their content that can be found in the `src/main/resources` folder.

Package	Description
<code>default</code>	Contains configurations for the XVSM Micro-Room Modeler itself.
<code>distribution</code>	Folder used as a basis for each provisioned application package (i.e. always included in the application package). Includes configuration templates and the XVSM Micro-Room Framework JAR.
<code>distribution/certificates</code>	Certificate files required for secure communication with the central OpenAM server.
<code>distribution/modules</code>	All default module files.
<code>distribution/modules/ui</code>	UI skeleton framing all module-specific UIs.
<code>distribution/modules/ui/css</code>	Default UI styling.
<code>distribution/modules/ui/js</code>	Required JavaScript libraries, including <code>xmrf.js</code> , allowing to access the REST endpoints of the XVSM Micro-Room Framework.
<code>static</code>	All static web files required by the modeler.
<code>static/css</code>	CSS files for the AngularJS client part.
<code>static/images</code>	Images for the AngularJS client part.
<code>static/js</code>	AngularJS files (controllers and services).
<code>templates</code>	Thymeleaf template HTML files for AngularJS.

Table 5.2: XVSM Micro-Room Modeler Resource Folder Structure

5.1.3 Classes & Interfaces

After describing where to find which parts of the XVSM Micro-Room Modeler, we will now outline the most important classes and interfaces and their relation to each other. Figure 5.1 shows them in a class diagram.

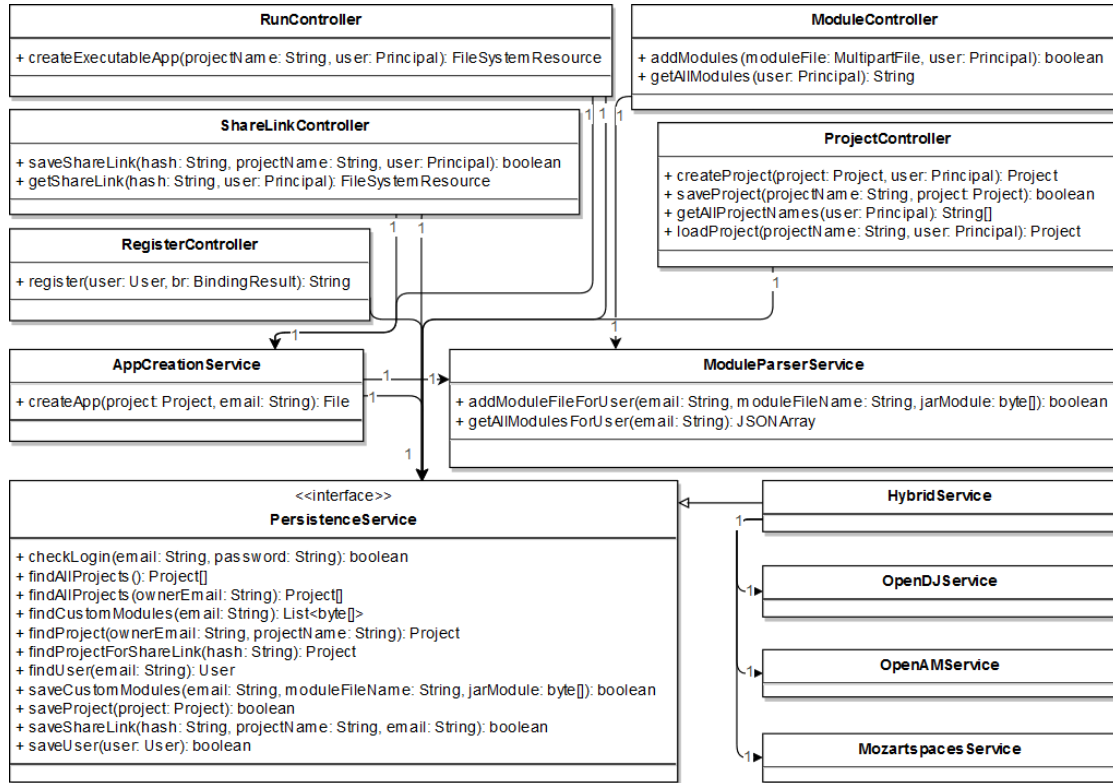


Figure 5.1: Class Diagram of the XVSM Micro-Room Modeler

As can be seen, we follow the strict *Model View Controller* (MVC) layering presented in Section 4.2. Controllers are called by the view (AngularJS JavaScript). No controller accesses another controller but they all only access services which are responsible for manipulating the model. Services do not access controllers but only other services. It also has to be noted that all of the classes presented represent singletons, thus explaining why there are only one-to-one connections and no attributes at all.

As we have three different services handling persistence, they are abstracted by the `PersistenceService` interface. All controllers and services access the `HybridService` singleton, which acts as a Facade for the three concrete implementations `OpenDJService`, `OpenAMService` and `MozartspacesService` to which it delegates, depending on the method called. E.g. the `checkLogin()` method will be delegated to the `OpenAMService` only whereas the `saveUser()` method will be forwarded to all three implementations.

The `AppCreationService` contains all the provisioning logic required and will be described in detail in Section 5.4. With the `ModuleParserService`, XVSM Micro-Room

Framework module JAR files can be converted to JSON, so that AngularJS can interpret the contained micro-rooms and plugins and show them in the modeler UI.

Regarding the controllers, the `RegisterController` is the only non-REST controller that simply interprets a form submit and creates the corresponding user. The `ShareLinkController` allows to create share links and can return the customized application package for a passed share link, resolving the project from the `PersistenceService`. The `RunController` returns the application package for the current project customized to the user currently logged in. Finally, the `ModuleController` provides an interface for AngularJS to access the parsed module files in JSON format and to upload new module files to the modeler whereas the `ProjectController` provides basic *Create, Read, Update, Delete* (CRUD) methods for end user projects.

5.1.4 Containers

After presenting the location and the functionality of the most important classes and interfaces, we will outline the XVSM containers used by the `MozartSpacesService` to persist modeler-specific data (cf. Table 5.3).

Container Name	Description	Coordinators	Entry
User	Used to store user information, e.g. email and name.	Key, Query	User object
Project	Used to store project information, e.g. project name, model, members and groups.	Key, Query, Lifo	Project object
Module	Used to store custom modules uploaded by an end user.	Label, Lifo	byte[]
ShareLink	Used to store correlation data for created share links.	Key, Lifo	String[]

Table 5.3: XVSM Micro-Room Modeler Containers

5.2 XVSM Micro-Room Framework Adjustments

To enable the whole feature set described in Chapter 4 the XVSM Micro-Room Framework has to be extended by several parts.

5.2.1 New Annotations

Previously, micro-rooms and plugins could be created by simple Java classes implementing the corresponding interfaces `MicroRoom` or `Plugin` provided by the XVSM Micro-Room Framework. For the XVSM Micro-Room Modeler this is not sufficient, as additional metadata is required. Therefore, several new annotations have been introduced to provide the metadata. They are parsed via reflection by the `ModuleParserService` in the XVSM Micro-Room

Modeler. An example of their usage is shown in Listing 5.1 and Listing 5.2.

```
1 @MicroRoom(name = "Archive", image = "/img/arc.png", ui = "/ui/arc")
2 public class Archive implements MicroRoom {
3     @Property(name = "Maximum Upload Size", defValue = "10MB")
4     private String maxUploadSize;
5
6     @Action(name = "add item", replicate = true, exposed = true)
7     public void store(Item item) {
8         ...
9     }
10
11     ...
12 }
```

Listing 5.1: Example of new Micro-Room Annotations in the XVSM Micro-Room Framework

```
1 @Plugin(name = "Recorder", image = "/images/rec.png", ui = "/ui/rec")
2 public class Recorder implements Plugin {
3     @Property(name = "Recorded action", defValue = "My Action")
4     private String targetAction;
5
6     ...
7 }
```

Listing 5.2: Example of new Plugin Annotations in the XVSM Micro-Room Framework

The new `@MicroRoom` and `@Plugin` annotations are placed directly on the class and specify the name of the module and the path of the image that is shown in the modeler. Also, they define the location of the UI folder for the module as described later in Section 5.3.3. Please note that both the image path and the UI folder is resolved as a classpath resource and thus should be provided under the `src/main/resources` directory.

The `@Property` annotation can be specified on a field which then is interpreted as a configurable micro-room or plugin property in the XVSM Micro-Room Modeler. It allows to specify the name and the default value that should be shown in the modeler for the property. Additionally, it has a configurable flag that is `true` by default. If it is set to `false` explicitly, the property will not be configurable in the XVSM Micro-Room Modeler but only be available as internal property within the micro-room and its UI.

Finally, the `@Action` annotation has to be placed on each method that should be interpreted as an action. This annotation will not only be interpreted by the XVSM Micro-Room Modeler, but also by the XVSM Micro-Room Framework. Formerly, the `@Replicate` annotation marked all actions that had to be replicated to other peers (cf. [Bin13]). This annotation has now been set to deprecated and replaced by `@Action(replicate = true)`. Hence, the `replicate` flag defines whether the action will be replicated or not and is only relevant for the XVSM Micro-Room Framework.

On the other hand, the `exposed` flag indicates whether the action is an exposed or an internal action. In the modeler, exposed actions are rendered with ports on the micro-room,

therefore being connectable to other micro-rooms with connections, whereas internal actions are not. Internal actions are only defined for UI interaction, i.e. they are callable via the JS wrapper library. If nothing is specified in the annotation, the action will neither be replicated nor exposed. Regardless of both flags, the name of the action is shown in the modeler as well as a possibility to define access restrictions for the action.

5.2.2 Browser Launch on Startup

To improve user experience, the XVSM Micro-Room Framework needs to show the UI generated by the XVSM Micro-Room Modeler on startup. Therefore, the XVSM Micro-Room Framework now launches the `ui/index.html` file in the system's default browser after starting up completely. This file is related to the XVSM Micro-Room Framework's configuration directory which points to `."` by default but can be altered with a `--configDir` command line argument.

More about UI generation by the XVSM Micro-Room Modeler will be described in Sections 5.3.3 and 5.4.5.

5.3 Modules

In this section several module-related issues in the XVSM Micro-Room Modeler are described.

5.3.1 Package Format

The package format of module files remains the same as described in [Bin13], i.e. a set of related modules is packed together into a JAR file. In this file, a new `manifest` file has to be provided, similarly to Listing 5.3.

```
1 name=Default
2 basePackage=org.xvsm.microroom.modules
```

Listing 5.3: Example Manifest of a Module JAR File

Thereby, the `name` is used for a root node in the tree view showing all available micro-rooms and plugins in the XVSM Micro-Room Modeler. Beneath this root node all modules of the module file will be listed, hence allowing end users to better distinguish between different module suites. The `basePackage` will be used in the `ModuleParserService` to scan for parsable classes with `@MicroRoom` or `@Plugin` annotations.

5.3.2 Parsing

When parsing a module JAR file, the `ModuleParserService` instantiates a new temporary class loader for the current user. With this class loader it scans for classes with the annotations, accesses their values via reflection and stores them in a JSON object. Afterwards, it destroys the classloader again, as it is of no further use.

By this procedure, no problems occur in the XVSM Micro-Room Modeler if two users upload custom module JAR files with equal fully qualified class names. If all module classes were loaded by a single class loader, negative side effects could occur. The class loader would return the module class uploaded first, regardless if the second class with the same fully qualified name was different or not.

The JSON result after successfully parsing a module JAR file looks as specified in Listing 5.4. It will be queried by the AngularJS client part which uses the model to render the list of available modules.

```
1 {
2   "manifest": {
3     "basePackage": "org.xvsm.microroom.modules",
4     "name": "Default"
5   },
6   "microRooms": [
7     {
8       "image": "iVB ... II=",
9       "ui": "/ui/micro-rooms/archive",
10      "name": "Archive",
11      "class": "org.xvsm.microroom.modules.rooms.Archive",
12      "actions": [
13        {
14          "method": "store",
15          "parameterTypes": [
16            "org.xvsm.microroom.framework.objects.Item"
17          ],
18          "name": "store item",
19          "exposed": true,
20          "returnType": "void"
21        }
22      ],
23      "properties": [
24        {
25          "field": "maxUploadSize",
26          "name": "Maximum Upload Size",
27          "fieldType": "java.lang.String",
28          "defValue": "10MB",
29          "configurable": true
30        }
31      ]
32    }
33  ],
34  "plugins": [
35    {
36      "image": "iVB ... II=",
37      "ui": "/ui/plugins/recorder",
38      "name": "Recorder",
```



```

39         "class": "org.xvsm.microroom.modules.plugins.Recorder",
40         "properties": [
41             {
42                 "field": "targetAction",
43                 "name": "Recorded action to be recorded",
44                 "fieldType": "java.lang.String",
45                 "defValue": "My Action",
46                 "configurable": true
47             }
48         ]
49     }
50 ]
51 }

```

Listing 5.4: JSON Structure of a parsed XVSM Micro-Room Framework Module

As can be seen, the created JSON structure corresponds directly to the methods, fields and the new XVSM Micro-Room Framework annotations presented in Section 5.2.1. The images are inlined into the model directly as Base64 encoded byte arrays, since they are not very big (128x128 pixel). As mentioned in Section 5.2.1 the `@Action(replicate)` flag is indeed only required by the XVSM Micro-Room Framework and ignored in the XVSM Micro-Room Modeler since it is neither parsed nor stored in the JSON model.

5.3.3 Integrated UI

As described in Section 3.1.4, previously the UI for an application based on the XVSM Micro-Room Framework had to be implemented by IT experts after the business logic files had been created. Thereby, a special JavaScript wrapper library could be used to call actions of micro-rooms and plugins.

To solve this problem, from now on UI files have to be added to the module JAR, so that the XVSM Micro-Room Modeler can use them as building blocks to assemble the end user's final UI. Therefore, for each module a folder can be specified in which the UI files are located.

This folder has to consist of at least an `index.html` file. Within the file, additional resources such as further HTML, CSS or JavaScript files can be included and accessed. The file should contain the required HTML to allow the end user to properly interact with the micro-room or plugin it represents. User actions such as clicked buttons or entered form elements can be sent via the provided JavaScript wrapper library to the XVSM Micro-Room Framework to call the corresponding actions. Listing 5.5 shows an example of a UI for the `Archive` micro-room draft shown in Section 5.2.1.

```

1 <div>
2     <p>Max Upload Size:<span id="mus"></span></p>
3     <form action="#">
4         <input type="button" id="sb" value="Store" />
5     </form>
6 </div>

```

```

7
8 <script>
9     $(document).ready(function() {
10         var prop = getProps($mr, "maxUploadSize");
11         $("#mus").html(prop);
12
13         $("#sb").click(function(e) {
14             var item = new Item([{"string":["text", "Hello!"]}]);
15             xmrf.action("Main", $mr.name, "store", msgType.
16                 ↪ASYNC_REQRESP, item, function(retVal){
17                 if(!(retVal instanceof Error)){
18                     alert("Success!");
19                 } else {
20                     alert("Error!");
21                 }
22             });
23         });
24 </script>

```

Listing 5.5: Example UI for a Module of the XVSM Micro-Room Framework

As can be seen, the `index.html` file contains simple HTML and JavaScript. When the UI (similar to the one shown in Figure 6.6) is loaded, the `maxUploadSize` of the micro-room defined by the end user is read and shown in the UI (cf. line 10 and 11). Therefore the provided helper function `getProps()` is called with the `$mr` variable and the name of the micro-room or plugin property to resolve (i.e. the name of the field annotated with the `@Property` annotation). The `$mr` variable is set in the UI frame provided by the XVSM Micro-Room Modeler and points to the currently active micro-room JSON model (cf. Section 5.4).

If the `Store` button is clicked, the `store` action of the current micro-room (`$mr.name`) is called with a new “Hello!” `Item` (cf. line 15). The scenario name of the only scenario will always be “Main” for applications created by the XVSM Micro-Room Modeler. The message type to interact with the XVSM Micro-Room Framework can be chosen from the four types specified in [Bin13]:

ONEWAY: Asynchronously write the request into the request container of the XVSM Micro-Room Framework. Do not wait for a response from the response container.

SYNC_REQRESP: Synchronously write the request into the request container, block until a response is available in the response container and return it from the function call. Please note that this will cause the UI to become unresponsive since JavaScript is single-threaded.

ASYNC_REQRESP: Asynchronously write the request into the request container and call a callback function upon an available response in the response container.

PUBSUB: Asynchronously write the request into the request container and asynchronously fetch correlated responses from the response container and forward them to the callback

function until the subscription gets cancelled. Therefore, a `PubSub` object is returned from the function call which allows to cancel the subscription by calling `unregister()`.

A full reference of the provided variables and functions for UI interaction will be given in Section 6.2.

5.4 Provisioning

A very important part of the implementation is provisioning, i.e. creating a fully executable application from the model designed by the end user. The required parts of the application package have already been described in Section 4.4. In this chapter we will describe how the actual implementation works. Therefore, we will outline the tasks performed by the `AppCreationService` step-by-step in the following Sections.

5.4.1 Derive from Template

First of all, the static files included in the modeler's `src/main/resources/distribution` folder are extracted to a new temporary directory which will be referred to as “working directory” from now on. The general folder structure of the distribution folder has already been described in Section 5.1.2. Figure 5.2 shows the detailed contents of the folder which are used as a template for the following steps.

In the `certificates` folder certificates required for *Hypertext Transfer Protocol Secure* (HTTPS) communication with the OpenAM server are located. The `default-modules.jar` file contains the module implementations including their UIs as described in Section 5.3.

All required CSS files for the default layout specified in the `index.html` file are located in the `ui/css` folder whereas the `ui/js` folder contains all required JavaScript libraries. Beside a date formatting library and several jQuery¹ libraries, the `xmrf.js` file is of major importance. It represents the JavaScript wrapper library providing helper functions for the UI to interact with the XVSM Micro-Room Framework, which has been used in Section 5.3.3.

The `index.html` file is the main UI file that will be shown to the end user. It provides a login dialog that verifies the end user via OpenAM and shows a generic UI with a navigation panel to the left and a placeholder for the currently navigated micro-room in the main area, followed by a placeholder for all plugins of the micro-room (cf. Figure 4.9). With the `menu.html` file the generic navigation panel and the logic to navigate between rooms is realized. Internally this works by injecting the required micro-room and plugin UIs into the `index.html` file on predefined positions.

The `config.properties` file configures the XVSM Micro-Room Framework and specifies properties like the location of the modules and business logic files, the own peer user name or the P2P discovery server location. The `mozartspaces.xml` file configures e.g. the ports of the internally used XVSM instance. Both configuration files contain template variables that will be replaced in a later step, e.g. to inject the actual user name or port.

¹<https://jquery.com/>

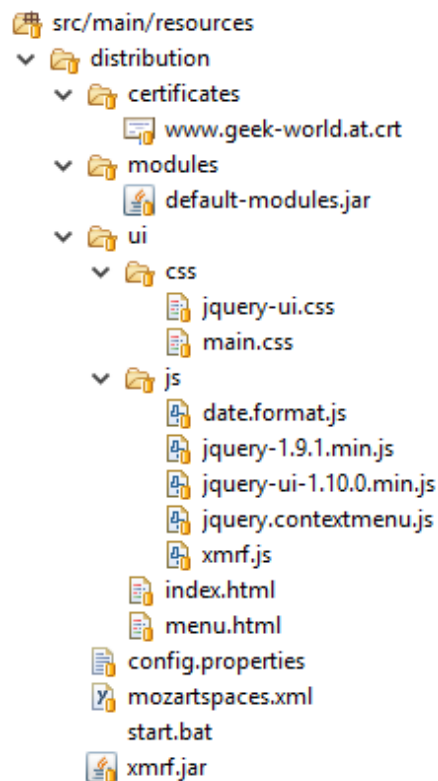


Figure 5.2: Content of the Distribution Folder in the XVSM Micro-Room Modeler

Finally, the `xmrf.jar` contains the fully executable XVSM Micro-Room Framework that will be launched via the `start.bat` file. To demonstrate its simplicity, the full content of the `start.bat` file is shown in Listing 5.6.

```
1 java -jar xmrf.jar
```

Listing 5.6: Content of `start.bat`

5.4.2 Adding Custom Modules

After extracting the template to the working directory, all modules that have been uploaded by the end user via the XVSM Micro-Room Modeler need to be added to the working directory as well. Therefore, they are read from the `Module` container (cf. Section 5.1.4) by the `ModuleParserService` and written to JAR files into the working-directory's `modules` folder.

5.4.3 Parsing the JSON Project

The project created by the end user with the XVSM Micro-Room Modeler, including the designed model, is stored in JSON format in the client part. The JSON format of the project is shown in Listing 5.7.

```
1 {
2   "name": "test",
3   "ownerEmail": "alice@test.com",
4   "groups": [
5     {
6       "name": "Editor",
7       "ownerEmail": "alice@test.com",
8       "projectName": "test",
9       "members": [
10        "alice@test.com"
11      ]
12    }
13  ],
14  "members": [
15    {
16      "name": "Alice",
17      "email": "alice@test.com",
18      "memberOf": [/* group objects */]
19    }
20  ],
21  "model": { /* model object */ }
22 }
```

Listing 5.7: JSON Structure of an XVSM Micro-Room Modeler Project Object

As can be seen, the model contains the basic project-specific settings such as project name (line 2) and project owner (line 3). Also, the created project members (line 14) and groups (line 4) are included, where each group is defined uniquely by the combination of its own name (line 6), the project name (line 8) and the owner of the project in which it is contained (line 7).

Users can be added to the project and assigned to groups at will, but will only be able to use the created software after registering an account in the XVSM Micro-Room Modeler. Please note that groups represent discrete entities that can exist on their own and should be usable anywhere, not only if embedded in the project context. They have a composite unique identifier as stated to clearly delimit each group from the other. Thus, the `ownerEmail` field in line 7 is not redundant to the one in line 3 as it might seem at first glance. Moreover, it is a mandatory part of the group's composite unique identifier to keep the group entity usable even if separated from the project context. Since in plain JSON there is no simple possibility to reference other objects, group objects containing only the composite key (and no `members` property) are used for defining the `memberOf` property for each member (line 18).

The designed model itself can be found under the `model` (line 21) property and is of the form shown in Listing 5.8.

```

1  {
2    "cells": [
3      {
4        "type": "devs.MicroRoom",
5        "inPorts": ["store item"],
6        "outPorts": [],
7        "size": {
8          "width": 200,
9          "height": 200
10       },
11       "ports": { /* graphical representation of ports */,
12       "position": {
13         "x": 535,
14         "y": 80
15       },
16       "angle": 0,
17       "mr": { /* micro-room object */ },
18       "id": "5076f78a-7a50-4730-9e52-973b5c5bcc6f",
19       "z": 1,
20       "attrs": {}
21     },
22     {
23       "type": "link",
24       "source": {
25         "id": "826981a4-d42e-4771-b602-ec4ec9f12c10",
26         "port": "finish document"
27       },
28       "target": {
29         "id": "5076f78a-7a50-4730-9e52-973b5c5bcc6f",
30         "port": "store item"
31       },
32       "router": {
33         "name": "manhattan"
34       },
35       "connector": {
36         "name": "rounded"
37       },
38       "id": "c4c5e1c2-9adc-440e-8a93-393fb2999c03",
39       "z": 6,
40       "attrs": {
41         ".marker-target": {
42           "d": "M 10 0 L 0 5 L 10 10 z"
43         }
44       }
45     } /* more cells here*/
46   ]
47 }

```

Listing 5.8: JSON Structure of an XVSM Micro-Room Modeler Model Object

The model contains an array of cells, i.e. micro-room wrapper objects (line 4) or links between them (line 23). A micro-room wrapper object contains all properties required to store its graphical representation in the model, e.g. its position (line 12), width (line 8), height (line 9), rotation (line 16) and stack order in case elements overlap (line 19). `inPorts` (line 5) and `outPorts` (line 6) are equal to the exposed actions of the micro-room and indicate the ports to be rendered. More details about their graphical representation is stored in the `ports` property (line 11), e.g. shape, color, position, connectability and so on.

A link contains the source (line 24) and target (line 28) room references (note the equal `id` values (line 18 and 29) and the source (line 26) and target (line 30) ports (i.e. action names). Additionally, rendering information about the link itself is specified (line 33), e.g. manhattan route calculation (cf. [BBL83] for more information), rounded corners (line 36) and the exact path of the link on the modeler pane (line 42).

So far we have only covered all properties regarding rendering of the model. The configuration itself is stored in a micro-room object in the `mr` property (line 17) within the micro-room wrapper object. Its structure is shown in Listing 5.9.

```

1  {
2      "image": "iVB ... II=",
3      "ui": "/ui/micro-rooms/archive",
4      "name": "My Archive",
5      "class": "org.xvsm.microroom.modules.rooms.Archive",
6      "actions": [
7          {
8              "method": "store",
9              "parameterTypes": [
10                 "org.xvsm.microroom.framework.objects.Item"
11             ],
12             "name": "store item",
13             "exposed": true,
14             "returnType": "void",
15             "permissions": [
16                 "$all"
17             ]
18         }
19     ],
20     "properties": [
21         {
22             "field": "maxUploadSize",
23             "name": "Maximum Upload Size",
24             "fieldType": "java.lang.String",
25             "defValue": "10MB",
26             "configurable": true,
27             "value": "5MB"
28         }
29     ],
30     "type": "Archive",

```

```

31     "roomCount": "$all",
32     "navigableBy": [
33         "role.editor"
34     ],
35     "plugins": [/* plugin objects */]
36 }

```

Listing 5.9: JSON Structure of an XVSM Micro-Room Modeler Micro-Room Object

The shown Listing is similar to the micro-room JSON structure created from the module JAR files (cf. Listing 5.4) but extended by several properties. One of these is the `permission` property to define which members or groups should be allowed to execute the corresponding action (line 15). In contrast to the JSON created from the module JAR, we now also have the configured values of all custom micro-room properties (line 27). The `roomCount` property (line 31) defines the creation strategy of the micro-room (cf. [Bin13]), which can be either “one for all”, “one per group” or “one per member”. With the `navigableBy` property (line 32), the end user configures which groups or members are allowed to enter this room. Unauthorized users will not see the micro-room in their UI and will not be able to execute any actions in the XVSM Micro-Room Framework due to XVSM Security constraints.

The plugins of the room are defined in the `plugins` property (line 35). The structure of a plugin is specified in Listing 5.10.

```

1  {
2  "image": "iVB ... II=",
3  "ui": "/ui/plugins/recorder",
4  "name": "Recorder",
5  "class": "org.xvsm.microroom.modules.plugins.Recorder",
6  "properties": [
7  {
8  "field": "targetAction",
9  "name": "Action to be recorded",
10 "fieldType": "java.lang.String",
11 "defValue": "My Action Name",
12 "configurable": true,
13 "value": "store"
14 }
15 ],
16 "type": "Recorder"
17 }

```

Listing 5.10: JSON Structure of an XVSM Micro-Room Modeler Plugin Object

As the micro-room JSON structure, also the plugin JSON structure is equal to the plugin JSON structure created from the module JARs (cf. Listing 5.4). Again, the `value` property (line 13) is the relevant new part configured by the end user.

When creating the executable application, all the shown JSON objects need to be parsed into Java objects for further use. Therefore, they are mapped with Jackson² to the corresponding POJOs. The created Java objects are equal to the JSON model regarding their structure and hence will not be discussed in further detail.

5.4.4 Transformation to Business Logic XML File

Now that the whole project is available as Java objects, the business logic XML file required as input for the XVSM Micro-Room Framework has to be derived. The desired target format of a business logic file is defined in a XML schema in [Bin13]. An example is shown in Listing 5.11.

```
1 <?xml version="1.0" encoding="UTF-8" ?>
2 <scenario>
3   <rooms>
4     <room name="My Storage" type="org.xvsm.microroom.modules.rooms.
5       ↪Storage" generateFor="$all" maxUploadSize="5MB">
6       <owner>alice@test.com</owner>
7       <plugins>
8         <plugin type="org.xvsm.microroom.modules.plugins.Recorder">
9           <property key="targetAction" value="store" />
10          </plugin>
11         <plugin type="org.xvsm.microroom.modules.plugins.Replication">
12           <property key="replicateTo" value="role.alice@test.com/test/
13             ↪editor"/>
14           </plugin>
15         <plugin type="org.xvsm.microroom.modules.plugins.Access">
16           <property key="store" value="role.alice@test.com/test/all" />
17         </plugin>
18       </plugins>
19     </room>
20     <room name="My Mail Room" type="org.xvsm.microroom.modules.rooms.
21       ↪MailRoom" generateFor="$all" />
22   </rooms>
23   <connections>
24     <connection>
25       <from roomName="My Mail Room" actionName="finish" />
26       <to roomName="My Storage" actionName="store" />
27     </connection>
28   </connections>
29 </scenario>
```

Listing 5.11: Example of a XVSM Micro-Room Framework Business Logic File

As can be seen, the structure is relatively simple and can be mapped quite easily from the JSON model (more specifically from the mapped Java model with the same structure) as this has been designed with the structure of the XML file already in mind.

²<https://github.com/FasterXML/jackson>

Each of the micro-room objects in JSON is mapped to exactly one `room` element in XML (line 4). Its attributes, i.e. `name`, `type`, `generateFor` (`roomCount` in JSON) and all custom properties can be extracted from the micro-room JSON object directly (formally more correctly from the Java object created from the JSON object). Every plugin of the micro-room in JSON is mapped to one `plugin` element in XML (line 7). Their properties are mapped to `property` elements where `field` in JSON is `key` in XML and `value` in JSON is `value` in XML.

Special handling is required for the `navigableBy` and the `actionPermissions` properties in JSON. `navigableBy` is mapped to the `Replication` plugin provided by the XVSM Micro-Room Framework itself. Its value is set as the value of the `replicateTo` key in XML (line 11), thereby defining to which peers the micro-room will be replicated. Internally, this enforces a hard access restriction, as the room will not even exist on all other peers, thereby being completely inaccessible.

The `permissions` property of each action is defined as a `property` element within the `Access` plugin provided by the XVSM Micro-Room Framework. Thereby, the `method` property in JSON is mapped to the `key` property in XML and the `permissions` property in JSON is mapped to the `value` property in XML (line 14).

Please note that all group-specific values such as `$all` or `role.editor` are replaced by the unique group identifier for the corresponding group during translation. The unique group identifier consists of the project owner email address, the project name and the group name within the project. This prevents naming collisions in the central OpenAM server due to same group names in different projects of different users. So, e.g. `$all` is mapped to `role.alice@test.com/test/all` and `role.editor` is mapped to `role.alice@test.com/test/editor`.

Technically, the mapping of the JSON model to the XML business logic file works with simple assignments from the JSON POJOs to the XML POJOs, followed by a serialization with *Java Architecture for XML Binding (JAXB)*³. The generated business logic XML file is then written to `business-logic/Main.xml` within the working directory.

5.4.5 Creating UI Files

After deriving the business logic file, the customized UI files have to be created. Therefore, the contents of each UI folder of all modules are extracted to a sub folder equal to the fully qualified class name of the module in the `ui` directory of the working directory. So e.g. for the `Archive` micro-room, a `ui/org.xvsm.microroom.modules.rooms.Archive/index.html` file would exist in the working directory after the operation. This step is required since the UI files need to be accessible directly on the file system by the browser and cannot be accessed within the JAR files.

As a side note, additional UI templates for a module could be added with ease, e.g. by defining each of them in a separate sub folder when creating the module. Then they could be used at this point instead of the mentioned `index.html` file.

³<https://jaxb.java.net>

Customization of the UI according to the user happens on startup via JavaScript. After login, the UI frame provided by the XVSM Micro-Room Modeler (i.e. the `index.html` in the `ui` directory) will scan the JSON model for all micro-rooms with a `navigableBy` property matching our user or his/her roles. The accessible micro-rooms will be shown in the navigation panel to the left and are stored in a global `$navi` variable which holds a micro-room name to micro-room object mapping. Later on, clicking onto one of the micro-room links or calling `navigate()` with a micro-room name will use this mapping to view the desired micro-room. Also, when calling this function, the global `$mr` variable is set to the navigated micro-room so that the UI of a micro-room can always rely on having access to its own configuration by simply using `$mr`.

5.4.6 Replacing Template Variables

Now that all files are in place in the working directory, all that is left is to replace the template variables in the files copied from the `distribution` directory. Those template variables are in the form of `${variableName}` and occur in the `index.html`, `mozartspaces.xml` and `config.properties` file.

In `index.html`, by replacing the template variables, the project name and the email of the user for whom the application is created are set. Also the model itself is injected into a global `$model` variable. Hence, each module UI has full access to the whole underlying model.

In the `mozartspaces.xml` file, the XVSM port (for direct P2P communication with other peers) and the HTTP port (for communication between UI and XVSM Micro-Room Framework) are injected via template variables. To enable end users to launch multiple instances of created applications at the same time, those ports are randomized upon each creation.

Finally, in the `config.properties` file the peer name (i.e. user email) is set, which is used to uniquely identify the current peer.

5.4.7 Creating the Zip File

Now that all files are in place in the working directory, it is packaged into a self-executable ZIP file that can be downloaded. More about starting up the created application will be described in Section 6.4.

5.5 Further Problems & Solutions

In this chapter, further interesting problems & pitfalls of the XVSM Micro-Room Modeler are described, including the implemented solution.

5.5.1 Workflow Navigation

In the created application, the workflow defined in the model is mapped as follows: If from within the UI a micro-room action is called (e.g. via button click) that is connected to another action, the XVSM Micro-Room Framework executes the target action with an input parameter equal to the return value of the source action (cf. [Bin13]).

After both actions have completed execution, the return value of the source action is returned to the calling UI function, including a special `redirect` parameter. This parameter contains the micro-room name to which should be redirected and will be removed from the response by the XVSM Micro-Room Framework JavaScript wrapper library automatically, before it is passed further on to the calling UI function.

If the library detects such a parameter, it calls the provided `navigate()` function with the micro-room name, thereby triggering a UI transition to this room and aborting the calling UI function's logic.

If no parameter is present or the user is not allowed to access the target room, the calling UI function returns normally and receives the return value of the called source action. Hence, for the calling UI function it appears as if no connection existed at all. Please note that for the message type `ONEWAY` (cf. Section 5.3.3) no redirects will be executed due to its design in which no responses are read at all.

It explicitly has to be stated that while the execution of connected actions are replicated within the XVSM Micro-Room Framework to other peers as designed, redirects in the UI are not replicated. First of all, the `redirect` parameter is stripped off by the `ReplicationManager` in the XVSM Micro-Room Framework, thereby the second peer will not know anything about the UI transition. However, even if such a parameter was present, the UI of the second peer will not be waiting for the response of the replicated action, so no automatic redirect could ever happen.

5.5.2 Group Assignment of Non-existent Users

When a user is assigned to a group in the XVSM Micro-Room Modeler, two actions are performed in OpenDJ: First, the `memberof` attribute of the user's `people` LDAP entry is extended by e.g. `cn=alice@test.com/test/editor`. Second, a new `uniqueMember` attribute of the `groups` LDAP entry is added containing e.g. `uid=alice@test.com, ou=people, dc=opensso, dc=java, dc=net`. This causes no problems as long as all users already have a corresponding `people` LDAP entry.

Now, what happens if e.g. Alice defines a project that contains Bob, but Bob is not registered yet, hence having no `people` LDAP entry? In this case, nothing is changed in LDAP but the group membership is only stored in the embedded `MozartSpaces` instance of the XVSM Micro-Room Modeler. If Bob registers his account later, on registration `MozartSpaces` is queried for all occurrences of Bob in all groups of all projects of all users. For each of these groups, the two LDAP operations described are performed, thereby creating a consistent state, required for XVSM Security to correctly process Bob's permissions in all projects.

5.5.3 Auto-Saving

As the XVSM Micro-Room Modeler is a web application, it could happen that a user refreshes the page or closes the browser tab unintentionally. Therefore, auto-save functionality should be provided to avoid the thereby caused loss of all unsaved changes of the currently edited project.

This is realized by storing the whole JSON object of the currently active project as a string in the browser's session storage every five seconds. When loading the page, it is first checked

whether a project is stored in the session storage. If so, it will be restored automatically, thus avoiding data loss.

5.5.4 Undo & Redo

To increase the usability of the XVSM Micro-Room Modeler, undo and redo functionality have to be implemented.

Therefore, on each user event in the model (i.e. adding, moving or deleting elements such as micro-rooms, plugins and connections or changing properties of micro-rooms and plugins) an undo event is recorded. Technically, simply a snapshot of the current model object (cf. Listing 5.8) is pushed onto a global undo stack before the action is executed.

Now, when the user clicks on the “undo” button, the current model object is pushed onto a global redo stack. Then, the last element is popped from the undo stack and the model is replaced with the previous version. As the model object does not only contain all properties but also the graphical representation, the user will see his/her last change being reverted in either the properties pane or the graph.

The end user can then redo the version undone or further undo his/her changes until the redo or undo stack is empty correspondingly. If the user performs an action beside undoing or redoing, the redo stack will be emptied.

5.5.5 Smart Links

Further usability improvements are enabled by providing so-called “smart links”. Such links can only be connected to target ports suiting to the source port. In terminology of a micro-room model, a source action can only be connected to a target action that takes a parameter of the type of the source actions return value. Otherwise, a connection will not be possible within the XVSM Micro-Room Modeler, which is a valuable feature, as such connections would lead to runtime errors in the created application.

Figure 5.3 shows how available target ports are highlighted in the implementation. The usability is even increased as the links are “magnetic” to such ports, automatically connecting to them when the link is dragged near them.

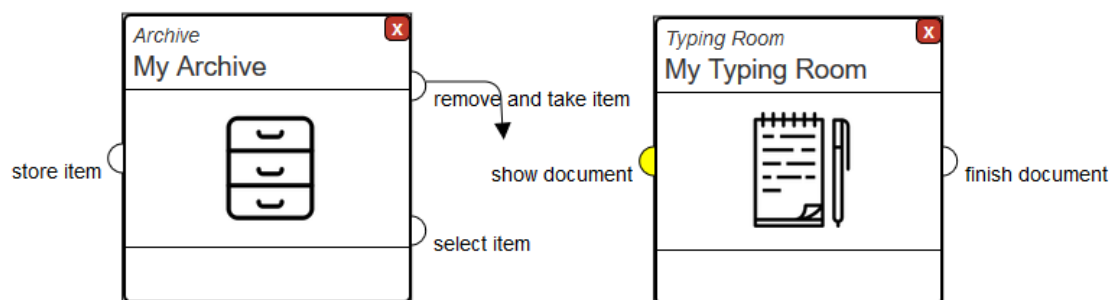


Figure 5.3: A Smart Link being dragged in the XVSM Micro-Room Modeler

Internally, JointJS⁴ is used for modeling the diagram. It allows to realize this feature by using the `validateConnection` function that can be specified for the whole graph. In it, one can decide for the currently dragged link which ports should be connectable and which not without a lot of effort. More information about this and similar code examples can be found in [14]. Regarding the arrangement of links, built-in manhattan routing of JointJS is used, which is described in more detail in [15].

5.5.6 Link Sharing

Finally, when sharing a link, it has to be ensured that the accessing user is allowed to download an executable package for the corresponding project. Therefore, the `ShareLinkController` checks whether the user is a member of the project before it generates the executable application package. If the user is not a member, he/she will see a corresponding error message.

⁴<http://www.jointjs.com>

User Guide

After describing the approach, the architecture and various implementation details of the XVSM Micro-Room Modeler, we will now discuss how it is used. Therefore, the following chapter will cover how the modeler itself can be deployed, how IT experts can extend it, how end users can use it to create their applications and finally, how these applications can be deployed and used.

6.1 Deploying the XVSM Micro-Room Modeler

Deploying the XVSM Micro-Room Modeler itself is relatively easy since it is an executable fat-JAR created via Spring Boot. As only prerequisite a JRE 8 has to be installed on the machine. Then, the JAR can be executed by calling the command shown in Listing 6.1.

```
1 java -jar XVSM-Micro-Room-Modeler-1.0-SNAPSHOT
   ↪--openAMUrl=http://your.server.com:8080/openam2
   ↪--openAMUserId=amAdmin --openAMPassword=pwdAmAdmin
   ↪--openDJUrl=your.server.com --openDJPort=1389
   ↪--openDJUsername="Directory Manager" --openDJPassword=pwdOpenDj
```

Listing 6.1: Deployment of the XVSM Micro-Room Modeler

The additional parameters are required to configure the OpenAM and OpenDJ servers to use. These servers are used by the modeler to handle login and to persist users, groups and project members (cf. Section 5.1.3). Please note that it is completely sufficient to deploy the modeler only once on a server that can be reached by all end users. The end user him-/herself does not need to deploy the modeler.

6.2 Extending the XVSM Micro-Room Modeler

The XVSM Micro-Room Modeler can be extended with functionality (i.e. new micro-rooms and plugins) at runtime by uploading new module files in the modeler UI. Such module files can

be created by any IT expert as follows:

First, the logic of the micro-rooms and plugins has to be specified by implementing the corresponding Java interfaces provided by the XVSM Micro-Room Framework as described in Section 5.2.1. Thereby, each micro-room and plugin is represented by a single Java file, containing simple annotations that configure which fields should be properties and which methods should be actions. Internally, it is recommended (but not required) to use XVSM containers for storing data as the XVSM Micro-Room Framework can then replicate it across peers automatically. Also XVSM aspects could be used such as described in Section 3.1.1.

Then, UI files for the micro-rooms and plugins need to be created, such as stated in Section 5.3.3. It is sufficient to provide a single HTML file for each micro-room and plugin that contains JavaScript calls to interact with the actions specified in the corresponding module. Therefore, the XVSM Micro-Room Framework JavaScript wrapper library can be used since it will be included in each created application by the modeler automatically. More information about this library and how actions are called can be found in Section 5.3.3 and [Bin13].

Beside the interaction capabilities provided by the XVSM Micro-Room Framework wrapper library, the XVSM Micro-Room Modeler itself also adds functionality that can be used within each UI file. First of all, the modeler injects several global variables that can be used (cf. table 6.1).

Global variable	Description
<code>\$email</code>	Email of the logged in user (i.e. the one that runs the created application).
<code>\$ownerEmail</code>	Email of the project creator.
<code>\$projectName</code>	Name of the project.
<code>\$model</code>	The complete JSON model created by the project owner, containing all information such as described in Listing 5.8.
<code>\$navi</code>	A map containing micro-room name to micro-room object mappings.
<code>\$isRedirect</code>	Is true if the current page initialization has been triggered by a <code>redirect</code> parameter coming from the XVSM Micro-Room Framework such as described in Section 5.5.1.
<code>\$mr</code>	Contains the current (i.e. the one related to the UI) micro-room object such as defined in Listing 5.9.

Table 6.1: Global JavaScript Variables Provided by the XVSM Micro-Room Modeler

Furthermore, the XVSM Micro-Room Modeler provides some helper functions that can be of use within the UI files (cf. Table 6.2).

Finally, the module manifest has to be added as shown in Section 5.3.1. It includes the name of the module collection and its unique namespace. After everything is in place, the files simply need to be packaged as a JAR file, which then can be distributed to the end users. Therefore, an

Helper functions	Description
<code>navigate(roomName)</code>	Replaces the current micro-room UI with the UI of another micro-room, i.e. navigates to another micro-room UI.
<code>getPlugin(pluginName)</code>	Retrieves a plugin (cf. Listing 5.10) by name.
<code>getProps(mr, propName)</code>	Extracts the property value of a given micro-room.

Table 6.2: JavaScript Helper Functions Provided by the XVSM Micro-Room Modeler

own platform might be created in the future (cf. Section 8.5). Each end user can then upload the module files he/she requires to the XVSM Micro-Room Modeler directly via its UI.

6.3 Using the XVSM Micro-Room Modeler

As the XVSM Micro-Room Modeler has been designed with usability in mind, it is relatively easy to use. After registering an account and logging in, a user can create a new project that requires not more than a name (cf. Figure 6.1).

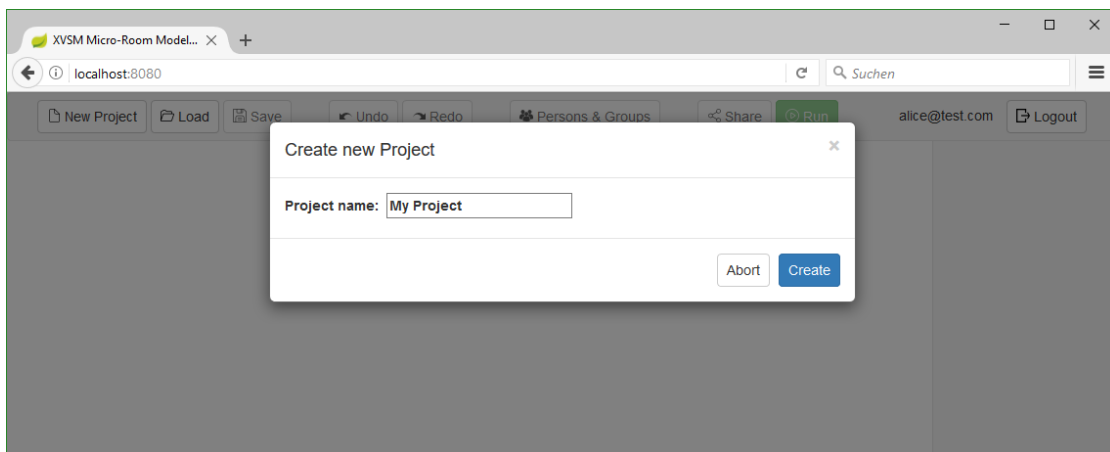


Figure 6.1: Creating a new Project in the XVSM Micro-Room Modeler

Once the project name has been entered, the user is directly forwarded to the groups and members management dialog. With this dialog, all members and groups can be created that should collaborate in the workflow to be designed. Based on these groups and members, room and action permissions can be configured later on.

As can be seen in Figure 6.2, members can be defined on the left while groups are defined on the upper right. When selecting a group (green background color), members can be assigned to the group via drag and drop or by checking the checkbox of the desired members and clicking the “>” button. Members of a group are then shown in the lower right area.

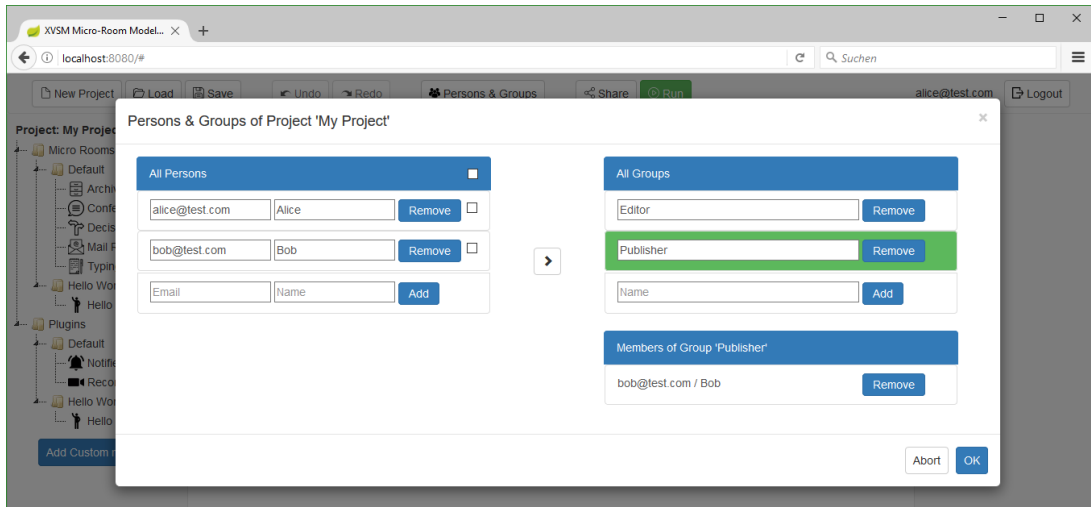


Figure 6.2: Configuring Groups and Members in the XVSM Micro-Room Modeler

Now, the model itself can be created. Figure 6.3 shows all features of the XVSM Micro-Room Modeler at a glance.

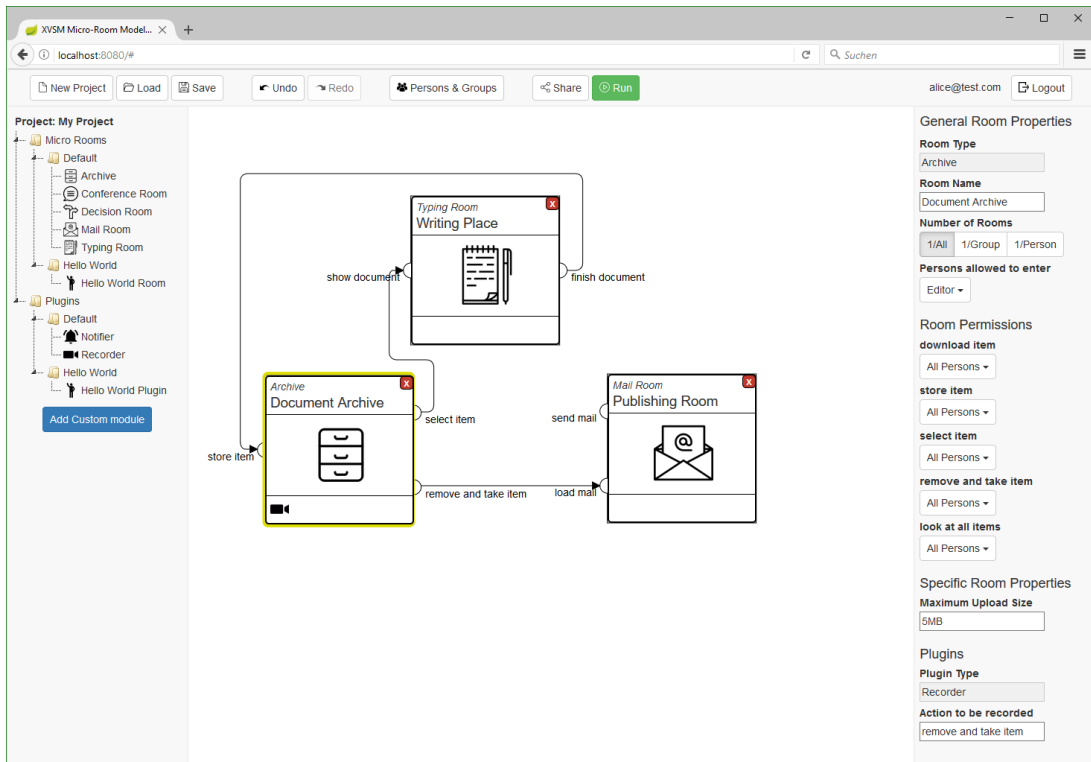


Figure 6.3: Creating a Model in the XVSM Micro-Room Modeler

In the menu at the top, projects can be created, loaded and saved (from/to the XVSM Micro-Room Modeler backend, i.e. XVSM containers). Every action can be un- and redone (cf. Section 5.5.4) and the groups and members dialog previously described can be opened again.

On the left, all available micro-rooms and plugins are shown in a tree view, where the root node is equal to the name of the module file in its manifest file (cf. Section 5.3.1). By clicking on the `Add custom module` button, a custom module JAR file can be uploaded, immediately extending the module tree with the new micro-rooms and plugins (cf. `Hello World` micro-room and plugin).

From the module tree, micro-rooms can be dragged anywhere into the modeling area while plugins can be dragged onto any micro-room placed in the modeling area. Each of the actions of a micro-room can be connected to a suitable other one (cf. smart links described in Section 5.5.5) by dragging a connection from the source port of the room to the target port of the other room. Also, micro-rooms and connections can be moved around anywhere in the model.

If a micro-room is selected via a single click, the properties pane appears in the right of the modeler. It allows to configure all generic micro-room settings, i.e. name, number of rooms, permission to enter and permission to access a specific action. Beside that, all room-specific properties can be set via text fields (e.g. `Maximum Upload Size`) and plugins that are assigned to the micro-room are also configured with this pane (e.g. `Recorder` plugin).

When the model is created as desired, it can be saved and shared with other project members. Therefore, a special invitation link can be created by clicking on the `Share` button (cf. Figure 6.4).

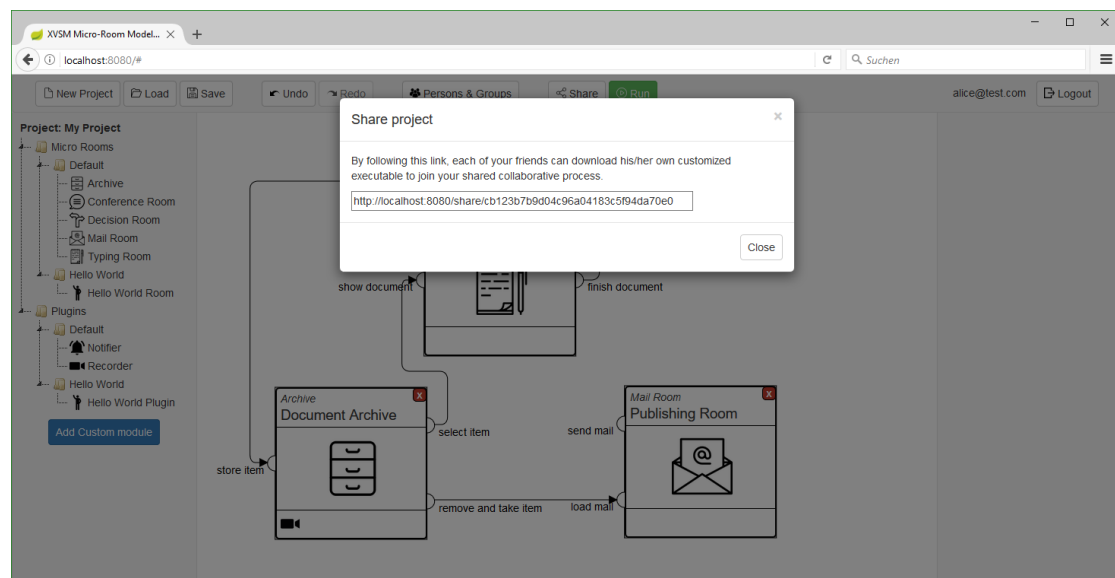


Figure 6.4: Creating an Invitation Link in the XVSM Micro-Room Modeler

Please note that the link will always be created relatively to the modeler's URI and is only `localhost:8080` in the screenshot because it has been deployed locally in this case. This link can be sent to all members. After opening it, they will need to log in and can then download

their customized application, based on the designed model (cf. Section 4.3). If they are not members of the project, they will see an error message (cf. Section 5.5.6). The user's own customized application can be downloaded and executed by clicking on the Run button.

6.4 Using applications created by the XVSM Micro-Room Modeler

After downloading the customized application, it can be executed by double clicking it. Then, it will extract its contents and launch the `start.bat` file, thereby starting up the XVSM Micro-Room Framework. As soon as the XVSM Micro-Room Framework has fully initialized, it will open the start page (i.e. `ui/index.html`) in the system's default browser (cf. Section 5.2.2).

On the start page, the user has to log in with the same credentials as in the XVSM Micro-Room Modeler (cf. Figure 6.5). The username cannot be altered in the shown input field as this application is customized for exactly this user (i.e. `alice@test.com`). Please also note that the UI files are indeed accessed directly from the file system without any web server in place.

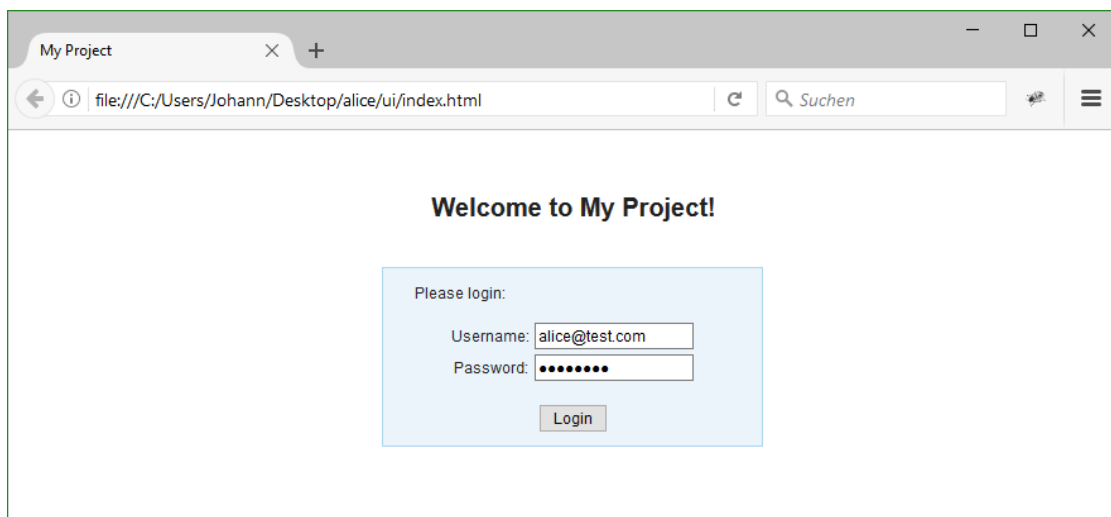


Figure 6.5: Logging into the Created Application

In the background, an authentication token is retrieved from OpenAM that is used to authenticate all action calls against the XVSM Security constraints of the XVSM Micro-Room Framework. After logging in, the first micro-room modeled and accessible by the user is shown (cf. Figure 6.6). On the left, a list of all accessible micro-rooms can be seen and navigated to at any time (cf. Section 5.5.1).

In the shown micro-room `Document Archive` one can see that several actions are possible. The `Upload` and `Download` buttons call internal actions (i.e. they are not exposed as described in Section 5.2.1), allowing users to upload or download files into/from the archive. Therefore, these actions are not connectible in the modeler, as can be seen in Figure 6.3.

The defined customized property `Maximum Upload Size` is used when uploading files and is shown as information below the action buttons. Please also note the `Show History`

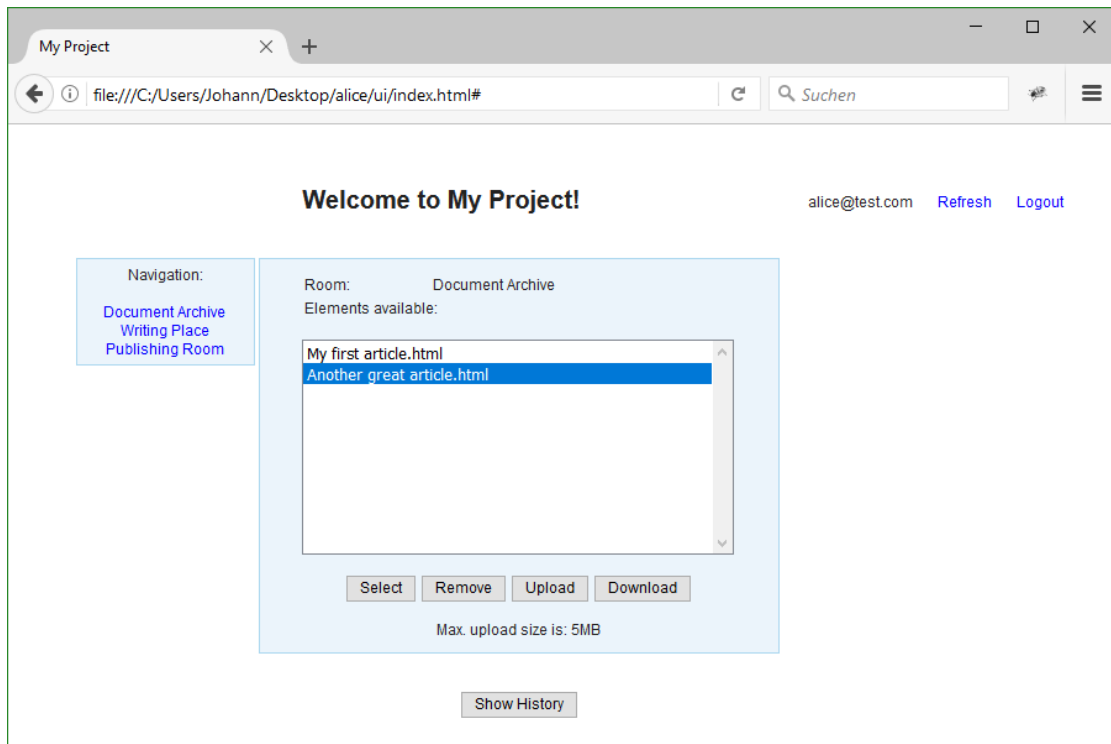


Figure 6.6: Using a Micro-Room in the Created Application

button on the bottom. It is not located in the actual UI area of the micro-room but beneath it in the plugin UI area (cf. Section 5.4.1). Here, all UIs of plugins are shown that have been assigned to the micro-room. In the example, the `Recorder` plugin records all calls of the `remove` and `take item` action, i.e. `Remove` button clicks. These recordings can then be shown by clicking on the `Show History` button provided by the plugin UI.

The `Select` and `Remove` buttons are directly related to the connected actions defined in the model (cf. Figure 6.3). Hence, if the `Select` button is clicked, the `select item` action of the `Document Archive` is called with the selected item. Within the `XVSM Micro-Room Framework`, this item will be forwarded to the `show document` action of the `Writing Place` and a `redirect` parameter will be returned to the UI, navigating it to the `Writing Place`. On load of the writing place, the current item to be shown will be loaded from the `XVSM Micro-Room Framework`, being exactly the item from the `Document Archive`, set just moments before.

From a user's perspective, none of these background processes will be visible. For him/her, it appears as if he/she simply takes the item from one room into the other. When switching to another room without following the flow, the currently selected item will remain in the `Writing Place` until he/she navigates to it again or replaces it by selecting another document from the `Document Archive`.

A very important fact is that all micro-room UIs are designed in a way so that they can

be used completely standalone, i.e. without any connections at all. E.g. the `Archive` can be used to up- and download files, thereby allowing file exchange with other members due to the automatic replication in the background. A `Typing Room` can be used to create HTML together with a WYSIWYG editor and the `Mail Room` allows to create and send mails to arbitrary persons. With the `Decision Room`, one can import an item and a second person can decide what to do with it, while a `Conference Room` allows to chat with all members in realtime.

All of this functionality can be used standalone but at the same time could be connected to actions of other micro-rooms. E.g. the items from other rooms can be stored into the `Archive` or the `Archive` can provide items for other rooms itself. Items can be sent to predefined users directly by passing them to the `Mail Room` and edited if they are passed to the `Typing Room`. A simple form of flow control can be achieved by sending items to a `Decision Room` where a user decides about the further destination of the item.

To sum things up, an application created with the XVSM Micro-Room Modeler can be used in various ways to achieve very different goals. End users can realize simple collaborative tasks like chatting or file exchange and at the same time create workflow-like scenarios containing multiple steps that can be performed by different users.

Evaluation

In this chapter, the evaluation of our work is performed. Therefore, first the modeler itself is evaluated on a theoretical basis. Then the general approach for the usability evaluation is outlined, followed by the results of the evaluation.

7.1 Theoretical Evaluation of the XVSM Micro-Room Modeler

First of all, we will answer research question 1: “*How can a suitable modeling tool for the XVSM Micro-Room Framework be designed and implemented?*”

In Chapters 4-6, we made several decisions to create a suitable modeler for the XVSM Micro-Room Framework. To evaluate whether they were expedient or not, the requirements for a decent modeling technique specified in Section 2.3 will be taken as a basis. In the following, each of the requirements will be evaluated regarding its fulfillment by the final implementation of XVSM Micro-Room Modeler.

Automation Support: Since the XVSM Micro-Room Modeler now allows end users to create a model just like any other modeling tool instead of writing XML files, the first drawback of the XVSM Micro-Room Framework, failing this criterion, is effectively fixed. The created model (i.e. business logic file) can be executed with the XVSM Micro-Room Framework automatically.

Human Task Support: The second drawback of the XVSM Micro-Room Framework is overcome by providing UIs with the module files, which has been enabled by our work (cf. Section 5.3.3). Hence, IT experts deliver corresponding UIs with their micro-room implementations, thus allowing end users to interact with their created workflow via these UIs.

Distributed Execution: This criterion is inherently fulfilled as the XVSM Micro-Room Framework is based on XVSM and heavily focuses on shared P2P workflows where multiple users all run their own instance of the created application.

Multiple Domains: To allow arbitrary domains to be modeled, end users can upload custom module files to the XVSM Micro-Room Modeler, thus extending its feature set (cf. Section 6.3). The problem of end users themselves not being able to create such module files could be solved by IT experts creating various module suites that are published on a platform in the Internet, where end users could choose those they need (cf. Section 8.5).

Tool Simplicity: The simplicity of the tool correlates directly with the simplicity of the used micro-room concept itself as well as the reduced feature set of the XVSM Micro-Room Framework if compared to e.g. the Peer Model, UML or BPMN. Additionally, great effort is taken to keep the modeler as simple as possible. The fulfillment of this requirement can be directly concluded by looking at the results of research question 4 (cf. Section 7.3.3).

Comprehensible Abstraction: To ensure that the abstraction of the modeling tool is easily understandable, we use the micro-room concept which has been designed for exactly this case in the first place. The justification of this assumption is done by answering research question 2 (cf. Section 7.3.1).

Easy Deployment: Finally, to enable end users to deploy and execute their applications as simple as possible, our modeling tool creates the whole executable application via a single click. The end user is then prompted to download it and can start it via double-click. Additionally the XVSM Micro-Room Modeler provides a possibility to “share” the model, i.e. send a link to a friend who can then download his/her customized executable application to join the specified collaborative workflow. Details about these features have been described in Sections 4.4 and 5.4.

As can be seen, the *XVSM Micro-Room Modeler* (XMRF) fulfills all of the requirements initially defined for creating a decent modeler targeting end users. Similar to Figure 2.20, Figure 7.1 shows the comparison of all evaluated modeling techniques, but this time including the XVSM Micro-Room Modeler.

		UML	BPMN	WS-BPEL	ISEA	HBPM	WPM	FWSC	WSCF	EUPM	PM	XMRF	XMRF
Features	Automation Support	⚠	✓	✓	✗	✗	✓	✓	✓	✓	⚠	✓	✓
	Human Task Support	✗	✓	✓	✗	✗	⚠	✓	⚠	✓	✗	✗	✓
	Distributed Execution	✗	✓	✓	✗	✗	⚠	✓	✓	⚠	✓	✓	✓
	Multiple Domains	✓	✓	✓	✓	✓	✓	✓	✓	✗	✓	✓	✓
End User Suitability	Tool Simplicity	✗	✗	✗	✓	✓	✓	⚠	✗	✓	✗	✗	✓
	Comprehensible Abstraction	✗	✗	✗	⚠	✓	✗	✗	⚠	✓	✗	✗	✓
	Easy Deployment	✗	⚠	✗	✗	✗	✓	✗	✗	✓	✗	✓	✓

Figure 7.1: Comparison of Modeling Techniques including the XVSM Micro-Room Modeler

Besides these main requirements, also other factors are of importance, such as performance and security.

7.1.1 Performance

Concerning the performance of the XVSM Micro-Room Modeler, no evaluation has been made. However, since it is a standard Spring Boot web application that is accessed via REST, other available performance benchmarks can be used as a reference. In [16] the author tested various microservice frameworks by simulating 200 users making 1000 requests each, both writing and reading. He comes to the conclusion that Spring Boot with an embedded Tomcat (i.e. exactly the technology used for the XVSM Micro-Room Modeler) is capable of serving 814 requests per second, which is only 8.3% less than the best result of the evaluation.

When looking at the XVSM Micro-Room Modeler, this means that 814 REST calls from end users could be made simultaneously per second. It has to be noted that such calls are only made for specific actions such as creating, saving and loading a project, adding custom modules, creating the executable application and creating the invitation link. Hence, it becomes obvious that 814 of such action calls per second would require a lot of simultaneously logged in users. Considering that a user spends most of the time by actually creating the model and not by executing one of the mentioned actions, it is estimated that an ordinary end user performs on average 1 of those actions per 20 seconds. Thereby, the XVSM Micro-Room Modeler could serve more than 16,000 simultaneously logged in users, which is far more than enough.

Regarding the created applications, performance is directly related to the performance of the underlying XVSM Micro-Room Framework. In [Bin13] the performance of the XVSM Micro-Room Framework has already been evaluated. Thereby, it has been found that the XVSM Micro-Room Framework can handle up to 100 simultaneous outgoing replication commands per second without any delay. E.g. a user of an executable application created by the XVSM Micro-Room Modeler could click 5 times on a button performing an action in the XVSM Micro-Room Framework that is replicated to 20 other users currently online without any delay. If more simultaneous outgoing replication commands are necessary because more users are online at the same time or more actions are executed simultaneously, the actions will be replicated with a delay.

This limitation is caused by poor performance of the underlying Paxos Commit protocol used for the distributed transactions across all peers. However, as in our work we are dealing with simple collaborative workflows between only few simultaneously online friends, this limitation is not of big concern. It is true that *“the limit of 100 transaction participants per second is sufficient for common small- to mid-scale scenarios”* [Bin13].

7.1.2 Security

The security of the XVSM Micro-Room Modeler is ensured by Spring Security¹. It restricts access to all pages except the login and the registration page to verified OpenAM users. A model is only accessible by the creator of the model, hence no harmful manipulations of other

¹<https://projects.spring.io/spring-security/>

users are possible. The invitation link usable by others only allows to download the executable application. Thereby, it is ensured that only users that are part of the model are allowed to download it (cf. Section 5.5.6).

Concerning the the created applications, their security is tightly depending on the security concept of the underlying XVSM Micro-Room Framework, such as described in [Bin13]. Basically, interaction between created applications of several users happens via replicated actions. Every time a user triggers an action (that should be replicated), it is replicated to all other users currently online. Users who start their application later on will receive the full history of replicated actions they missed upon startup.

A user could try to manipulate these actions, e.g. by sending fake replicated actions to other users in which he/she calls an action that he/she has no permissions to according to the model. To prevent this, all replicated actions received are validated against the central OpenAM server via XVSM Security. If a replicated action is not permitted, it will not get executed on the remote user's application.

Another possibility would be that the user changes the model or specific JAR files of the created application before startup to e.g. allow him-/herself access to all micro-rooms. The user would then have access to his local set of micro-rooms but will not see any data since he/she does not receive any replicated actions. The reason for this is that the replication targets are derived from the model on the sending side, where the model is still unmanipulated.

7.2 Usability Evaluation Approach

To answer research questions 2-5, a usability evaluation has been performed. Details about the specific approach are outlined in the following.

At the first glance, regarding evaluation of BPM languages and tools, a lot of related work exists. In [SC14] Shitkova analyzed and compared 62 papers published between 2005 and 2013 dealing with usability evaluation of BPM languages and tools. She comes to the conclusion that *“the number of thorough usability studies in BPM is still quite small”* and that *“mostly these solid studies are conducted for BPM languages, but not for BPM tools”*.

Nevertheless, we try to evaluate both our language and our tool to answer the research questions stated in Section 1.2. Our approach is based on [ESJK11], [SRDS00] and [SK15]. The basic procedure relies on the approach of usability evaluation of BPM tools (cf. Figure 7.2) presented in [ESJK11].

In contrast to the approach we skip the heuristic evaluation and the improvement of identified shortcomings. These will be resolved in the future (cf. Section 8.2). Also, we do not iterate the process several times but only perform it once. So our approach consists of a usability testing, followed by questionnaires and the evaluation of the results.

As in [SRDS00] we perform the usability testing with two products of similar functionality, i.e. the XVSM Micro-Room Modeler and Signavio Process Editor (cf. Section 2.1.2) and give each of the users a short theoretical introduction about each tool. Like Scheller and Kühn in [SK15] we capture each user session on video, hence allowing detailed analysis of usability problems afterwards. E.g. the time a user required for modeling a task can be measured, as well as at which parts of the UI they required help or at which tasks they completely gave up.

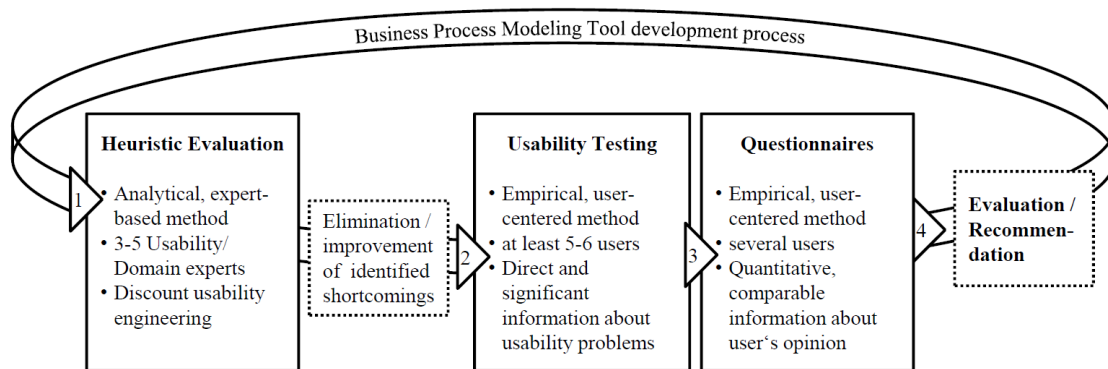


Figure 7.2: Approach of Usability Evaluation of Business Process Modeling Tools [ESJK11]

According to [Jor08] several evaluation methods are available during usability testing. These are think-aloud protocol, video analysis, input protocols, eye tracking, logfile recording, interviews and observation. Beside video analysis, we also use think-aloud protocol (i.e. end users describing what they think when performing a task), observation and partial interviews (several questions as specified in Section 7.2.1).

Regarding the number of users to test, Eiffinger et al. state that at least 5-6 users should perform the usability testing [ESJK11]. Lewis calculated via binomial distributions that five to eight users are enough to discover 85% of all usability problems available [Lew06]. Nielsen developed a mathematical model from various user evaluations, describing the percentage of available problems found, depending on the number of users [NL93]. Curves of this model can be seen in Figure 7.3, where λ represents the probability of finding an average usability problem by an average user in a single evaluation.

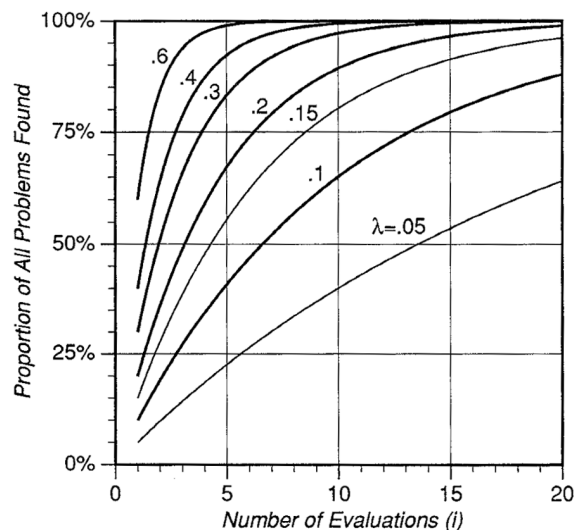


Figure 7.3: Proportion of Usability Problems Found Depending on the Number of Users [NL93]

According to Nielsen, “the typical value of λ is 31%, averaged across a large number of projects we studied” [17], hence, 5 test users would lead to about 85% of all problems found.

Based on the literature provided we use 10 test users for our usability testing. According to Figure 7.3 this leads to approximately 95% of all usability problems found when assuming λ being 31%.

In the following sections, more about the setup of the usability testing and the questionnaire used will be outlined.

7.2.1 Usability Testing Set-Up

The usability testing is performed as follows: First, the user is shortly informed about the capabilities of the XVSM Micro-Room Modeler, i.e. modeling collaborative applications by coarse-grained building blocks and execute them by a single click. Then, the user is questioned for which use cases he/she might need such a tool, before even seeing the XVSM Micro-Room Modeler for the first time.

After that, the user models the following *Tasks* (T) with the basic set of micro-rooms provided in the XVSM Micro-Room Modeler (cf. Section 4.3) and explains everything he/she thinks:

- T 1: Chat with your friends Alice and Bob in a global chat room.
- T 2: Chat with your friends Alice and Bob in a private chat room. Chat with your co-workers Carol and Dave in another private chat room.
- T 3: Exchange files with Alice.
- T 4: Create documents together and exchange them with your friends Alice and Bob.
- T 5: Compose and send mails to a mailing list (list@mail.com) cooperatively with Alice and Bob.
- T 6: Log who sent mails to the mailing list.
- T 7: Editors (Alice and you) compose mails together and store them while publishers (Bob and you) can choose and send them to the mailing list.
- T 8: Publishers can reject mails to the editors.

While executing the tasks, the user is recorded on video with the integrated camera of the testing notebook using Free2X Webcam Recorder². The user’s screen is recorded at the same time with Free2X Screen Video Recorder³. If the user cannot solve a task he/she can either ask for help or abort the task. Both of these actions are recorded, as well as the task execution time. Also, special attention is on detecting where users have difficulties with the current terms, explanations and abstractions.

²<http://www.free2x.com/webcam-recorder/>

³<http://www.free2x.com/screen-video-recorder/>

After all tasks have been completed, the user has to fill out a questionnaire concerning usability of the XVSM Micro-Room Modeler (cf. Section 7.2.2). Then, he/she is asked the question from the beginning again, i.e. for which use cases he/she might need such a tool. An execution of the created models would not benefit to answer any of the research questions of this work and hence is not performed. Also, the overall procedure already takes about 2 to 2.5 hours and should not be extended unnecessarily.

Now, the same tasks defined above have to be modeled with the Signavio Process Editor, a BPMN modeling tool. Thereby, large abstractions are allowed, e.g. users may define coarse-grained activities like “create document” or “send mail” without specifying any further detailed sub-processes. Again, the execution time for each task is recorded, as well as when the user asks for help or aborts a task. Afterwards, the same questionnaire has to be filled out as with the XVSM Micro-Room Modeler.

It might be that working with the first modeling tool might influence the way subjects try to solve tasks with the second modeling tool, i.e. they might try to solve them in the same way. To avoid biasing the results into our favor, 5 subjects use the XVSM Micro-Room Modeler first and the other 5 subjects use the Signavio Process Editor first. Also, those 5 subjects do not receive any information about the XVSM Micro-Room Modeler or the micro-room concept until they finished modeling all tasks with the Signavio Process Editor and answered the corresponding questionnaire.

Finally, the following set of *Evaluation Questions* (EQ) are asked as they are valuable for our research questions:

- EQ 1: Did you have problems to understand the micro-room concept?
- EQ 2: Did you find it more or less familiar than the second approach?
- EQ 3: For which problems would you use the first tool, for which the second?
- EQ 4: Which tool did you find easier to use and why?

The whole process has been transformed to an evaluation sheet that is used by the evaluator during the usability testing.

7.2.2 Questionnaire

The questionnaire used is based on the official ISO 9241/10 standard [Deu95], defining principles of designing UI dialogs. Currently, there are two main questionnaires available that rely on these principles: ISONORM [18] and IsoMetrics [Ged99].

In [Fig09], Figl evaluates both questionnaires via user testing. The results of the evaluation state that IsoMetrics is slightly better when it comes to expressiveness of the users’ thoughts as well as total time required to answer all questions. Hamborg compares IsoMetrics to other user evaluation methods such as think-aloud protocol, video recording or heuristic evaluation in [Ham02]. He comes to the conclusion that with IsoMetrics, 3 to 5 times more notes will be recorded regarding usability problems than with the other approaches. Thus, he states that

IsoMetrics is the most effective method for usability testing. Based on these findings, we choose IsoMetrics in our work.

IsoMetrics contains a total of 75 items (i.e. statements about the evaluated software) distributed among the seven dialog principles defined in [Deu95]:

1. Suitability for the task: The user can realize his/her task effectively and efficiently. Only those parts of the software are shown that are required to perform the task.
2. Self descriptiveness: Every step is understandable and intuitive. In case of errors, immediate feedback and support is provided.
3. Controllability: The user is in charge of the start, direction and speed of the actions he/she performs.
4. Conformity with user expectations: The dialog is consistent and takes the user's knowledge into account. General conventions are considered.
5. Error tolerance: In case of errors, no or only minimal additional effort is required to reach the former goal.
6. Suitability for individualisation: The system allows customization regarding the task as well as the individual preferences and capabilities of the user.
7. Suitability for learning: The effort for learning is as low as possible. The user is accompanied through all stages of knowledge until fully understanding the dialog.

There are two version of IsoMetrics, i.e. IsoMetrics^L and IsoMetrics^S. IsoMetrics^S is the short version of the questionnaire consisting of only 8 pages. Thereby, each item can be answered by selecting the level of agreement from 1 (predominantly disagree) to 5 (predominantly agree). Additionally, it is possible to select "No opinion". Figure 7.4 shows an example.

		Pre-dominantly disagree		So - so		Pre-dominantly agree			
Index		1	2	3	4	5	No opinion		
A.1	suitability for the task The software forces me to perform tasks that are not related to my actual work.								

Figure 7.4: Example Item in IsoMetrics^S [GHW98]

IsoMetrics^L on the other hand is the long version, having a total of 78 pages. Beside the answering possibilities of IsoMetrics^S, users can also specify the importance of the item for themselves from 1 (unimportant) to 5 (important). Furthermore, an example justifying the answer should be given for each item. An example item can be seen in Figure 7.5.

		Pre-dominantly disagree		So - so		Pre-dominantly agree	No opinion
A.1	The software forces me to perform tasks that are not related to my actual work.	1	2	3	4	5	
		Un-important		So - so		Important	No opinion
	Please rate the importance of the above item in terms of supporting your general impression of the software?	1	2	3	4	5	
Can you give a concrete example where you can agree with the above statement?							

Figure 7.5: Example Item in IsoMetrics^L [GHW98]

Due to the large number of pages of IsoMetrics^L, we use IsoMetrics^S. However, we adjust it slightly, also allowing users to specify the importance of each item by writing the corresponding number from 1 (unimportant) to 5 (important) to the right of the item. The full questionnaire can be seen in the appendix of this work.

7.3 Usability Evaluation Results

In this section we will present our findings based on the usability evaluation approach described in Section 7.2 with a target group of 10 subjects. Out of these subjects, 4 were male and 6 were female. All of them were end users without IT knowledge and their age varied from 25 to 40.

The recorded data was accumulated via Microsoft Excel to perform data analysis and to generate graphs. In the following we will use the data to answer each of the remaining four research questions independently.

7.3.1 RQ 2: Abstraction of the Micro-Room Concept

Research question 2 is answered in this section: *‘Is the abstraction used by the micro-room concept more comprehensible than those of common process modeling languages?’*

To achieve this, several aspects are considered. In the questionnaire, the items shown in Table 7.1 target the underlying concept (i.e. the micro-room concept or BPMN) of the tool under evaluation.

Index	Item
S . 12	The terms and concepts used in the software are clear and unambiguous.
L . 1	I needed a long time to learn how to use the software.
L . 4	So far I have not had any problems in learning the rules for communicating with the software.
L . 5	I was able to use the software right from the beginning by myself, without having to ask coworkers for help.
L . 6	I feel encouraged by the software to try out new system functions by trial and error.

Table 7.1: IsoMetrics^S Items Related to Abstraction of the Underlying Concept

When accumulating the average score for each of these items, it can be seen in Figure 7.6 that the XVSM Micro-Room Modeler achieved with 20.1 points twice as much as the Signavio Process Editor with 10 points. Both could have reached a maximum of 25 points (5 items with a maximum of 5 points per item). Please note that the points of item L . 1 have been inverted since it is negatively formulated. More about this can be read in Section 7.3.3.

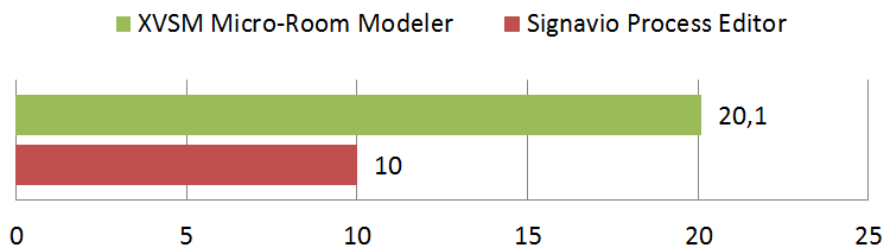


Figure 7.6: Accumulated Average Score for S.12, L.1, L.4, L.5, L.6

Besides that, the comparison of the time to complete all tasks allows to draw conclusions about the comprehensibility of the abstraction used. It can be argued that a more comprehensible abstraction reduces the time to perform the tasks, since the subject requires less thinking, has to ask fewer questions and can avoid time-consuming trial and error.

Before looking at the details, please note that special attention has been paid to not bias the results regarding time taken. Therefore, every subject got about one minute time to make proper progress before the evaluator interrupted and gave a hint. This also holds for subjects that asked for help immediately. They ought to at least try to find the UI elements they needed instead of letting the evaluator explain everything. Therefore, the individual frustration levels of the test subjects could be neutralized and have been converted into more objective “one minute penalties” instead.

Figure 7.7 shows the total average time taken for a test subject to finish all tasks as specified in minutes.

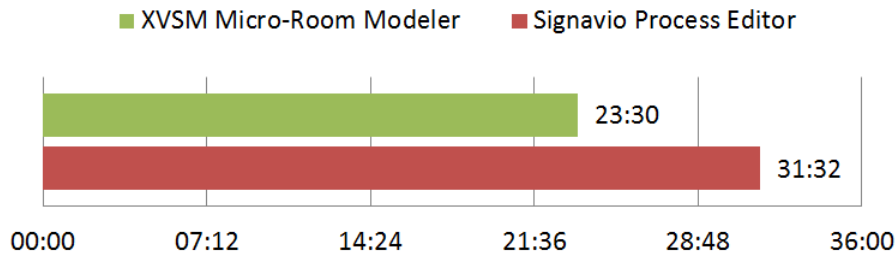


Figure 7.7: Average Time for all Tasks in Minutes

It can be seen that the test subjects needed about a third longer to model all tasks in the Signavio Process Editor when compared to the XVSM Micro-Room Modeler. The difference can be further evaluated when looking at the concrete durations for modeling each task separately. These numbers are shown in Figure 7.8.

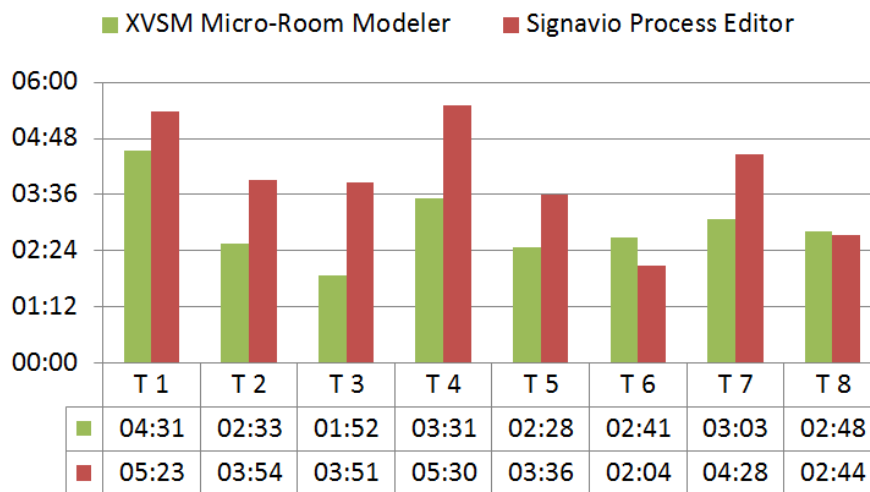


Figure 7.8: Average Time / Task

All tasks except task 6 and task 8 could be completed about one to two minutes faster with the XVSM Micro-Room Modeler when compared to the Signavio Process Editor. This can be explained when looking at how these tasks are modeled with the XVSM Micro-Room Modeler.

Task 6 introduces the `Plugin` concept, which has not been explained by the evaluator beforehand. Hence, test subjects were struggling and trying to solve the task with micro-rooms before either discovering the `Recorder Plugin` themselves (6 subjects) or asking the evaluator (4 subjects). It can be stated, that the `Plugin` concept is not as easy to understand for end users as the micro-rooms and needs special explanation to not confuse them.

Task 8 requires to use a new room (i.e. `Decision Room`) which takes the test subjects not significantly longer than in the previous tasks, hence indicating no direct problem in the underlying concept. The equal time taken in both tools can be explained more likely by looking at the Signavio Process Editor part. Here, subjects simply added a new “reject mail to editors”

task or modeled the decision itself with a “decide about mail” task having two outgoing edges, hence creating very informal models.

Tasks 3 and 4 took significantly shorter to model with the XVSM Micro-Room Modeler, indicating that the abstraction was especially easy to understand for these tasks. When looking at the tasks, task 3 introduces the `Archive`, hence end users seem to correctly interpret what they can do in this room at first glance, i.e. store and take out files.

Task 4 introduces the possibility to connect micro-rooms with each other, thereby creating workflows. Here, all of the subjects wanted to connect both rooms required (i.e. `Typing Room` and `Archive`), thereby indicating no problem with the micro-room concept itself. However, half of them failed to do so because of the implementation of the modeler and needed help by the evaluator, pointing them to try dragging a port (i.e. an action) of a micro-room instead of e.g. trying to drop the micro-rooms on each other to connect them (3 subjects). This explains the slightly increased time taken in contrast to the other tasks. Nevertheless, task 4 could be modeled still a lot faster with the XVSM Micro-Room Modeler than with the Signavio Process Editor. Hence, it can be concluded that the concept of connecting rooms itself is understandable by end users.

Finally, we use EQ 1 and EQ 2 as well to answer this research question. When looking at EQ 1, seven of ten subjects answered that they had no problems at all to understand the micro-room concept. One subject stated that she understood the concept except for plugins, another one stated that he did not understand it completely until seeing and using the XVSM Micro-Room Modeler. One subject did not understand the concept, stating that it was too abstract for her to use. Regarding EQ 2, nine subjects answered that the micro-room concept was more familiar, while one subject preferred Signavio Process Editor’s underlying BPMN.

After looking at all results, it can be clearly stated that the micro-room concept indeed uses a more comprehensible abstraction than common process modeling languages (represented by BPMN). First, the average score of all items regarding the underlying concept of the modeling tool is higher for the XVSM Micro-Room Modeler than for Signavio Process Editor. Second, tasks could be modeled significantly faster with the XVSM Micro-Room Modeler, thereby indicating that the underlying concept was easier to understand. Third, when looking at the tasks individually, subjects demonstrated that they understood what they were doing. Finally, nine of ten subjects state that they understood the concept (EQ 1) and that they find it more familiar than the the other approach (EQ 2).

7.3.2 RQ 3: Use Cases suiting the Micro-Room Concept

This section will answer research question 3: *“In which domains or use cases is the micro-room concept better or worse than its competitors?”*

It can be stated that the micro-room concept directly targets end users whereas BPMN targets business users. So, from a theoretical point of view, we can deduce that the micro-room concept has been designed to model simple, collaborative interaction scenarios that can continue endlessly, while BPMN is predestinated for more complex processes with a defined start and end point.

To verify whether these assumptions hold, all subjects were asked for which use cases they would use a tool like the XVSM Micro-Room Modeler. The answers and their naming count can be seen in Figure 7.9. Interestingly, three of the top five use cases mentioned (i.e. Chatting, Document Reviews and File Sharing) were already thought of by the author of the work and can already be modeled with the XVSM Micro-Room Modeler. Many of the other use cases have already been thought of in Section 4.3.

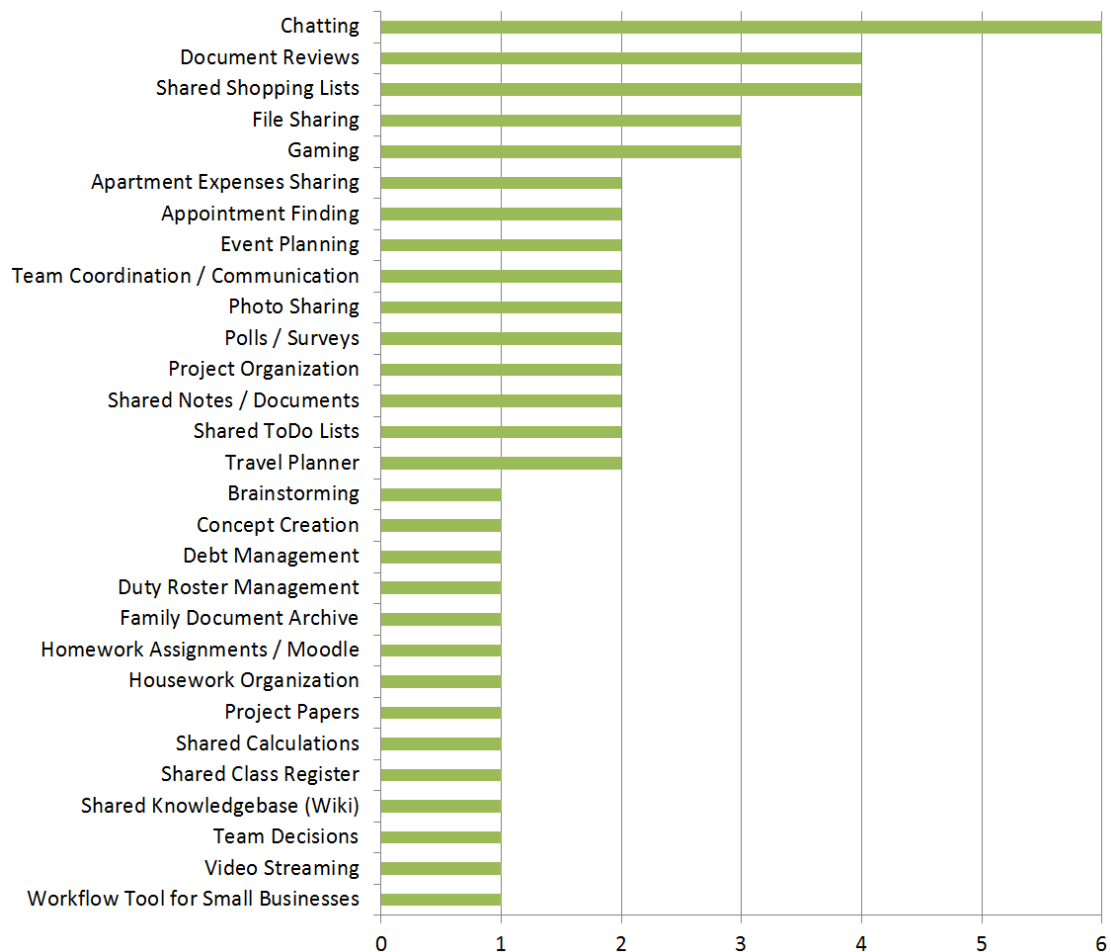


Figure 7.9: Use Cases for the Micro-Room Concept and their Naming Count

What they all have in common is that they cover simple collaborative tasks that end users face regularly in their daily life. When looking at the answers of EQ 3, this finding becomes prevalent. Figure 7.10 shows the answers of all ten subjects for which problems they would use which tool.

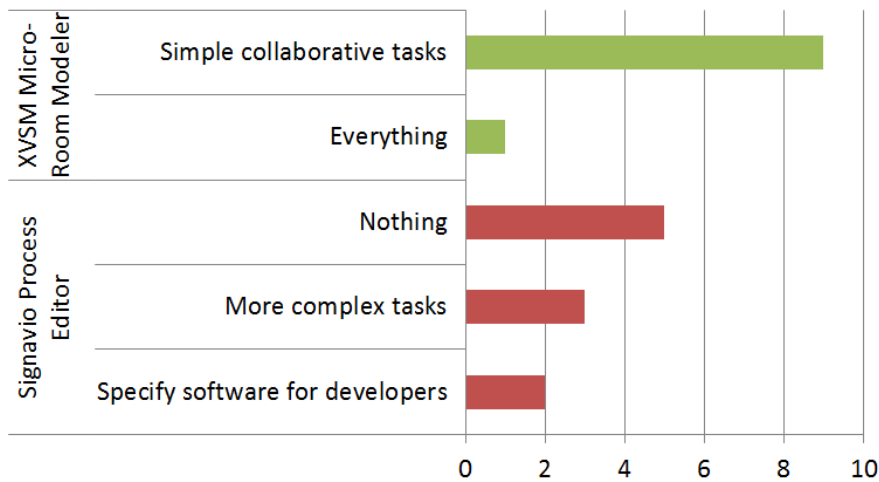


Figure 7.10: Preferred Usage per Tool (EQ 3)

The results clearly state the suitability of the XVSM Micro-Room Modeler and its underlying micro-room concept for simple collaborative tasks, such as the use cases listed in Figure 7.9. Half of the subjects could not imagine any problem where they would use Signavio Process Editor and its underlying BPMN. Three of them could think of more complex tasks that would be difficult to model with the micro-room concept due to its limitations (e.g. only one-to-one connections between micro-room actions). Two subjects concluded that this kind of software might only be useful to specify software for software developers, as the evaluator initially told them this was the main purpose of Signavio Process Editor.

Summarizing the findings concerning research question 3, the micro-room concept is better suited for modeling simple collaborative tasks out of the life of ordinary end users. Such tasks range from chatting, file exchange, document reviews, shared shopping lists, appointment finding and event planning over polls, photo sharing, project organization to travel planning, debt management and housework organization. When it comes to more complex tasks, such as e.g. logistics workflows or highly technical specifications of software for software developers, clearly other modeling approaches such as BPMN are better suited than the micro-room concept.

7.3.3 RQ 4: Usability of the XVSM Micro-Room Modeler

Now, research question 4 is discussed in further detail: *“Is the usability of the new modeling tool more intuitive in contrast to other modeling tools?”*

To compare the usability of both tools under test, first of all, the IsoMetrics questionnaire is evaluated as specified by the authors in [GHW98]. Therefore, all items answered with “no opinion” are set to median (i.e. 3 points) and all answers to negatively formulated items are transformed by $r'_i = 6 - r_i$ points. Afterwards, all points are summarized across all subjects for each tool. The final scores are shown in Figure 7.11.

As can be seen, the XVSM Micro-Room Modeler clearly outperforms the Signavio Process Editor concerning overall usability by more than 25%. When looking at the seven dialog princi-

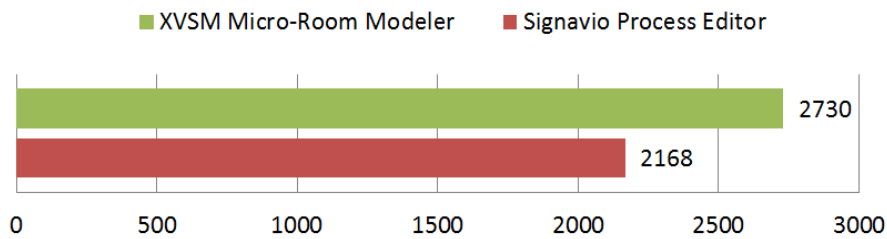


Figure 7.11: Total IsoMetrics Score per Tool

ples of the IsoMetrics questionnaire separately, we can further evaluate this result. Figure 7.12 shows the average points per dialog principle across all users per tool. A maximum of 5 points could be reached for each principle, meaning that all subjects answered all items of the principle with 5 points for the tool. Additionally, the average score per item can be seen, i.e. 3.64 for the XVSM Micro-Room Modeler and 2.89 for Signavio Process Editor.

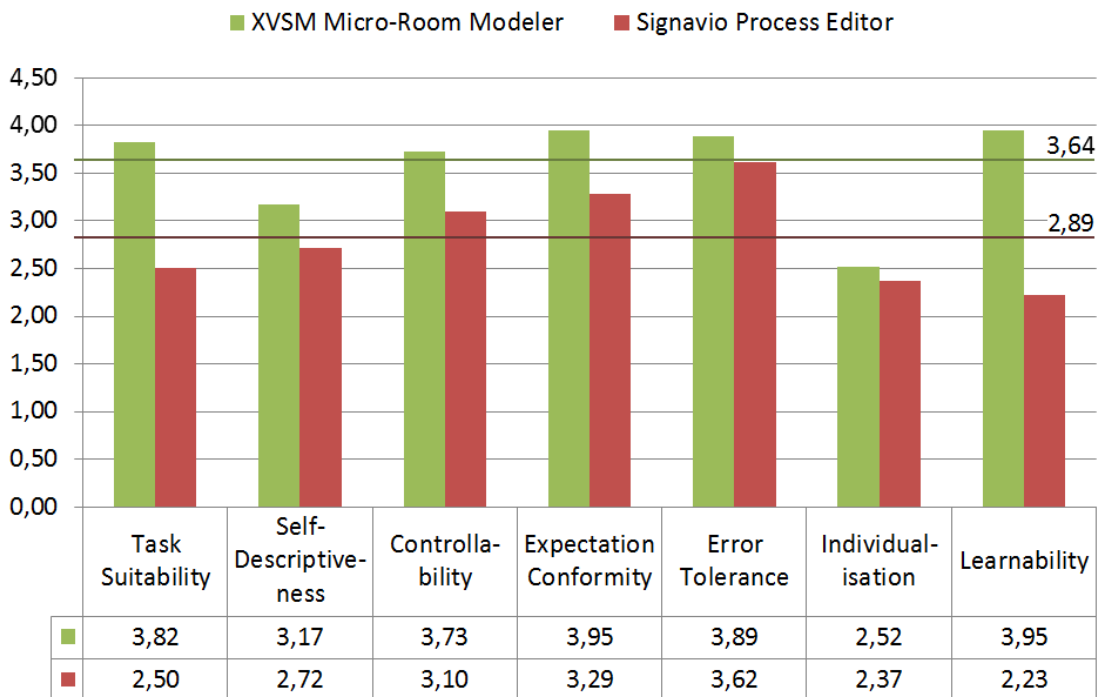


Figure 7.12: Average IsoMetrics Score per Principle

The XVSM Micro-Room Modeler outperforms the Signavio Process Editor in every single dialog principle. Most important is the almost twice as big score in *Learnability*, which indicates that the XVSM Micro-Room Modeler has a very flat learning curve and can be understood by end users without a lot of documentation, i.e. it is “intuitive”. *Task Suitability* directly shows that the XVSM Micro-Room Modeler is better suited for the given tasks (i.e. end

user related simple collaborative tasks) than the Signavio Process Editor (cf. research question 2).

Comparing the scores of *Individualisation*, the score of the XVSM Micro-Room Modeler is significantly lower than in all other principles. There are two reasons for this: First, the XVSM Micro-Room Modeler has been implemented as a prototype, hence no individualisation capabilities such as customizable menus, hotkeys or panels have been added, as these clearly do not belong to the core functionality. Second, as stated in Section 4.1.1, the tool to be implemented should be as simple as possible. Therefore, individualisation logic would be counterproductive as it adds more elements to the UI that might confuse the user.

Besides looking at the scores of *IsoMetrics*, also other metrics can be compared. Of all 80 tasks performed by the ten subjects per tool, zero have been aborted when modeling with the XVSM Micro-Room Modeler while three have been aborted when modeling with the Signavio Process Editor. Additionally, the number of questions asked during the usability testings is informative. In Figure 7.13 the overall number of questions asked by all ten subjects are shown.

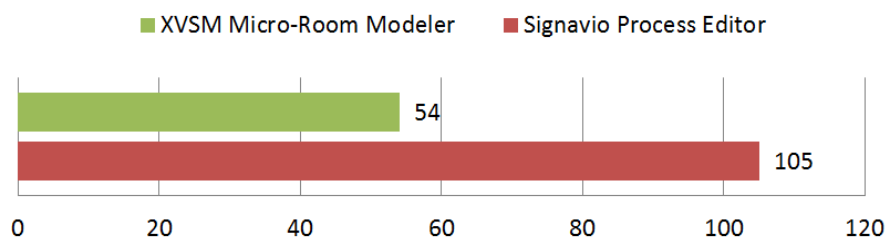


Figure 7.13: Total Number of Questions Asked

It becomes apparent that the XVSM Micro-Room Modeler is far easier to use than the Signavio Process Editor when looking at the number of questions asked. This also corresponds to the high *Learnability* score of the XVSM Micro-Room Modeler shown in Figure 7.12.

Finally, also EQ 4 can be used to answer this research question. Here, nine of ten subjects stated that they would use the XVSM Micro-Room Modeler rather than the Signavio Process Editor while one subject found the Signavio Process Editor less abstract. This is comprehensible, because the subject was allowed to model the BPMN models in a very simple informal way. If formally correct BPMN models were demanded, none of the subjects would have been able to solve the tasks with the Signavio Process Editor.

Covering up the results for research question 4, the XVSM Micro-Room Modeler clearly outperforms the Signavio Process Editor concerning usability. Not only does it reach 25% more points in the overall *IsoMetrics* score, but also does it achieve a higher score in each of the seven dialog principles separately. When looking at the number of aborted tasks (0 vs. 3) and the number of questions asked (54 vs. 105), the findings are further acknowledged. Finally, asking the test subjects directly about their opinion, nine of ten users state that the XVSM Micro-Room Modeler is more usable than the Signavio Process Editor.

7.3.4 RQ 5: Key Factors of a Modeling Tool for End Users

The last research question to be answered is as follows: “If so, what are the key factors for the intuitiveness of our modeling tool, or a modeling tool for end users in general? “

In contrast to research question 1, which has been argued on a theoretically defined set of requirements, the aim of this question is to find key factors empirically. Regarding the first part of the question, we can use EQ 4 to answer it. Figure 7.14 shows the main reasons stated by the test subjects why they would choose the XVSM Micro-Room Modeler over the Signavio Process Editor for modeling their tasks.

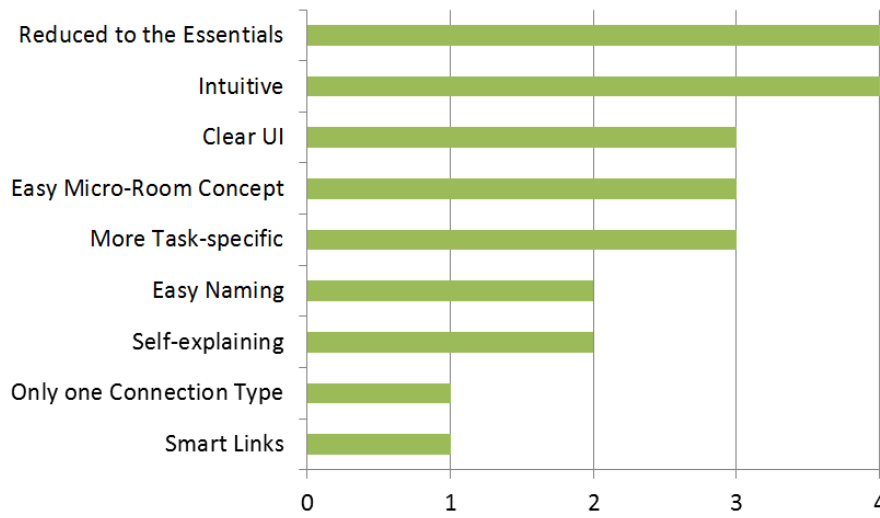


Figure 7.14: Reasons to use the XVSM Micro-Room Modeler and their Naming Count

Based on these answers, we can derive the key factors for the intuitiveness of our modeling tool and put them into relation to each other. Due to the micro-room concept, the modeling tool is more task-specific and allows an easier naming than the general-purpose BPMN. Also, it enables the modeler to be reduced to its essentials, which itself is mandatory for a clear UI.

The micro-room concept only allows one connection type in contrast to Signavio Process Editor, where five different connection types exist. This benefits the UI to be reduced to its essentials and also allows to provide smart links. All of the mentioned key factors contribute to the XVSM Micro-Room Modeler being self-explaining, which itself is the reason for the modeler’s intuitiveness. Figure 7.15 summarizes the described relations between the key factors for the XVSM Micro-Room Modeler’s intuitiveness.

Thereby, the term “intuitive” has been explained by a subject as follows: “I knew what I had to do and what each element was meant to do.” This ultimately can be reached by providing a self-explaining UI, which itself is depending on several other factors that all basically source in the micro-room concept. One subject put it in a nutshell by stating: “I can imagine what I am doing a lot easier with rooms and people.”

Considering the second part of the research question, we use the IsoMetrics questionnaire to identify the most important usability items for the subjects. As described in Section 7.2.2 we

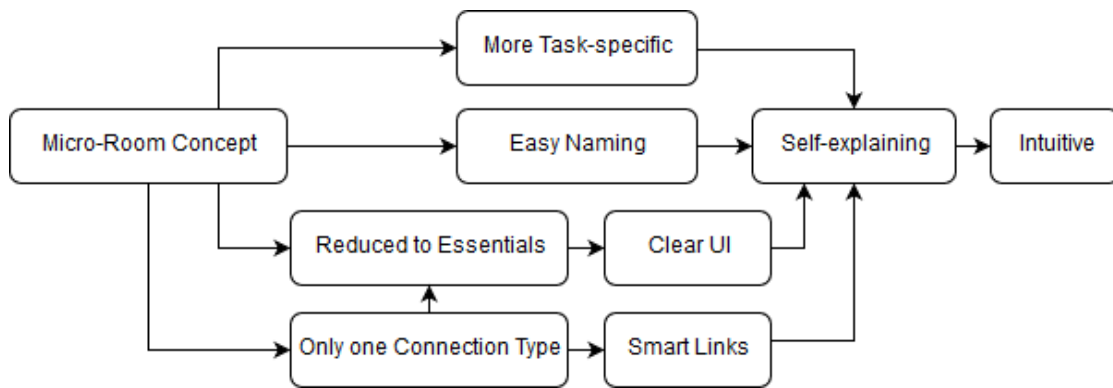


Figure 7.15: Relations between Key Factors for the XVSM Micro-Room Modeler’s Intuitiveness

extended the IsoMetrics^S questionnaire by allowing users to specify the importance of each item from 1 (unimportant) to 5 (important). The ten most important items are shown in Table 7.2 as well as their average importance across all subjects.

Index	Item	Average Points
A . 4	The functions implemented in the software support me in performing my work.	4.7
A . 3	The software lets me completely perform entire work routines.	4.3
A . 10	The software is well suited to the requirements of my work.	4.2
F . 3	If I make a mistake while completing a form, I can easily restore everything to its previous state.	4.2
F . 7	No system errors (e.g. crashes) occur when I work with the software.	4.2
L . 1	I needed a long time to learn how to use the software.	4.2
S . 12	The terms and concepts used in the software are clear and unambiguous.	4.1
F . 2	Even if I make a mistake, the information(e.g. data, text, and graphics) which I have just entered is not lost.	4.1
F . 8	If I make a mistake while performing a task, I can easily undo the last operation.	4.1
F . 9	I have never made an entry that caused a software error (e.g. a system/program crash or an undefined dialog state).	4.1

Table 7.2: Most Important IsoMetrics^S Items Across all Test Subjects

Effinger et al. state that “the ISO criteria self-descriptiveness, suitability for learning and error tolerance are highly relevant” for novice users of business process modeling tools [ESJK11]. According to our findings, this holds true except for some very fundamental requirements to the software (i.e. items A . 4, A . 3 and A . 10). These items form the newly defined key factor

“Required Functionality”. All other items come from one of the three Sections stated above.

Items F . 3 and F . 8 can be combined to the key factor “Undoability”, while F . 7, F . 2 and F . 9 reassemble to “System Stability”. L . 1 is transformed to “Easy Learnability” and S . 12 to “Understandability”.

Finally, Table 7.3 summarizes our derived key factors for an intuitive modeling tool targeting end users.

Key Factor	Description
Required Functionality	The modeler contains all functionality needed by end users to achieve their goal, but not more.
Undoability	Every action in the modeler can be undone easily.
System Stability	No error causes data loss or the modeler to crash.
Easy Learnability	Handling of the modeler can be learned easily.
Understandability	The terms and concepts used by the modeler are clear.

Table 7.3: Key Factors for an Intuitive Modeling Tool Targeting End Users

The performance of the XVSM Micro-Room Modeler and the Signavio Process Editor in these key factors can be evaluated by summarizing the points of the underlying items of the questionnaire and dividing them by the number of items. Figure 7.16 shows the results.

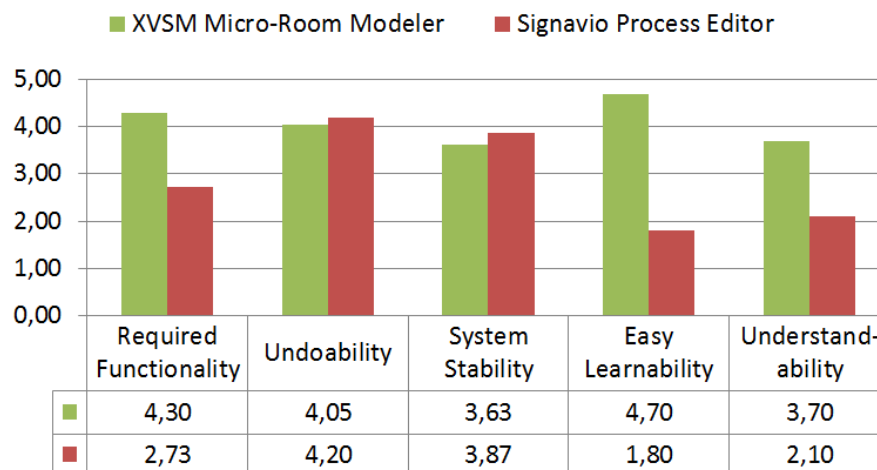


Figure 7.16: Evaluated Tool Performance in each of the Key Factors

It can be seen that the XVSM Micro-Room Modeler reaches a decent score of more than 3.5 points in each of the key factors whereas the Signavio Process Editor does so only for key factors “Undoability” and “System Stability”. Regarding the key factors “Required Functionality” and “Understandability” it cannot compete with the XVSM Micro-Room Modeler and trails by more than 1.5 points in these categories respectively. Special focus has to be set on “Easy Learnability”, where the XVSM Micro-Room Modeler outperforms the Signavio Process

Editor significantly by scoring 4.7 of a maximum of 5 points whereas the Signavio Process Editor only reaches 1.8 points.

Concluding research question 5, it has been shown that the micro-room concept is the basis for the intuitiveness of the developed modeling tool. Besides the key factors identified for the XVSM Micro-Room Modeler, a list of five more key factors has been derived from the importance ranking of items across all test subjects. These key factors should be considered when developing a modeling tool targeting end users.

Future Work

In this chapter, remaining problems are discussed and it is outlined how they could be solved in the future.

8.1 Realization with the Peer Model

After evaluating the XVSM Micro-Room Modeler in Chapter 7 it becomes apparent that the micro-room concept is very useful for creating a tool with which end users can create their own applications. We realized the modeler by using the XVSM Micro-Room Framework in our work, but since it uses similar concepts like the Peer Model (cf. Section 3.1.4), the modeler could also be based on the Peer Model directly.

To achieve this, two approaches exist: First, one could re-implement the whole modeler from scratch with the Peer Model and also build all modules with the Peer Model directly. This causes more effort but completely removes the dependency on the XVSM Micro-Room Framework. Fortunately, a lot of groundwork for this approach has already been done in [Csu14] where the author developed a Peer Model monitoring tool. Based on this work, a full featured Peer Model modeling tool might be implemented that can then be simplified in a second version to the feature set required by the micro-room concept.

In the second approach, custom modules could be created with the Peer Model directly and used within the existing XVSM Micro-Room Modeler. This is possible, since the XVSM Micro-Room Framework does not limit how micro-rooms and plugins work internally, as long as they implement the given Java interfaces (cf. Section 5.2.1). Hence, a micro-room could internally start up a whole Peer Model runtime environment without problems.

This approach could also be simplified by creating a general-purpose peer micro-room in the first place that represents a peer of the Peer Model, containing all required structures, e.g. PIC and POC. It could then read all input from an incoming micro-room action and put it into its internal PIC. The peer model that should be executed internally could be configured via room properties (e.g. a file name). After executing the internal workflow, the results from the internal POC could be forwarded to the outgoing micro-room action.

Thereby, multiple such peer micro-rooms could be connected via their micro-room actions to create the corresponding workflow. It has to be noted that this approach also has several drawbacks. First, the XVSM Micro-Room Modeler limits the Peer Model to the use of only one to one connections, i.e. strict sequential flows. Also, the UI would have to be customized via room properties such as the Peer Model to use (e.g. by specifying a HTML file), which is tedious. Finally, a general-purpose peer micro-room is not intuitive for end users and hence could only be used for testing purposes.

8.2 UI Improvements to the XVSM Micro-Room Modeler

Based on the usability tests described in Chapter 7, some usability problems of the XVSM Micro-Room Modeler became apparent that should be improved in the future. First of all, the groups and members dialog should be simplified as most users struggled to correctly assign persons to groups. Therefore, the list with all members of a group should be placed in the middle between persons (left) and groups (right) instead of beneath the groups list. Also, there will be a clear information such as “This group has currently no members.” if the selected group has no members.

Some users thought that they were already included in each group and did not create a person entry for themselves. This has been designed intentionally for the case that a person wants to model an application for others only, not including him-/herself. As this is only a corner case and end users want to create models for themselves most of the time, an entry for the modeling user him-/herself should be added automatically upon project creation.

In the usability testing, end users tried to guess which module has which functionality only from its name and icon. Surprisingly, they did succeed most of the time but if more modules are added this might not be the case anymore. Therefore, each module should include a description that will be shown as tooltip, giving clear information about the modules capabilities in one or two sentences. Technically, the description can be added to the `MicroRoom` and `Plugin` annotations as additional field (cf. Section 5.2.1).

Furthermore, the values of the action permission dropdown fields could adjust automatically to the selected members allowed to enter the room. So, instead of still showing “All persons”, a action permission dropdown field should be set to “Alice, Bob” automatically, if those two have been selected to be the only persons allowed to enter the room.

Finally, a short step-by-step guided tour should be started upon first login in the XVSM Micro-Room Modeler. It should quickly explain all elements by short descriptions and has to take less than one minute to complete. Thereby, problems with understanding plugins or how to connect micro-rooms with each other can be circumvented. It could be implemented with the help of given libraries such as `Intro.js`¹.

¹<http://introjs.com>

8.3 Port Forwarding

A problem that has already been stated in [Bin13] and still remains is that the XVSM peer instances connect directly to each other via XVSMP. Hence, if end users operate behind routers or firewalls (which they will certainly do), they need to configure a port forwarding to allow incoming traffic to their local XVSM peer instance.

This is a big drawback right now, effectively preventing the use of the XVSM Micro-Room Modeler on a larger scale. To overcome the problem, *Universal Plug and Play* (UPnP) support could be added to XVSM. Thereby, it could automatically configure port forwarding for the required XVSMP connection in the end user's router without him/her even knowing it. Fortunately, several Java libraries exist that provide UPnP support (e.g. Cling²). Hence, only little effort is required to implement this feature.

8.4 Download JRE if not Installed

Right now, if the end user does not have a JRE installed, the created application will not start up. To overcome this problem, the XVSM Micro-Room Framework JAR file that is packaged in the executable ZIP archive could be transformed, e.g. by using packr³. Packr allows to pack the JAR and a simplified JRE, containing only those classes required by the JAR, into an executable file. Another approach would be to use tools like launch4j⁴ to create an executable wrapper around the XVSM Micro-Room Framework JAR file. This wrapper searches for the specified JRE on the system and if none is found it allows to easily install it upon execution.

8.5 Publish Platform for Custom Modules

As stated in Section 7.1, a platform in the Internet should be set up where IT experts can upload module files, i.e. suites targeting specific use cases. End users could then download those module files they need and use them in the XVSM Micro-Room Modeler. To further improve usability, the publish platform could be directly integrated into the XVSM Micro-Room Modeler, such that end users could choose from all module suites directly instead of down- and uploading the files.

8.6 UI Templates

Referencing Section 4.3, support for different UI templates for each micro-room could be provided. A UI template defines how the micro-room should be rendered in the created executable application. By allowing different UIs per micro-room type, their flexibility can be increased even further, e.g. by rendering a `Conference Room` in form of a live chat with one UI template or in form of a message board with another UI template.

²<https://github.com/4thline/cling>

³<https://github.com/libgdx/packr>

⁴<http://launch4j.sourceforge.net>

This feature can be realized as stated in Section 5.4.5. Inside the micro-room’s UI folder, a special sub folder (e.g. `templates`) has to be defined. Per UI template a sub folder named according to the UI template has to be provided within this folder. Within each of these folders an `index.html` file has to be specified that implements the UI. The available sub folders in the `templates` folder (i.e. all available UI templates), can be parsed by the `ParserService` when the module is loaded and rendered by the XVSM Micro-Room Modeler in the micro-room’s property pane (e.g. in form of a dropdown element).

For example, a `Conference Room` uses the `ui/org.xvsm.microroom.modules.rooms.Conference Room/index.html` file as its default template. Additionally, the file `ui/org.xvsm.microroom.modules.rooms.Conference Room/templates/Message Board/index.html` could be provided as the “Message Board” UI template.

8.7 Further Modules

Finally, additional modules should be implemented to further increase the XVSM Micro-Room Modeler’s feature set. Based on the use cases stated by the test subjects in Section 7.3.2 and those already outlined in Section 5.3, modules can be created and uploaded to the publish platform.

Conclusion

The main goal of this thesis was to create a modeling tool with which end users can create collaborative applications by themselves. Therefore, related work has been researched for classical process modeling techniques and process modeling tools targeting end users. It has been found that currently no other tool exists for the specified requirements.

Hence, the XVSM Micro-Room Modeler has been designed and implemented, laying a special focus on providing good usability and hiding complex tasks like module support and provisioning. To achieve this, it makes use of the XVSM Micro-Room Framework and its underlying micro-room concept. Thereby, end users can create simple collaborative workflows by creating rooms that provide a predefined set of actions. Those rooms can be connected with each other, extended by plugins and configured regarding security. After creating the model, it can be transformed into a fully executable P2P application with a single click. Friends can be invited to download their customized application to collaborate according to the modeled workflow.

Evaluation has shown that the XVSM Micro-Room Framework fulfills all of the requirements for a decent modeling technique targeting end users. Additionally, it and its dynamically generated applications provide a proper level of performance and security. Another finding was that the micro-room concept indeed uses an abstraction that is more comprehensible than those of classical modeling languages. Also, it has been justified that the micro-room concept is best suited for modeling simple collaborative tasks whereas it is not suited to create complex workflows or formal software specifications. It has been found that the usability of the developed XVSM Micro-Room Framework outperforms those of competing modeling tools significantly. Additionally, the key factors for the intuitiveness of our modeling tool have been derived which basically all are achieved by using the micro-room concept. Finally, the key factors of a modeling tool for end users in general have been outlined, i.e. “Required Functionality”, “Undoability”, “System Stability”, “Easy Learnability” and “Understandability”. It has been found that the XVSM Micro-Room Modeler accomplishes all of these factors satisfyingly.

References

- [AAA⁺07] Alexandre Alves, Assaf Arkin, Sid Askary, Ben Bloch, Francisco Curbera, Yaron Goland, Neelakantan Kartha, Sterling, Dieter König, Vinkesh Mehta, Satish Thatte, Danny van der Rijn, Prasad Yendluri, and Alex Yiu. Web Services Business Process Execution Language Version 2.0, 2007.
- [ASCP13] Pedro Antunes, David Simiés, Luis Carriço, and José A. Pino. An End-user Approach to Business Process Modeling. *J. Netw. Comput. Appl.*, 36(6):1466–1479, November 2013.
- [Bar10] Martin-Stefan Barisits. Design and Implementation of the next Generation XVSM Framework – Operations, Coordination and Transactions. Master’s thesis, Vienna University of Technology, 2010.
- [BBL83] Brenda S. Baker, Sandeep N. Bhatt, and Frank Thomson Leighton. An Approximation Algorithm for Manhattan Routing. In *Proceedings of the Fifteenth Annual ACM Symposium on Theory of Computing*, STOC ’83, pages 477–486, New York, NY, USA, 1983. ACM.
- [BEMP07] Catriel Beeri, Anat Eyal, Tova Milo, and Alon Pilberg. Monitoring Business Processes with Queries. In *Proceedings of the 33rd International Conference on Very Large Data Bases*, VLDB ’07, pages 603–614. VLDB Endowment, 2007.
- [Bin13] Johann Binder. Introducing the XVSM Micro-Room Framework – Creating a Privacy Preserving Peer-to-Peer Online Social Network in a Declarative Way. Master’s thesis, Vienna University of Technology, 2013.
- [Brü13] Andreas Brückl. Relaxed non-blocking Distributed Transactions for the eXtensible Virtual Shared Memory. Master’s thesis, Vienna University of Technology, 2013.
- [BS07] Steen Brahe and Kjeld Schmidt. The Story of a Working Workflow Management System. In *Proceedings of the 2007 International ACM Conference on Supporting Group Work*, GROUP ’07, pages 249–258, New York, NY, USA, 2007. ACM.
- [CDJ⁺13] Stefan Craß, Tobias Dönz, Gerson Joskowicz, Eva Kühn, and Alexander Marek. Securing a Space-Based Service Architecture with Coordination-Driven Access Control. *Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications (JoWUA)*, 4(1):76–97, 2013.

- [CDJK12] Stefan Craß, Tobias Dönz, Gerson Joskowicz, and Eva Kühn. A Coordination-Driven Authorization Framework for Space Containers. In *Proceedings of the 2012 Seventh International Conference on Availability, Reliability and Security, ARES '12*, pages 133–142, Washington, DC, USA, 2012.
- [CJK15] Stefan Craß, Gerson Joskowicz, and Eva Kühn. A Decentralized Access Control Model for Dynamic Collaboration of Autonomous Peers. In *Security and Privacy in Communication Networks - 11th International Conference, SecureComm 2015, Dallas, TX, USA, October 26-29, 2015, Revised Selected Papers*, pages 519–537, 2015.
- [CK12] Stefan Craß and Eva Kühn. A Coordination-based Access Control Model for Space-based Computing. In *Proceedings of the 27th Annual ACM Symposium on Applied Computing, SAC '12*, pages 1560–1562, New York, NY, USA, 2012.
- [Cra10] Stefan Craß. A Formal Model of the Extensible Virtual Shared Memory (XVSM) and its Implementation in Haskell – Design and Specification. Master’s thesis, Vienna University of Technology, 2010.
- [Csu14] Maximilian Alexander Csuk. Developing an Interactive, Visual Monitoring Software for the Peer Model Approach. Master’s thesis, Vienna University of Technology, 2014.
- [DDD10] Gero Decker, Remco Dijkman, Marlon Dumas, and Luciano García-Bañuelos. The Business Process Modeling Notation. In Arthur H. M. Hofstede, Wil M. P. Aalst, Michael Adams, and Nick Russell, editors, *Modern Business Process Automation*, chapter 13, pages 347–368. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.
- [Deu95] Deutsches Institut für Normung. DIN En ISO 9241 Ergonomische Anforderungen für Bürotätigkeiten mit Bildschirmgeräten Teil 10: Grundsätze der Dialoggestaltung (ISO 9241-10), Deutsche Fassung, 1995.
- [Dön11] Tobias Dönz. Design and Implementation of the next Generation XVSM Framework – Runtime, Protocol and API. Master’s thesis, Vienna University of Technology, 2011.
- [DP06] Brian Dobing and Jeffrey Parsons. How UML is Used. *Commun. ACM*, 49(5):109–113, May 2006.
- [ESJK11] Philip Effinger, Sandra Seiz, Nicole Jogsch, and Eberhard Karls. Evaluating single features in usability tests for business process modeling tools. In *In Proceedings of the Informatics 2011 conference (Berlin, Germany)*, 2011.
- [Fig09] Kathrin Figl. Usability-Fragebögen im Vergleich. In *Tagungsband Mensch & Computer*, Berlin, September 2009. Oldenbourg.

- [FRS14] Agnès Front, Dominique Rieu, and Marco Santórum. A Participative End-User Modeling Approach for Business Process Requirements. In *BMMDS/EMMSAD*, pages 33–47, 2014.
- [Ged99] G. Gediga. The IsoMetrics usability inventory: an operationalization of ISO 9241-10 supporting summative and formative evaluation of software systems. *Behaviour and Information Technology*, 18(3):151–164, May 1999.
- [Gel85] David Gelernter. Generative Communication in Linda. *ACM Trans. Program. Lang. Syst.*, 7(1):80–112, 1985.
- [GHW98] Günther Gediga, KC Hamborg, and Heinz Willumeit. The IsoMetrics Manual (Osnabrücker Schriftenreihe Software-Ergonomie Nr. 7). *Osnabrück: Universität Osnabrück, Fachbereich Humanwissenschaften*, pages 73–84, 1998.
- [GS06] Ralf Gitzel and Michael Schwind. Experiences with Hierarchy-based Code Generation in the J2EE Context. In *Proceedings of the 4th International Symposium on Principles and Practice of Programming in Java, PPPJ '06*, pages 216–223, New York, NY, USA, 2006. ACM.
- [Ham02] Kai-Christoph Hamborg. Gestaltungsunterstützende Evaluation von Software: Zur Effektivität und Effizienz des IsoMetricsL Verfahrens. In *Mensch & Computer 2002: Vom interaktiven Werkzeug zu kooperativen Arbeits- und Lernwelten, Hamburg, Germany, September 2-5, 2002*, pages 303–312, 2002.
- [HBR00] William Harrison, Charles Barton, and Mukund Raghavachari. Mapping UML Designs to Java. In *Proceedings of the 15th ACM SIGPLAN Conference on Object-oriented Programming, Systems, Languages, and Applications, OOPSLA '00*, pages 178–187, New York, NY, USA, 2000. ACM.
- [ICK⁺10] Dave Ings, Luc Clement, Dieter König, Vinkesh Mehta, Ralf Mueller, Ravi Rangaswamy, Michael Rowley, and Ivana Trickovic. WS-BPEL Extension for People (BPEL4People) Specification Version 1.1. OASIS Committee Specification, August 2010.
- [Iso01] Sadahiro Isoda. Object-oriented real-world modeling revisited. *Journal of Systems and Software*, 59(2):153–162, 2001.
- [Jor08] Philipp Jordan. Selection of an appropriate Usability Evaluation Method. Project. Department of Design Engineering, University of Stuttgart, Stuttgart, Germany, 2008.
- [KCJ⁺13] Eva Kühn, Stefan Craß, Gerson Joskowicz, Alexander Marek, and Thomas Scheller. Peer-Based Programming Model for Coordination Patterns. In Rocco De Nicola and Christine Julien, editors, *15th International Conference on Coordination Models and Languages (COORDINATION), held as part of the 8th International Federated Conference on Distributed Computing Techniques (DisCoTec)*,

volume 7890 of *Lecture Notes in Computer Science*, pages 121–135, Florence, Italy, June 3-5 2013. Springer.

- [KCS15] Eva Kühn, Stefan Craß, and Gerald Schermann. Extending a Peer-Based Coordination Model with Composable Design Patterns. *2015 23rd Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP)*, 00, 2015.
- [Küh16] Eva Kühn. Reusable Coordination Components: Reliable Development of Cooperative Information Systems. *International Journal of Cooperative Information Systems (Elsevier)*, 25(4), 2016.
- [Lew06] James R. Lewis. Sample Sizes for Usability Tests: Mostly Math, Not Magic. *interactions*, 13(6):29–33, November 2006.
- [LV10] Jean-Philippe Lombardi and Jürgen Vogel. Wizard-based Process Modeling for Business Users. In *Proceedings of the International Conference on Advanced Visual Interfaces, AVI '10*, pages 406–406, New York, NY, USA, 2010. ACM.
- [MAKS12] Jürgen Münch, Ove Armbrust, Martin Kowalczyk, and Martin Soto. *Software Process Definition and Management*. Springer Publishing Company, Incorporated, 2012.
- [MGWD09] Milan Milanovic, Dragan Gasevic, Gerd Wagner, and Vladan Devedzic. Modeling service orchestrations with a rule-enhanced business process language. In Patrick Martin, Anatol W. Kark, and Darlene A. Stewart, editors, *CASCON*, pages 70–85. ACM, 2009.
- [MKS10] Richard Mordinyi, Eva Kühn, and Alexander Schatten. Space-Based Architectures As Abstraction Layer for Distributed Business Applications. In *Proceedings of the 2010 International Conference on Complex, Intelligent and Software Intensive Systems, CISIS '10*, pages 47–53, Washington, DC, USA, 2010.
- [MSMP11] Sonja Meyer, Klaus Sperner, Carsten Magerkurth, and Jacques Pasquier. Towards Modeling Real-world Aware Business Processes. In *Proceedings of the Second International Workshop on Web of Things, WoT '11*, pages 8:1–8:6, New York, NY, USA, 2011. ACM.
- [MTJ⁺10] Hafedh Mili, Guy Tremblay, Guitta Bou Jaoude, Éric Lefebvre, Lamia Elabed, and Ghizlane El Boussaidi. Business Process Modeling Languages: Sorting Through the Alphabet Soup. *ACM Comput. Surv.*, 43(1):4:1–4:56, 2010.
- [NL93] Jakob Nielsen and Thomas K. Landauer. A Mathematical Model of the Finding of Usability Problems. In *Proceedings of the INTERACT '93 and CHI '93 Conference on Human Factors in Computing Systems, CHI '93*, pages 206–213, New York, NY, USA, 1993. ACM.

- [ODA⁺09] Chun Ouyang, Marlon Dumas, Wil M. P. Van Der Aalst, Arthur H. M. Ter Hofstede, and Jan Mendling. From Business Process Models to Process-oriented Software Systems. *ACM Trans. Softw. Eng. Methodol.*, 19(1):2:1–2:37, 2009.
- [OMG13] OMG. Business Process Model and Notation (BPMN) Version 2.0.2, December 2013.
- [OMG15] OMG. Unified Modeling Language (UML), Version 2.5.0, March 2015.
- [Pro11] Christian Proinger. Design and Implementation of a REST-Interface for Mozartspaces. Bachelor Thesis, Vienna University of Technology, 2011.
- [RLRA12] Gianna Reggio, Maurizio Leotta, Filippo Ricca, and Egidio Astesiano. Business Process Modelling: Five Styles and a Method to Choose the Most Suitable One. In *Proceedings of the Second Edition of the International Workshop on Experiences and Empirical Studies in Software Modelling*, EESSMod '12, pages 8:1–8:6, New York, NY, USA, 2012. ACM.
- [SC14] Maria Shitkova and Leonardo Campus. On the usability of business process modelling tools—a review and future research directions. In *EMISA*, pages 117–130, 2014.
- [SK15] Thomas Scheller and Eva Kühn. Automated Measurement of API Usability. *Inf. Softw. Technol.*, 61(C):145–162, May 2015.
- [SRDS00] Alistair G. Sutcliffe, Michele Ryan, Ann Doubleday, and Mark V. Springett. Model mismatch analysis: Towards a deeper explanation of users' usability problems. *Behaviour & IT*, 19(1):43–55, 2000.
- [SS07] Martin Soukup and Jiri Soukup. The Popularity Cycle of Graphical Tools, UML, and Libraries of Associations. In *Companion to the 22Nd ACM SIGPLAN Conference on Object-oriented Programming Systems and Applications Companion*, OOPSLA '07, pages 753–756, New York, NY, USA, 2007. ACM.
- [SSF08] Todor Stoitsev, Stefan Scheidl, Felix Flentge, and Max Mühlhäuser. From Personal Task Management to End-User Driven Business Process Modeling. In Marlon Dumas, Manfred Reichert, and Ming-Chien Shan, editors, *BPM*, volume 5240 of *Lecture Notes in Computer Science*, pages 84–99. Springer, 2008.
- [TYI05] Muhammad Adeel Talib, Zongkai Yang, and Qazi Mudassir Ilyas. A Framework Towards Web Services Composition Modeling and Execution. In *Proceedings of the IEEE IEEE05 International Workshop on Business Services Networks*, BSN '05, pages 4–4, Piscataway, NJ, USA, 2005. IEEE Press.
- [Ver04] Laury Verner. BPM: The Promise and the Challenge. *Queue*, 2(1):82–91, 2004.

- [WAD⁺06] Petia Wohed, Wil M. P. van der Aalst, Marlon Dumas, Arthur H. M. Hofstede, and Nick Russell. On the Suitability of BPMN for Business Process Modelling. In Schahram Dustdar, Jose Luiz Fiadeiro, and Amit P. Sheth, editors, *Business Process Management*, volume 4102 of *Lecture Notes in Computer Science*, pages 161–176. Springer, 2006.
- [WPB13] Ingo Weber, Hye-Young Paik, and Boualem Benatallah. Form-Based Web Service Composition for Domain Experts. *ACM Trans. Web*, 8(1):2:1–2:40, December 2013.
- [Zar12] Jan Zarnikov. Energy-efficient Persistence for Extensible Virtual Shared Memory on the Android Operating System. Master’s thesis, Vienna University of Technology, 2012.
- [ZLC⁺12] Liping Zhao, Keletso Letsholo, Erol-Valeriu Chioasca, Sandra Sampaio, and Pedro Sampaio. Can Business Process Modeling Bridge the Gap Between Business and Information Systems? In *Proceedings of the 27th Annual ACM Symposium on Applied Computing*, SAC ’12, pages 1723–1724, New York, NY, USA, 2012. ACM.

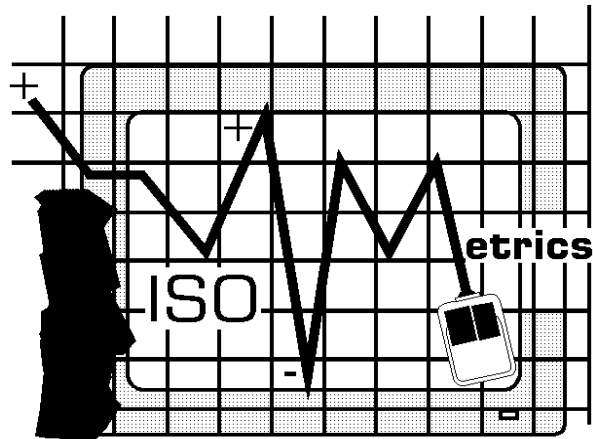
Web References

- [1] Marco Di Natale. Code generation using Acceleo. <https://www.youtube.com/watch?v=42jkrOWA9RE>. Accessed: 2017-03-20.
- [2] Sparx Systems Europe. What is the pricing of my Enterprise Architect Edition? <https://www.sparxsystems.eu/enterprise-architect/ea-pricing-purchasing/>. Accessed: 2017-03-20.
- [3] Sparx Systems Europe. Sparx Systems Enterprise Architect. <http://idownload.ws/software/large/enterprise-architect-for-uml-2-1-9527.jpg>. Accessed: 2017-03-20.
- [4] Signavio. Signavio Process Editor. <http://donar.messe.de/exhibitor/cebit/2017/J358799/gallery-868x0-342915.jpg>. Accessed: 2017-03-20.
- [5] Activiti. Activiti User Guide. <http://activiti.org/userguide/index.html>. Accessed: 2017-03-20.
- [6] Activiti. Activiti Designer. <https://3.bp.blogspot.com/-1fit03hjZcY/Tf-vpniW7yI/AAAAAAAAAPE/MAfNAtpONaQ/s1600/Screenshot-Activiti+-MyProcess.activiti+%2528BPMNdiagram%2529+-+Eclipse+SDK+.png>. Accessed: 2017-03-20.
- [7] NetBeans. Enterprise Service Oriented Architecture Project Home. <https://soa.netbeans.org/soa/>. Accessed: 2017-03-20.
- [8] SOA4AllProject. BPM using SOA4All. <https://www.youtube.com/watch?v=ZpBPqDv5rss>. Accessed: 2017-03-20.
- [9] Eclipse. Overview of GMF. <http://wiki.eclipse.org/images/5/59/Overview.png>. Accessed: 2017-03-20.
- [10] Eclipse. GMF modeler example. http://wiki.eclipse.org/images/c/c0/Basic_mindmap.png. Accessed: 2017-03-20.
- [11] Eclipse. GMF Lite Runtime. http://wiki.eclipse.org/Graphical_Modeling_Framework/Tutorial/Part_4#Exploring_the_Lite_Runtime. Accessed: 2017-03-20.

- [12] Microsoft. Welcome to the Visio SDK. <http://msdn.microsoft.com/en-us/library/office/ff758690.aspx>. Accessed: 2017-03-20.
- [13] Sawyoo. Photos of Visio Flowchart Examples. http://www.sawyoo.com/postpic/2015/03/visio-process-flowchart-examples_206703.png. Accessed: 2017-03-20.
- [14] JointJS. Working with Ports. <http://resources.jointjs.com/tutorials/joint/tutorials/ports.html>. Accessed: 2017-03-20.
- [15] JointJS. API Documentation. <http://resources.jointjs.com/docs/jointjs/v1.0/joint.html#dia.Link.prototype.presentation>. Accessed: 2017-03-20.
- [16] Cdelmas. Performance of Microservice Frameworks. <https://cdelmas.github.io/2016/06/20/Performance-of-Microservices-frameworks.html>. Accessed: 2017-03-20.
- [17] Nielsen Norman Group. Why You Only Need to Test with 5 Users. <https://www.nngroup.com/articles/why-you-only-need-to-test-with-5-users/>. Accessed: 2017-03-20.
- [18] Prof. Dr. Jochen Prümper. Fragebogen ISONORM. http://www.ergo-online.de/site.aspx?url=html/software/verfahren_zur_beurteilung_der/fragebogen_isonorm_online.htm. Accessed: 2017-03-20.

APPENDIX **A**

IsoMetrics^S



IsoMetrics^S

**Questionnaire
for the evaluation of graphical user interfaces
based on ISO 9241/10**

(short version)

Copyright © by Heinz Willumeit (1993) / K.C. Hamborg, G. Gediga (1995 .. 97).

All rights reserved including the right of reproduction in whole or in part in any form.

Version: 2.01e October 1997

Contact:
Universität Osnabrück
Fachbereich Psychologie
Seminarstr. 20
D - 49069 Osnabrück
Germany

email: isometric@luce.psych.uni-osnabrueck.de

About the questionnaire 'IsoMetrics'

Dear study participant:

The purpose of this questionnaire is to assess the usability of software products. By completing it, you are enabling us to identify and remedy any shortcomings, with the aim of enhancing its user-friendliness.

The questionnaire contains statements about the user-friendliness of software products. Please indicate the extent to which you agree or disagree with each of these statements, making use of the scale provided in each case. Here is an example:

		Pre-dominantly disagree		So - so		Pre-dominantly agree	
Index	dialogue principle	1	2	3	4	5	No opinion
0	Computers are useful work aids.					X	

If you think the statement "Computers are useful work aids" is true, then mark column "5" for "**Predominantly agree**" (as shown) with an X. If you find you cannot agree with this statement, then mark column "1" for "**Predominantly disagree**". You can also indicate various degrees of agreement between these two poles by marking the corresponding numbers (column 2 or 4). If for some reason you cannot or do not wish to reply, you should mark the last column "**no opinion**" with an X.

Thank you very much for your cooperation.

IsoMetrics^S

		Pre- dominantly disagree		So - so		Pre- dominantly agree	
Index	suitability for the task	1	2	3	4	5	No opinion
A.1	The software forces me to perform tasks that are not related to my actual work.						
A.3	The software lets me completely perform entire work routines.						
A.4	The functions implemented in the software support me in performing my work.						
A.6	The way in which data is entered is suited to the tasks I want to perform with the software.						
A.7	I perceive the arrangement of the fields on-screen as sensible for the work I do with the software.						
A.8	Too many different steps need to be performed to deal with a given task.						
A.9	The way in which data is output is suited to the tasks I want to perform with the software.						
A.10	The software is well suited to the requirements of my work.						
A.11	In a given screen, I find all of the information I need in that situation.						
A.12	The terminology used in the software reflects that of my work environment.						
A.14	The software provides me with a repeat function for work steps that must be performed several times in succession.						
A.15	I can easily adapt the software for performing new tasks.						
A.16	The important commands required to perform my work are easy to find						
A.17	I am able to adjust the presentation of results (on the screen, to printer, plotter etc.) to my various work requirements.						
A.18	The presentation of the information on the screen supports me in performing my work.						

IsoMetrics^S

		Pre- dominantly disagree		So - so		Pre- dominantly agree	
Index	self-descriptiveness	1	2	3	4	5	No opinion
S.2	I can call up specific explanations for the use of the system, if necessary.						
S.3	I understand immediately what is meant by the messages displayed by the software						
S.5	It is easy to retrieve information about a certain entry field.						
S.6	When menu items are not available in certain situations, this fact is visually communicated to me.						
S.7	If I want, the software will display not only general explanations but also concrete examples to illustrate points.						
S.8	The explanations the software gives me clearly refer to the specific situations in which they are output.						
S.9	If I want, the software displays basic information about conceptual aspects of the program.						
S.10	The software provides me with enough information about which entries are permitted in a particular situation.						
S.11	I can tell straight away which functions are invoked by the various menu items.						
S.12	The terms and concepts used in the software are clear and unambiguous.						
S.13	The software always visually marks the current entry location (e.g. by a highlight, a contrasting color, a blinking cursor, etc.).						
S.14	I can easily tell the difference among feedback messages, requests to confirm inputs or commands, warnings, and error messages.						

IsoMetrics^S

		Pre- dominantly disagree		So - so		Pre- dominantly agree	
Index	controllability	1	2	3	4	5	No opinion
T.2	The possibilities for navigating within the software are adequate.						
T.3	The software makes it easy for me to switch between different menu levels.						
T.4	The software lets me return directly to the main menu from any screen.						
T.5	I can interrupt any dialog at any time.						
T.6	It is always easy for me to evoke those system procedures that are necessary for my actual work.						
T.7	It's easy for me to move back and forth between different screens.						
T.8	The software allows me to interrupt functions at any point, even if it is waiting for me to make an entry.						
T.10	The navigation facilities of the software support optimal usage of the system functionality.						
T.12	In order to perform my tasks, the software requires me to perform a fixed sequence of steps.						
T.13	When selecting menu items, I can speed things up by directly entering a letter or a command code.						
T.15	It is always possible to abort a running procedure manually.						

IsoMetrics^S

		Pre- dominantly disagree		So - so		Pre- dominantly agree	
Index	conformity with user expectations	1	2	3	4	5	No opinion
E.8	The software is inconsistently designed, thus making it more difficult for me to do my work.						
E.1	I can anticipate which screen will appear next in a processing sequence.						
E.2	I have no difficulty in predicting how long the software will need to perform a given task.						
E.3	The designations are used consistently in all parts of the software I am familiar with.						
E.4	I find that the same function keys are used throughout the program for the same functions.						
E.5	When executing functions, I have the feeling that the results are predictable.						
E.6	My impression is that the same possibilities are consistently available for moving within and between different parts of the software.						
E.7	The messages output by the software always appear in the same screen location.						

IsoMetrics^S

		Pre- dominantly disagree		So - so		Pre- dominantly agree	
Index	error tolerance	1	2	3	4	5	No opinion
F.1	When working with the software, even small mistakes have sometimes had serious consequences.						
F.2	Even if I make a mistake, the information (e.g. data, text, and graphics) which I have just entered is not lost.						
F.3	If I make a mistake while completing a form, I can easily restore everything to its previous state.						
F.4	When I attempt to perform a destructive operation (e.g. deletion of data etc.), I am always first prompted to confirm the action.						
F.5	My impression is that very little effort is involved in correcting mistakes.						
F.6	When I make entries, they are first checked for correctness before further processing is initiated.						
F.7	No system errors (e.g. crashes) occur when I work with the software.						
F.8	If I make a mistake while performing a task, I can easily undo the last operation.						
F.9	I have never made an entry that caused a software error (e.g. a system/program crash or an undefined dialog state).						
F.10	The software includes safety features to help prevent unintended actions (e.g. critical keys spaced well apart, highlights, designations that are not easily confused).						
F.12	The software provides me with useful information on how to recover from error situations.						
F.13	I perceive the error messages as helpful.						
F.14	In some situations the software waits too long before calling attention to wrong entries.						
F.15	The software warns me about potential problem situations.						
F.16	The software lets me keep the original data even after it has been changed.						

IsoMetrics^S

		Pre- dominantly disagree		So - so		Pre- dominantly agree	
Index	suitability for individualization	1	2	3	4	5	No opinion
I.1	The software lets me adapt forms, screens and menus to suit my individual preferences.						
I.4	The software can be easily adapted to suit my own level of knowledge and skill.						
I.6	I am able to adjust the amount of information (data, text, graphics, etc.) displayed on-screen to my needs.						
I.7	The software lets me change the names of commands, objects and actions to suit my personal vocabulary.						
I.8	I can adjust the attributes (e.g. speed) of the input devices (e.g. mouse, keyboard) to suit my individual needs.						
I.11	I can adjust the software's response times to my own personal working speed.						

		Pre- dominantly disagree		So - so		Pre- dominantly agree	
Index	suitability for learning	1	2	3	4	5	No opinion
L.1	I needed a long time to learn how to use the software.						
L.2	It is easy for me to relearn how to use the software after a lengthy interruption.						
L.3	The explanations provided help me understand the software so that I become more and more skilled at using it.						
L.4	So far I have not had any problems in learning the rules for communicating with the software, i.e. data entry.						
L.5	I was able to use the software right from the beginning by myself, without having to ask coworkers for help.						
L.6	I feel encouraged by the software to try out new system functions by trial and error.						
L.7	In order to use the software properly, I must remember a great many details.						
L.8	I find it easy to use the commands.						