Model-based
(Remote) Usability Evaluation

Gregor Buchholz

Putbus, 27.06.2008

Remote Usability Testing

- What is Remote Usability Testing?
  - situations where the user and the usability evaluator are not located in the same place
- Two groups of methods
  - „same-time but different-place“ (contact by phone / VOIP possible)
  - „different-time and different-place“
  - common methods:
    - Real-time design walk-throughs *(same-time, different-place)*
    - Surveys *(Web based / paper)*
    - Automated usage tracking *(different-time, different-place)*

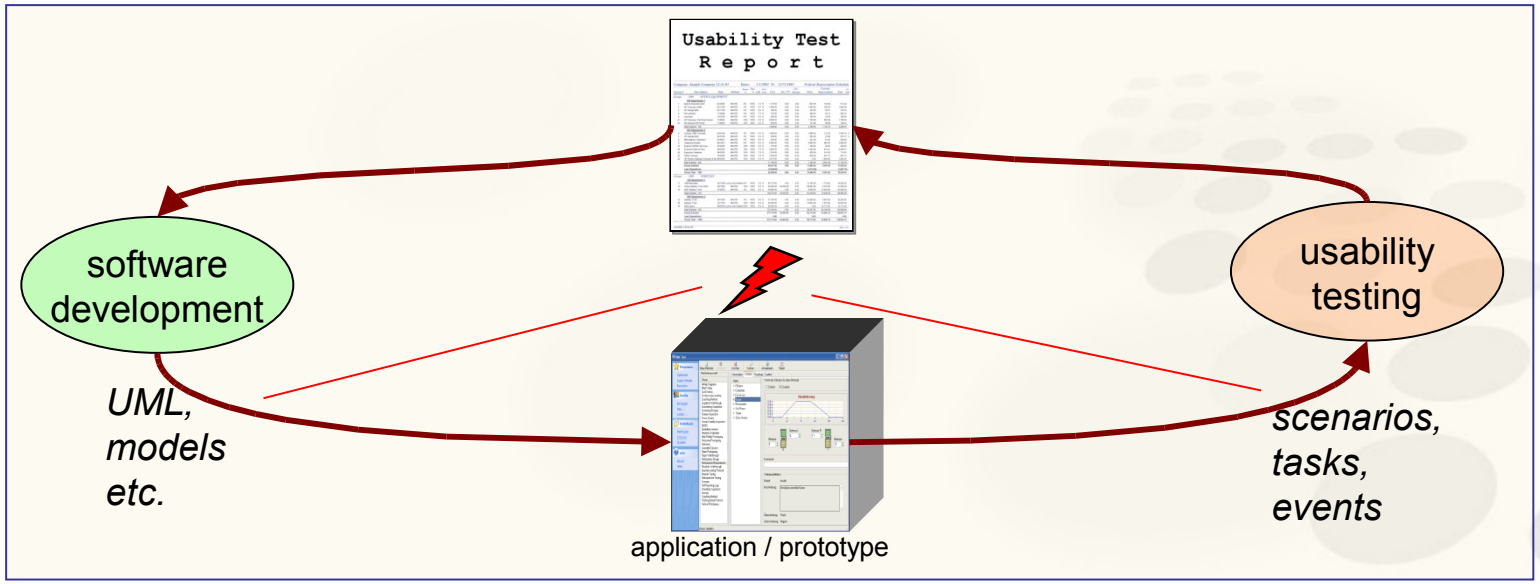
Remote Usability Testing

- Advantages
  - users are in their own work environment
  - → will behave more normally
  - environment's technical aspects are like during actual usage
  - savings of cost and time

- Disadvantages
  - much bandwidth needed when conducted in real-time
  - configuration of the user‘s firewall
  - users are on their own
  - → no assistance while preparing the test (software installation etc.)

Structural Disadvantages of common remote techniques

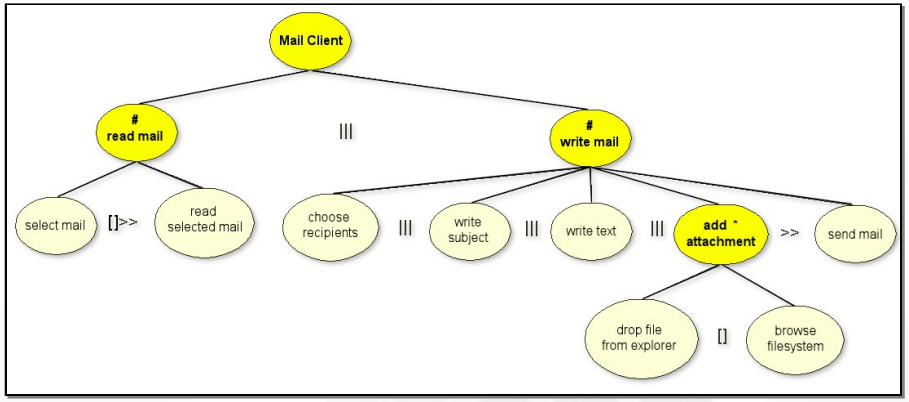
- Restricted to loggable interface events
- Laborious subsequent mapping of logged data to structural data
- Unbridged gap between developers‘ and usability experts' methods:



- → **use concepts of model-based software development for (remote) usability tests**

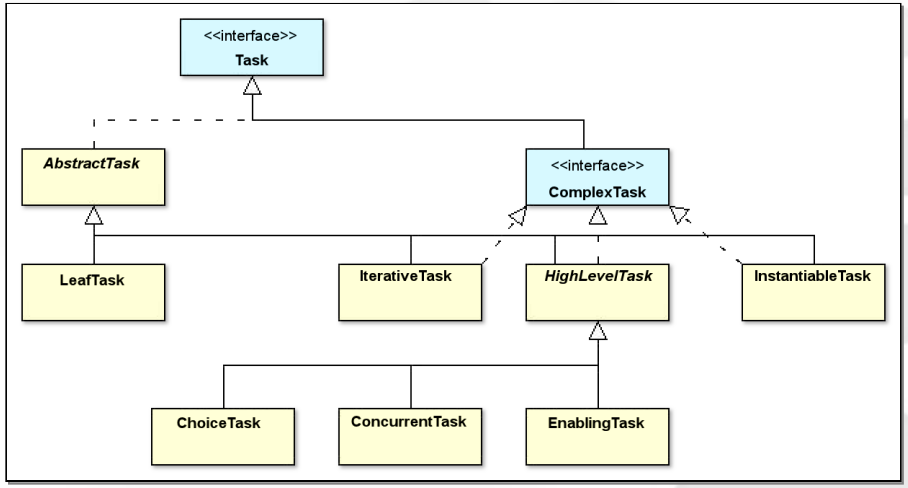
Background

- Task model
  + User model
  + Object model
  + Device model



Task model example: Mail Client Application

- Framework for the rendition of temporal and structural relations between tasks

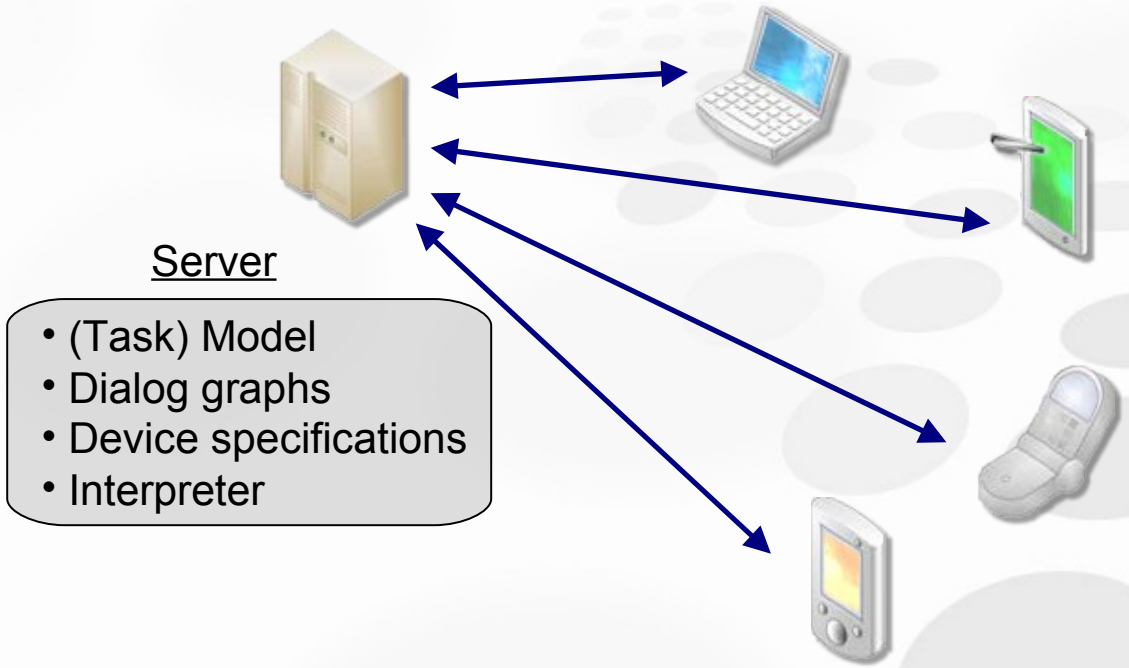


Java framework m6c.taskmodel.runtime

Architecture

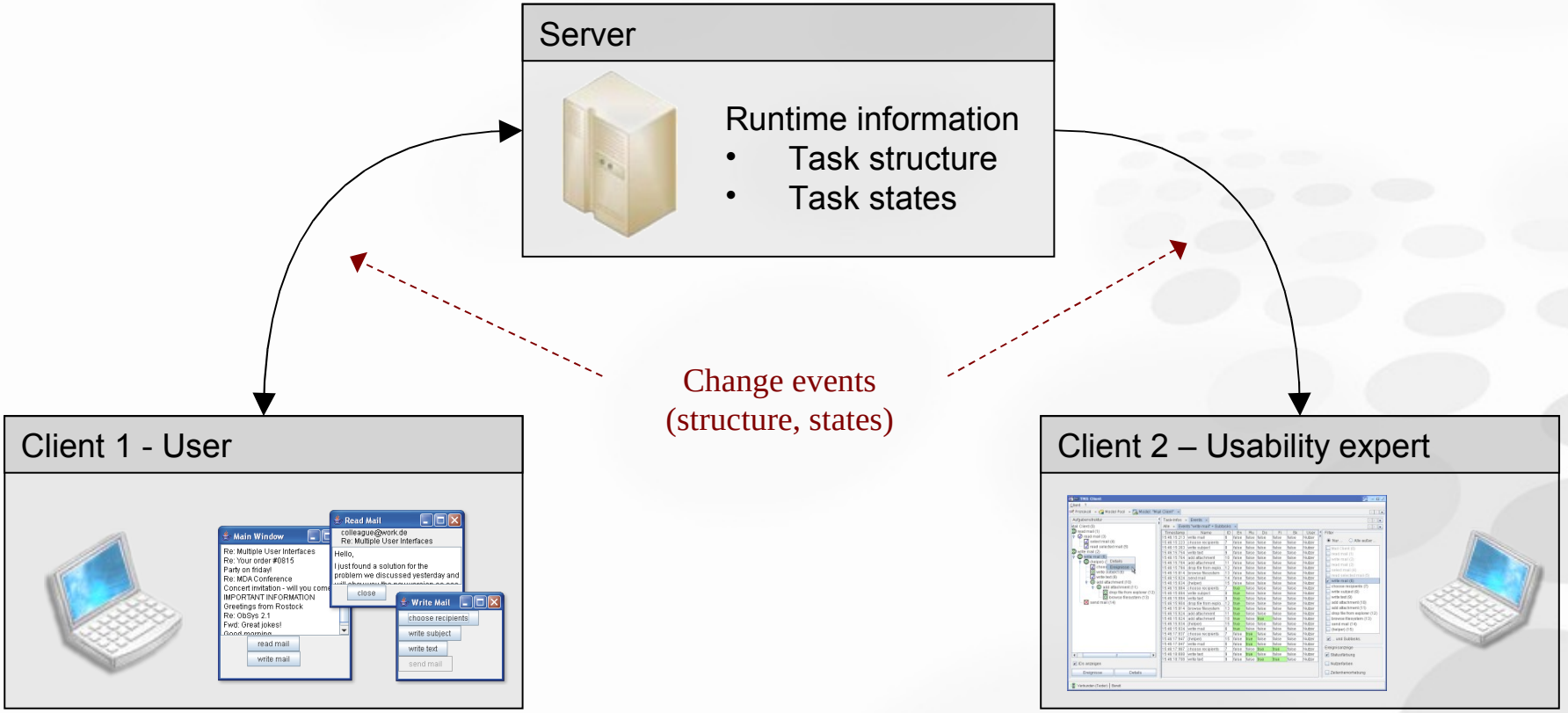
- Goals:
  - optimal support for mobile interactive applications
  - support for different devices (with different UI capabilities)

- Generation based on
  - tasks
  - objects
  - roles
  - devices
  - context of use



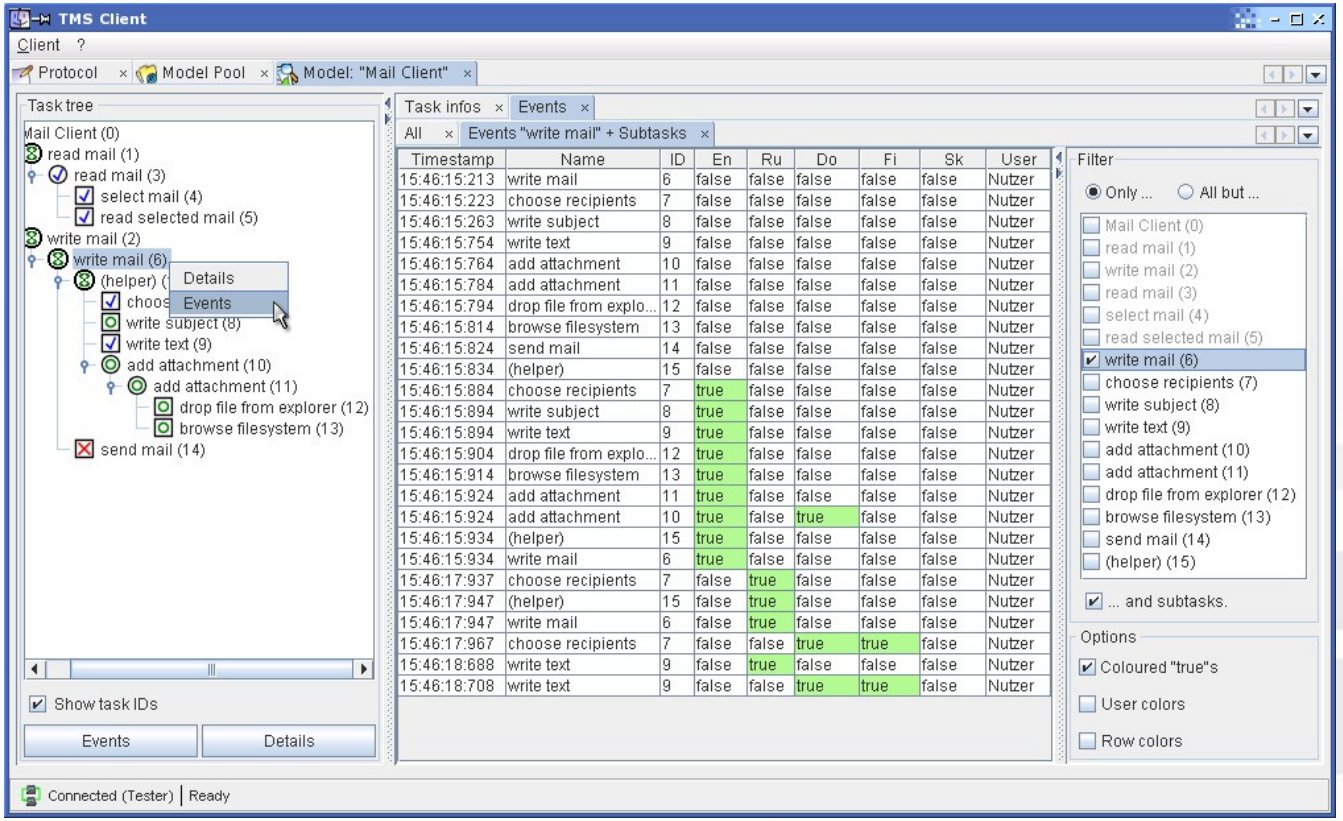
Schematic Setup

- Use runtime information about model instances to observe and record the user‘s interaction with the system



ReModeEl

- Dynamic task tree with state visualization
  - ○ Inner task
  - □ Leaf task
  - ○ Waiting
  - ⧖ Running
  - ✓ Done
  - ✕ Inactive
- Log view (table) of state change events
  - Filter options
  - View options



Usability expert can observe the „execution“ of a task model instance

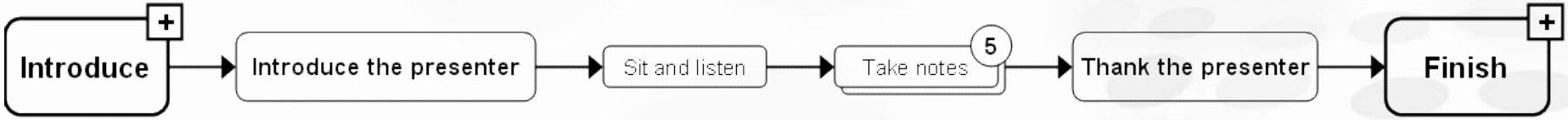
ReModEl – current state

- Tool support for software development by interactively transforming models is available
- Server-client architecture for the execution of task-model based software systems
  - Generated applications can be executed and produced events are sent to the server
- Observer client to watch the execution of a model based application was implemented
  - Shows a dynamic tree view of instantiated tasks
  - Lists all state and structure events
  - Provides basic communication
  - Several visualizations

Enhanced task trace view

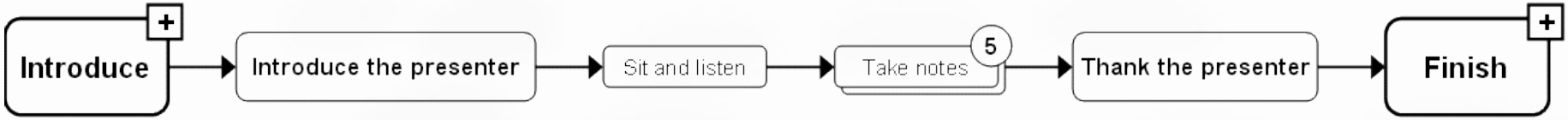
Current work: In-time notification of unexpected user behavior

Enhanced task trace view
(presented to observers for usability evaluation purposes)



- Sequence of executed tasks from left to right
- Higher level in task tree → bigger rectangles
- Repeated tasks: Stacked, with execution counter
- Aggregated by semantic lense: marked with „+“

Current issue: Expected Behaviour



ExpectedTaskTraceRecord

- TaskID (according to task model)
- (optional: runtime ID, for suspend-resume-sequences)
- timestamp
  - # relative to static or dynamic ID, absolute time from session start
  - # tolerance, absolute or relative
- preconditions
  - # task state *(for resuming e.g.)*
  - # 0{object, property}N
- (instances, refers to static ID)
- duration *(overall or per instance)*
- postconditions
  - # task state
  - # 0{object, property}N

Ongoing work

- Evaluation of Tool Support
- Open toolset for systems not developed using the task model framework
- Track e.g. web site usage
- Protocols → Models ?

Protocols

- Level ob abstraction of user interaction [Hilbert et al., 1997]

- Physical Events (key pressing fingers, mouse moving hands)
- Input Device Events (hardware generated events, interrupts)
- UI Events (input focus changes, keypresses)
- Abstract Interaction Level (value input)
- Domain/Task-Related (e.g. input of address data)
- Goal/Problem-Related (place an order)

Event categories

- *synchronous events* (High Frequency Band)
  - logger-based capturing (synchron)
    - requires system‘s source code or a sufficient degree of self-reporting (web server [sometimes: UI, DB])
- *asynchronous events* (Low Frequency Band)
  - manual tracking (asynchron)
    (files, documents, mails, other [collaborative] artefacts)
- often: low frequency band events = composition of high frequency band events
  - “moving”: eye movements, gestures, comments
  - “meeting event”: sum of movements
  - „input“: sequence of selecting, keypressing, …

From Protocols to Models

- inferences between frequency band boundaries:
  - log low-frequency-band events only
    → loss of information: components (sub events) of low frequency band events hard to find
  - log high-frequency-band events only
    → insufficient for detection of low frequency band events

  - Issue: How much additional knowledge (intelligence) and effort is needed to construct a reasonable (task) structure of low frequency band events out of a sequence of comparatively easy trackable high frequency band events?

From Protocols to Models

- synchronization
- transformation
- analysis
  - statistical analysis
  - sequence detection
  - sequence comparison
  - sequence categorizing
- visualization
- integration

Synchronization

- cross-linking to other sources
  - e.g. from different frequency bands
  - Simple but effective and - necessary.

- examples: ObSys/Yacob
  - capturing and analysis of different event types

transformations

- selection, filtering
  - *noise* removal, only keep *interesting* events
  - hide sub sequences
  - positive vs negative filtering
- Abstracting, aggregating
  - construct higher level structures
- Recoding
  - create sequences of higher abstraction

analysis 1/2

- statisical analysis
  - Feature use count
  - Error frequencies
  - Use of help system
- sequence detection
  - find concretely or abstractly defined *target* sequences in *source* sequences, that may indicate usability issues
  - (automatically) detect sequences, that differ from expection
  - Sequence detection as first step towards abstraction
  - **Fisher‘s cycles**, **multiple repeating pattern analysis**, **TAG**
- sequence comparison
- sequence categorizing

Fisher‘s Cycles

- *Source* sequence: ABACDACDBADBCACCCD
- Start action: A
- End action: D
- output:
  - AB**ACD**ACDBADBCACCCD
  - ABACD**ACD**BADBCACCCD
  - ABACDACDB**AD**BCACCCD
  - ABACDACDBADBC**ACCCD**
- Parallel activities?

	Cycle	Count
1	ACD	2
2	AD	1
3	ACCCD	1

Lag sequential analysis (LSA)

- *Source* sequence `ABACDACDBADBCACCCD`
- Key action: `A`
- Target action: `D`
- Lag(s): -4 to +4
- Output:

Lag	-4	-3	-2	-1	1	2	3	4
occurrences	0	1	1	1	1	2	0	1

Maximal Repeating Pattern (MRP)

- Assumption: Repeated occurrence of same sequence ...
  - ... indicates strong relationship
    → candidates for abstraction/aggregation
    or
  - ... Indicates possible usability issues
- *Source* sequence: `ABACDACDBADBCACCCD`
- Output:

	Pattern	Count
1	ACD	2
2	AC	3
3	CD	4
4	BA	2
5	DB	2

Related
(Exploratory Sequential Data Analysis, ESDA)

- „Expectation Agents“ EA and
  „Expectation-Driven Event Monitoring“ EDEM [Hilbert, Redmiles]
  - detect composites in sequences
  - Rules between different levels (frequency bands)
  - Real time: report incidence of sequences and non-incidence of expected sequence
- EBBA [Bates]
  - Determine if the system behaves as specified in terms of higher level events (e.g. by pattern matching)
- disadvantages
  - extensive output
  - sometimes hard to interprete
  - sometimes not leading to the goal

Task Action Grammar [Payne, Green]

- Parameterized description of user activities
  - Start symbol (task)
  - Terminals (list of actions)
  - Non terminals
  - Substitution rules (T-rules decompose tasks; other rules → actions)

Example: Word processor: Move cursor right/left by one word/character

action:	Move cursor
dimensions:	direction, extend
values:	left, right; word, character

T rule:

```
Task(direction, extend) -> symbol(direction) + character(extend)
```

Weitere Regeln:

```
Symbol(direction = right)  -> „CTRL“
Symbol(direction = left) -> „ALT“
Character(extend = character)  -> „c“
Character(extend = word)     -> „w“

Task(direction = right, extend = word) -> CRTL – w
```

Task Action Grammar

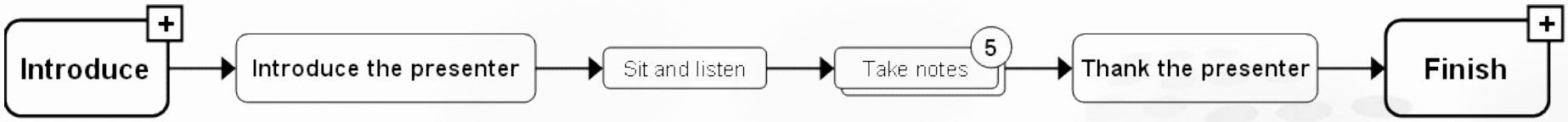
- When having only slightly differences or (~) semantical equal *source*-sequences
  - Interpretation of differing lexical elements as different values of same dimension
- Example:
  - `KEY_PRESSED Alt-F` A
  - `MENUITEM_SELECTED „Print“` B
  - `RADIOBUTTON_SELECTED „All pages“` C
  - `RADIOBUTTON_SELECTED „Current Page“` D
  - `BUTTON_PRESSED „OK“` E
  - `KEY_PRESSED Alt-P` F
  - sequences: `ABDE, FCE, ABCE`
    - `T1(id = „Current Page“, par=„1“) -> ABDE`
    - `T1(id = „All Pages“, par=„1“) -> ABCE`
    - `T2(id = „All Pages“, par=„2“) -> FCE`
    - `T(id, par=„1“) = T1(id)`
    - `T(id, par=„2“) = T2(id)`

Analysis 2/2

- Statistical analysis
- Sequence detecion
- Sequence comparison
  - Determine degree of equalness (unit?)
  - Differences between abstract *target* sequences and *source*-sequences
  - Differences between concrete *target* sequences (by expert user) and *source*-sequences
  - Visualization of best fitting between *target* and *source*
  - → *After model construction*
- Sequence categorization
  - Construct abstract models from *source* sequences to combine them to describe „interesting“ sequence properties
  - Construct probability matrix for transitions based on categories
  - Construct grammar models or state machines to describe the sytactical structure of the *source* sequence [Olsen et al.]
    similar to Task action Grammar, but more knowledge neccessary

Visualization

- Show results of transformations an analysis (interactive per definition interaktiv)



Integration

- Desired state (combine existing tools)
- Cross-linking

Summary

- ReModEl
  - Client server system for early and late usability tests
  - Next steps:
    - Expected behaviour
    - Visualizations
    - Projects
- Allow not model-based developed systems to benefit from model-based evaluations
  - Model construction based on usual UI event protocols

Thank you very much for your attention.

Model-based
(Remote) Usability Evaluation