Generating Data from Highly Flexible and Individual Process Settings through a Game-based Experimentation Service

Georg Kaes[1], Stefanie Rinderle-Ma[2]

Abstract: The ability to adapt process instances to changing requirements has long been recognized as a fundamental research topic. While in some settings, process flexibility is only required in exceptional situations, in other settings it is the key component which drives the process design. Examples can be found in multiple domains, including the nursing domain, where each patient requires his own, individual therapy process which may change on a regular basis. In this paper, such flexible and individual process settings (FIPS) are analyzed and the basic building blocks are defined based on expert interviews and relevant literature. The building blocks are then mapped onto a game-based experimentation service which offers a simulation and evaluation environment for FIPS. The data generated in this game are evaluated by a comparison with data from a real world FIPS.

Keywords: process flexibility, process logs, process mining, experimentation service

1 Introduction

Process flexibility is fundamental to many domains and thus a well researched topic [RW12]. Depending on the process setting there are different degrees of flexibility [Sc08]. In some settings, multiple process instances follow the same basic schema which has to be adapted if some exceptional situation occurs. In other cases, each process instance is different by design. Take the therapy process of a patient in a nursing home as an example (cf. [Ka14]). For each patient there exists one long-running process instance executing the therapy steps specific to this patient. There is no common schema, but each instance develops based on ad hoc adaptations. Whenever a patient shows new symptoms, the nurse has to adapt the therapy process accordingly. Such settings are referred to by *highly Flexible and Individual Process Settings (FIPS).*

FIPS can be found in many domains [Ka14]. Examples range from regular customer care in a hotel setting, over the medical domain, to schools for children with special needs. In FIPS, end users are regularly supposed to decide on, specify, and conduct adaptations [We09]. Hence, system support becomes crucial [KR15]. In the nursing domain, for example, nurses could be supported in finding the best adaptation for the current patient's therapy process based on his current symptoms and medical history [Ka14]. Existing approaches on user support for process flexibility [Aa09; Gü08; KR15; We09] exploit information about previous process executions and adaptations, typically captured in process *execution* and

[1] University of Vienna, Faculty of Computer Science, Waehringerstrasse 29, 1090 Vienna, Austria, georg.kaes@univie.ac.at
[2] University of Vienna, Faculty of Computer Science, Waehringerstrasse 29, 1090 Vienna, Austria, stefanie.rinderle-ma@univie.ac.at

change logs. The goal is to learn from previously applied changes and to utilize additional knowledge such as context information, e.g., patient age and medical history, to identify the best adaption for the situation at hand.

Providing system-based user support requires to understand building blocks that are characteristic to FIPS independent of the particular domain. In [Ka14] requirements for highly flexible process settings have been collected from four different application domains, i.e., *subject data*, *environmental data*, *goals*, *trigger conditions*, and *process fragments*. The first question is whether these requirements can be generalized as building blocks for FIPS. A follow-up question is whether the building blocks can be mapped and integrated into an experimentation service. The goal of such a service would be to enable the simulation of FIPS independent of any application domain. Such an experimentation service would enable the simulation of FIPS with a holistic view on data. It would also enable the creation of process execution and change logs in case no real-world logs are available due to, for example, legal or privacy reasons.

Overall, this paper tackles the following research questions:

Q1 Which building blocks are common to FIPS (following up on [Ka14])?

Q2 How to reflect FIPS requirements in a game-based experimentation service?

Q3 Is a game-based design suitable for generating the data common to a FIPS?

The research questions are tackled by elaborating a service-based design of a tower defense game that simulates FIPS. In a tower defense game, players defend towers from incoming enemies. To do this, players can select from multiple defense systems which can be placed at the towers. These defenses then fight the incoming enemies. At an abstract level, players conduct comparable actions to nurses in a nursing home: They plan individual process adaptions in order to deal with a certain problem. In the nursing home, these problems are the symptoms a patient shows; in the tower defense game, they are the incoming enemies. While playing the game, process change and execution logs and related data for analyzing the logs are logged in a data repository. Additionally, the tower defense game itself provides a setting which can be used to evaluate approaches which support users in the sequel.

The paper is structured as follows: In Section 2, building blocks and concepts of a FIPS are analyzed based on expert interviews and explained including data sources and components which have to be represented in the tower defense game ($\rightarrow$ Q1). Section 3 describes the design of the service-based tower defense game: Here, we present the mapping of FIPS requirements, data sources, and components to the concepts of the tower defense game ($\rightarrow$ Q2). Further on, the service-based architecture of the game and details of the generated log files are presented. The feasibility of the data generated by the game is evaluated by comparing the generated data elements with data generated by a real world FIPS ($\rightarrow$ Q2, Q3). Finally, Section 5 discusses related work and Section 6 concludes the paper.

2 Generalization of Highly Flexible and Individual Process Settings

FIPS can be found in many domains. As shown in previous work [Ka14], the basic properties of such settings are quite similar. Following up on [Ka14] and considering further work on flexible processes [RW12; We09], we conducted several expert interviews with practitioners from different domains which we identified as possible FIPS. Our goal was to enhance the basic building blocks we found in [Ka14] and to identify basic properties of FIPS. In this section, we first present the resulting building blocks (Section 2.1), followed by the outcome of the expert interviews and individual properties of FIPS (Section 2.2).

2.1 FIPS Building Blocks

Based on related work [RW12; We09] and the results from the expert interviews, the following requirements are considered to build the basis for a general FIPS representation ($\rightarrow$ Q1). These building blocks are domain independent abstractions from general concepts such as data sources or procedures which exist in specific domains which can be seen as FIPS. We found implementations of these concepts in any domain we investigated which we considered as a FIPS. The rest of the section defines these building blocks and gives examples from the nursing domain. Following, the domains will be described in detail and domain specific implementations of these concepts will be presented.

- **Subjects** define the individual persons, objects or concepts that are subject to the process execution [Ka14]. In the nursing domain, these individuals are the patients for which the process instance exists and is being adapted.
- **Process Instances** [RW12] capture and execute the process logic for a subject. In FIPS, instances typically run for a long time and are affected by process change operations whenever necessary. In the nursing domain, the process instances contain the therapy tasks which have to be executed in order to make the patient feel good or better.
- **Organizational Units** reflect the organizational perspective of a process. They can be used to define relationships between actors, define skills and possible actions for those units etc. [RW12]. In the nursing domain, the organizational units are the nurses, doctors, assistant nurses, and all other employees of the nursing home which are in some way related to a patient's therapy plan.
- The **Environment** [Ka14] of a FIPS comprises all data that is related to the subject, but not directly incorporated by the subject, for example, the limited resources which are required to execute the tasks related to the subject. In the nursing domain, the environment would be the nursing home.
- **Process Fragments** are parts of a process which get inserted into, deleted from, or moved during a process change operation [Ka14; RW12]. In the nursing domain, these process fragments refer to therapies which are added to or removed from a patient's therapy plan.

- **Problem List:** For each fragment a set of conditions where the fragment should not be used can be defined. In [We09], such a problem list item has been introduced in an example from a medical domain, where a certain adaptation could not be made because the patient has a cardiatric pacemaker.
- **Bonus List:** In contrast to the problem list, the bonus list for each fragment describes situations where the fragment is known to perform well. This building block has been identified during the expert interviews. In the nursing domain, this bonus list contains details about the patient which give a hint that a certain therapy may perform well.
- **Triggers:** In [RW12], the necessity of ad hoc modifications to a process instance is defined by exceptions which have not been thought of when designing the process model. Whenever such an exception occurs, the process instance has to be changed in order to deal with this exception. In FIPS, such events occur on a regular basis, thus we argue that the term *exception* does not fit in this context. Since these situations trigger a change operation, we will use the term *trigger* instead. In the nursing domain, symptoms are the triggers. Whenever a patient shows new symptoms, something has to be changed in his therapy process.
- **Positive Goals [KK97]:** There are always reasons which justify a process change. These reasons can be defined as *goals* which should be reached by executing the tasks which have been inserted, or not executing the tasks which have been removed from the process instance. In the nursing domain, positive goals can be to make the patient feel better, more confident, or at least to stabilize him in his current condition.
- **Negative Goals [KK97]:** Process changes can also be conducted in order to avoid certain situations and hence they address negative goals. In the nursing domain, examples include the deterioration of the patient's condition or even the death of the patient.

2.2 Expert Interviews

We conducted interviews with experts from different domains in order to identify the basic building blocks of a FIPS presented in the last section. Highly individual settings can be considered as FIPS if they require some kind of process to be adapted in order to react properly to certain situations. In [Ka14] the basic high-level components for FIPS were analyzed for four domains, i.e. *manufacturing*, *software development*, *hotel and event management* and *care planning*. For evaluating the building blocks and general properties of a FIPS, we chose *software development* and *hotel and event management* as domains already evaluated in [Ka14]. In order to emphasize the diversity of settings where FIPS plays a key role, we decided to evaluate two additional domains, namely a *special needs school* and the *PhD program* of a university.

The interviews started with the experts describing the domain they are working in. We asked them for examples where they had to deal with special individual situations, and what they could personally learn from these situations. In the following, we presented them the basic idea of FIPS based on the nursing scenario and discussed possible commonalities between their domain and the nursing domain as a FIPS. We chose the nursing domain as a reference, since it is easy to empathize with, even if the interviewed person is no expert in this domain. Based on this information, we identified in cooperation with the domain experts the basic building blocks which have been presented in the last section. The exact allocation of each building block for each evaluated domain can be found on our project website[3].

2.2.1 Expert Interview: Software Sales and Support

We interviewed two sales managers from mesonic, an austrian company which develops enterprise resource planning (ERP) and customer relationship management (CRM) software for small and medium enterprises. The contact to the customers works to a big part over a network of retailers which are in geographical vicinity of their customers. The support of mesonic mainly works over an internal system where problems with the software itself (e.g., bug reports), individual wishes, and additional requirements from certain customers and retailers are gathered. Depending on the information the company's support department has about the case at hand, different measures are taken: If a bug is identified (which is already known or reported by many other customers), it will be discussed in the next developer meeting, where further steps will be defined. Depending on its priority, it may be solved right away, or as soon as resources are free. A customer's whish for an additional feature may be implemented in a future version of the product, depending on the workload of the developers, the usefulness to other customers, and several other parameters.

Discussion: After introducing the nursing domain as a point of reference, we talked about commonalities between the software sales and support and the nursing domain on an abstract level. It became clear that commonalities between the individual problems of a patient in the nursing home and the problem description in a support case exist indeed: In both scenarios, there is a problem which has to be solved. In order to identify the best course of action (i.e. the best process fragment), parameters from the case itself as well as from the environment are analyzed. While in the nursing home the circumstances of the patient's problem are analyzed, including his medical history and common diseases, in the software domain the circumstances of the support case are identified, including the targeted version of the software and other, similar reported problems. When the problem itself is identified and the goals are clear, the next steps are planned. In the software domain these steps include work to be done by software developers, the support and sales departments. After these steps have been conducted the person in charge of the case evaluates whether the goals have been reached or not.

2.2.2 Expert Interview: Hotel Management and Guest Care

Hotel management also deals with individual subjects (i.e., guests) having specific requirements. We interviewed two receptionists from a local hotel and seminar location which

[3] http://cs.univie.ac.at/project/apes

focuses on business as well as leisure guests. The guest profiles range from typical seminar groups with a trainer and multiple employees or managers, to couples or families who want to spend some days in a hotel. Obviously, these groups have different requirements for the location: While for the leisure guests, for example, outdoor activities or wellness offers have a higher priority, for the business guest a well-equipped seminar location or a smooth procedure of their workshop is more important. Hence, the hotel has to provide different offers depending on the guests' profiles and demands.

Discussion: Each interaction with a guest can be seen as part of a FIPS. Depending on the guest profile and the current offers of the hotel and seminar location, different things can be offered to the guests. For example, if a guest is known to be quite exhausting, he will be treated with special care in order to keep him at bay. A seminar group who is known to have problems with technical equipment will receive special support right from the start. This behavior again shows commonalities to the nursing or the software development domain: First, the situation and the problem are identified, then, based on knowledge about the subject at hand and what has worked in a similar situation before, the best course of action is identified.

2.2.3 Expert Interview: Special Needs School

We interviewed two teachers from a special needs school who teach children with special needs from ages 10 to 15. The special needs range from language issues, over physical up to psychical conditions of various kinds. Depending on the specific needs of the children different teaching concepts have to be chosen. While there exists a general plan for the school year as a whole, it has to be presented to each child in a different way. Of course, due to resource limitations, this is not always possible. Hence, the teachers are required to balance the needs of all children.

Whenever a child shows some ad hoc needs, for example his or her concentration is going down, the teachers have to find the root causes for this symptom, and find a way to deal with it. Depending on the child and his or her needs, different causes can lead to the same symptoms. For example, learning problems may indicate family or physical issues. It is the teacher's job to find countermeasures which will help the child to feel better and more confident again. These countermeasures differ from child to child, and a great deal of experience is required to find a the best one.

Discussion: In this scenario, the parallels to the nursing domain and to FIPS in general were obvious: Just as patients who have their individual problems, children with special needs also require attention for their individual needs. If a child shows some symptoms (as described above) it is the teacher's job to identify the problem and to find an effective way of dealing with it. If the teacher is experienced in his field of work, he had similar cases before, and uses this knowledge in order to find the best way to help the child.

2.2.4 Expert Interview: Doctoral Program Supervision

We interviewed two participants in the doctoral program of the University of Vienna. During the interviews it became clear that the doctoral program can also be seen as a FIPS. While all students have their individual project, there are some common characteristics between the different students and projects which can be utilized to analyze problems and find solutions. The basic research plan is typically laid out during the first year of the PhD studies, but it has to be adapted many times during the course of the work, whenever issues or new ideas come up. For example, if a student realizes he or she requires some special equipment which is hard to get, he or she may have to adapt his or her work plan. There might be different solutions for the same challenge. For example, if a certain kind of technical equipment, like a special microscope, is required to complete a part of the work, for some projects it may be it easier to adapt the work in a way that this equipment is no longer needed. For other projects, the students may have contacts to institutions which can provide such a microscope.

Discussion: During the interviews it became clear that the doctoral program can also be seen as a FIPS. In contrast to other domains however, each student only has one individual project (i.e. one subject in terms of a FIPS) he is working with. While in the other three domains people were identifying and handling problems for several different subjects (e.g. guests, support cases, children) each student only deals with one subject.

Conclusion: Altogether 8 experts from 4 different domains were interviewed. All of these experts were able to identify some commonalities between the domains they are working in, and a FIPS. Together with the domain experts, we identified the basic building blocks of a FIPS which we discussed in the last section. For each of the four domains described above, these building blocks have been discussed in detail and allocated with data from each domain.

In addition to the building blocks which are based on related work, and the *bonus list* identified during the expert interviews, the following special properties of a FIPS were encountered while analyzing these basic building blocks with the domain experts (c.f. Section 2.2.) The following list defines these properties:

- **Individualism:** Each subject, and each situation where a trigger occurs, has its very individual properties. For example, in the software development domain, each support case has its own set of data elements which describe the situation and the case itself.
- **Limitation of available resources:** In each setting, the resources which can be used to solve the problematic situation are limited. This usually leads to a prioritization of the problems: For those which are more important, more resources are spent. In the software domain, a crucial support case naturally receives more resources in order to solve it. On the other hand, in the special needs school setting, such a prioritization is not that simple: Every child needs attention, and the available resources have to be split up in a good way.

- **Ambiguity of triggers:** The same trigger does not have to mean that the same problem has to be solved: In the special needs school, two children can show the same symptoms (e.g. they cannot concentrate any more), but the reasons behind those symptoms can be very different.
- **Diversity of possible solutions:** If we know what the actual problem is, still multiple solutions are possible. It highly depends on the current situation which one is chosen, i.e. the workload of the available resources, the history of the individual etc. In the special needs school it depends to a big part on the condition of the child which solutions can be chosen in order to make him or her feel better.

3 Designing a Tower Defense Game as an Experimentation Service for FIPS

The main goal of this work is to design a game-based experimentation service that reflects the building blocks introduced in Section 2 and enables the simulation of FIPS in a generalized way (→ Q2, Q3). Specifically, the service enables the collection of data that is essential for user support features [Gü08; KR15; We09], i.e., process execution and change logs.

The following section first discusses why we decided to implement a tower defense game as an experimentation service for FIPS. Following, we introduce the basic ideas and gameplay. After that it is explained how this gameplay relates to FIPS building blocks and procedures. Finally, the basic architecture are shown and we introduce the data players generate while playing the game.

3.1 Using a Tower Defense Game as an Experimentation Service

Tower defense games (cf. e.g. [Av11]) are strategy games where the player has to defend a set of towers. Usually, the player has some defense systems to choose from (build a cannon, train some soldiers, etc.) and sends them to the towers. In certain intervals, waves of enemies attack these towers and try to destroy them. By placing his defense systems, the player tries to counter the enemy forces and avoid his towers from being destroyed.

We chose a tower defense game as a basis for our evaluation service for several reasons: First, tower defense games are a well known type of game, thus, the basic game mechanics will be quite easy to understand for new players. Second, tower defense games can be round based. This means that the player can take as much time as he requires to make his decisions. This stands in contrast to real-time strategy games, where the player has to make his decisions very fast in order to keep up with the game. A very popular representative of a round based strategy game is chess: Here, both players can usually take a long time to decide what to do next. Since the goal of our game is to represent a FIPS, we decided to implement a round based strategy game: As in a real FIPS, where the responsible actors have to plan their next steps, the player should have enough time to plan what to do next. Third, a tower defense game does not depend on a real domain: For most FIPS, the person responsible for deciding what to do next has to be a domain expert. Depending on the domain, it can take a long time to acquire the required knowledge. In a domain independent setting such as a tower defense game, no long training is required: Of course, it takes some time for new players to know exactly when to use which defense system, but in contrast to learning the details of nursing science, this can be done very fast. Fourth, the domain independence also leads to the advantage that anyone interested can download and play our game. Thus, we have a broad base of possible players who generate the data we can use for further evaluation.

Aside from the players who actually play the game, our targeted audience are scientists: First, the generated data can be used to develop and evaluate algorithms work with FIPS data. Second, using our game as an experimentation service, scientists can evaluate approaches to support users in FIPS domain independently. Imagine a service which aims at supporting FIPS users. Our game service could be utilized to compare two groups of FIPS users: Those who receive some kind of support when making decisions, and those who do not receive any support. If the supported group outperforms the group who did not receive support, it can be a hint that the support service might be useful. Due to the domain independence of our game service, a broad range of users can be included in such an experiment.

3.2 Gameplay

The idea of the game to be developed[4] is that the player slips into the role of a commander who has to conquer and defend villages on an island. (The villages are equivalent to the towers in the definition given above.) Each of these villages has its own set of parameters which make them individual. While some villages are closer to the sea, others are in the middle of the woods, and others close to mountains. These villages are attacked by enemy armies in varying intervals. Who these armies are and how they attack largely depends on the parameters and the history of the villages. Some enemies will preferably attack villages closer to the sea, while other enemies are more often seen in the woods and will preferably plan their attacks there.

The game is round based, meaning that the player can take as much time for his decisions as he needs. In the game each round is referred to as a day. In order to defend his villages, the player can plan the defenses for the days to come. For each village, there exists exactly one defense plan, which is implemented as a highly flexible and individual process instance. In this process instance, all the actions which have to be carried out in order to defend the village are defined. On each day, the same things happen:

1. New enemies show up at some of the villages.
2. The players defenses which are already in place (because they have been planned on previous days) fight the enemies. If the enemy wins, the village loses population. If the population reaches zero, the village is lost.
3. The player can plan new defenses for the following days for each of his villages.

[4] Available at: http://cs.univie.ac.at/project/apes

The enemy armies attack the villages over multiple days. On the first day, they usually show up with light units who do not do much damage, but the player already notices them. The longer the player does not react properly to the threat, the stronger the enemy army gets; ultimately destroying the village.

The player controls four different buildings which cannot be attacked by an enemy directly. Their sole purpose is to produce different parts relevant for defending the villages. All buildings have a maximum number of items they can produce on a single day. If their workload is fully used, they cannot produce any more on that day.

- **Barracks:** In the barracks, warriors of different types are trained. Among others, the player can choose between a knight with a lot of health points, an archer who does additional ranged damage, a mage who is a proficient spellcaster, and a soldier who does melee damage.
- **Forge:** A warrior cannot do much without a weapon. In the forge, the player can create multiple weapons such as swords, maces, axes, polearms, longbows etc. Each of these weapons has individual properties, which make them more or less effective depending on the warrior who carries them.
- **Mage Tower:** In the mage tower the player can create spells which are cast by the defense units either to protect themselves or to harm their enemies.
- **Alchemy Lab:** In the alchemy lab the player can prepare oils which add an additional effect to the weapons, and thus enhance their effectivity.

Each time a new enemy army shows up at a village for which the player has not planned any defenses yet, he is supposed to do the following steps:

1. **Choose Warriors:** In a first step, the player chooses the warriors which make up his army. These warriors are from one of three different races (human, elf and dwarf) which all have advantages and disadvantages depending on the terrain of the village and the enemy they are facing. The player can choose as many warriors as he wants for each day, as long as the total number of warriors doesn't exceed the maximum workload for the barracks for this day.
2. **Prepare the Army:** Before sending his army into battle, the player has to allocate the weapons and spells to his warriors. Basically, all infantry can carry all kinds of weapons and spells, but some will perform better with certain types of weapons than others. For example, an archer will do more damage using a longbow than a mage would do with the same weapon. Also, the effectiveness of the weapons depends on the enemy the player is facing: Some weapons will perform good against certain races, while others will not yield good results.
3. **Execute Adaptation:** When the player has set up his army for the following days, he adapts the village's process instance and sends his troops to the village which is attacked.

4. **Evaluate Results:** Each day, the enemy armies will first fight the defending armies and - if they should destroy the defenses - inflict a certain amount of damage to the village by killing its population. If the village's population reaches zero, the village is lost for the remainder of the game. If the player has lost all his villages, the game is over.

Fig. 1 shows the interface for choosing the troops and units for the defense of a village called Merasus in the game. The player can select for each day which soldiers, weapons, and defense systems he wants to use for his defense.

Magical Alignment: The spells and oils which can be created in the alchemy lab have a certain magical alignment. There are three axes of magical alignment: **Heat vs Cold**, **Order vs Chaos**, and **Nature vs Corruption**. Aside from the topology of a village, magical alignment is the second concept which creates individual situations. Each time a spell of a certain alignment is being cast at a village (either in order to defend or attack the village), the alignment of this village shifts into the direction of the spell's alignment. For example, if the player decides to send units who cast the spell *Fireball* to the battlefield (which is a *Heat* spell), the magical alignment of said village will slowly change to the *Heat* alignment, and away from the *Cold* alignment. Fig. 1 shows an overview over the magical alignment of a village. The magical alignment of a village influences how good or bad a spell or an oil of a certain alignment works. If the *Heat vs Cold* alignment of a village is more on the heat side, heat spells will generally be more effective, and cold spells won't work as well. Spells from the other alignments such as *Nature*, *Corruption*, *Order* or *Chaos* are not affected. Magical alignment adds a great deal of individualism to a village.

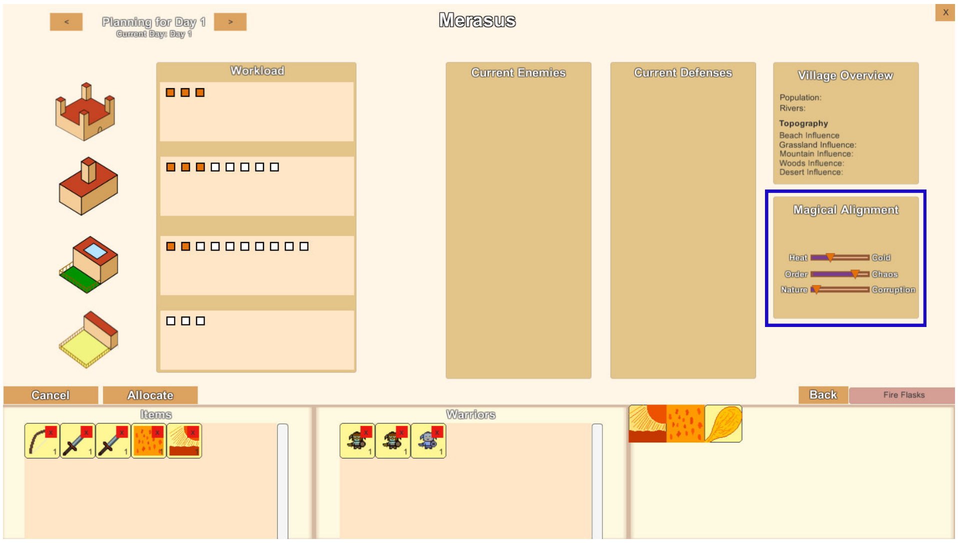


Fig. 1: Choosing the defense mechanisms; framed in blue: magical alignment

3.3 Mapping the Tower Defense Game Concepts to FIPS Buidling Blocks

In the following we describe how the gameplay is mapped onto FIPS.

- **Subjects** map to villages which differ according to their topology, magical alignment and their history of enemy attacks.
- A **process instance** maps onto exactly one defense plan for each village which should be adapted every time a new enemy army shows up.
- **Organizational units** map onto buildings under the player's command that carry out the steps defined in the process instance, namely the barracks, the forge, the alchemy lab and the mage tower.
- An **environment** subsumes all resources and organizational units that have to be shared for the execution of a certain process, for example, a nursing home. In the tower defense game, all production buildings have to be shared for the set of villages in the game. Hence, each game forms its own environment.
- **Process fragments** map onto the defense plans a player creates in order to stop the enemy army. These defense plans / process fragments are inserted into the village's defense process instance.
- **Problem list:** For each fragment which can be inserted into a process instance there exist some situations where it will not be optimal to execute its steps, for example if the magical alignment of the village is contradicting the spells' alignment.
- **Bonus list:** While some spells may be problematic in some situations, they may perform well in others. If a village's magical alignment is the same as the spells in a fragment, carrying out the adaptation may yield more promising results.
- **Triggers:** Every time a new enemy army shows up at a village, its defense process instance should be adapted.
- **Positive goals:** If the enemy is destroyed or chased away, the battle is won.
- **Negative goals:** If the village loses population or too many resources have been used, the battle may not have found its optimal outcome.

- **Individualism:** Each village has its own, individual properies. This includes the terrain, number of rivers, and magical alignment.
- **Limitation of available resources:** The buildings can only produce a limited amount of soldiers and weapons to defeat the enemies each day. Their resources have to be split up among all villages.
- **Ambiguity of triggers:** When a certain enemy shows up, it does not necessarily mean that always the same kind of enemy will follow. For example, if some troll scouts show up at a village, it does not always mean that the same troll army is about to attack. The players will have to take a look at the village's topological parameters, magical

alignment and history to find out which army they may be facing. Additionally, the player can send some scouts to find out more about the enemy.

- **Diversity of possible solutions:** Each enemy can be faced with many diverse defense tactics. Choosing the optimal one depends on the current situation of the village and the workload of the environment. Depending on the magical alignment of a village and the enemy army, different solutions may perform better or worse.

3.4 Comparing the game's adaption planning procedure to a real world FIPS

In Section 3.2 we presented the basic procedure of planning a defense strategy in our game. We showed what a player has to do as soon as he finds out that an enemy is attacking one of his villages, and how his plan is executed. In this section, we show how this basic procedure of planning a defense strategy for a certain village is similar to planning the upcoming actions in a real world FIPS. Table 1 shows the mapping between planning upcoming actions in a FIPS and in the game. This is shown by an example from the special needs school and compared to the actions a player has to take in the game.

Tab. 1: Mapping between the Tower Defense Game and a real world FIPS

Step	Special Needs School	Game Setting
Step 1: Trigger	teacher notices problems with a student's behavior	player sees attack of the village by some enemy
Step 2: Adaptions	teacher plans what to do to help his student	player plans the defenses for the next days
Step 3: Execution	teacher executes his plan	buildings send the planned units into battle
Step 4: Evaluation	teacher evaluates whether the problems have been solved or not	player gets feedback about the result of his defense

First, some problem has to be identified. In the special needs school, this is done by the teacher: Whenever he discovers some problems with one of his students, he has to do something about it. In our game, this is implemented as the arrival of a new army: The player knows he has to plan some kind of defense in order to defeat it. Following this initial analysis, some adaptions have to be planned: The teacher has to prepare some kind of support for the child, and the player has to plan his defense strategy. Third, the plans are executed, followed by the evaluation: In the special needs school, the teacher has to find out whether his plan has worked and his student's problems are solved or not. In the game setting, the player immediately sees whether the enemy has been defeated or not.

3.5 Service Based Architecture

The game architecture shown in Fig. 2 consists of several independent services which interact via RESTful interfaces. It consists of a *game server*, which provides the calculations for the fights and when which enemy army shows up where, a *process engine* which manages the defense process instances for the villages, a *logging service* which logs all information into the *data repositories* and a *game interface*. In the following, we will describe each service in detail.

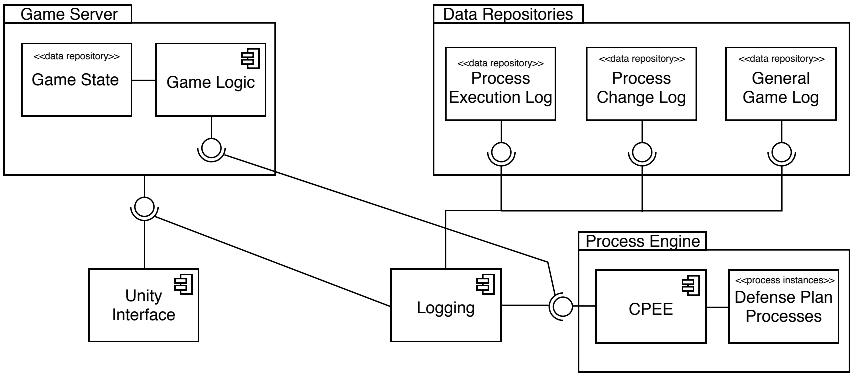


Fig. 2: Service Components and their Interactions

The Cloud Process Execution Engine (CPEE) [MR14] is a service-oriented process execution engine which itself is a RESTful web service. It manages the defense process instances and provides multiple interfaces for interacting with the process instance and logging various kinds of information.

The game server provides all calculations for the fights and the villages and stores all relevant information for the games which do not directly belong to the defense process instances. This includes among others the population, magical alignment and topography of the villages. It also communicates with the CPEE when the player updates the defense plan of a village and controls the enemy: Each round it calculates which villages of a player will be attacked by which enemy, and calculates the damage these enemies do.

The logging service generates the change and execution logs[5] for the defense plan processes which are provided by the CPEE. Additionally, it logs relevant information from the game server, which can be used to further understand and evaluate these change and execution logs. For example, the logging service logs when which enemy arrives at a certain village, how many units were lost during the fights, how the magical alignment shifts etc. In combination with the change and execution logs, the logging service provides a complete picture of each situation, which then can be used for further studies.

The user interface has been developed in Unity. It accesses the data and services provided by the game server web service over RESTful interfaces.

[5] Log format: XES, cf. http://www.xes-standard.org/

3.6 Generated Log Files

While players play the game, three types of log files are generated and updated for each village: The general log, the change log and the execution log. In this section we describe these log files in detail.

In the **general log** all information, ranging from the arrival of new enemies, over which defense units have been added, up to magical alignment shifts are shown. This log file sums up the development of a certain individual - on the one hand contingent on the decisions the player has made, on the other hand contingend on internal factors, on which the player has no influence (such as new enemy armies which arrive, population lost etc.). In our nursing example, this log file would contain some data about the development of the patient, maybe including data like his body temperature, weight, blood sugar level, or other parameters which are relevant for his treatment. Listing 1 shows an example general log of the village:

List. 1: General log example

```
<?xml version="1.0"?>
<log>
  <event id="1" type="started" day="" text="Village founded on day "/>
  <event id="2" type="newday" day="2" text="New day: 2"/>
  <event id="3" type="incoming" army="3" day="2" text="Army 3 incoming on day 2"/>
  <event id="4" type="alignmentshift" day="2" alignment="heat" shift="10" newvalue="70"/>
  <event id="5" type="defense" day="2" defensetype="weapon" defensename="Longbow" building="forge"/>
  <event id="6" type="defense" day="2" defensetype="weapon" defensename="Round Shield" building="forge"/>
  <event id="7" type="defense" day="2" defensetype="weapon" defensename="One Handed Sword" building="forge"/>
  <event id="8" type="defense" day="2" defensetype="unit" defensename="Knight" building="barracks"/>
</log>
```

The **change log** shows the process change operations the player has conducted while planning his defenses. These change operations represent the next steps which will be taken in order to save the village from the threat. Coming back to our nursing example, this log file represents the planned therapy steps which will be taken in the future in order to support the patient in dealing with a certain condition. Listing 2 shows an example for a change log from the game as it has been logged by the logging service. Here, the change shows the insertion of a new defense strategy for a specific village.

List. 2: Change log example

```
<?xml version="1.0"?>
<log xes.version="2.0" xes.features="arbitrary-depth">
  <extension name="Concept" prefix="concept" uri="http://www.xes-standard.org/concept.xesext"/>
  <extension name="Time" prefix="time" uri="http://www.xes-standard.org/time.xesext"/>
  <extension name="Change" prefix="change" uri="http://leonardo.wst.univie.ac.at/acaplan/change.xesext"/>
  <global scope="trace">
    <string key="concept:name" value=""/>
  </global>
  <classifier name="Change" keys="change:type change:subject"/>
  <trace>
    <string key="concept:name" value="Change Log of Village 1"/>
    <event>
      <date key="time:timestamp" value="2016-05-11 17:36:09 +0200"/>
      <date key="day" value="2"/>
      <int key="change:transaction" value="0"/>
      <string key="change:type" value="insert"/>
      <string key="change:position" value="root"/>
      <string key="change:fragment" value="17"/>
      <string key="change:rationale" value="troll scouts"/>
      <string key="change:goal" value="defeat army"/>
    </event>
  </trace>
</log>
```

The **execution log** sums up the process steps which have been executed in order to help the village to overcome the enemy army. These steps have been planned beforehand (as can be seen in the change log). In the nursing domain, there has to exist an execution log for each patient due to legal obligations: For each patient, each therapy step which has been executed has to be strictly logged in order to trace back any errors which have may been done if something goes wrong. Listing 3 depicts an execution log that reflects how the units are added to the villages defense.

List. 3: Execution log example

```
<?xml version="1.0"?>
<log xes.version="2.0" xes.features="arbitrary-depth">
  <extension name="Concept" prefix="concept" uri="http://www.xes-standard.org/concept.xesext"/>
  <trace>
    <event>
      <int key="day" value="1"/>
      <string key="concept:name" value="archer trained"/>
      <string key="unit" value="archer"/>
      <string key="building" value="barracks"/>
    </event>
    <event>
      <int key="day" value="1"/>
      <string key="concept:name" value="knight trained"/>
      <string key="unit" value="knight"/>
      <string key="building" value="barracks"/>
    </event>
    <event>
      <int key="day" value="1"/>
      <string key="concept:name" value="fireball created"/>
      <string key="spell" value="fireball"/>
      <string key="building" value="alchemylab"/>
    </event>
  </trace>
</log>
```

These logs provide valuable data sources for deriving insights on the effects of previously applied changes based on process analysis techniques. In this section we have shown some parallels between our log files and the nursing domain. In the next section, we will evaluate the log files with equivalent data from the software sales and support domain.

4 Evaluation

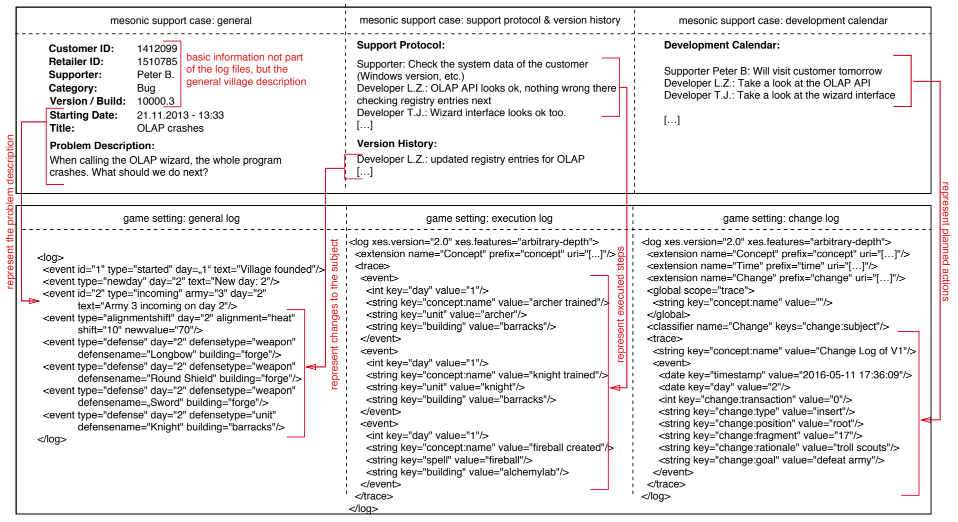
The goal of the game-based setting is to provide log data that are not subject to any disclosure restrictions and a test setting which can be used to develop algorithms and approaches which support users when adapting process instances in a FIPS. The different types of log files, namely change, execution and general logs, have been introduced in the last section. In this section we want to show that the information gathered in these log files are actually comparable to the set of information which is gathered in a real world FIPS. We claim that if the data generated in our game matches the data generated during an individual situation in a real world FIPS on a conceptual level, our data can be used to evaluate approaches and algorithms to support process adaptions in FIPS.

For our evaluation we analyzed support cases from mesonic[6], the ERP / CRM software developer introduced in the expert interviews in Section 2.2.1. In their *support network*, mesonic has over 200.000 different support cases describing the respective problem in free text form. Each support case is classified in one of four groups, ranging from bugs, where the

[6] Due to data privacy issues we only publish anonymized parts of the actual data gathered from these support cases in this paper. Since the original support cases are in German free text form, we shortened them and translated the relevant parts.

development team has some work to do, over wishes, where it has yet to be decided whether the desired feature will be implemented or not, over general support cases up to questions which are already answered in the manual. Depending on the case's classification, the next steps are decided: If it is a real bug or a wish, it will be discussed with the responsible developer or, if necessary, the whole team. If it is a general support case, it will be handeled in the support department internally. If it is something which can also be found in the manual, the responsible supporter handles the case on his own. This classification represents the first steps in the planning stage: Based on a cases "symptoms", the next steps are planned. In addition to this classification, information such as the customer's name and id, the retailers name and id, the version and build number of the product are logged.

Figure 3 summarizes the comparison of the data elements in the real world FIPS with the data elements in the game: The top section shows an example of such a support case: Here, some problems with the OLAP component have appeared which have been classified as a bug. On the bottom, example log files from the game are shown. The mappings between the data elements are indicated by the arrows. The general data of the support case (i.e. customer id, retailer id etc.) is not part of any log file, but static information which is saved separately.


mesonic support case: general
basic information not part of the log files, but the general village description
represent the problem description
mesonic support case: support protocol & version history
represent changes to the subject
represent executed steps
mesonic support case: development calendar
represent planned actions
game setting: general log
game setting: execution log
game setting: change log

Customer ID: 1412099
Retailer ID: 1510785
Supporter: Peter B.
Category: Bug
Version / Build: 10000.3
Starting Date: 21.11.2013 - 13:33
Title: OLAP crashes

Problem Description:
When calling the OLAP wizard, the whole program crashes. What should we do next?

Support Protocol:

Supporter: Check the system data of the customer (Windows version, etc.)
Developer L.Z.: OLAP API looks ok, nothing wrong there checking registry entries next
Developer T.J.: Wizard interface looks ok too.
[...]

Version History:
Developer L.Z.: updated registry entries for OLAP
[...]

Development Calendar:

Supporter Peter B: Will visit customer tomorrow
Developer L.Z.: Take a look at the OLAP API
Developer T.J.: Take a look at the wizard interface

[...]

```
<log>
  <event id="1" type="started" day=„1“ text="Village founded"/>
  <event type="newday" day="2" text="New day: 2"/>
  <event id="2" type="incoming" army="3" day="2"
          text="Army 3 incoming on day 2"/>
  <event type="alignmentshift" day="2" alignment="heat"
          shift="10" newvalue="70"/>
  <event type="defense" day="2" defensetype="weapon"
          defensename="Longbow" building="forge"/>
  <event type="defense" day="2" defensetype="weapon"
          defensename="Round Shield" building="forge"/>
  <event type="defense" day="2" defensetype="weapon"
          defensename=„Sword" building="forge"/>
  <event type="defense" day="2" defensetype="unit"
          defensename="Knight" building="barracks"/>
</log>
```

```
<log xes.version="2.0" xes.features="arbitrary-depth">
 <extension name="Concept" prefix="concept" uri="[...]"/>
 <trace>
  <event>
   <int key="day" value="1"/>
   <string key="concept:name" value="archer trained"/>
   <string key="unit" value="archer"/>
   <string key="building" value="barracks"/>
  </event>
  <event>
   <int key="day" value="1"/>
   <string key="concept:name" value="knight trained"/>
   <string key="unit" value="knight"/>
   <string key="building" value="barracks"/>
  </event>
  <event>
   <int key="day" value="1"/>
   <string key="concept:name" value="fireball created"/>
   <string key="spell" value="fireball"/>
   <string key="building" value="alchemylab"/>
  </event>
 </trace>
</log>
```

```
<log xes.version="2.0" xes.features="arbitrary-depth">
 <extension name="Concept" prefix="concept" uri="[…]"/>
 <extension name="Time" prefix="time" uri="[…]"/>
 <extension name="Change" prefix="change" uri="[…]"/>
 <global scope="trace">
  <string key="concept:name" value=""/>
 </global>
 <classifier name="Change" keys="change:subject"/>
 <trace>
  <string key="concept:name" value="Change Log of V1"/>
  <event>
   <date key="timestamp" value="2016-05-11 17:36:09"/>
   <date key="day" value="2"/>
   <int key="change:transaction" value="0"/>
   <string key="change:type" value="insert"/>
   <string key="change:position" value="root"/>
   <string key="change:fragment" value="17"/>
   <string key="change:rationale" value="troll scouts"/>
   <string key="change:goal" value="defeat army"/>
  </event>
 </trace>
</log>
```

Fig. 3: Mapping of the log files to a real world FIPS scenario

Data Source Comparison 1: The change log: At mesonic, next steps are planned with the responsible employees and shown in their calendars. In the game, the change log shows the planned steps for solving a village's problem (i.e. the incoming enemy army).

After the next steps for handling the case are planned, they are executed by the responsible employees. The executed steps are documented in the support case itself in free text format.

Data Source Comparison 2: The execution log: Mesonic's support case documents what has been done, who has done it, and when it has been done in order to solve the problem at hand. The execution log from our game represents the very same type of information: It is shown when which step has been executed in order to deal with the village's problem. This information includes the time, what has been done, and which role has executed the steps.

All actions influence the state of the product. These state changes are documented in the support protocol as well as the company's version control system.

Data Source Comparison 3: The general log: The version history of mesonic logs all information about changes to the product's source code. These changes may be because of actions which have been applied because of planned steps within the process, or factors not related to the support case. In the game, the general log summarizes all state changes of a village's parameters. These changes happen because of the player's actions, or because of other events which are not influenced directly by the player (but e.g., by the enemy army).

Conclusion: The comparison between the three generated log files from the game and the data gathered in a real world FIPS, namely the software sales and support setting, shows that all relevant information to a FIPS can be found in the game logs. In the real world setting, however, many data sets are free text, thus making further analysis difficult. Here we see a big potential of the game: The structured data in XES respectively XML can be easily used for further analysis, while representing all information relevant to a real world scenario.

5 Related Work

Virtual settings have already been used for process modeling. In [BRW11] the authors have created a 3D virtual world to enhance the collaborative modeling of business processes - even if the participants are not in geographical vicinity. By using avatars - 3D representations of humans who engage in the modeling of business processes, this approach offers various forms of communication which would be typically only possible if the participants were in geographical vicinity, i.e. speech, artifacts, gestures etc. [Ha16] presents another virtual world approach for generating process models. In contrast to other approaches, participants do not have to know any modeling language in order to generate a process model. Instead, they behave as they would do in the real world. Based on their actions in a realistic environment, a process model is generated.

While developing the game-based environment, we had to design the game mechanics according to the FIPS requirements, while still making the game fun to play. This balancing act is also an issue in serious games such as educational games [WK11], where fun game mechanics have to be balanced with a serious background [Mo08].

Data obtained from computer games, especially from strategy games, has been used for the development and validation of artificial intelligence approaches on multiple occasions. In [Av11], the authors show that tower defense games provide a good test environment for numerous computational intelligence scenarios, ranging from map generation, over enemy strategies, good defense strategies up to dynamic game balancing which keeps the player engaged and the enemies challenging. In [On13], the authors provide an overview over the possibilities of artificial intelligence in a real time strategy environment.

There exists a multitude of approaches on flexible process management [RW12]. First requirements for FIPS have been introduced in [Ka14]. This paper extends these requirements to a comprehensive list of building blocks for FIPS. Moreover, to the best of our knowledge, there is no approach that maps building blocks of FIPS to a game-based experimentation service. Logs produced by the experimentation service can be utilized by existing approaches on user support in changing business processes such as [Aa09; Gü08; KR15; We09]. However, developing user support techniques are outside the scope of this work. Adaptive Case Management (ACM) [MS13] also handles individual cases which evolve over time. In contrast to FIPS, where a process instance evolves around a certain subject and its environment, in ACM each case is handled individually.

6 Conclusion

FIPS can be found in many different domains, ranging from the nursing sector, over hotel guest and software customer interaction to special needs schools and the PhD program. Since all of these domains deal with highly sensitive data, it is almost impossible to get a full picture of a situation where such a process instance has to be adapted without running into data privacy and legal issues. For this reason we have developed a tower defense game where the players are put into a comparable situation: They adapt highly individual process instances in order to deal with problematic situations. The data thus generated can be used without any issues, may they be legal or otherwise. Additionally, the setting itself can be used as an evaluation method for support methodologies: A set of players who received support while adapting their process instances can be compared with a set of players who did not receive any support. If the supported group of players performs way better, it may be an indicator that the support is actually helpful. In the future we will use the game in order to develop and enhance support and analytical capabilities for FIPS. This includes analyzing change logs, finding out when the desired goals are reached, and, ultimately, finding the best adaptation for a given situation.

Acknowledgment This research has been funded by the Vienna Science and Technology Fund (WWTF) through project ICT15-072.

References

[Aa09] van der Aalst, W. et al.: Declarative workflows: Balancing between flexibility and support. Computer Science-Research and Development 23/2, pp. 99–113, 2009.

[Av11] Avery, P.; Togelius, J.; Alistar, E.; van Leeuwen, R. P.: Computational intelligence and tower defence games. In: 2011 IEEE Congress of Evolutionary Computation (CEC). Pp. 1084–1091, June 2011.

[BRW11] Brown, R.; Recker, J.; West, S.: Using virtual worlds for collaborative business process modeling. Business Process Management Journal 17/3, pp. 546–564, 2011.

[Gü08] Günther, C.; Rinderle-Ma, S.; Reichert, M.; van Der Aalst, W.; Recker, J.: Using process mining to learn from process changes in evolutionary systems. International Journal of Business Process Integration and Management 3/1, pp. 61–78, 2008.

[Ha16] Harman, J.; Brown, R.; Johnson, D.; Rinderle-Ma, S.; Kannengiesser, U.: Augmenting process elicitation with visual priming: An empirical exploration of user behaviour and modelling outcomes. Information Systems/, 2016.

[Ka14] Kaes, G.; Rinderle-Ma, S.; Vigne, R.; Mangler, J.: Flexibility Requirements in Real-World Process Scenarios and Prototypical Realization in the Care Domain. In: OTM 2014 Workshops. Pp. 55–64, 2014.

[KK97] Kueng, P.; Kawalek, P.: Goal-based business process models: creation and evaluation. Business Process Management Journal 3/1, pp. 17–38, 1997.

[KR15] Kaes, G.; Rinderle-Ma, S.: Mining and Querying Process Change Information Based on Change Trees. In: Service-Oriented Computing. Pp. 269–284, 2015.

[Mo08] Moreno-Ger, P.; Burgos, D.; Martínez-Ortiz, I.; Sierra, J. L.; Fernández-Manjón, B.: Educational game design for online education. Computers in Human Behavior 24/6, pp. 2530–2540, 2008.

[MR14] Mangler, J.; Rinderle-Ma, S.: CPEE - Cloud Process Exection Engine. In: Int'l Conference on Business Process Management. CEUR-WS.org, Sept. 2014.

[MS13] Motahari-Nezhad, H. R.; Swenson, K. D.: Adaptive Case Management: Overview and Research Challenges. In: 2013 IEEE 15th Conference on Business Informatics. Pp. 264–269, July 2013.

[On13] Ontañón, S.; Synnaeve, G.; Uriarte, A.; Richoux, F.; Churchill, D.; Preuss, M.: A Survey of Real-Time Strategy Game AI Research and Competition in StarCraft. Computational Intelligence and AI in Games 5/4, pp. 293–311, Dec. 2013, ISSN: 1943-068X.

[RW12] Reichert, M.; Weber, B.: Enabling Flexibility in Process-Aware Information Systems - Challenges, Methods, Technologies. Springer, 2012.

[Sc08] Schonenberg, H.; Mans, R.; Russell, N.; Mulyar, N.; van der Aalst, W.: Process flexibility: A survey of contemporary approaches. In: Advances in Enterprise Engineering I. Springer, 2008, pp. 16–30.

[We09] Weber, B.; Reichert, M.; Rinderle-Ma, S.; Wild, W.: Providing Integrated Life Cycle Support in Process-Aware Information Systems. International Journal of Cooperative Information Systems 18/01, pp. 115–165, 2009.

[WK11] Wallner, G.; Kriglstein, S.: Design and Evaluation of the Educational Game DOGeometry: A Case Study. In. ACE '11, ACM, Lisbon, Portugal, 14:1–14:8, 2011, ISBN: 978-1-4503-0827-4.