Exploring the Design Space for Dynamic Interfaces

Dynamic interfaces in an Automated Software Generation context

A thesis submitted in partial fulfilment for the degree of
Master of Science of Information Sciences
At the Faculty of Science
Institute for Computing and Information Sciences

August 2015

Author:	Joeri Arendsen
	s3033066
Supervisor Radboud:	Marko van Eekelen
Related company:	CGI
Supervisor CGI:	Edwin Hendriks

Radboud University

CGI

Abstract

This master thesis researched the field of dynamic interfaces and tries to determine designs that are compatible with an existing automated software generation system (SPADE) that generates working code based on business specifications. The research explores opportunities that address the issue at a practical level, but also the overall problem in this automated software development process. The proposed solutions vary on three levels. Firstly, based on theory, several guidelines are constructed for a designer to keep in mind when working towards interfaces with dynamic capabilities. Secondly, several designs are created and explored to identify a solution that best fits the current automated software development process. Furthermore, some of these designs also function as guidelines themselves, on a more practical level, involving usability. Thirdly, a detailed description of the desired functionality for CGI is provided in the form of use cases describing how the interface aids their end-users in editing or adding GUI-components to serve the end-users' needs while working with CGI's automated generated software.

Acknowledgements

While creating this section, I gradually begin to realize the work is almost done. The last year has been a challenge. The number of question marks to be solved increased at every turn. Several people have supported me during this long process.

I would like to thank my supervisor **Marko van Eekelen** for helping me to overcome challenges I encountered while writing this thesis and providing feedback and guidance. Several conversations solved seemingly complex problems in a matter of minutes. Next, I would like to thank **Edwin Hendriks** from CGI, his energy and perspective was very inspirational. The brainstorm sessions and discussions were the best of the project.

Lastly, I would like to thank the following people who have supported this thesis or me:

Marcus Klimstra – CGI developer
For his insights.

Nikos Makris – Fellow student on the project
For the Thesis Team sessions.

Ina de Vries – Student advisor Radboud University
For helping me to find my way to CGI.

Vera Kamphuis – Student advisor Radboud University
For new perspectives.

Nadine Arendsen – My sister
For the support.

•••

Table of Contents

List of figures

Extended Table of Contents

General Introduction

Two related theses are the product of a joint venture between CGI[1] and two master students at the Radboud University. This thesis is one of them. The research questions were devised together with Edwin Hendriks working at CGI in software development using their SPADE generator. The other thesis is realized by my fellow student Nikos Makris[2] with whom I have worked together on this project. The areas researched in our theses overlap at times, however, as each thesis has his own focus points and direction, they differ greatly.

This thesis is structured in the following order.

In chapter 1 the tool 'SPADE[3]' developed by Edwin Hendriks and Marcus Klimstra at Logica and CGI is explained. It describes the domain and the problem. The chapter ends in stating the research questions. Chapter 2 is an overview of literature about the research areas providing a stepping stone for chapter 3 which consists of the operationalisation of theory and own attribution resulting in certain statements (Guidelines) to keep in mind when designing a dynamic Graphical User Interface (GUI). Chapter 4 consists of several design ideas addressing several aspects of the problem. One the designs has been worked out in detail in chapter 5, consisting out of a description of functionality in the form of use cases. Chapter 6 ends with conclusions and future work. Chapter 7 mainly lists references.

[1] Company founded in 1976. 'CGI' is an French abbreviation of "Conseillers en gestion et informatique" which means "information systems and management consultants".
[2] For more information about Nikos Makris, see section 7.2 (p.49).
[3] Spade is an abbreviation of Smart Process Acceleration Development Environment

1. Introduction into the Domain of SPADE

In this chapter the domain software of this project will be explained.

1.1 SPADE the transformation engine

“SPADE” is an abbreviation of *Smart Process Acceleration Development Environment.* SPADE is a transformation engine[4] that has the ability to convert fully smart business requirements[5] to business processes which consist of a flow of user processes and system processes. Furthermore it has the ability to create all the software that is required to execute all those processes. A second major part is the SPADE server side which consists of (a large set of) generic functionality which, together with the generated functionality, creates a complete operational system.

The pre-programmed framework that consists of the front-end and user role management system that load the generated modules, has standard user interface capabilities. Extending this part, the user interface, is the focus of this project.

1.2 SPADE Software Development

As stated above – in a simplified manner – SPADE is a transformation engine. However, when you put this engine (or the correct term ‘SPADE Generator’) in perspective, the subject is a whole process, specifically a process of software engineering. As you can see in Figure 1 below, marked with red, using SPADE development takes a shortcut compared to traditional development paths (orange). The SPADE generator (almost) automates all the other steps.

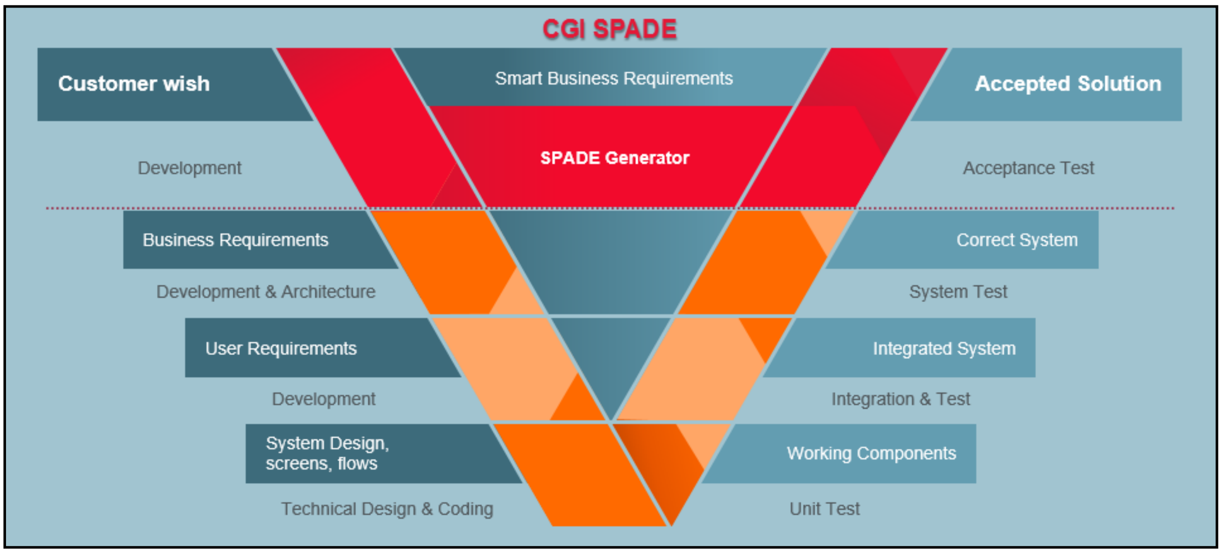
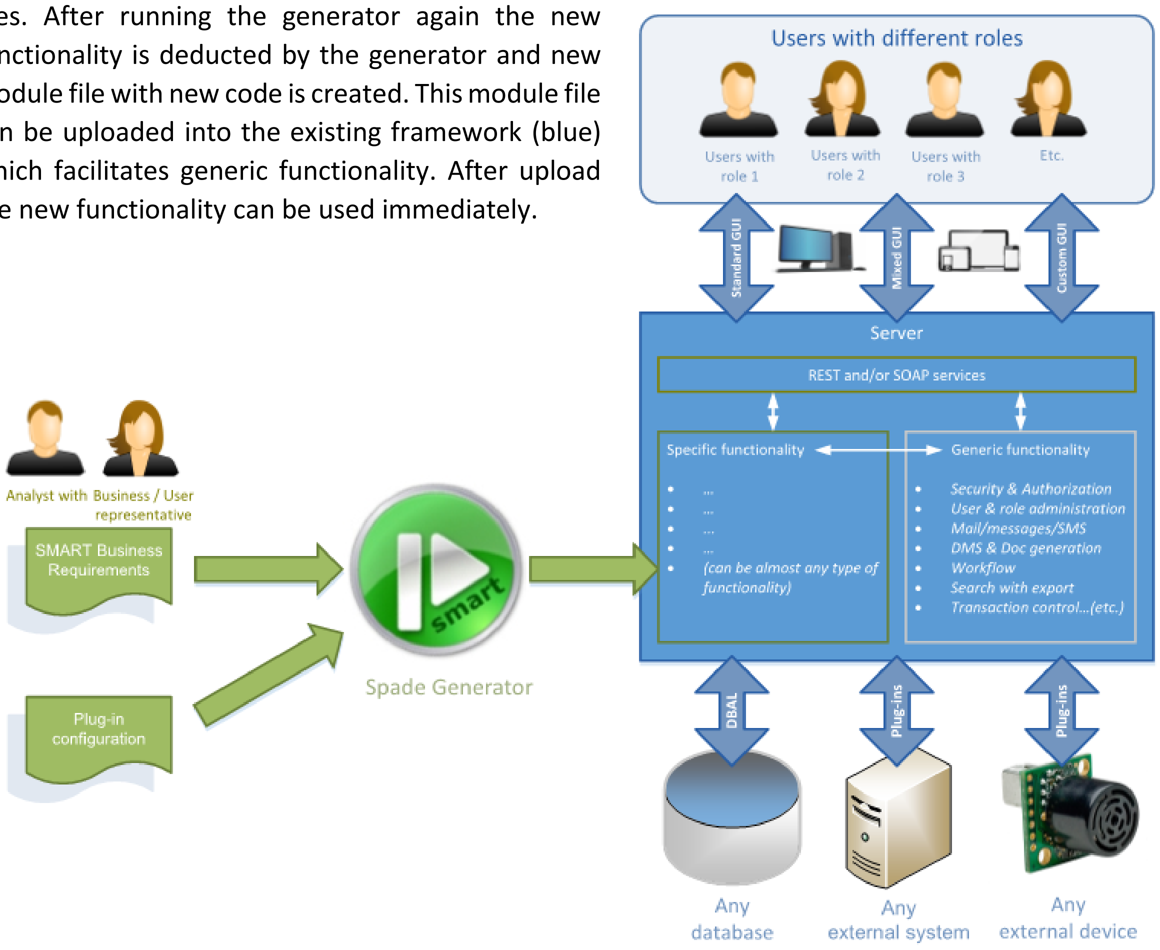


Figure 1: Spade development versus regular development (source: CGI)

[4] Also called SPADE Generator or SPADE Transcompiler.
[5] Level 4 conform Smart Req. 2.0 (Aydinli, Hendriks, & Zandvliet, 2013)

Any required changes are specified in terms of end-results in the business requirements specification files. After running the generator again the new functionality is deducted by the generator and new module file with new code is created. This module file can be uploaded into the existing framework (blue) which facilitates generic functionality. After upload the new functionality can be used immediately.



1.3 SPADE's Philosophy

SPADE development is founded on the principle that some information must be stated, but a lot can be deducted from that initial information. A compact description of SPADE development would be that it is end-result orientated development.

An engineer only needs to define:

- what the end-result is;
- what information is needed to realize that end-result;
- who is required to realize that end-result.

Whatever information is needed, is a new (sub)end result which the engineer has to describe in the same way. The engineer does **not** define: how information is represented or when exactly information is presented or asked, SPADE will deviate that. SPADE is master in deviating a BPM model from SMART level 4 business requirements (Aydinli et al., 2013) and converting that into code that together with the framework results in a operational system. The result is a workflow type system that can be used by users in different roles.

1.4 Using the SPADE engine

To better understand SPADE, by suggestion of CGI, an example case was devised. The required result was a system that creates University Diploma's or in other words, tracks progress of students through their educational careers until they graduate.

1.4.1 Using SPADE with the University Diploma Case

As an exercise the SPADE's business level 4 specification language was used to model the University process starting from enrolling into a university until receiving your diploma. After a few minutes the number of relations was increased drastically. What might have looked like a simple test case for SPADE, seemed to enforce different types of relations that initially seemed straight forward. For example, keeping SPADE's philosophy active, focussing on end results: the end result is a student receives a diploma. In order to realize that, a student needs to gather a total of 180 study points. In order to realize 180 points, the student needs to take exams and receive a grades that is equal or higher than the border set by the institute of that course. Now the domino effect is starting. In order to take an exam, a student needs to follow a course, but formally this is not mandatory for every course: in some of the courses one is allowed to skip classes and only take the exam. In some courses the end-grade also exists from a (mandatory) group or individual assignment(s); in the latter case, only taking the exam, is obviously not allowed. The end grade – which can be a number or something like 'sufficient' – is determined by the teacher who is part of a research institute and of an education institute. These institutes are part of a faculty which are part of the university. Not all grades rules apply in every institute resulting in differences between courses as a student can follow courses from different institutes.

These specifications contained a high level of detail. Ignoring a detail or a step would have resulted in a system that did not match with the reality. Sometimes reality has a way of circumventing the system. For example: when a student received 180 points, the student does not magically receive a diploma by mail or anything. No, the student has to *ask for it*, or in more appropriate words, the student needs to ask for a final Bachelor or Master exam which actually is request for the diploma by filling-out a form. However, this request (sometimes) needs to be sent before all of the points are registered in the system. When the student finally has all the points registered, the student has to e-mail to the administration: "NOTICE OF COMPLETION" with all relevant information. So though there is a system in place that could detect when a student is ready, there are these mechanisms in place around the system that the administration office enforces simply to see the difference between somebody who is ready and wants his or her diploma or somebody who for some reason is not ready to receive his or her diploma. – And after all these examples, we haven't even got to the part of different master diplomas which can represent 60 or 120 points.

Though the SPADE philosophy forces the user to think in end-results and work back from there, in a system with many different processes or steps, it would be effective to have an BPM model before the user starts writing the specifications because it is more easy to think from beginning to end than from end to the beginning. The ironic part is that SPADE is able to generate such a BPM model after one has created the specifications.

1.4.2 Excerpt of *.spade file describing end-results in SMART business requirements.

Note that this is an excerpt of a lager file and that not all discussed details are processed.

Below an example of input of the SPADE generator which it uses to deduct a BPM model and generate code that can be loaded into the framework resulting in a working system.

```
Process 'Create diploma'

// ============== individual results ==============

The following applies:
    "Grades have been added"
    and
    "Student has requested a diploma"
    and
    "The diploma is created for the student if all required points are
registered"
    and
    "The diploma is printed after being verified by an employee"


// ============== SMART details about the individual results ====


"Grades have been added" =
    Several EXAM_RESULT exist in EXAM_RESULTS with:
        STUDENT  = input from TEACHER
        COURSE = input from TEACHER //Should be limited to the teacher's courses.
        grade = input from TEACHER
        gradetext = input from TEACHER
        earnedPoints =  if EXAM_RESULT.grade >= 6
                        then EXAM_RESULT.COURSE.studypoints
                        else if EXAM_RESULT.gradetext == 'Sufficient'
                        then EXAM_RESULT.COURSE.studypoints
                        else if EXAM_RESULT.gradetext == 'Great'
                        then EXAM_RESULT.COURSE.studypoints
                        else 0

[…]

"all required points are registered" =
    DIPLOMA_REQUEST.STUDENT.AllEarnedPoints >= DIPLOMA_REQUEST.requiredPoints

[…]
```

1.4.3 Blueriq Side step

To gain a better understanding in the field of SPADE, additional related software was also reviewed. In the course Business Rules Specifications[6] students gain experience with several software systems by performing different cases. One of the software systems is Aquima, though nowadays the software has the name Blueriq. Here are my observations. Please note that in the meanwhile, the software could have been updated.

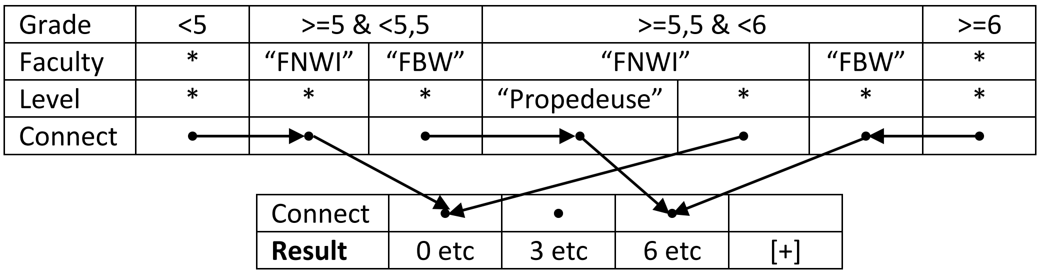
The interface is very complex and consists many buttons. Even with the proper introduction, one could still be lost in all the different tabs and (sub)buttons, side bars any other interface components. There is no beginner/expert mode, there are just many buttons. An upside that was encountered is that you are able to model a process from beginning to the end, whereas SPADE forces you to think in end results and therefore also work backwards, which felt counter-intuitive.

Another usability perk is the editable decision table which is a nice representation of potentially elaborated if-else-statements that makes it accessible for users. A downside is that it also forces you to do **double work** due to the tree-structure. It is hard or impossible to merge table columns together later on if they have the same end result. Even it is possible that two end-results can be joined together, the *design* of using a table to visualize an if-else-tree is limited itself. Even if one could merge the adjacent cells because they have the same end value, what about non-adjacent cells? The reason why linking end-result-values can have added value, becomes visible if you have to update one particular end-result-(group) in a decision table with for example 25 columns. Now you need to check every column. Below an example of such a decision table, applied to assign study points to grades in the University case. Please mentally extrapolate the complexity of the example yourself by adding extra variable rows and extra possible end results, as in this example there are only two possible end results.

Grade	<5	>=5 & <5,5		>=5,5 & <6			>=6
Faculty	*	"FNWI"	"FBW"	"FNWI"		"FBW"	*
Level	*	*	*	"Propedeuse"	*	*	*
Result	0 etc	0 etc	6 etc	6 etc	0 etc	6 etc	6 etc

Figure 2: Example of decision table

From a technical or business rules standpoint it is understandable how powerful this tool can be, from a usability standpoint: providing courses to one's clients is not the only strategy one can use to explain how to use the product. The product(design) could also do that by itself, for example: My advice for improvement of a decision table (on a design level) would not be to put variables in the result row or merging columns, it would be *connectable end-results*, which I will explain, by only showing the self-explanatory drawing (read design) itself:



Grade	<5	>=5 & <5,5		>=5,5 & <6			>=6
Faculty	*	"FNWI"	"FBW"	"FNWI"		"FBW"	*
Level	*	*	*	"Propedeuse"	*	*	*
Connect							

Connect		•		
Result	0 etc	3 etc	6 etc	[+]

Figure 3: Example of improved design decision table.

[6] Course at Radboud University provided by Dr. Stijn Hoppenbrouwers for Information Science master students.

1.5 Paradox of SPADE

SPADE – being end-result-orientated – absolves the requirements specialist and the client from answering the *how*-question when it comes to the user-interface and creating meaningful effective affordances on the screen to objects or concepts in the real world. Not answering the how-question 'back-fires' because using affordances is part of the how-answer and these were not specified in the first place.

Because the philosophy used in combination with SPADE which works very well on a business level and system level, sometimes, SPADE is unable to address certain issues at user-level simply because this information is unknown. Conversations focus on the end-result; not the *how* at user-level. (This could be defined as a desired end-result on user-level, but nevertheless SPADE is not (made) aware of these low-level desired end-results and even if it was, it would not be able to interpreter it to a specific GUI design.

The question remains how to detect the need or desire for specialized screens early on in the requirements process which is usually between a super user or business administrator and CGI employee. The majority of the process of requirements elicitation takes place at a higher level. How do you extract these requirements that are related at interactions at a low user-level? How can existing techniques of dynamic interfaces adapt to its user's needs without needing to specify the *how* question beforehand? That is the question the paradox creates. Please continue reading about the established Research Questions in section 1.7 (p.10).

1.6 Exploring Interface Adaptation Ability

Preliminary exploration of research

The following parts are preliminary results of the exploration of the solution space of the first expected Research Question: Which interface adaptation abilities can be identified […]. The list was created as a brainstorm before accessing any literature; it provided a search angle in the literature later on.

1.6.1 Part 1: Context factors

An overview of factors influencing the context:

1. Device dependencies
   a. Type of device
   b. Screen size + orientation
   c. Accessibility
   d. Consume versus produce capability
2. User dependencies
   a. Expert level
   b. Frame of knowledge
   c. Frame of reference (legacy)
   d. State of mind
   e. Native language or language capabilities
3. Time-dependency
   a. Day time versus night time (sunlight)
   b. Working versus non-working hours
   c. Awake hours versus sleeping hours
   d. Special: lunch hours or break times.
4. Location-dependency
   a. Home
   b. Work
   c. Travelling
5. Use case dependencies
   a. Goal of task
   b. Consumption of information versus production of information
   c. Links between the screen objects and objects in real life.
   d. Legacy processes (user expects it to go like X because …)
6. Data dependencies
   a. Number of data records
   b. Sensitivity of data (private / sensitive / public)
   c. Complexity of data (number of additional conditional columns)
   d. Accessibility of data (meaningfulness)

The following questions arise, among the first question that only has been answered partially:

1. What influences the context of the interface?
2. Which factors influence the interface the most?
3. Which of these factors would add value for the client if the system was context-aware of them?
4. Which of these factors could the system handle dynamically? (Required information streams)

(Added value is defined as where efficiently or effectiveness of the user interface (and process) is increased for the client of CGI.)

5. How can these factors be translated to a generic level? (Identification problem)
6. Which usage patterns can be identified that could be translated to a generic level?

1.6.2 Part 2: GUI-components

Reviewing the context factors and looking forward it seems that the solution-area focusses on linking context-factors to GUI-components or design choices. A linked factor with a GUI-component is an *interface adaptation ability*. (See example at arrow → at the bottom of this page.)

These abilities can be sorted in two groups:
- Generic Context Adaptation
- User-Specific Context Adaptation

These group names are potentially confusing because we would also like to see the specific context abilities to become generically (deductible) and some of the generic context abilities are user-depended, but require no additional information to work. The group names might be replaced in the future with terms that better cover their intended meaning, however, distinguishing between the two types of adaptation abilities is important because one group opens the door for *cross-project feedback-loops* updating interfaces among several projects using the generic functionality part of SPADE while the other requires a *mechanism to retrieve more information* about the interaction at user-level.

Generic Context Adaptation - Every interface adaptation ability that is (read should be) deductible from the Business specification realizing Generic Context Adaptation.

Context-factors: SENSORS/TRIGGERS
- Data dependency: Number of records
- Data dependency: Sensitivity of data
- Data dependency: Complexity of data

The context factors have an implicit indicator, which will make sense when we provide example of context factors that require an explicit indicator:
- User dependency: Expert level (indicator: ExpertLevel in user profile)
- Use case dependency: Links between objects on the screen and objects in real life where structure is important.

How would one measure the existence of links between objects and real life objects? How would one measure if it is structured and when it is important? This complex context factor cannot be easily detected. Either multiple different indicators have to be identified or extra information needs to be provided by the end-user which will place this context-sensor in Group B.

Examples of interface elements that can be manipulated: ACTOR
- Display size
- Display orientation
- Text size
- List or Dropdown
- Column(s)
- Grouping
- Information density
- Labelling

→ An example of Interface Adaptation Ability
Linking the context factor (sensor-side) *number of data records* to an interface-component or aspect (actor-side) *DisplayListAsDropdown* because the number for data records influences the choice whether a list is displayed as a list or as a (searchable) dropdown bar. In this case, a number determines between two different types of displaying data. Obviously more values could be used to more effectively address the need of the user. Question: When does the system 'measure' the amount of

data records? If the system measures this during the CODE phase then either the list or dropdown-menu will not be dynamically updated when the number or records changes later on. The moment of measurement determines if this interface ability is dynamic only during the SPADE transformation or is real-time dynamic during run-time.

User-Specific Context Adaptation – Every interface adaptation ability which is not deductible (easily) from the Business specification without additional information.

These group is very specific towards one specific implementation of one of CGI's client. Because most of the requirements elicitation occurs rather at a higher level than user-system interaction level, usually at system level or business level, it is not easy to predict anything without talking to the user. This brings us to the next research question, which focusses on the whole process of software engineering.

What design for *specific* IntelliGUI is best suited for SPADE's process of software engineering?

1.7 Research Questions

Because the project covers several large research areas, with limited project time, the project was divided between two master students each answering their own research question and together working towards an answer to the main question, which is:

How can a generic and specific Intelligent Graphical User Interface be incorporated in SPADE's framework?

The work of my fellow master student Nikos Makris and my own work will therefore overlap; the domain is identical and even though the focus points are different, they are related. The adaptation abilities of research question 1 (RQ1) might provide potential input for RQ4's statistical feedback mechanism. RQ3 implementation research will be an operationalized part of RQ2's design. Nikos will focus on RQ3.

Several sub research questions were identified in this project, **only one of them** will be the main focus point in this thesis:

1. Which *interface adaptation abilities* can be identified based upon the information which is gathered or modelled during the different phases of SPADE's process of software engineering?
2. What design for *specific IntelliGUI* is best suited for SPADE's process of software engineering?
3. How to *manipulate* the GUI-components *during run-time* within SPADE framework?
4. How can existing *statistical feedback methods* provide data for automatic UI-updates or design-choices in a *generic IntelliGUI* within SPADE's framework?

The focus of this thesis will be the research question 2:
*What design for specific IntelliGUI is best suited for SPADE's process of software engineering***?**

2. Dynamic Interfaces Theory

Multiple areas of research are related to the whole project. Not every area will be explored as thoroughly as others (or none at all) in this thesis due to time restriction, but also due to certain focus points and remember that the main research question is divided among two different theses.

In this chapter an overview of the related theory will be provided. Please consult chapter 3 to see how the theory is activated and discussed to identify guidelines for designing dynamic interfaces.

2.1 Static versus Dynamic interface

In their experimental study, Kumar and Sekmen (2008) proved that adaptive interface can reduce the mental load of the user. Participants were asked to execute tasks with individually controlling a mobile robot with a static Graphical User Interface and a specially designed intelligent Graphical User Interface. In their method they made sure that the 20 participants had the same prior knowledge and experience before the main experiment by performing certain ‘equalizing’ tasks and measurements, like performing a Mental Rotation test to determine the participants’ spatial reasoning abilities. The 20 participants were asked to perform tasks with the static interface and the intelligent interface controlling the mobile robot. The conclusions were that 90% of the participants preferred the intelligent interface over the static interface and that adaptivity in interfaces may be used to compensate for the inverse effect of spatial reasoning ability of the user. Furthermore, the authors stated that not too many components should be adaptive to avoid negative effects and that the timing of adaptation is important, as well as providing the user with an opportunity to reject interfaces changes fast and efficiently because if an intelligent user interface is not designed carefully, automatic adaptivity may cause the user to feel the loss of control. Thus a combination of static and adaptive interface components might be more desirable. (Kumar & Sekmen, 2008, pp. 463, 464).

2.2 Model-Driven Engineering

Research of Criado, Iribarne, Padilla, Troya, and Vallecillo (2011, p. 707) showed that model-driven engineering (MDE) and development (MDD) play an important role in user-interface design and development, especially when context- and user-awareness come into play. (Schaefer, 2007) MDE can be even more effective for user-interface generation at run-time: different user-interfaces can be generated at run-time from the same model according to context-properties such as platform, user roles, component states and environmental conditions.

Criado et al. pointed out – while setting the stage for introducing their own new level of user-interface adaptation – that Sottet et al. (2007) described an example to adapt the user interface transformation at run-time through the selection of different types of rules. Criado et al. introduced observer objects that monitor the state and behaviour of the components that realize the user-interface architecture. These observers are used to trigger the model transformations that accomplish the adaptation process. Furthermore, they can trigger a lower-level adaptation process whereby the global architecture does not need to be changed, but only one of its components. Resulting in an effective mechanism for triggering changes in software architecture of the user interface application. (Criado et al., 2011)

2.3 Adaptivity versus Adaptability

According to researchers Stephanidis, Paramythis, Karagiannidis, and Savidis (1997) there are two different types of GUI adaptations in time: during the initiation of interaction named ‘adaptability’ and continuously at run-time named ‘adaptivity’. In other words: dynamic interface behaviour that is realized before run-time (adaptability) and during run-time (adaptivity). To provide a more practical example. First, one can have a different Interface Design for every User Group which represent adaptations during initiation of interaction (just before run-time). Second, within every initiated instance, one can provide adaptations at run-time to further improve the quality of the interaction (Stephanidis et al., 1997). Similar work with the identical separation was realized by Kramer, Oussena, Komisarczuk, and Clark (2013, p. 93), calling it “static variability derived at compile time” and “dynamic variability that must be realised at run-time.”

2.4 User models & Statistics

Paskalev and Serafimova (2011) and Jameson (2009) discussed how a system can adapt to individual users by using information about the user. System adaptation is realized by “[…] user model acquisition and application […]” that require aggregated data and decision making. (See figure.) The different kinds of information that is gathered, partly matches what was mentioned earlier in section 1.6.1 about Context Factors where equivalent factors are listed.

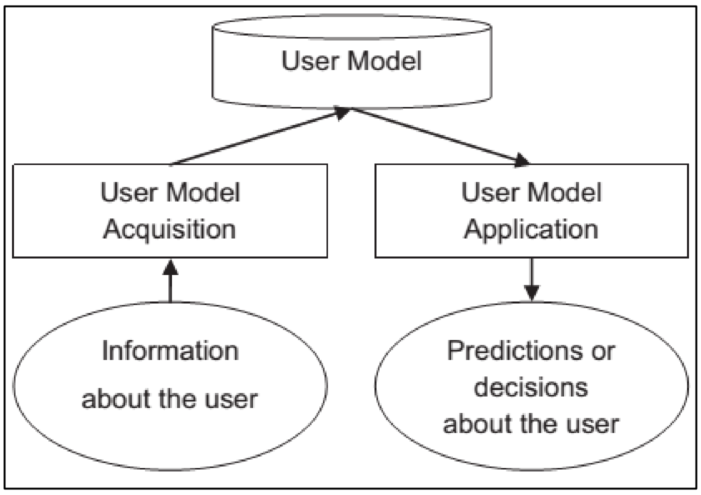

User Model
User Model Acquisition
User Model Application
Information about the user
Predictions or decisions about the user

Figure 4: User model (Paskalev & Serafimova, 2011)

Learning about the user is achieved by using ‘user traces’ which are actually a combined set of actions performed by the user with the user interface. The system gathers this information to fire adaptation rules. For example, detecting repeatable combinations of actions in sequence, GUI changes can take place which could be the merging of three actions into one new button displayed to the user.

Other researchers have found similar tactics to facilitate (now in the words of these authors) “multi criteria approach for dynamic reasoning […]” (Kabassi, Despotis, & Virvou, 2005). Where Paskalev & Serafimova discuss “user traces”, Kabassi et al. talk about “dominating actions” which seems identical. In their research they also tried to infer the user’s goals, which is called “Predictions […] about the user” in the figure above. Though the terminology defers, there seems to be an overlap between the papers. Worth mentioning is that Kabassi et al. performed an experiment with an e-mail application GUI as a test case; the system generates alternative candidate actions the user can do or actually “might have intended to do” based on observations of the system. Aggregated is used to create user profile. Karuzaki and Savidis (2014) take it a step further and present an approach that combines these profiles of different services to aid the adaptive interactive system.

2.5 Adaptive User-Interface Composition

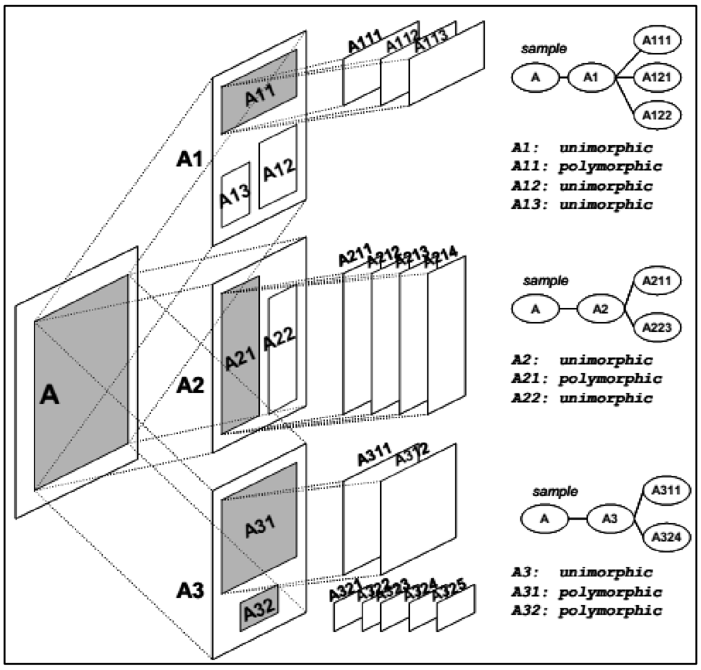
Savidis and Stephanidis (2010) say that there are no structured processes to reform existing non-adaptive systems towards systems with adaptive behaviour. It seems their research was born to fill that void. They provide a refactoring process to enable adaptive dynamic interface replacement. It consists out of 7 steps, listed below:

1. • Identify roles and requirements
2. • Model profiles and define stereotypes
3. • Identify adaptation decision logic
4. • Encapsulate adaptation alternatives
5. • Abstract containment and deployment
6. • Support factories and repositories
7. • Enable adaptive dynamic replacement

Figure 5: Adaptation-orientated user-interface refactoring process (Savidis & Stephanidis, 2010)

Please take a moment to look at the following quote:

> "[…] adaptive composition involves at runtime the adaptation-driven selection and activation of dialogue components from a pool of related alternatives with the design to aim to optimally support the respective user tasks." (Savidis & Stephanidis, 2010, p. 22)



Looking at their work on adaptive composition, the following figure is used to explain adaptive behaviour. It describes a user interface design that operates with alternatives. Different GUI-components or actually, different versions of GUI-components are created and a different version is activated depending on certain factors from the user-profile or context-profile. This can effect different parts of the screen with not all components having the same level of variety.

Figure 6: Concept of polymorphic user-interface containment (Savidis & Stephanidis, 2010, p. 20)

2.6 Cross platform interface generation

Another field of research sheds light on a whole new area (cross platform generation) that is worth mentioning because of the potential future relevance for SPADE, but will not be continued because it is beyond the main scope in future sections in contrast to the previous sections of chapter 2 which will be referred to in chapter 3 numerous times.

Cross platform interface generation refers to the automatic generation of interfaces for different platforms using models. Earlier related work by Nilsson, Floch, Hallsteinsen, and Stav (2006). One can create an interface for different platforms (iOS, Desktop browser, etc.) or use a generator to create all saving on maintenance time and development time. Macik, Cerny, and Slavik (2014) creatively propose a system that realizes this. They have the advantages of native applications while they avoid platform-specific user interface implementations. Generated forms use the benefits the native capabilities without the need of having two (or multiple) different development tracks. The kicker is this is a run-time solution, meaning the interfaces (forms) are generated for different platforms at run-time[7]. (Macik et al., p. 228)

[7] Future research: Combining SPADE's dynamic interface with Macik's work.

3. Creating Guidelines for Dynamic Interfaces

In this chapter the theory of chapter 2 is discussed to work towards design guidelines of a dynamic (intelligent) user interfaces. Obviously the Research Questions in chapter 1.7 provide direction. The guidelines are on their own part of the product of this thesis, some of them are also incorporated into certain designs (or ideas) shown in chapter 4.

3.1 Adaptive interfaces that decrease the mental load

As mentioned in §2.1, Kumar and Sekmen (2008) performed an experiment with a dynamic (intelligent) graphical user interface for controlling a mobile robot. Generalizing the participants' measured preference of a dynamic interface over static interface onto other contexts without further experimental research, might be a bridge too far. However, since the goal of this thesis is not to prove that adaptive interfaces are beneficial over static interface, but to design an adaptive interface that best fits SPADE's process of software engineering, the focus can be on what that adaptive (intelligent) design should be, rather than the need to proof that an adaptive interface is better. Kumar and Sekmen's work provides advice to keep in mind when designing dynamic interfaces. For example, Kumar warned that:

> "Although an [Intelligent User Interface] is expected to decrease the mental load for the user and enhance the performance, it should be designed carefully considering the fact that multiple adaptive interface components might cause interruption that may have negative effect.
> […]
> … automatic adaptivity may cause the user to feel loss of control.
> […]
> Therefore, an interface with adaptive and non-adaptive interface components might be more desirable"
>
> (Kumar & Sekmen, 2008, p. 464).

3.1.1 A Guideline for creating Dynamic Interfaces:

One can interpret Kumar's first warnings and operationalize in guidelines that have subtle differences:

- *Do not change too many components as that may cause confusion*.
- *Do not change too many components at the same time as that may cause confusion*.
- *Do not change too many components to prevent increasing the mental load for the user.*

I argue however that it is not specifically the number of adaptive components which should be limited per se, but that one provides the user with enough static parts or a static structure in the interface so the user can rely on that. Beside that static part, the interface can have as many multiple adaptive interface components as required, as long as the static 'frame' remains the same or within the borders of the user's perception of expected behaviour of the interface. This supports Kumar's assessment of combining adaptive and non-adaptive components, even though Kumar's reason to *why* the combination is sometimes more desirable, seems more HRI[8]-only-related, the combination should avoid the previously mentioned user's unwanted feelings of loss of control.

[8] Human-Robot-Interaction. General term to describe all types of interactions an human and robot can have.

To expand on this issue, I think the user can handle pushing a bit beyond the user's expectation of the interface's behaviour, into new territory, otherwise users would never be able to work with new interfaces. However, the trick is to find balance: it is no problem if the user is a bit surprised by some elements changing on the screen (as this might be a new specific experience for the user), but just after the surprise there must not be any confusion for the user about *why* this has happened or at least about the purpose of the change; meaning it should be self-explanatory. The users should continue with the task at hand, either using the shortcut an adaptive interface provides or (if possible) the normal basic course of events. Therefore, I state that the new updated guideline is:

> *The interface must – one way or the other – relate to the existing frame of reference of the user, where the behaviour of adaptive components of the interface decreases the mental load of the user, but does not turn surprise into confusion; thus limiting the difference between the user's expectation of the interface's behaviour and the user's observation of the interface's behaviour.*

To show the reasoning and stating this more explicitly, the following points provide more insight:

- Being surprised for a short period of time, is acceptable, however:
- Being surprised for a long period of time, is not acceptable, because:
    - Being surprised can turn into confusion and should be prevented, because:
        - Confusion can turn into frustration and this must be prevented, because:
            - Besides the fact that this last scenario will most likely *already have neutralized any beneficial effect of the dynamic interface* – either the decreased mental load or decreased time to total task completion – it might also negatively influence the user's emotional state which can result in termination of the interaction or worse, permanent abandonment.

In conclusion, when designing an interface and therefore the interaction between a user and a system, the designer usually never intends to cause confusion let alone frustration, however, this is a side effect of design decisions that do not take into account several usability guidelines or best practices or causality scenario running that could have prevented (or predicted) the aforementioned negative effects or at least minimized the chance they occurred. This is true for every type of interface, not exclusively for graphical interfaces or dynamic graphical interfaces. A dynamic interface – which has parts that can change – has to pay special attention to prevent confusion. Taking the expanded guideline into account is a step in the right direction, especially because when you try to apply the guideline, you should notice that whatever dynamic design is created, it must be related to the frame of reference of the user. That makes the design of the dynamic interface dynamically dependent of the user's frame of reference. It is a double dependency, designers should realize that.

3.2 Introducing Statistical Feedback Methods

Kumar and Sekmen (2008) also mentioned that an interface is made intelligent by inferring from the user model. “One way of developing the model is to collect the metrics of users’ interaction with the interface into a database accessible at run-time. After collecting the metrics, a user model is developed using the learning algorithms of Bayesian networks from the data.” (Kumar & Sekmen, 2008, p. 459). In section 2.4 (p.12) related work by Paskalev and Serafimova (2011) and Kabassi et al. (2005) were mentioned. Their acquisition technique focussing on repeatable action patterns (or dominating actions) seems compatible with a higher level[9] of dynamic interface.

This strategy, hereafter referred to as *Statistical Feedback Methods*, is required for the interface-engine to make sensible choices based on user action patterns. Though the same method could be used for a system engineer to evaluate interfaces, like with A-B testing for example, the added value for SPADE’s process of software engineering is the possibility of creating one or more feedback loops that have a positive effect on the usability of the dynamic interface.

What needs to be determined is where the feedback loop is exactly tied into, how many stages the feedback loop crosses. Furthermore, also whether the feedback mechanism can occur at real time or only for every transformation of business requirements[10] prior to run-time.

3.2.1 Feedback loops in SPADE’s process

Identifying these relations between business rules and adaptive components is tricky because one cannot always predict what kind of (combination of) business rules might need to result in certain desired forms at run-time. Furthermore, it might be too complex to identify these components without using some measurement tool that can identify relations in a (large) data set. A statistical feedback method might provide insight, however, the problem is that the feedback loop (A) becomes very long, meaning: spread out over many steps. (See figure below.) Writing the business rules specifications and letting SPADE convert it into code and loading the resulting module into the framework is not done without human intervention. So whatever statistics can be gathered before or during the transformation, this takes time and has to be manually uploaded to the framework so the aggregated information can be used at run-time to facilitate – over time – intelligent GUI behaviour. Alternatively it has to be processed to improve the generator itself which is very labour intensive.

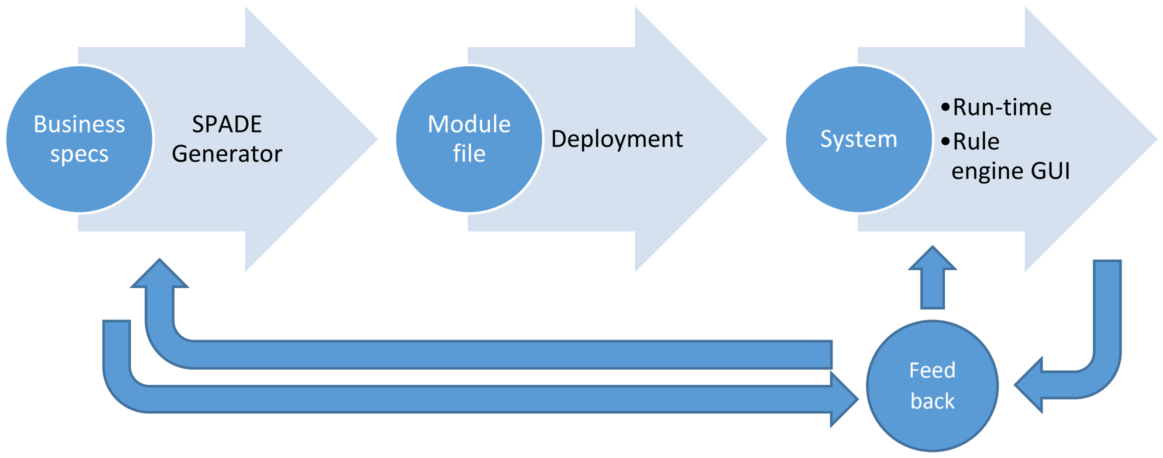


Figure 7: Feedback loop across different stages in the SPADE process.

[9] Read section 3.3 (or more specific §3.3.1) for an explanation on “a higher level of dynamic interface” (p.20).
[10] The end results described in smart business requirements level 4. For an example, see §1.4.2 (p.6).

Another feedback loop is the one that only exists at run-time, which can run continuously and could facilitate intelligent GUI behaviour immediately.

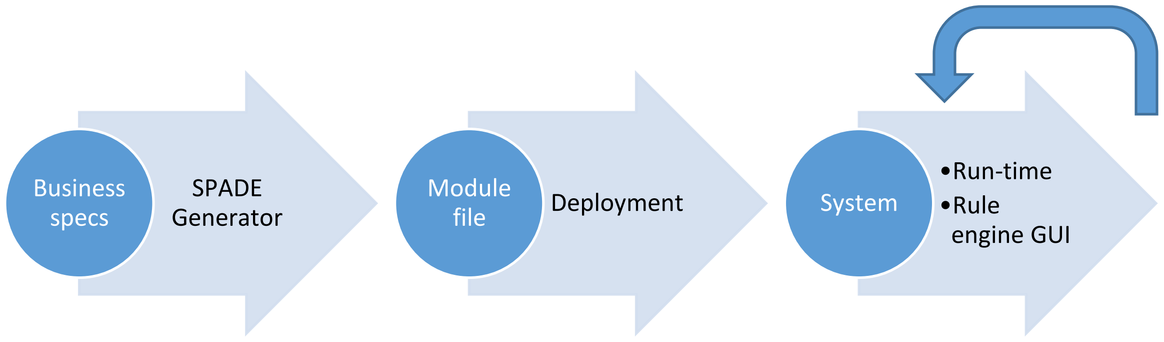


Figure 8: Feedback loop during run-time.

The separation between these two feedback loops might be artificial. From a technical standpoint it is just a rule-engine making decisions on aggregated data where the source or the "distance between the rule-engine and information" is not relevant. However, they are separate feedback loops with different characteristics and practical limitations and options and should be addressed separately until a practical plan of merging them has been identified. As mentioned, it would seem logical, from a technical point of view, that the rule-engine makes interface composition decisions, simply has multiple sources of input upon which decisions can be based or learned. From a design perspective however, due to the difference between the loops, one being crossing multiple stages, requiring human intervention and the other running continuously during run-time it is better to approach them as separate entities because most likely they will require different solutions and deliver different amounts of added value (if any) to the end user.

3.2.2 Guideline for identifying desired dynamic interface behaviour.

I suspect the bigger loop to improve the ability for the tool takes an educated guess on how certain forms should be created, more having a long term effect in the generic IntelliGUI. This while the smaller loop is more of a short-term solution providing the user with a lot of power and facilitating the user in everyday use by *observing and reacting*.

To combine this with section 3.1.1 (p.15), the more complex, stage overlapping feedback loop has the ability to mainly improve the adaptability of the system whereas the run-time feedback loop improves the adaptivity of the user interface.

Based on desired dynamic behaviour, a designer should divide the requirements among these two groups and with help of a developer determine carefully the actually costs of development of these groups and determine with the client what actually is required.

The next guideline, mentioned in section 3.3.2 (p.21), extends this guideline perfectly.

3.3 Interface Adaptation 101

Researchers have shown a difference between adaptivity and adaptability (section 2.3) where timing of the 'changeability' determines which type of GUI adaptation is applicable. When designing dynamic interfaces it is in important to realize the difference because they require different approaches and have different results. Within adaptability thus before run-time the designer has the opportunity to address differences between larger user groups, by designing different starter points, different screens whereas the continuously GUI updates during run-time provide the opportunity to address the users unique needs, as well as any other context factors that could not have pro-actively determined.

However, there is an additional aspect that has not been discussed yet. Which parts on the screen are exactly dynamic? In section 3.1.1 it was determined that a combination of static and dynamic interface components was best, however, this still can be operationalized in different ways. For the purpose of addressing this issue: when imagining a basic set-up, a static part (menu) with a dynamic part (content). The most simplistic view would be that the menu would always be the same and influence what is displayed in the content part. In the context of dynamic interfaces however, there is nothing stopping the designer from making the menu itself dynamic. Either based on the content itself or based on other factors. In either way this makes the previously labelled static part suddenly dynamically depended on the content. Resulting in the separation between static and dynamic GUI-components becoming more artificial, or at least so it seems. The problem lies in repositioning of focus points and the limitations of language. In other words: when talking about interfaces and making certain components dynamic, the actual focus point can jump from one component (group) to another when navigating back and forth from concrete to abstract levels of thinking (or design). Our language unfortunately does not provide an explicit difference, but it can be clarified:

To go back to the basic example of a menu and a content part. The menu-component without any menu-items provides structure implicitly. As soon as a user identifies these two major differences on the screen, the menu and the content part, we are inferring from his existing frame of reference and the menu will provide structure. That the menu itself (zoom-in on menu component) has dynamic behaviour does not matter because on a higher level it is a static component providing structure. (The menu is not going to jump around the screen and take different forms or change its operating procedure.) A component can be static and dynamic at the same time; this is why it so hard for our language to describe desired dynamic behaviour. The reference to the menu-component in relation to the whole screen is a different than the reference to the menu-component in relation to its own inner workings and dependencies (to content for example), but mortal humans still call this "menu".

When looking at Figure 6 at page 13, from the work of Savidis and Stephanidis (2010), it becomes suddenly more clear how a component can be dynamic and static and the same time. Component A1 and A2 are examples of such components whereas component A3 is completely dynamic; or to use the words of the author, polymorphic. It is about the relation to its surroundings.

3.3.1 What is an IntelliGUI?

Intelligent Graphical User Interface merely refers to an interface that has dynamic capabilities; in this project it was first used a working title. Along the way it became clear what kind of IntelliGUI CGI was looking for or actually what kind of dynamic behaviour was desired. (Akoumianakis, Savidis, and Stephanidis (2000) provide an elaborate overview, including this term.) As discussed in the previous section, one can easily get confused when talking about dynamic behaviour because of shifting focus points; the section provided a view towards dynamic interfaces, extending that view the include the notion of 'IntelliGUI' is the goal of this section.

In essence the concept of IntelliGUI results in a discussion on what is and what is not intelligent dynamic GUI behaviour. However, a growing knowledge base due to research forced to address this term. To facilitate this, below a borrowed layered technology view used to describe an artificial intelligence (or sophisticated rule engine) that interacts with users to control several utilities of a house or office.

Legend: an actor is any device that can be operated, this can vary from a simple light to climate control in a large skyscraper, to sprinklers in the garden, to screens etc. A sensor is a device that can detect a certain exact conditions, for example, the sun going under or the temperature or a person walking up the stairs or entering a room.

- Level 1: Control of actors based upon user-input.
    - An app on your phone providing a digital switch on your phone, no intelligence.
- Level 2: Automatic rule based control of actors based upon input or sensor data.
    - Timer based or sensor based, predefined, no intelligence.
- Level 3: Adaptive automatic rule based control of actions based upon sensor data and corrections made by user.
    - Learning rule engine, pseudo intelligence due the effect of memory.
- Level 4: Adaptive, predictive automatic control of actors based upon artificial intelligence, sensor data and previously learned rules.
    - Intelligence can extract information and combine in with big data, user-profile and domain resulting in feedback or control-action of actor(s).
- Level 5. Speech interaction* between Avatar and user resulting in actions based upon speech and all of the above.
    - Digital switch with a lot of possibilities: voice command:
    - Intelligence can extract information from spoken words and combine in with big data, user-profile and domain resulting in feedback and control-action of actor(s).

Figure 9: Layered technology view to determine level of intelligence. (Arendsen, Kuznetcov, Jadidian, & Gadjieva, 2014)

When you project these levels onto dynamic interfaces, the following can be stated.

1. When applying level 1, the user can change the GUI by clicking a button. (Manual action, run-time.)
2. When applying level 2, this would be the alternate GUI versions realized before run-time. (Adaptability)
3. When applying level 3, this would add run-time adaptivity and a feedback loop with memory.
4. When applying level 4, this would add predictive behaviour based on SFM and models.
5. Level 5 is not applicable.

When does a dynamic interface become intelligently dynamic? One can argue that level 4 still an elaborate rule engine with no intelligence. On the other hand, starting from level 3 the memory effect could be construed as the equivalent of mimicking behaviour which could be perceived as a basic from of intelligence. Nevertheless, it seems logical to keep these different levels in mind when designing a dynamic interface, as each level turns up the level of sophistication and therefore also provides a sense of direction on where to start. A designer should link functional requirements to these levels and determine what level of sophistication (or 'intelligence') is required.

3.3.2 Guideline for determining level of desired dynamic interface capability.

A designer should link functional requirements to the previously mentioned levels of dynamic interface capability (see above) while being aware that the difference between static and dynamic is not that straight forward due to the fact that a GUI component can be both: $menu_{outer}$, representing the static menu GUI-component in relation to the whole screen providing structure and $menu_{inner}$ representing the inner workings of the menu GUI-component that can have dynamic behaviour based on content, user action or other factors.

4. Idea Branches: Creating Designs for Dynamic GUIs

In this chapter the freedom is provided to go beyond certain design decisions and see what other cascade effects (on functionality) this might have when development would go down a certain path (read idea branch[11]). This chapter contains several design ideas of different magnitude that address the issues at hand. One of these designs is exemplified in chapter 5 with a very high level of detail in the form of use cases.

4.1 Introduction

CGI might change their mind about a design decision, however the functionality can still be ported to a new idea branch. Ideally one wants to port (e.g. migrate, copy over, "cherry-pick") all the desired functionality towards a branch without any of the limitations the original branch has. This provides the most freedom and flexibility for the system. However, since there is an existing system this resulting branch always needs to be compatible with the existing idea branch.

Extracting functionality from an idea branch and projecting it on the current system with the easiest road to implementation might result in loosing nuances or overall usability perks that really could add value for the end user. Such a projection should therefore always be executed with the utmost care.

[11] "idea branch" is about exploring ideas going in different directions. The subject is not actual code nor a SVN/GIT repository.

4.2 The Ribbon versus The Magic Wand

The Ribbon resembles an idea of a static part of the interface that is used to manipulate other parts of the GUI. This magic wand resembles the idea of a context-dependent menu-button to manipulate the GUI-component(s) it is linked to.

The Ribbon provides more points of extensibility than the magic wand. Furthermore, the design of the ribbon itself can guide the user in using it whereas the magic wand, though easily accessible, requires the user the understand more before using it. Though the implicit context of a magic wand, which is equal to any (right click) context menu, can provide filtered actions only applicable or useful to that context or object, it cannot provide the stature or the reliability of a ribbon that is always there, on which a user also can passively rely on by seeing the structure and grouping of the actions on the ribbon which can direct the user through a process. A context menu (read magic wand) shifts the responsibility more to the user. The user needs to know what he would like to do before "he right-clicks" or in our case, clicks the magic wand next to the GUI-component. The ribbon however, taking a part of this responsibility, is susceptible to an overload of options[12] due to the nature of high extensibility and freedom on the ribbon itself. This can be mitigated by defining some guidelines on the ribbon and applying it to all the tabs or by projecting a bit of context behaviour on the ribbon: as soon as a GUI-component is selected, the relevant options become available on the ribbon. A correct balance between using the ribbon with the structure and context-aware buttons needs to be found.

Below preliminary mock-ups of the Ribbon design are displayed. Please be aware that when building a mock-up not all aspects can have the same level of focus at the same time. The titles of the tabs were made overly explicit because at the time these mock-ups were created the difference between read-only GUI-components and read-and-write GUI-components was discussed in a brainstorm session with CGI[13] and the main focus was about what will be on the ribbon rather than how to name a tab, which has been picked up later on in the design process.

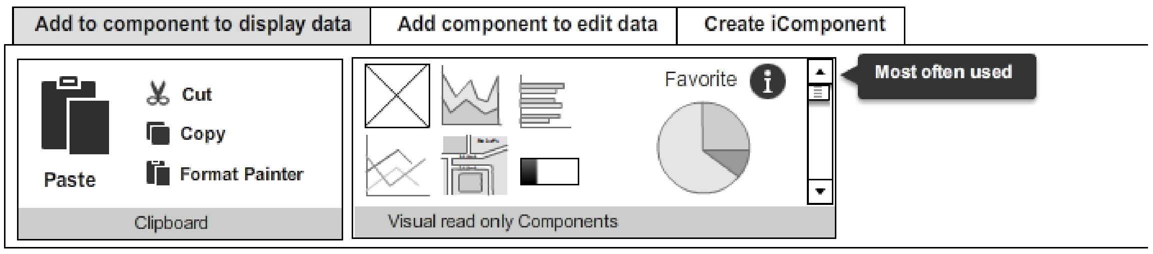

Add to component to display data
Add component to edit data
Create iComponent
Paste
Cut
Copy
Format Painter
Clipboard
Favorite
Most often used
Visual read only Components

Figure 10: Early mock-up of Ribbon design – Tab for adding GUI components that visualize data.

On the right side, the ribbon provides different GUI-components that can be added to display data. There is a representation of an image component, several types of graphs, a map, a quantity indicator and a pie chart. Furthermore you can see a favourite part which facilitates the listing of the components that are used the most for easy access for the end-user.

The general clipboard section was added to improve the reference of the ribbon idea. This has no other pre-determined practical value. However, the 'format painter' could be hijacked to copy GUI-component's characteristics to other GUI-component saving the user repeatable actions. Imagine a graph for every month (so 12) with special component settings. Manually it would take 12 x (6 mouse clicks + 2 entered values) = 96 actions, while using the 'format painter' or rather the 'setting painter' it

[12] An example of option overload has been previously mentioned in section 1.4.3 Blueriq Side step (p.5).
[13] CGI, the company providing direction for the research of this thesis.

would be possible in 8 actions to set is once + number of extra components x 2 clicks to copy settings which reduces the total number of actions.

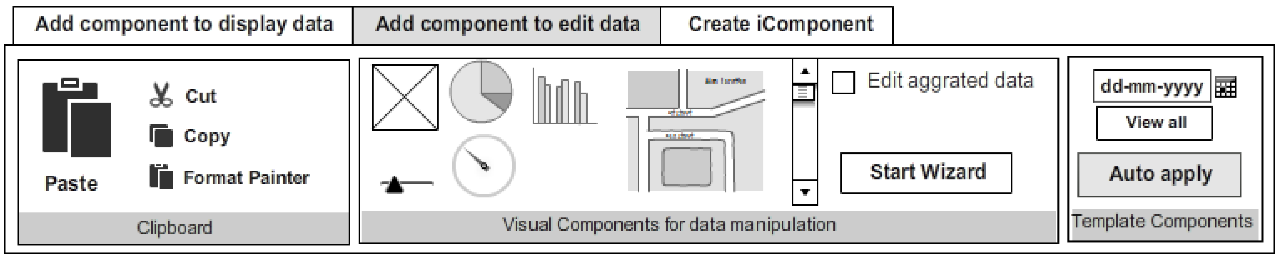


Figure 11: Early mock-up of Ribbon design – Tab for adding GUI components that visualize and manipulate data.

In this mock-up (above) of the second tab, you can see slight variations. Sliders are introduces to edit numerical values. This allows form-fields to be manipulated by these GUI components providing better control as well as a better overview of range and outer limits. – The "Edit aggregated data" option exists to facilitate the situation where a GUI-component must edit (manipulate) date that is actually a calculated value. It will require the end-user to point to values that can be edited which will have an effect on end-value in question. For example, if the GUI-component manipulates the total reserved spots of parking spaces which is combined number of three different parking garages. Editing the total value can only be realized if the end-user points the values of the individual parking garages. The idea is that the ribbon guides the user through this process.

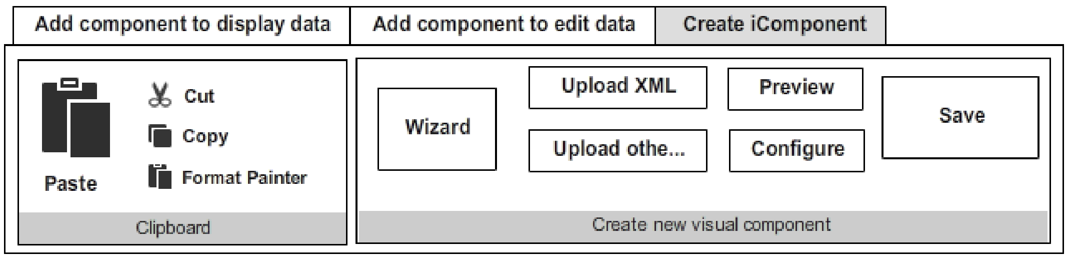


Figure 12: Early mock-up of Ribbon design – Tab for adding custom GUI components.

In this mock-up the third of creating an custom 'intelligent' component is displayed. The idea is that the ribbon provides options for extensibility for users to add their own components.

No mock-ups were created of the magic wand. The Magic Wand was simply a button that would become visible if you select a component. When a user clicks on this magic wand, a menu appears on the spot with options for that component. Options that were not relevant, would have been hidden in the menu.

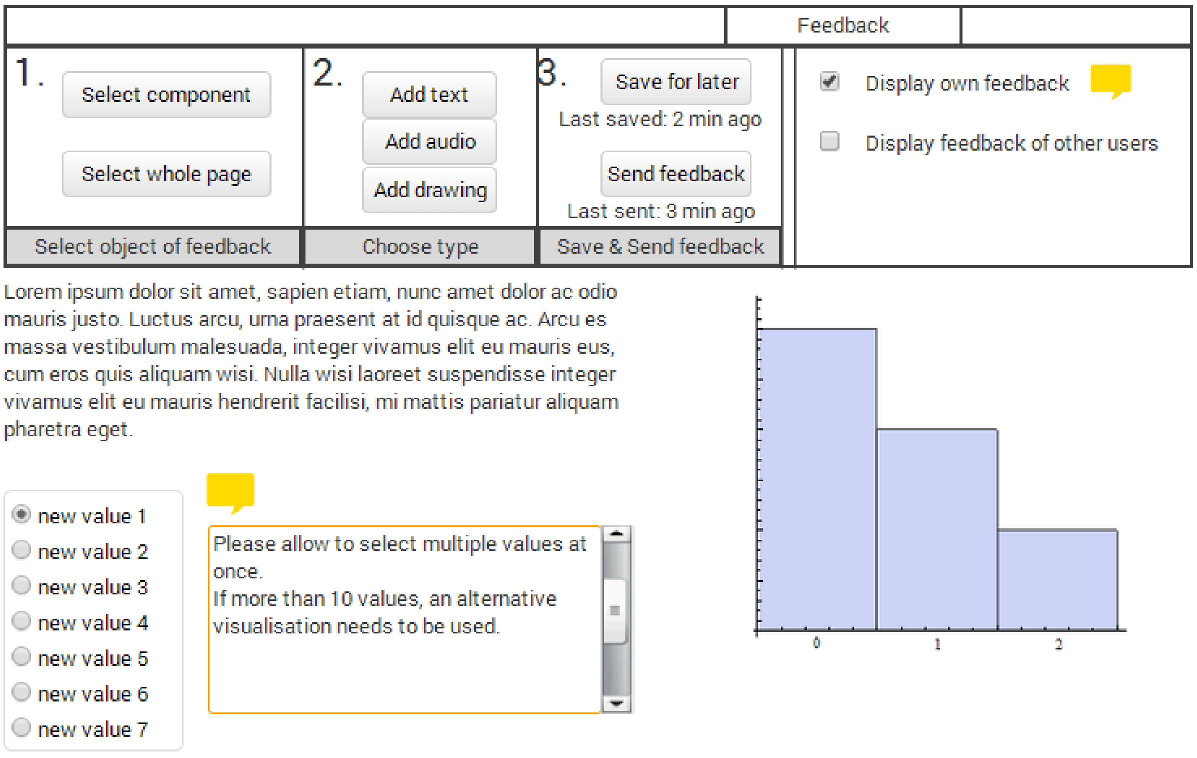
4.2.1 Acceptance

A perceived downside of the Ribbon idea in comparison to the Magic Wand idea was that it would take a longer time to develop. A CGI developer told that the development time between the two ideas did not have to be necessarily different. The company preferred the Ribbon idea over the Magic Wand idea and the context-awareness characteristic of the Magic Wand menu was projected onto the Ribbon creating a context-aware tab or at least context-aware options. Other use cases, like combining two different fields into a new GUI component seemed more logical in the Ribbon idea then in the Magic Wand idea because the latter was oriented per field/component. For a full description of described functionality, please consult the uses cases in chapter 5.

4.3 Ribbon extensibility: Feedback form

In a nutshell, for every new feature group, a new tab can be created. Ignoring for the moment if that improves the overall usability. The tabbed design facilitates this by design. For the purpose of exploring different ideas it is acceptable to think in different Ribbon Tabs, however, when arriving at a later stage of development it would be prudent to re-evaluate the artificial separation by tabs. That this separation is logical during the design phase or even the development phase does not guarantee that this separation is also logical during run-time for the end-user. It would not be the first time an unwanted separation like this, found its way to run-time and completely frustrate the end-users. That being said, let's extend:

The Ribbon Feedback tab, facilitating a communication channel between end-user and developer (or super-user). During run-time, the end-user has the opportunity to provide feed-back in different forms. Because this occurs at run-time, there exists an opportunity to transfer useful information that makes it easier for the recipient to understand what the original author means. Imagine feedback being transferred via a phone call or an e-mail. Because you have to refer (point) to certain components on the screen, the end-user might name these components in such ways that the developer has no idea what the end-user talking about and vice versa by the way! – In the Ribbon feedback example. The system can either provide the user with a screenshot of the current page where the feedback is about or, even better, facilitate an overlay (over the page) where the user can add text and draw. The recipient of the feedback can open the page and see the overlay. References to components are now made visual and direct. The language challenge still exists, but it is mitigated a bit by using this run-time feedback mechanism. – The developer can update the form very quickly and maybe the end-user does not even have to wait until the next release of software.



Lorem ipsum dolor sit amet, sapien etiam, nunc amet dolor ac odio mauris justo. Luctus arcu, urna praesent at id quisque ac. Arcu es massa vestibulum malesuada, integer vivamus elit eu mauris eus, cum eros quis aliquam wisi. Nulla wisi laoreet suspendisse integer vivamus elit eu mauris hendrerit facilisi, mi mattis pariatur aliquam pharetra eget.

Please allow to select multiple values at once.
If more than 10 values, an alternative visualisation needs to be used.

Figure 13: Mock-up of feedback tab - Ribbon design

The figure above is a rough sketch a the Ribbon's feedback tab. As you can see the Ribbon tab has 1-2-3 structure to guide the user through the process of providing feedback. So the design (the way every component is put together) explains the use of each component to accomplish the goal of providing feedback. That is why is not deemed necessary to explain the main workings of this interface to you, the reader, as they are self-explanatory. However, for a more elaborate description of the interaction between the user and the proposed interface, please consult the related use case at section 5.1.3 (p.37). – Going back to the higher level of design decisions, besides the *Structured Design should explain Function* guideline, the idea of feedback at run-time itself addresses the issue of the earlier mentioned paradox (section 1.5), namely that the SPADE development track how a new approach on answering the how question. *How do you want your information to be displayed?* You can either gather this information before the generator or after. This design followed the path of answering any how questions at the end of track, during run-time. The benefit of this situation is that the end-user is able to visualize the interface or actually look at the interface. Normally, in a traditional software development track, the user can have problems with visualizing what a certain decision looks like. This problem is mitigated with user stories and mock-ups etc., but having a direct communication line (like the feedback tab described above) to the developer while looking at the interface as part of the development process can be beneficial! Yes, the developer might still get a headache from the end-user, but he does not have to 'guess' where the end-user is talking about or pointing at. That is incorporated in the way the feedback is delivered to him.

4.3.1 Acceptance

CGI prioritized and gave the relevant use case the lowest priority. In context to developing the Ribbon or a similar GUI adaptation system, it seems logical to put this off until the primary functions are implemented. However, based on the brainstorm session considering this feedback mechanism I expect the whole aspect of incorporating feedback will get future attention when they arrive at that stage of development.

4.4 Handlers

To manipulate GUI-components on the screen, one can use handlers (icons next to a component facilitating certain actions). The benefit of using handlers is to avoid confusion between normal interaction and (for lack of a better word) meta-interaction. For example, activating a form text field by a click and click-and-move operation might cause confusion. Will the field activate? Or will it be moved? Or if a Google Maps component is added on the form then click-and-move results in what behaviour? Expected behaviour might be that click-and-move (click-and-drag) results in changing the focus of the map area, so navigating the map whereas the user might have desired to move the whole component to another position on the screen. The recap, to explicitly differentiate between these two type of interactions (normal and meta), certain 'hot-zone' for specific actions should be created; also known as: handlers. An example is shown below.

4.4.1 Acceptance

The idea of handlers was accepted by CGI when the usability challenges mentioned above were recognized. During a brainstorm session at CGI the handler-idea resulted in a combined 'handler-bar'. So all the handlers would be combined into one bar facilitating the following actions:

- moving component (relocating on the screen-grid),
- select component (also for actions that require selection of multiple components.)
- hiding or deleting component (auto-generated components are hidden, others deleted.[14])

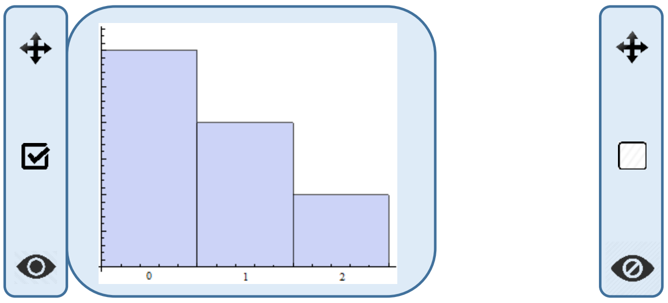


Figure 14: Mock-up of handlers for a GUI-component. (Component hidden on the right.)

Obviously when one is using the form these handler-bar is not visible. They become available when a certain mode is activated or when the Ribbon is activated if it is not on by default.

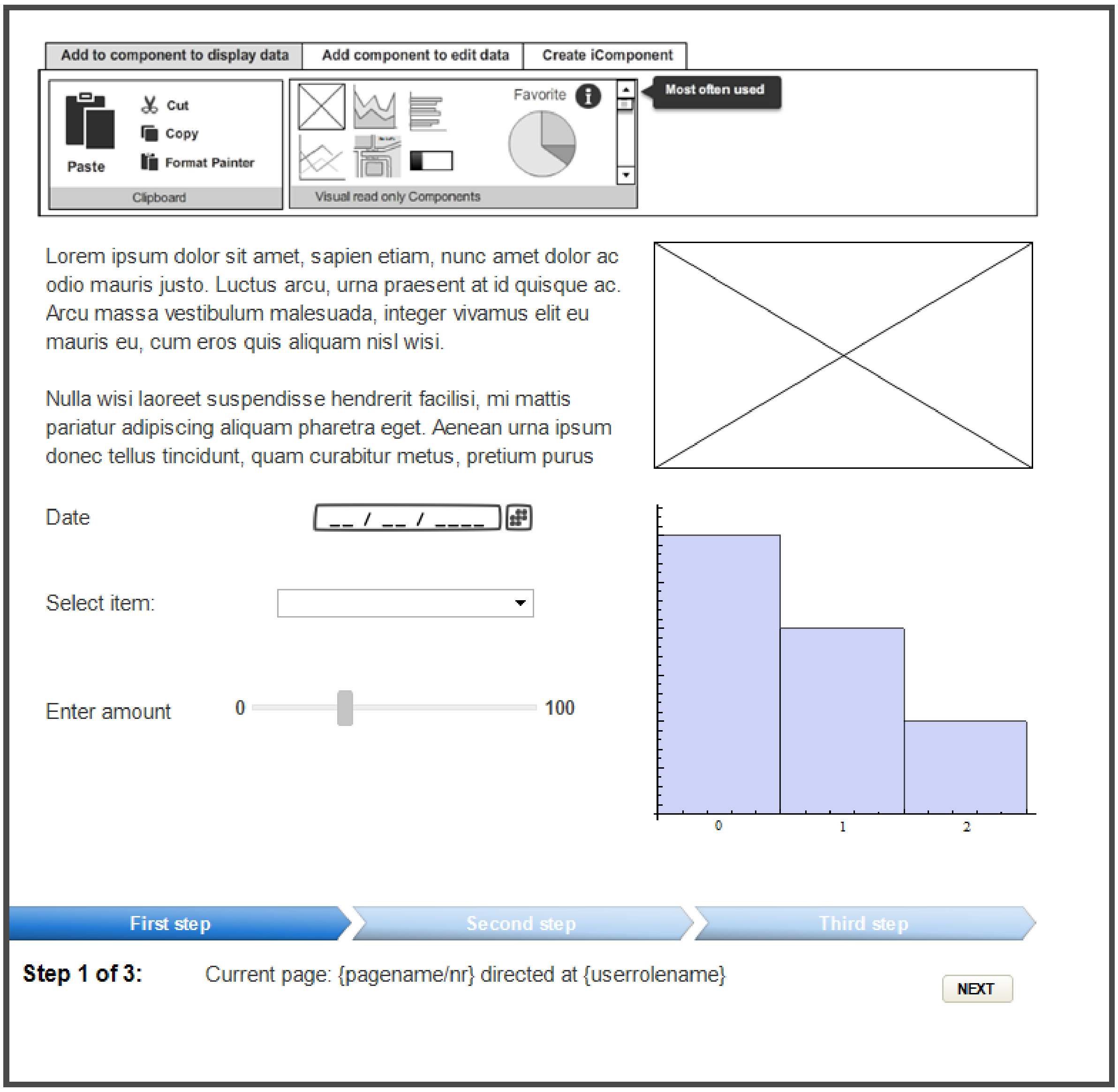
[14] The hide-icon in can be replaced by a trash bin icon if the related GUI-component can be deleted.

4.5 Simulation mode

In simulation mode a process is simulated step by step. The screens can be re-configured to set the default start configuration for the end-users. The end-users are still able to change components during run-time; but setting these pre-settings of default behaviour can save time so that not every user has to use the ribbon when starting a process.

Furthermore, during simulation mode, future interactions can be identified that require additional visualisations not yet available to the system.

Data entered during simulation mode is not saved. While a user is in simulation mode, a user jumps from one screen to the next. At the bottom in the navigating section it is visible to what user role the resulting screen will be directed at during 'real' run-time. This supports different GUI versions for different user roles if required. Only alterations to the GUI are saved. Setting overall conditional adaptations are also possible in this mode.



Lorem ipsum dolor sit amet, sapien etiam, nunc amet dolor ac odio mauris justo. Luctus arcu, urna praesent at id quisque ac. Arcu massa vestibulum malesuada, integer vivamus elit eu mauris eu, cum eros quis aliquam nisl wisi.

Nulla wisi laoreet suspendisse hendrerit facilisi, mi mattis pariatur adipiscing aliquam pharetra eget. Aenean urna ipsum donec tellus tincidunt, quam curabitur metus, pretium purus

Step 1 of 3: Current page: {pagename/nr} directed at {userrolename}

Figure 15: Mock-up of simulation mode (navigator at bottom).

The problem with this idea that going through the forms without saving data or temporary saving data creates all sort of database challenges while the added value of the mode is uncertain.

4.5.1 Acceptance

CGI has no problem with going through 10 forms, filling out all fields on every form, to edit a component on the 11th form page. Directly jumping to page 11 seems to be troublesome due to the data restrictions. However, instead of finding a way to implement this, their preference was that no simulation mode required at this time.

4.6 Squashed BPM model for navigating

In Figure 16 a BMP model generated by SPADE is displayed. As explained in chapter 1, the SPADE generator deducts this BPM model from the business level 4 specifications. As you can see the BPM model is vertically orientated.

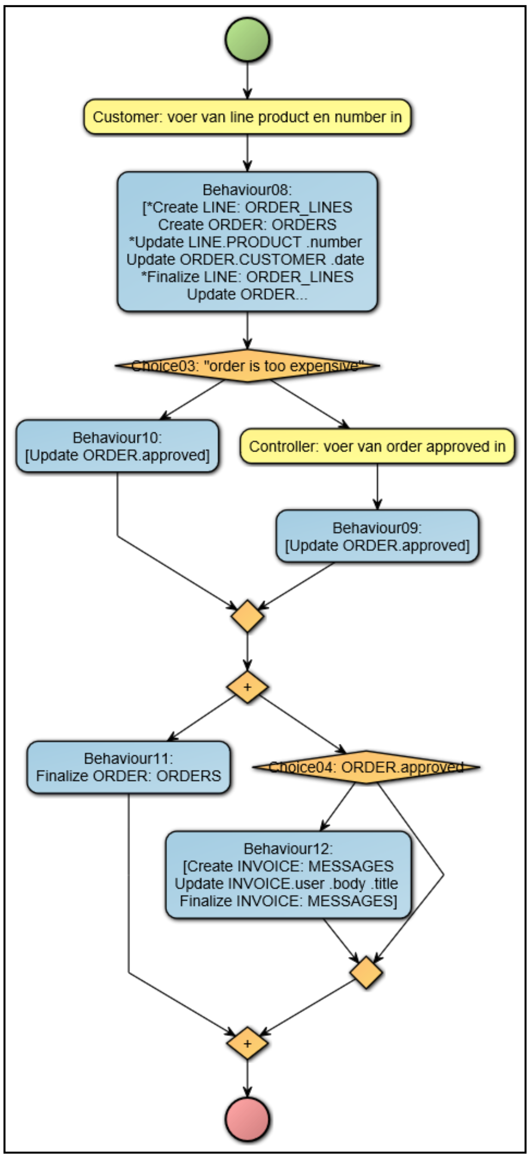

Customer: voer van line product en number in
Behaviour08:
[*Create LINE: ORDER_LINES
Create ORDER: ORDERS
*Update LINE.PRODUCT .number
Update ORDER.CUSTOMER .date
*Finalize LINE: ORDER_LINES
Update ORDER...
Choice03: "order is too expensive"
Behaviour10:
[Update ORDER.approved]
Controller: voer van order approved in
Behaviour09:
[Update ORDER.approved]
Behaviour11:
Finalize ORDER: ORDERS
Choice04: ORDER.approved
Behaviour12:
[Create INVOICE: MESSAGES
Update INVOICE.user .body .title
Finalize INVOICE: MESSAGES]

Figure 16: BPM Model created by SPADE

An idea to improve navigating through the several forms of a process, could be realized with an alternate version of the BPM model. To provide context, the previous section explains simulation mode. At the bottom of the simulation mode page (Figure 15), there is a part dedicated to navigation. The idea of incorporating a “squashed and rotated version” of the BPM model is to aid the end-user to keep track of where they are in the process on the same screen and provide the ability to jump around more quickly, but also to see dependencies between forms (pages) (and therefore GUI-components) that can be affected if one adapts the current form in simulation mode.

The idea originated from maps on busses, trains and metros where there is horizontally more space available then vertically. On those maps the sequence is more important than being topographically correct; for a BPM model for navigation the design-properties that are relevant are:

- Adapt orientation to available (screen) space;
- Display only minimum of information;
- Only differ when you really have to.

This means that the BPM-model for navigation will be different in some ways, besides the orientation. Not all text will be there. (Not relevant by default; maybe in mouse-over.) Not all different elements will be required, as the end-user primarily needs to be able to link forms (pages) to elements on the BPM model. In Figure 17, as an example of structure, a partial map of train station tracks is visible. The result of BPM model for navigation would look similar, only displaying the relevant information, keeping the design-properties in mind.

4.6.1 Acceptance

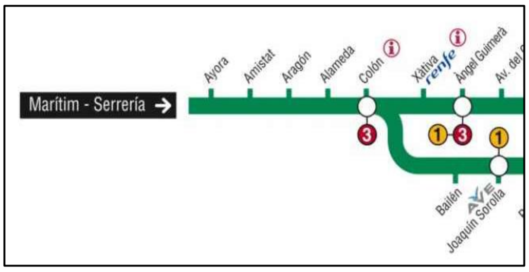


Figure 17: Map as example for navigational BPM

As the idea was part of simulation mode that was not accepted, this idea has consequently been parked. However, developers at CGI have shown creativity in extracting functionality with smaller development times than the larger designs in which they were presented; so maybe this navigation BPM-model idea finds its way to other places in their software development tracks in the future.

4.7 Priority conflict resolution of GUI preferences

The following problem was predicted: with the different factors all trying to influence the composition of the interface, collisions can occur, or rather conflict in different settings. This chapter proposes a preliminary solution to this problem, namely a priority listing.

The manager and screen-manager are in this example two different types of roles.
Screen-users are provided with screens by the system within any launched process of the loaded tar module. Going down the list increases priority.

1. Front-end system default setting (this would be the result of adaptability[15], initiation.)
2. (In case of conflict overruled by) *If-this-than-that* rule engine. (For now, perceived as separate entity from default #1; occurring at run-time.)
3. (In case of conflict overruled by) admin setting (referring to role that loads the module-tar file into the framework.)
4. (In case of conflict overruled by) screen-manager setting (referring to role appointed by admin that is able to set default decisions and default GUI configuration for its staff/colleagues/users in case the end-user does not have the required knowledge or requires a preferred starting point.)
5. (In case of conflict overruled by) screen-user setting during run-time (referring to the role that is used to fill out forms, note that this can be a user, manager, or whatever name is appointed.
6. (In case of conflict overruled by) any enforced GUI setting that should not be overruled set by admin or screen-manager.

The summary would be: System default (now), IFTTT-rule engine, admin tar-loader(1), screen-manager(1), screen-user(*), enforced settings

The issue to be addressed, if agreed on the layered approach, is that the roles already known in the system, will not 100% overlap with the roles used to adapt the GUI. One could enforce that it in such a way that it does match, but this is a bit weird because the manager cannot edit the screens of user because he does not see them during run-time. This is also valid for the admin.

So it seems it is important to have a difference between editing GUI components during run-time and editing GUI components as a preparation before run-time.

Normally when looking at rights, the admin setting overrules every role lower than itself. In the case of GUI-adaptation, the closer you get to the screen and the end-user, the more the system should ‘listen’ to this end-user, simply because the end-user is experiencing what is and is not working! This is why linking GUI-priority rules to the current roles might be troublesome because it defies normal expectations of roles and what they do especially the expected hierarchy.

The question remaining is *how to handle this?*

A proposed solution is to separate the roles as we know it from GUI edits and then you get a priority list like this:

[15] This has been discussed in section 2.3 (p.12) and 3.2.2 (p.19).

1. Front-end system default setting
2. (In case of conflict overruled by) If this than that rule engine. (For now perceived as separate entity from default #1.)
3. (In case of conflict overruled by) **pre-set settings** set by screen-manager[16]
4. (In case of conflict overruled by) **run-time setting**s set by screen-user
5. (In case of conflict overruled by) any enforced GUI setting that should not be overruled.

4.7.1 Acceptance

CGI argued that a lower number of levels would be sufficient. There would be a super-user and an end-user at the time of discussing this subject. In hindsight, it might have been premature to have this discussion with this level of detail. However, it did address certain challenges of how GUI-adaptation data is saved and who edits the GUI for whom.

If later on Statistical Feedback Methods are hooked up to the rule-engine, I hope there is a slot available so the user does not have to fight the *automatic mechanism* re-setting his GUI every time he loads a form that has exceptional GUI requirements. It is worth repeating that determining the level of required dynamic interface capability[17] before making hard design choices, can have a beneficial effect on the development by taking into account future extension points.

4.8 The questionnaire for clients

As previously stated, information on how to the interface should look like, comes second, focussing primarily on the end-results. This section tries to identify a simple method to find information about interfaces that add value for the end-user.

The suggestion is to construct a special interview with the goal of identifying effective affordances between screen and (objects in) the real world that could add value for the end-user by optimizing information consumption (or processing) from (or to) the screen. This interview could be automated and therefore also be answered by (several) users performing system interaction tasks the business administrator is not aware of in detail. The questionnaire itself, could be created with SPADE itself. This also provides the opportunity to not only use it as mechanism to identify specific desired GUI elements, but also to provide users with information about the future system. The process could even be part of change management.

Lacking a strategy to identify the right questions, this idea branch was abandoned to continue working on idea branches which had CGI's preference anyway because of practicality. The other idea branch (Ribbon) that addresses the issue after the transformation, has more value because the questionnaire only delivers information on how the interface should be designed (for the end-user), but does not provide a method of manipulating the interface itself; which the Ribbon does do. Two totally different strategies. However, besides the fact that one occurs before and the other after transformation, the first addresses the unknown information problem, while the second circumvents that by providing the end-user with tools to manipulate the end result (read interface) itself thereby absolving (for a part) the developer to be aware of interface requirements.

[16] Also known as 'super-user'.
[17] See Guideline for determining level of desired dynamic interface capability. (Section 3.3.2 p.22)

4.9 SPADE Inception

Because the philosophy used in combination with SPADE is end-result-orientated which works very well on a business level and system level, sometimes, SPADE is unable to address certain issues at user-level simply because this information is unknown (conversations focus on the end-result; not the *how* at user-level). In other words: though the requirements specialist and the client are absolved by the SPADE engine of answering the *how*-question, when it comes to the user-interface and creating meaningful effective affordances on the screen to objects or concepts in the real world this abstraction of how 'back-fires' because using affordances is part of the how and these were not specified. (In theory one could see these affordances as specific end-results at user-level.) Now a lot of the proposed ideas address this issue at the later stages of the SPADE development process; this idea addresses the issue before the SPADE generator. It is an example of how the same language as the business level specification can be used to define desired end result on the screen.

```
SCREEN 'TUBE RACKS' // Example about screen resembling an actual tube rack which matches a real one.

// === INDIVIDUAL RESULTS ===
The following applies:
        "A rack on the screen has identical configuration as real-life object."
        and
        "A rack on the screen has the same information content as real-life object."
        and
        "A rack is filled with a number of tubes."
        and
        "A tube on the screen provides information content of the tube by click."
// === SMART DETAILS ===
        "A rack on the screen"=
                One RACK exists in RACKS with:
                xcoordinate     =
                ycoordinate     =
                infofield       =

                One TUBE exits in TUBES with:


        "Tube rack on screen has identical configuration as real-life object"=
        RACK.columns            = AFFORDANCE.Rack.columns
        RACK.rows               = AFFORDANCE.Rack.rows
```

However, though the language has a lot of freedom; the generator still needs to able to interpret it, so this idea seems to continue towards extending the generator itself.

Due to time limitations and other focus points, this idea is not explored further in this thesis, however, because of the higher relevance for my fellow student Nikos Makris, he explored the idea further: can we use a Domain Specific Language to define specific GUI components? Please consult his research to learn more about this area of the project[18].

[18] More information about this related thesis can be found in section 7.2 (p.49).

5. Exemplifying the Dynamic IntelliGUI Design for SPADE

5.1 Use cases

In this chapter functionality is described that provides an answer to the research question: what design for specific IntelliGUI would best fit SPADE's process of software engineering. Every use case is related to providing the user more control over the user interface.

The work of Kulak and Guiney (2008) was used as a guide to provide CGI with a description of the desired functionality. Take note that though some use cases are a direct result of meetings about the desired functionality, some use cases were provided to CGI as an exploration beyond a certain decision point or a potential future extension point. Changing the outcome of a decision point might have cascade effects on functionality. The most important part is the described desired functionality, which can be operationalized in different ways, though some are built upon interconnected usability ideas presented in the previous chapter(s).

Traceability in use cases.

The use case lay-out has been updated to suit needs of the situation at hand. For example, because the main research question has multiple areas of research and multiple angles, ideas come very quickly during brainstorm sessions. To keep a sense of the source of ideas the field of *Source* and *Trace* were added to the use case to improve traceability of ideas and functionality but also to identify design decision points.

Example

Use case:	Edit order of rows/columns or sorting in table
Source: Trace:	Provided by Edwin in session, Basic course of events are own interpretation, to be verified by Edwin[Yes, June 5, Priority:2]. RibbonSession2.ExamplebyEdwin("GUIadaptationMode").InterpretedbyJoeri. FeedbackbyEdwin[19]

"Source" provides information about where or when the use case idea came from, but also what the priority is and the verification status is. "Trace" is traceablity-over-the-top tactic, it uses language in a sequential way to create a thread of versions; every new version (or relation) adds a new dot in the line. A very easy way to keep track of ideas but also to identify assumptions and interpretations! In the example above, reading from right to left (ish), the current version of the use case is the result of Feedback by Edwin[19] on Joeri' interpretation of an earlier idea by Edwin called "GUI adaptation mode" during the second Ribbon brainstorm session. The trace makes explicit where the idea came from and how it developed.

Numbers in use cases

Every step in Basic Course of Events or Additional or Exception paths has a unique number so that you can refer to this step explicitly without ambiguity. Therefore different types of sequences are used. (1,2,a,b,i,ii,I,II.)

[19] Edwin, referring to Edwin Hendriks, supervisor of the project at CGI.

5.1.1 Use Case: Edit order of rows/columns or sorting in table

Use case:	**Edit order of rows/columns or sorting in table**
Source: Trace:	Provided by Edwin in session, Basic course of events are own interpretation, to be verified by Edwin[Yes, June 5, Priority:2]. RibbonSession2.ExamplebyEdwin("GUIadaptationMode").InterpretedbyJoeri. FeedbackbyEdwin
Status:	Tier 1 use case. Based on communicated idea of Edwin, interpretation verified
Actors:	GUI-user
Trigger:	A user desires to rearrange table column order or sorting.
Short description:	User can reorder columns and rows and sorting to their own preferences.
Pre-conditions:	(Ribbon is activated, dragging handlers are visible.)
Basic course of Events:	1. (System displays drag and drop handlers above/next to every column/row.) 2. User drags columns (or rows) to desired positions. 3. System updates table after users has let go of drag-handler. 4. User clicks on the column-arrow-icon to sort the table with that column. 5. System reorders table ascending on the related column. 6. System saves changes of the new arrangement of table.
Alternative path:	a. = step 4. User clicks on the column-arrow-icon to sort the table. b. User clicks on the arrow-icon again to toggle ascending/descending. c. System reorders table.
Exceptional path:	
Extension points:	Extension A: User clicks [Show column totals] to add a row with totals.
Assumptions:	*Manually re-ordering a row with drag-handler, will disable default sorting and disable sorting based on a column.*
Post-conditions:	New arrangement of table is saved to combination of user and page. Revisiting the page will automatically display the table with the new arrangement.
Related BR	
Author:	Joeri Arendsen
Created on:	Spring 2015
Version:	2
Last updated on:	

5.1.2 Use Case: Edit order of GUI-components (fields) on the form.

Use case:	**Edit order of GUI-components (fields) on the form.**
Source: Trace:	Provided by Edwin in call session, to be verified by Edwin [Yes, June 5, prio:1]. CallwithEdwin.VariationOn:DragColumn.ExamplebyEdwin.IntepretedbyJoeri .FeedbackbyEdwin
Status:	Tier 1 use case. Based on communicated idea of Edwin, verified by Edwin.
Actors:	GUI-user
Trigger:	A user desires to reorder the GUI-components on the form.
Short description:	User can reorder read-only and edit GUI-components on the form.
Pre-conditions:	(For instance: Ribbon is activated, dragging handlers are visible.)
Basic course of Events:	1. User drags GUI-component towards desired position. 2. System indicates where component would be placed if user would let go of handler at the current position. 3. User lets go of drag-handler and desired position. 4. System updates form's new arrangement. 5. System saves the changes of the form's new arrangement. 6. (User deactivates ribbon.) 7. (System hides dragging handlers.)
Alternative path:	a. = step 5. System saves the changes of the form's new arrangement. b. User navigates away from current form to next or previous form. c. System will continue to display dragging handlers until (for instance) the ribbon is deactivated. i. = User clicks [Return order to defaults] (for instance on the ribbon). ii. = System prompts "Are you sure you want to undo the manual changes of this current form and revert to default order?" iii. User chooses between [Revert to default] or [Keep changes]. iv. System either refreshes form to default order or closes prompt.
Exceptional path:	
Extension points:	Extension A: Allowing also components to be dragged horizontally, allowing components to be displayed next to each other. Components should be snapped into place. Extension B: Default sorting based upon groupings of based-on and input-from combinations. Extension C: On GUI-component context ribbon (so requires selection) also have button group that move the component, up, down, top, bottom, left and right.
Assumptions:	It is desired that dragging handlers are not visible by default so that one cannot accidentally re-order components. This choice matches Edwin's separation between normal mode and 'GUI-adaptation-mode'. *Manually re-ordering will disable the automatic order.*$^{EXT:B}$
Post-conditions:	New arrangement of GUI-components are saved and automatically used next time the form is visited.
Related BR	
Author:	Joeri Arendsen
Created on:	Spring 2015
Version:	2
Last updated on:	

5.1.3 Use Case: Send feedback about the page or GUI-component

Use case:	**Send feedback about the page or GUI-component**
Source: Trace:	Suggested by Joeri, part of qualitative feedback mechanism idea, answering the need for CGI to create specific GUI-components. By allowing GUI-users to provide feedback about visualisation of the form('s components) itself, either a super-GUI-user can change the visualisation or the GUI-user has a line of communication to a CGI-tech to create a new specific GUI-component. To be verified by Edwin[Yes, June 5, priority:4] ResearchQuestion1.StatisticalFeedbackMethods. ExtententedTo:QualitativeFeedbackMethods + RQ2.SolutionSpace:ProcessChange).IdeabyJoeri.UCsuggestion .FeedbackbyEdwin
Status:	Tier 2 use case. Extrapolated, verified by Edwin.
Actors:	GUI-user, Super-GUI-user, CGI-techs.
Trigger:	A GUI-user desires to submit feedback.
Short description:	User can enter text (or audio$^{EXT:A}$ or drawing$^{EXT:C}$) linked to a form or GUI-component to submit as feedback to fellow GUI-users or super-GUI-users or CGI-techs.
Pre-conditions:	User is viewing a form of any process. User has the need to provide feedback about the GUI towards the super-GUI-user or colleague-GUI-users.
Basic course of Events:	1. User selects the GUI-component on the form. 2. User clicks the [Add Feedback] button. 3. System displays a textbox$^{EXT:A,B}$. (In the Ribbon or Near GUI-Component) 4. User enters text as feedback. 5. User clicks save. 6. System saves the text. 7. System will display a feedback icon (with badge counter) next to GUI-comp.. 8. […] 9. A user while being on that form can click on the feedback icon to read it. 10. Any user can add a reply to the feedback, creating a conversation.
Alternative path:	a. User clicks on the [Add Feedback] button. b. System changes mouse indicator (for example) to indicate user can place feedback at desired position. c. User clicks at desired position. d. Continue from step 3. I. = step 9. II. User(author) clicks on the [edit] button to edit the feedback. III. The user alters the text. IV. The user clicks [Save]. V. The system adds "last updated with date/time stamp" to the feedback and saves the feedback.
Exceptional path:	i. = step 4. User enters text as feedback. ii. User clicks cancel. iii. System asks "Sure you want to cancel and remove your feedback?"

	iv. User clicks either [Remove feedback] or [Save feedback][20] or [Continue writing feedback]. v. System discards or saves feedback or closes prompt depending on choice above.
Extension points:	Extension A: In step 3 the system provides also a button to record an audio message. Extension B: In step 3 the system also provides a button to add a drawing. (Might become useful on touch capable devices.) Extension C: During step 4 the system auto-saves the typed text every 30 #TBD seconds. Extension D: Request a new component: new use case.
Assumptions:	
Post-conditions:	Post 1: Feedback is saved to #TDB.
Related BR	
Author:	Joeri Arendsen
Created on:	Spring 2015
Version:	2
Last updated on:	

[20] Please do not ask the user: “Are you sure you do not want to save the feedback?” with only a [Yes] or [No]. As a ‘yes’ or ‘no’ are ambiguous. Please refer to the new prompts of for example Office 2013 making it explicit what a button does. If anyone has to count the number of negations in the sentence and calculate what yes or no means, this might result in unintended loss of text.

5.1.4 Use Case: Request new GUI component with Feedback form

Use case:	**Request new GUI component with Feedback form**
Source: Trace:	Suggestion by Joeri, needs to be verified by Edwin [Yes, June 5, Priority:4] (RQ2(SpecificIntelliGUI).ReqExtraction.IdeaJoeri:AffordanceDetection:QuestionList & UseCase(Send feedback).LibrarySession(Nikos) & SPADE.endresulttargeted.questions).Combine.IdeaJoeri.UCsuggestion & Session(June5).FeedbackbyEdwin("For superuser to CGI")
Status:	Tier 2 use case. Extrapolated, verified by Edwin.
Actors:	Super-GUI-user
Trigger:	A GUI-user desires to request a new GUI component
Short description:	
Pre-conditions:	(Ribbon is activated.)
Basic course of Events:	1. User clicks on [Request new GUI component] (for instance, on the feedback tab on the ribbon.) 2. System displays the relevant form questions (for example by expanding a new part bellow the ribbon.) 3. User answers or enters: a. If the GUI-component should only display data or edit data. b. Short description of new required GUI component. c. What the source of the data is. (Actual real source data names) d. Any attachments (picture of real life object; affordance!) e. Quick draw. 4. System auto saves for later (for instance) when changing focus of field. 5. User clicks [Save & Send Request]. 6. System saves form and sends e-mail. (For instance to CGI-techie.)
Alternative path:	a. = step 5. System auto saves for later when changing field b. User clicks [Save draft] c. System displays "Last saved on [date and time stamp] near save button. d. User clicks [Close]. e. System hides the form. f. […] g. User clicks on [Request new GUI component] h. System expands the ribbon (step 4) and loads saved data. i. User continues entering information. j. From here step 4. i. = step 5. System auto saves for later when changing focus of field. ii. User clicks [Cancel changes]. iii. System prompts "Are you sure you want to discard any changes?". iv. User clicks [Discard all my changes] or [Save changes] or [Continue working on request]. v. System discards any feedback, saves changes and closes form or closes the prompt leaving the form open, depending on choice user.
Exceptional path:	
Extension points:	
Assumptions:	The super GUI user is able to interpret the source names of data he can select.
Post-conditions:	System has saved and has send the new component request (for instance) via e-mail to (for instance) a CGI-techie.
Related BR	

Author:	Joeri Arendsen
Created:	Spring 2015
Version:	2
Last updated on:	

5.1.5 Use Case: Add GUI-component based on selected parts on the screen

Use case:	**Add GUI-component based on selected parts on the screen.**
Source: Trace:	Edwin, interpretation needs to be verified. [Yes, June 5, Priority: 1] RibbonSession.ExamplebyEdwin.InterpretationbyJoeri.FeedbackbyEdwin
Status:	Tier 1 use case. Verified by Edwin.
Actors:	
Trigger:	User desires to combine to data groups together into a new visualisation.
Short description:	
Pre-conditions:	(Ribbon is activated, cross component(subpart) selectors are visible.)
Basic course of Events:	1. User selects (parts of) GUI-components (for example with help of selector checkboxes per part.) 2. User clicks [Combine selection to new component] (for instance, on the context ribbon tab). 3. System only displays compatible GUI-components with selection. 4. User clicks on desired combination GUI-component. 5. System will add GUI-component and change selection to that component. 6. (For instance, system will show context-ribbon to manipulate properties.) 7. User updates properties (for instance, on the ribbon). 8. System updates GUI-components and saves changes.
Alternative path:	
Exceptional path:	
Extension points:	
Assumptions:	
Post-conditions:	New GUI-component is saved to (Edwin:) form and user combination.
Related business rules	
Related requirements:	
Author:	Joeri Arendsen
Created:	Spring 2015
Version:	2
Last updated on:	

5.1.6 Use Case: Change visualisation (or property) of GUI-component.

Use case:	**Change visualisation (or property) of GUI-component.**
Source: Trace:	Edwin, interpretation to be verified by Edwin [Yes, June 5, Priority:1]. PrimaryRequirement.ExamplebyEdwin.InterpretationbyJoeri & UseCase(“Edit order of GUI-components”).VariationbyJoeri.UCsuggestion .FeedbackbyEdwin
Status:	Tier 1 use case, verified by Edwin.
Actors:	GUI-user
Trigger:	User desires to change the visualisation of field on the form.
Short description:	A GUI-component can be displayed differently
Pre-conditions:	(Ribbon is activated.)
Basic course of Events:	1. User selects (parts of) GUI-component(s). 2. System ~~greys-out or~~ hides incompatible visualisations (on context-ribbon.) 3. User clicks on alternative visualisation. 4. System updates GUI-component to new visualisation. 5. (System will show context-ribbon to manipulate properties.) 6. User updates properties. 7. System updates GUI-component and saves changes. 8. System displays: “Last saved on: _” with date and time stamp.
Alternative path:	a. = 6. User updates properties. b. User selects [Save changes for every user]. c. System prompts “This will overwrite any changes of other users made to this component. Do you want to continue?” d. System displays “Last saved on” with date and time stamp. e. Continue from step 9.
Exceptional path:	i. (During any step user decides to revert to default.) ii. User selects GUI-component. iii. System displays context-ribbon to manipulate properties. iv. Users clicks [Revert GUI-component back to default]. v. System prompts: “Are you sure you want to revert to default visualization?” vi. User clicks [Revert to default, delete changes]. (Or [Keep changes].) vii. System removes additions. (Or only closes prompts.)
Extension points:	
Assumptions:	
Post-conditions:	New GUI-component is saved to combination of user and page OR saved to page if user selected so.
Related BR	
	F1
Author:	Joeri Arendsen
Created on:	Spring 2015
Version:	2
Last updated on:	

5.1.7 Use Case: Add tag to component to facilitate identical component

Use case:	**Add tag to component to facilitate identical component**
Source: Trace:	Suggestion by Joeri, needs be verified by Edwin [Yes, June 5, Priority: none]. RQ.[…]ProblemOf("Defining what is an identical GUI-component results in over- or under specifying thus double work or flipflop save behaviour") .PotentialSolution.UCsuggestionbyJoeri
Status:	Tier 2 use case
Actors:	GUI-user
Trigger:	
Short description:	A user wants to manage identical GUI-components more easier. By applying relevant GUI-components with the same tag, the user can mass-edit the components all at once.
Pre-conditions:	Ribbon is activated
Basic course of Events:	1. User selects GUI-component 2. User clicks [Add tag] 3. System shows available tags and [Create new tagname]. 4. User chooses existing tag or [Creates new tagname]. 5. System applies tag to GUI-component. A. User changes visualisation of GUI-component (or properties). B. User sets Save preference to [All Components with tag]. C. User selects tag. D. (System saves GUI preferences to components with identical tag) E. User navigate to other pages with component with same tag. F. System applies earlier saved preferences to those GUI-components with the same tag and the same compatibility type.
Alternative path:	a. (Heads up, this will effect… choose continue or not)
Exceptional path:	i. _
Extension points:	List of al GUI-component with same tag
Assumptions:	
Post-conditions:	Tags are saved to GUI-components.
Related BR	
	F1
Author:	Joeri Arendsen
Created on:	Spring 2015
Version:	1.1
Last updated on:	

5.2 Additional design recommendations

- In interaction, use explicit buttons instead of buttons that are relative towards to the question or in some manner ambiguous.
    - As described in the use cases, do not put “Yes” and “No” on the buttons, but label the buttons itself with what they do. (Simple example: [Save] or [Discard changes].)
- Universal lay-out in the ribbon. Start each tab with a ℹ button explaining what it does with text or video. The power of this idea is that people can rely on the Ribbon to tell them how the ribbon works if it is not explanatory enough already by the structure itself.
    - However you operationalize the help button, please do not put the resulting help-element (or page) behind a log-in wall.

5.3 Use case evaluation

The use cases primarily answered CGI’s need for a new GUI design. Ironically, when CGI was asked to prioritize the different use cases, the use cases describing functionality that addressed issues of SPADE’s development process were cut out or ranked at the bottom.

6. Conclusions & Discussion

Take note that this thesis acquired different types of results. This chapter will be a summary of all the presented results, varying from guidelines on a higher level, to specific suggestions on a more practical level. Guidelines are the result of chapter 3, some of them are put into practice in chapter 4 and 5. Chapter 4 delivers the designs for future development of SPADE; chapter 5 delivers a detailed description of the desired functionality hinting at the Ribbon design.

My fellow student Nikos Makris[21] and I have explored the possibility of extracting the user's desired GUI information before and after transformation and reviewed opportunities to communicate or use these desired GUI requirements. Separately we have explored different areas and created designs to solve a part of the problem. Please consult his work to acquire a full overview of his part of the project[21].

The earlier mentioned main research question:

How can a generic and specific Intelligent Graphical User Interface be incorporated in SPADE's framework?

The sub research question that has been the focus of this thesis:

What design for specific IntelliGUI is best suited for SPADE's process of software engineering?

6.1 Conclusion

Due to the nature of the SPADE's process of software engineering (automatic transformations and end-result orientation), the best suited design, aligned with the desires of CGI, is to allow the end-user to provide the system with information about what the GUI should look like at run-time, resulting in a system design realizing a GUI that can manipulate certain components of itself. Design guidelines[22] are provided on how to best construct such a GUI. Furthermore, designs[23] are presented to update the overall development process of which some are incorporated into the GUI design. These GUI guidelines are put into practice in the case study described in chapter 5. [24] The use cases describe very thoroughly the desired functionality and mock-ups provide an alternative form of functionality insight.

To continue, different ideas have been discussed in the previous chapters that occur before and after the transformation by the SPADE generator. Most of them require future work. This will be a summary of those presented ideas.

6.1.1 Presented solution paths before transformation

One of the ideas to get a desired GUI design was describing GUI components in a domain specific language (DSL) or in smart requirements level 4 itself. A superficial example of using the latter is provided in section 4.9 ("SPADE inception"). Despite the language that is used to specify the system (process) itself, it might not be suitable for describing visual GUI components. However, in case the

[21] For more information about Nikos Makris, see section 7.2 (p.49).
[22] Chapter 3: Creating Guidelines for Dynamic Interfaces (p.15).
[23] Section 4.3 Ribbon extensibility: Feedback form, 4.5 Simulation mode, 4.8 The questionnaire for clients (Starting from page 25).
[24] Chapter 5: Exemplifying the Dynamic IntelliGUI Design for SPADE (p.26).

client is trained in using the DSL, at least the language might be used for communicating a GUI component idea from human to human. Furthermore, because the language forces the creator to be specific, this aids the human component builder in understanding the creator. Future research should determine if this language is even suitable to describe GUI components and potentially provide an extension on the existing notation. This idea branch was abandoned, because of more practical solutions. Describing a GUI component in text (or code) is still hard for the majority of users. However, fellow student Nikos Makris experimented with a DSL language describing the behaviour of a dynamic GUI in the form of rules. Please consult his work to read more about this topic (section 7.2).

Another idea was presented in section 4.8, discussing the Questionnaire strategy where SPADE's workflow system itself is used to identify valuable information interface requirements. This idea branch was also abandoned because there were other idea branches with more practical value. The idea branch on how to manipulate GUI-components later in the process of development, provided a more practical opportunity, which will be discussed in the next section.

6.1.2 Presented solution paths after the transformation:

The idea branch describing the Ribbon with all its extensions like the feedback tab and handlers provides CGI with a design that provides structure and yet still facilitates dynamic behaviour when this is required. The different design guidelines redirect attention to specific design decision points that have to be made. Like determining the required level of dynamic capability as well as the type of requirements asking for dynamic interface behaviour. Do the requirements ask for functionality that has to be realized before or after run-time? This separation needs to be addressed because the technical set-up influences what the system can and cannot do, but also what the time of development will be. Furthermore, the design guidelines addresses certain dangers that come with developing dynamic interfaces. For example, phrased poetically: *what to make dynamic and what not to make dynamic?* As research turned out, this question is not that easily answered as one might expect initially[25].

To put that in order, recommended is that a designer first determines the required level of dynamic interface capabilities. Secondly, determine technically in which stage (run-time or before; adaptability versus adaptation) the solution of dynamic behaviour will be realized. Requirements should be matched, this could be problematic, because certain requirements might require a sophisticated system with a certain level of dynamic capability while previously determined the level of dynamic capability is not required. This is exactly why one must do this, to align the *real* desired dynamic behaviour with all the technical consequences and vice versa. After this stage, thirdly, one can start designing a dynamic interface upholding and remembering all the guidelines helping the designer to create an interface that does not confuse or frustrate the user, but that actually helps the user in accomplishing the task at hand. The result should be improved usability for the end-user.

[25] As explained in section 3.3 Interface Adaptation 101 (p.19).

6.2 Discussion

6.2.1 General discussion about main issues

When looking at the overall picture of SPADE's process of software development and remembering the Paradox of SPADE (section 1.5 p.6), that explained the essence of the problem, the transformation process itself limits knowing more about the desired GUI. One can either try to identify valuable desired GUI information from end-users before or after the transformation. As explained in section 4.8, one can circumvent the problem of not knowing this GUI information and provide the end-user with the capability to adapt the interface on-the-fly. Though this is the best design for SPADE's system of automated software generation, it does not mean there are no other solutions to the described problem. A transcompiler, first absolves the user from divulging certain detailed information due to deductive capabilities and next lacking the capability to take into account certain aspects of human limitations[26] (Wupper, 2012) in system interaction and context specificity. Other solutions have been briefly explored, however, not with the same determination as the dynamic interface solution path, because the company CGI provided a directed question. (Not "if", but "how".) Future work should determine if a dynamic interface is the answer to limited transcompilers[27] or that an adaptation of either the transcompiler itself or the software development process is the best solution. This thesis surveyed all of these areas; though the level of depth differ.

Once on board on the solution path for a dynamic interface, the internal validity of this thesis' idea branches and designs have been supported by the case study that has been performed at CGI. The created use cases are the result of several sessions with the developers at CGI working with the automated software generation tool SPADE every day. The applicability of these designs to other automated software generations tools are unknown at the moment. However, the designs (and guidelines) are applicable to anyone developing a dynamic graphical user interface.

6.2.2 Discussion about acceptance of designs

In the Acceptance sections of chapter 4 and section 5.3, the acceptance by CGI of the ideas or designs were briefly discussed. This section tries to put those sections in perspective. Why were some designs denied or put on hold whereas other designs accepted? The answer is relatively simple. The developers of the SPADE software development tool are the same people who are hired to build software at other companies with the SPADE tool. In a nutshell: time and resources are limited. The use cases expecting to add value quickly, were accepted. The use cases not expecting to add value quickly or seemed very labour intensive, were not accepted. Some of the use cases (or designs) were built upon unproven ideas, which makes them even harder to accept, like incorporating a scouting mechanism[28] or a feedback mechanisms[29] in the framework itself, actually becoming part of the software development process. This is new territory, or at least, it is in the SPADE development process, resulting in several starting points for future research.

After the Ribbon is implemented, the developers will most likely reprioritize and determine if the designs and use cases with low priority are worth it. I do expect (and advise) the developers to re-evaluate the extension points in their software, especially when they are provided with time and resources to work on their own software development process.

[26] Referring to a concept in the context of architecture, in Dutch: "De menselijke maat" literally translated to: "Human size" or "Human dimensions", the intended meaning is related to usability.
[27] In the context of automated software generation.
[28] Section 4.8: The questionnaire for clients (p.32)
[29] Section 4.3: Ribbon extensibility: Feedback form (p.25) and 4.5: Simulation mode (p.28).

7. References

7.1 Literature

Academic Peer Reviewed

Akoumianakis, D., Savidis, A., & Stephanidis, C. (2000). Encapsulating intelligent interactive behaviour in unified user interface artefacts. *Interacting with Computers, 12*(4), 383-408.

Criado, J., Iribarne, L., Padilla, N., Troya, J., & Vallecillo, A. (2011). *Adapting Component-based User Interfaces at Runtime using Observers*. Paper presented at the Conference on Software Engineering and Databases, A Coruña, Spain.

Jameson, A. (2009). Adaptive interfaces and agents. *Human-Computer Interaction: Design Issues, Solutions, and Applications, 105*.

Kabassi, K., Despotis, D. K., & Virvou, M. (2005). A multicriteria approach to dynamic reasoning in an intelligent user interface. *International Journal of Information Technology & Decision Making, 4*(01), 21-34.

Karuzaki, E., & Savidis, A. (2014). *Consolidating diverse user profiles based on the profile models of adaptive systems*. Paper presented at the Proceedings of the 2014 ACM SIGCHI symposium on Engineering interactive computing systems, Rome, Italy.

Kramer, D., Oussena, S., Komisarczuk, P., & Clark, T. (2013). Using document-oriented GUIs in dynamic software product lines. 85-94. doi:10.1145/2517208.2517214

Kumar, S., & Sekmen, A. (2008). Single robot – Multiple human interaction via intelligent user interfaces. *Knowledge-Based Systems, 21*(6), 458-465. doi:http://dx.doi.org/10.1016/j.knosys.2008.03.008

Macik, M., Cerny, T., & Slavik, P. (2014). Context-sensitive, cross-platform user interface generation. *Journal on Multimodal User Interfaces, 8*(2), 217-229.

Nilsson, E. G., Floch, J., Hallsteinsen, S., & Stav, E. (2006). Model-based user interface adaptation. *Computers & Graphics, 30*(5), 692-701. doi:http://dx.doi.org/10.1016/j.cag.2006.07.003

Paskalev, P., & Serafimova, I. (2011). *Rule based framework for intelligent GUI adaptation*. Paper presented at the Proceedings of the 12th International Conference on Computer Systems and Technologies, Vienna, Austria.

Savidis, A., & Stephanidis, C. (2010). *Software refactoring process for adaptive user-interface composition*. Paper presented at the Proceedings of the 2nd ACM SIGCHI symposium on Engineering interactive computing systems, Berlin, Germany.

Schaefer, R. (2007). A Survey on Transformation Tools for Model Based User Interface Development. In J. Jacko (Ed.), *Human-Computer Interaction. Interaction Design and Usability* (Vol. 4550, pp. 1178-1187): Springer Berlin Heidelberg.

Sottet, J.-S., Ganneau, V., Calvary, G., Coutaz, J., Demeure, A., Favre, J.-M., & Demumieux, R. (2007). *Model-driven adaptation for plastic user interfaces.* Paper presented at the Proceedings of the 11th IFIP TC 13 international conference on Human-computer interaction, Rio de Janeiro, Brazil.

Stephanidis, C., Paramythis, A., Karagiannidis, C., & Savidis, A. (1997). *Supporting interface adaptation: the AVANTI Web-Browser.* Paper presented at the Proceedings of the 3rd ERCIM Workshop on User Interfaces for All, Obernai, France.

Other Sources

Arendsen, J., Kuznetcov, M., Jadidian, F. S., & Gadjieva, N. (2014). Brainotica (pp. 24). Nijmegen.

Aydinli, Ö., Hendriks, E., & Zandvliet, J. (2013). *SMART Requirements 2.0*. Den Haag: Sdu Uitgevers BV.

Kulak, D., & Guiney, E. (2008). Use Cases: Requirements in Context. Boston: Addison-Wesley.

Wupper, H. (2012). *Architectuur voor de digitale wereld!* (1 ed.). Berlin: epubli GmbH.

7.2 Parallel Thesis

Because the two theses are written in parallel, a reference to the final work of Nikos Makris cannot be made at the time of writing. Therefore, a reference to the work of Nikos Makris can be found here, which can be updated in the future:
http://linkbox.zapto.org/hub/redirect_thesis_Makris.html

Please be aware that our theses partly overlap and will have similar topics. Though we had our own focus points, we have worked together on the project numerous times at CGI in Arnhem and in Nijmegen at Radboud University.

7.3 Latest version

Current document version: 2015_Thesis_v1.01.docx

The latest version of this document can always be found here:

Mirror 1: http://linkbox.zapto.org/hub/redirect_thesis_Arendsen.html
Mirror 2: https://www.screencast.com/t/90I0Bpaw3i

7.4 Contact information

Joeri Arendsen (s3033066)
joeriarendsen@student.ru.nl

Prof. dr. Marko C.J.D. van Eekelen
Contact information at:
http://www.cs.ru.nl/~marko/

Edwin Hendriks
edwin.hendriks@cgi.com